НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук

_____/Голуб Б.Л., доц., к.т.н. /
підпис ПІБ, вчене звання і ступінь

«_04_»__червня_____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмне забезпечення автоматизованої системи по обліку мешканців населеного пункту»

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми
_____к.т.н., доцент_____ / Голуб Б.Л./
Науковий ступень та вчене звання підпис ПІБ

Керівник бакалаврської кваліфікаційної роботи :_____ / Панкратьєв В.О/
підпис ПІБ

Виконала:_____ / Величко К.В./
підпис ПІБ

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерних наук

_____ / Голуб Б.Л., доцент, к.т.н /
підпис

"16_" грудня 2024 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студентці Величко Кірі Валентинівні

Спеціальність 121 «Інженерія програмного забезпечення»

1. Тема роботи: Програмне забезпечення автоматизованої системи по обліку мешканців населеного пункту

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

2. Термін подання завершеної роботи на кафедру 2025 . 05 . 25
рік, місяць, число

3. Вихідні дані до роботи: опис програмного забезпечення

4. Перелік питань що розглядаються:

1. Аналіз проблемної області.
2. Вибір та обґрунтування засобів для розробки системи.
3. Проектування інформаційної системи.
4. Висновки.

Керівник бакалаврської кваліфікаційної роботи_____ / Панкратьєв В.О.
підпис ініціали та прізвище

Завдання прийняла до виконання _____ / Величко К.В. /
підпис ініціали та прізвище

Дата отримання завдання 2024 . 12 . 16
рік, місяць, число

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмний застосунок;

ORM – Object-relational mapping;

SQL – Structured Query Language;

СУБД – Система управління базами даних;

ПІБ – Прізвище, ім'я, по батькові;

ІТ – Інформаційні технології;

CRM – Customer relationship management;

UML – Unified Modeling Language;

TCP – Transmission Control Protocol.

ВСТУП

У сучасному світі, цифровізація поступово охоплює усі сфери людської діяльності, зростає потреба в ефективних інформаційних системах, здатних забезпечувати швидкий доступ до актуальної інформації, її зручне оброблення та надійне зберігання. Ця проблема важлива в контексті місцевого самоврядування, де необхідність точного й оперативного обліку мешканців населених пунктів стає важливою передумовою для прийняття обґрунтованих управлінських рішень, реалізації соціальних програм, організації виборчих процесів, планування ресурсів, обліку пільгових категорій громадян тощо. Традиційні паперові форми або розрізнені табличні файли вже не відповідають вимогам часу, не забезпечують належного рівня захисту даних, а також є надзвичайно ресурсоємними для обслуговування. Тому розробка ПЗ для автоматизованого обліку мешканців населеного пункту є актуальною.

Мета дослідження полягає у створенні надійного, функціонального та масштабованого програмного забезпечення, яке дозволяє здійснювати повний облік мешканців певного населеного пункту з можливістю внесення, редагування, пошуку та аналітичної обробки даних.

У процесі розробки програмного забезпечення буде застосовано сучасні технології та інструменти, а саме, мову програмування C# разом з платформою .NET, що дозволяє досягти високої продуктивності та стабільності системи. Для взаємодії з базою даних буде застосовано ORM-технологію Entity Framework, яка спростить роботу з даними та забезпечить гнучкість архітектури. Система керування базами даних Microsoft SQL Server відповідатиме за збереження структурованої інформації з високим рівнем захисту та масштабованості. Інтерфейс користувача планується реалізувати на базі Windows Forms.

Після завершення реалізації системи передбачено її апробацію шляхом моделювання умов роботи в межах умовного населеного пункту. Під час

тестування буде перевірено усі ключові функціональні можливості системи, оцінено стабільність роботи, зручність використання та відповідність поставленим вимогам.

Структура програмного продукту включатиме декілька основних модулів: модуль аутентифікації користувачів, модуль обліку мешканців, модуль аналітики та звітності, модуль резервного копіювання та адміністративний інтерфейс.

У першому розділі дипломної роботи представлено системний аналіз предметної області, включаючи опис її особливостей, аналіз вимог та існуючих рішень, формулювання завдання та побудову моделі. У другому розділі розглядається логічне проєктування даних, обґрунтування вибору СУБД і створення структури інформаційної бази. Третій розділ присвячено прикладному рівню – вибору інструментів, програмуванню модулів, опису реалізації алгоритмів. У четвертому розділі розглядаються питання тестування, технічних вимог до середовища експлуатації та опису інсталяційного пакету.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Облік мешканців населеного пункту є однією з ключових функцій, що забезпечують ефективне функціонування системи місцевого самоврядування. У межах цієї діяльності збираються, зберігаються, обробляються та аналізуються дані про фізичних осіб, які проживають на відповідній території. Ці дані використовуються для цілого спектру завдань – від планування бюджету громади до формування соціальних програм, організації виборчих процесів, надання адміністративних послуг та інформування державних органів вищого рівня. Предметна область охоплює широкий набір процедур, починаючи від реєстрації нового мешканця і завершуючи зняттям з обліку у разі зміни місця проживання або інших юридичних підстав.

У більшості невеликих населених пунктів така система функціонує або у вигляді паперових картотек, або ж ведеться у форматі електронних таблиць, які створюються і оновлюються вручну працівниками сільських або селищних рад. Подібний підхід має суттєві недоліки – обмежений функціонал пошуку та фільтрації, висока ймовірність помилок під час ручного введення даних, складнощі із забезпеченням конфіденційності інформації, відсутність централізованого доступу та синхронізації даних. У великих містах використовуються більш складні програмні рішення, однак вони часто є комерційними, дорогими або мають надлишкову функціональність, що ускладнює їхнє впровадження у малих громадах.

Облік мешканців – статистична процедура та важливий інструмент соціальної політики. Знання точної кількості мешканців, їхнього вікового складу, соціального статусу, місця працевлаштування чи спеціальних потреб дозволяє органам влади більш точно прогнозувати потреби громади [1]. Це впливає на розподіл державних субвенцій, наповнення бюджету, розвиток інфраструктури, забезпечення освітніх та медичних послуг.

Інформаційні ресурси предметної області можуть включати персональні дані мешканців – ПІБ, дату народження, адресу проживання, контактні дані, паспортну інформацію та інше [2]. Робота з такою інформацією вимагає обережності та дотримання чинного законодавства України у сфері захисту персональних даних. Тому ключовими вимогами до майбутньої системи є функціональність, забезпечення конфіденційності, контроль доступу, а також можливість створення резервних копій.

З огляду на специфіку діяльності органів місцевої влади, система обліку має бути адаптована до реальних умов експлуатації: мати простий та інтуїтивно зрозумілий інтерфейс, не потребувати високої кваліфікації персоналу для роботи, забезпечувати швидкий доступ до потрібної інформації, а також містити інструменти для генерації звітів. Ураховуючи вищенаведене, предметна область обліку мешканців потребує впровадження зручного та ефективного програмного рішення, яке відповідатиме сучасним стандартам і зможе забезпечити автоматизацію всіх основних функцій із мінімальними витратами ресурсів та зусиль з боку користувача.

Це програмне забезпечення має об’єднати в собі базові операції – реєстрацію, перегляд, редагування, видалення записів та генерацію звітів.

1.2 Аналіз вимог до програмної системи

Програмна система повинна відповідати реальним потребам місцевих адміністрацій, забезпечуючи зручний, безпечний та ефективний інструмент для роботи з персональними даними. Основною вимогою є автоматизація ключових операцій: реєстрації, редагування та перегляд інформації про мешканців. Програма має бути простою у використанні, мати зрозумілий інтерфейс та не вимагати високої технічної підготовки від користувачів.

Також, підтримувати зберігання великого обсягу даних з можливістю формування звітів. Обов’язковою є наявність механізмів резервного копіювання та захисту інформації. Програмне забезпечення повинно

працювати локально без обов’язкового підключення до Інтернету, що особливо важливо для невеликих сільських громад.

1.3 Огляд інформаційних джерел та існуючих рішень

У процесі розробки програмного забезпечення важливим етапом є дослідження інформаційних джерел, нормативно-правової бази та аналіз уже існуючих програмних рішень, які використовуються для виконання аналогічних або близьких за функціональністю завдань. Таке дослідження дозволяє визначити сильні та слабкі сторони наявних систем, оцінити ступінь їх відповідності сучасним вимогам і сформувати обґрунтовані технічні рішення для майбутнього продукту.

Основними джерелами інформації при вивченні предметної області стали нормативні документи, зокрема Закон України «Про захист персональних даних», який регламентує порядок збору, зберігання, обробки та використання інформації про фізичних осіб [3], а також Закон України «Про місцеве самоврядування», що визначає повноваження сільських, селищних і міських рад у частині ведення обліку мешканців [4].

На українському ринку програмних рішень для обліку мешканців можна виділити декілька систем, які активно використовуються в практиці органів місцевого самоврядування. Деякі з них мають державне походження або створені у співпраці з Мінцифрою, інші є приватними розробками, адаптованими під потреби громад. Найбільш поширеними системами є: «Соціальна громада», «Єдина інформаційна система населення» (ЄІСН) [5], «M.E.Doc» у специфічних конфігураціях [6], а також спеціалізовані рішення на базі 1С:Підприємство [6]. Ці продукти реалізують частково або повністю функціонал, необхідний для демографічного обліку, хоча мають суттєві відмінності у вартості, зручності, технічній реалізації та підтримці.

Система «Соціальна громада» є однією з найвідоміших в Україні і розроблена на замовлення Міністерства соціальної політики. Вона передбачає

ведення обліку різних категорій громадян, зокрема осіб, що потребують соціальної підтримки. Однак її функціонал орієнтований насамперед на соціальну сферу, а не на загальний облік мешканців. Тим не менш, система підтримує реєстрацію, зберігання, оновлення персональних даних, а також формування звітності, що робить її важливою у контексті вивчення аналогів.

Єдина інформаційна система населення (ЄІСН) розроблена для державного використання з метою централізації даних про громадян. Вона забезпечує облік мешканців, інтеграцію з іншими реєстрами, має сучасну архітектуру, однак є закритою для безпосереднього доступу з боку окремих територіальних громад і може бути використана лише у рамках взаємодії з державними структурами.

Рішення на базі 1С:Підприємство зустрічаються в багатьох громадах, однак вони зазвичай є результатом кастомізації стандартних конфігурацій бухгалтерського обліку. Такі системи здатні вести довідники, формувати звіти, зберігати великі масиви інформації, однак часто мають громіздкий інтерфейс, потребують спеціальної підготовки користувачів і ліцензування, що є бар’єром для малих громад із обмеженим бюджетом.

Також деякі громади використовують CRM-подібні рішення або прості Microsoft Access-бази, створені власними IT-фахівцями. Проте такі підходи майже завжди супроводжуються проблемами масштабування, відсутністю технічної підтримки та нестачею документації. Подібні рішення рідко відповідають вимогам до захисту персональних даних, оскільки не мають вбудованих механізмів контролю доступу, шифрування, журналювання дій тощо.

Для систематизації наведеного вище аналізу доцільно представити порівняльну таблицю (табл. 1), яка демонструє особливості найвідоміших вітчизняних систем обліку мешканців за критеріями, що є критичними з огляду на потреби дипломного проєкту.

Таблиця 1.

Порівняльна таблиця найвідоміших вітчизняних систем обліку мешканців

Назва системи	Орієнтація системи	Простота використання	Можливість кастомізації	Безпека персональних даних	Вартість впровадження	Наявність підтримки
Соціальна громада	Соціальна політика громади	Висока	Низька	Високий рівень	Безкоштовна (державна)	Так
ЄІСН	Централізований держреєстр	Висока	Відсутня	Дуже високий рівень	Немає даних	Так
1С: Підприємство (кастом)	Універсальна система	Середня	Висока	Середній рівень	Висока	Частково
Локальні Access-бази	Тимчасові рішення громад	Середня/Низька	Залежить від розробника	Низький рівень	Низька	Відсутня

Проведений аналіз демонструє, що жодна з наявних систем не є універсальною і повністю адаптованою до потреб кожної конкретної громади. Державні рішення мають обмежений доступ, приватні – надмірну складність або високу вартість, а локальні ініціативи часто є нестійкими в експлуатації. Це підтверджує доцільність створення нового програмного забезпечення, яке врахує реальні потреби адміністрацій, забезпечить простоту, безпеку та відповідність українським нормативним вимогам.

1.4 Постановка завдання

У межах предметної області підлягає обробці широкий спектр інформації, що включає персональні дані мешканців (прізвище, ім’я, по батькові, дату народження, стать), адресу проживання, контактні дані та інші уточнюючі відомості, необхідні для прийняття управлінських рішень на рівні місцевого самоврядування.

Операції, які повинні бути автоматизовані у межах системи, охоплюють реєстрацію нового мешканця, редагування наявної інформації, видалення

записів з відповідним підтвердженням, а також формування звітів. Ці звіти повинні мати можливість збереження у форматі PDF.

Система, яку передбачається реалізувати, належатиме до класу інформаційно-аналітичних систем локального рівня. За характером використання інформації вона належить до систем оперативного введення, накопичення та обробки персоніфікованих даних з подальшим формуванням звітів і наданням результатів у вигляді довідок і аналітичних матеріалів. За масштабом система розрахована на використання в межах однієї адміністративної одиниці – сільської, селищної або міської ради, з можливістю подальшого масштабування до рівня об’єднаної територіальної громади.

Система повинна працювати у настільному форматі з можливістю автономного використання без постійного підключення до мережі Інтернет. Це обумовлено специфікою роботи місцевих адміністрацій у сільській місцевості, де стабільне інтернет-з’єднання не завжди доступне. Комунікаційні технології, що підтримуються, мають включати локальні з’єднання для обміну даними між робочими місцями, а також можливість експорту й імпорту інформації через зовнішні носії або файли.

До системи висуваються наступні вимоги: зручний і логічно побудований інтерфейс користувача, що не потребує тривалого навчання; багаторівнева система доступу з можливістю розмежування прав користувачів; можливість резервного копіювання та відновлення даних; підтримка історії змін записів; висока продуктивність при роботі з великими обсягами даних; сумісність із сучасними операційними системами Windows; можливість легкої адаптації до змін у законодавстві чи адміністративних регламентах.

1.5 Моделювання предметної області

У процесі розробки будь-якого програмного забезпечення постає необхідність не лише створити функціональний продукт, а й сформувати комплексну модель системи, яка б з різних сторін описувала всі етапи та

учасників процесу, характер даних, потоки інформації, послідовність дій користувачів і логіку функціонування системи. На основі аналізу предметної області визначено, що система повинна автоматизувати цілу низку операцій, пов’язаних із реєстрацією мешканців, збереженням їхніх даних, підготовкою звітів. Для досягнення такого рівня формалізації проєкту необхідно реалізувати декілька типів моделей, кожна з яких надає окремий погляд на структуру та логіку функціонування майбутнього програмного продукту.

Діаграма прецедентів (Use Case Diagram), яку можна побачити нижче, показує що основні користувачі взаємодіють з процесами у реальному житті до запровадження автоматизованою системою. Вона демонстру шаблонні сценарії взаємодії мешканців та співробітників сільради з інформацією для обліку та допомагає при визначенні функціональних потреб майбутної системи [7].

Побудова такої діаграми дозволяє зрозуміти поточний стан речей, окреслити ролі користувачів та проаналізувати, які функції слід автоматизувати. Прецеденти представлені у вигляді еліпсів, а актори — у вигляді абстрактних фігур, що символізують реальних учасників процесу. Такий підхід дозволяє сформувати основу для подальшого проектування функціональності системи відповідно до реальних потреб.

На діаграмі прецедентів чітко візуалізовано основні взаємодії трьох ключових акторів: мешканця, секретаря та голови сільської ради (рис. 1.1). Мешканець виступає ініціатором дії, приносячи документи на реєстрацію або зміну даних, а також отримуючи готові документи. Секретар приймає документи, вносить інформацію в архіви, готує документи, формує звіти, виконує пошук в архівах. Голова сільради переглядає, формує, затверджує звіти і дає розпорядження щодо змін.

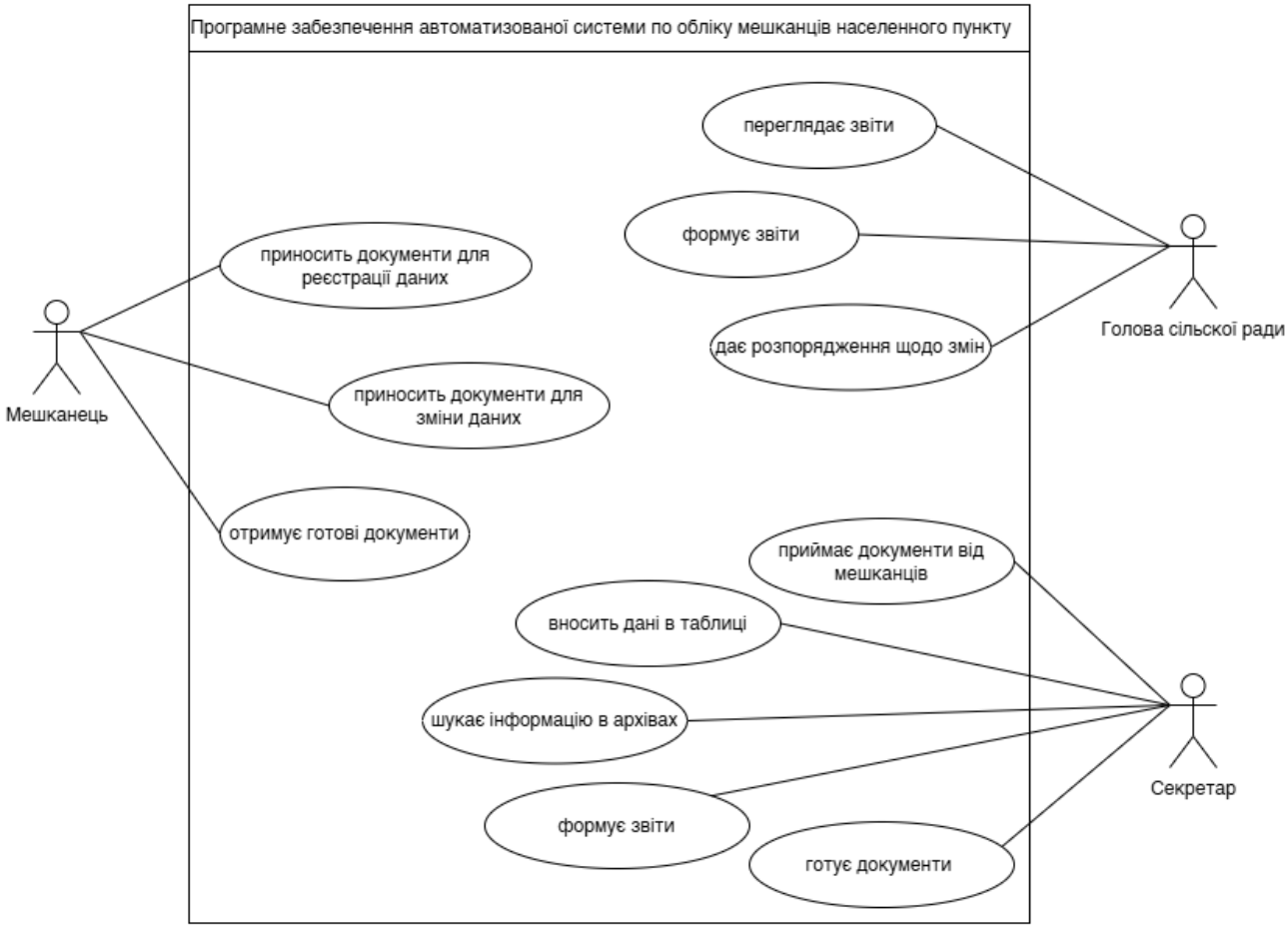


Рис. 1.1 Діаграма прецедентів

Дана діаграма прецедентів дозволяє формалізувати функціональні вимоги до програмного забезпечення та чітко розмежувати повноваження між користувачами різних рівнів доступу. Акторам надано лише ті функції, які відповідають їх реальним повноваженням у контексті управління населеним пунктом.

Прецеденти, пов’язані із запитом, переглядом, формуванням і затвердженням звітів, дозволяють забезпечити регулярний контроль стану справ, а розпорядження щодо змін виступає механізмом ініціації оновлень даних, що відображає управлінську роль голови.

Секретар, як основний виконавець дій, взаємодіє з найбільшою кількістю сценаріїв. Внесення даних у таблиці, пошук архівної інформації та формування вихідних документів – це операції, які потребують відповідної кваліфікації й

чітко налагоджених бізнес-процесів. Секретар також виступає зв’язковою ланкою між мешканцем і таблицями, де зберігається вся інформація.

Діаграма послідовності (Sequence Diagram) – динамічна UML-діаграма, що відображає порядок взаємодії об’єктів у рамках певного сценарію. Вона зображає об’єкти, розміщені по горизонталі, та повідомлення, які передаються між ними у хронологічному порядку, відображеному по вертикалі [8]. Така діаграма дозволяє чітко показати часову послідовність викликів методів, обмін повідомленнями та порядок обробки подій. Вона особливо корисна під час моделювання складної логіки взаємодії між компонентами, коли необхідно з’ясувати, які елементи ініціюють дії, як обробляються відповіді та які залежності виникають між об’єктами.

Діаграма послідовності доповнюватиме загальне уявлення про обмін повідомленнями між основними користувачами до впровадження системи шляхом ясного визначення порядку подій. Це дозволятиме глибше розглянути існуючі процеси та сформувати повне уявлення про функціонування системи у майбутньому на підставі інформацій з того, як цей процес виглядав раніше (рисунок 1.2). Серед ряду кроків - від надходження документів від мешканця до їх обробки і видачею секретарем або формуванню і схваленню звіту - показано логічний ланцюжок взаюмодій між мешканцем, секретарем, головою сільради та таблицею.

Завдяки використанню діаграми послідовності, можна детально простежити логіку роботи системи та зрозуміти часову залежність між запитами та реакціями компонентів.

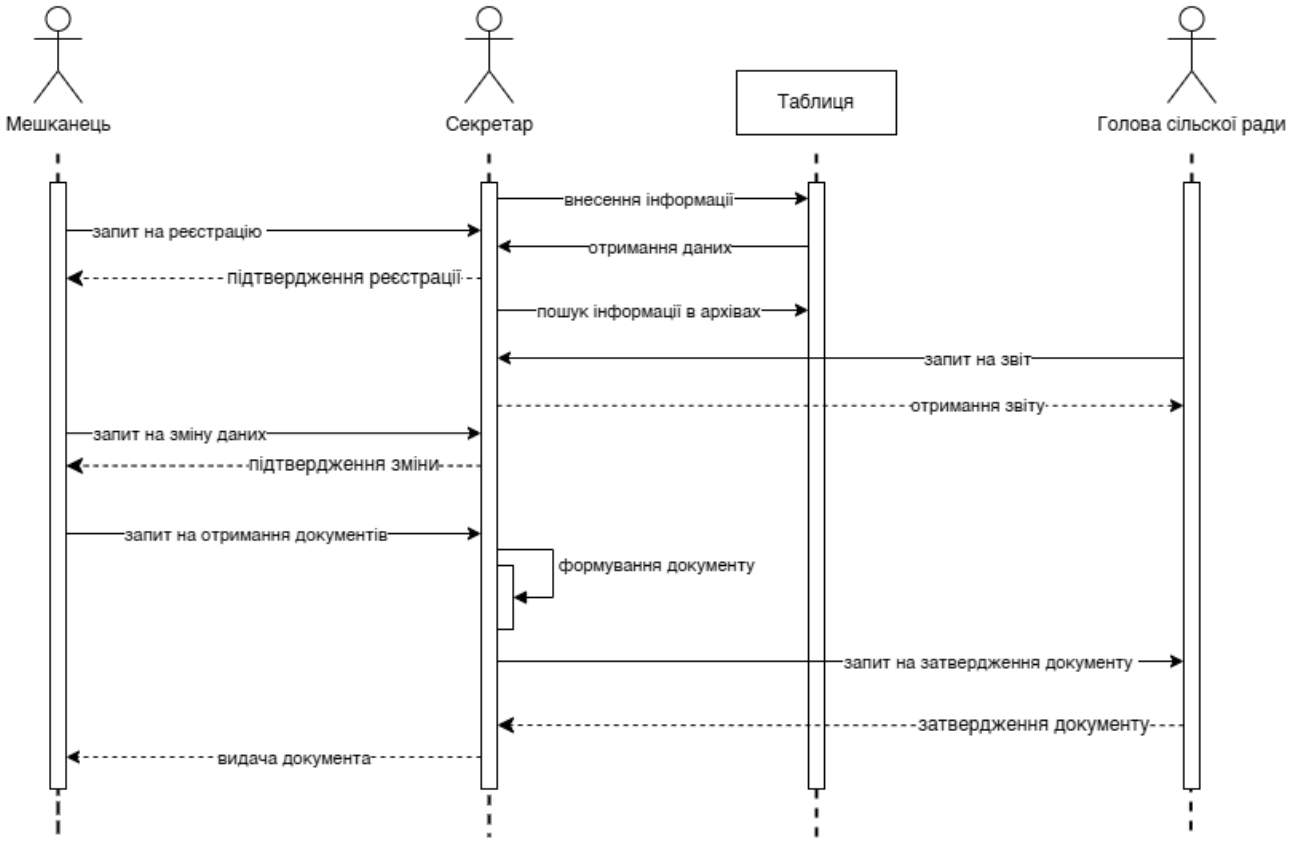


Рис. 1.2 Діаграма послідовності

На даній схемі помітна централізована роль секретаря як проміжної ланки, що координує майже всі дії між мешканцем, внутрішніми функціями та головою сільської ради. Це вказує на необхідність ретельного опрацювання інтерфейсу для цього користувача, забезпечення високої зручності й мінімізації рутинних операцій. З технічної точки зору, така діаграма дозволяє проєктувальникам і розробникам чітко визначити, у який момент мають викликатися ті чи інші функції додатку, як обробляються запити на сервері, і коли відбувається запис або оновлення даних у базі.

На основі цієї діаграми можна розробити сценарії автоматизованого тестування – зокрема, перевірку коректності передачі даних між об’єктами, часових затримок, послідовності викликів функцій і валідації відповіді системи на нетипові дії користувача.

Діаграма активності (Activity Diagram) використовується для моделювання бізнес-процесів або алгоритмів, що мають логіку розгалуження, циклів і паралельного виконання дій. Вона є своєрідним аналогом блок-схеми,

однак у контексті об’єктно-орієнтованого підходу та UML [9]. Кожна дія зображується як прямокутник із заокругленими кутами, а потоки керування між діями – стрілками. Також діаграма може містити умовні вузли, розгалуження, об'єднання та маркери паралельного виконання. Основною перевагою є її здатність наочно відображати логіку дій, прийняття рішень та умови, за яких відбувається перехід між етапами. Вона широко використовується для моделювання як і дій до створення самої системи так і для користувацької взаємодії або внутрішніх процесів системи.

Діаграма активності дає змогу побачити загальну логіку процесу від подачі документів до їх отримання або проходження затвердження (рис. 1.3). Вона дозволяє чітко виділити розгалуження між реєстрацією нових даних і зміною вже наявної інформації, а також визначити, в якому випадку процес вимагає схвалення з боку голови сільської ради.

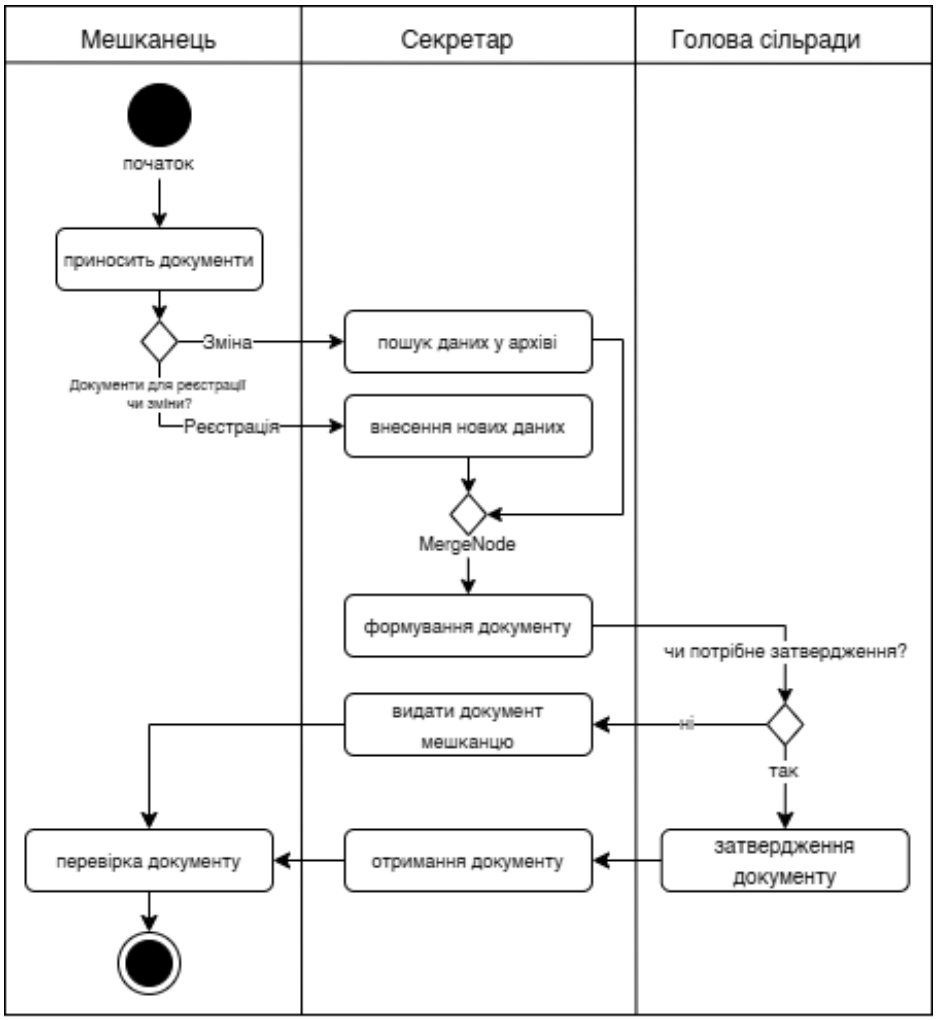


Рис. 1.3 Діаграма активності

Діаграма класів (Class Diagram) – фундаментальна структурна діаграма у UML, яка відображає статичну структуру предметної області. Вона відображає класи, їх атрибути, методи, а також зв’язки між ними, включаючи асоціації, агрегації, композиції та спадкування [10]. Діаграма показує основні сутності, які приймають участь у документообігу в конкретно декількох основних сценаріїв, які виконуються в межах сільської ради, а також зв’язки між ними. З цим можна краще зрозуміти принцип роботи до введення інформаційної системи та використовувати таку модель як основу для майбутнього проектування, що допомагають розробникам, тестувальникам та технічним письменникам краще розуміти структуру майбутньою системи.

Діаграма класів (рис. 1.4) дає уявлення про основні сутності, які будуть реалізовані в програмному забезпеченні, та зв’язки між ними. Визначено такі класи: «Мешканець», «Документ», «Секретар», «Голова сільради», «Звіт», «Архів». Кожен клас має відповідні атрибути та методи, які відображають логіку його роботи в предметній області. Зв’язки між класами побудовано відповідно до реальних бізнес-процесів: мешканці подають документи, секретарі вносять інформацію, секретар або голова сільради формує звіт, документи передаються на затвердження.

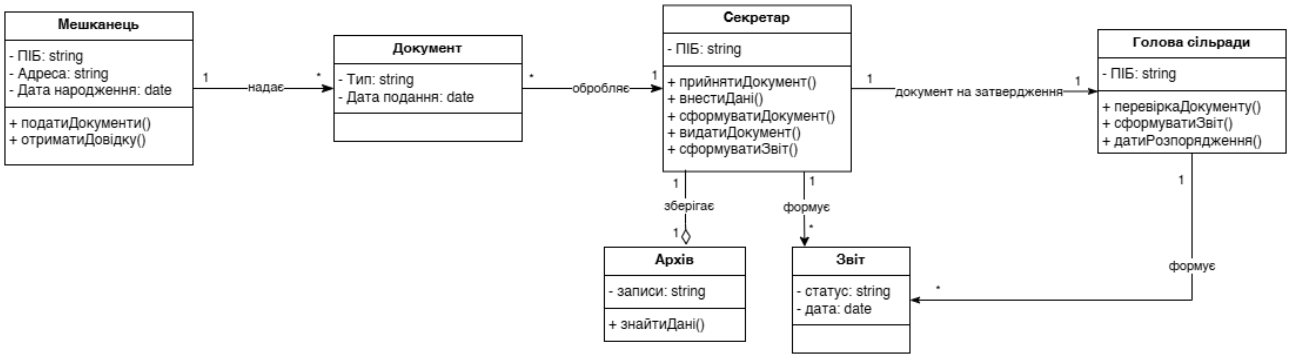


Рис. 1.4 Діаграма класів

Аналіз діаграми класів дозволяє глибше зрозуміти логіку взаємодії компонентів системи. Зокрема, видно, що об'єкти класу «Документ» мають

множинний зв'язок як з мешканцем, так і з секретарем. Це означає, що один мешканець може подавати кілька документів різного типу, кожен з яких підлягає перевірці й подальшій обробці секретарем.

Методи класів чітко структуровані відповідно до обов’язків реальних посадових осіб. Наприклад, метод датиРозпорядження() у класі «ГоловаСільради» дозволяє моделювати процес прийняття управлінських рішень на основі затверджених звітів.

Зв'язок класу «Звіт» із класом «Архів» також є концептуально важливим: кожен створений звіт може бути збережений у архіві для подальшого пошуку або перевірки. Наявність атрибута статус у звіті дозволяє відслідковувати етапи його життєвого циклу (чернетка, на перевірці, затверджено), що є необхідним для реалізації ролей та контролю доступу до функцій системи.

Варто підкреслити, що дана діаграма класів не лише визначає як відбувалися процеси надання документів, формування звітів, видача документів, та інше, а ще їй допомагає зрозуміти як в майбутньому треба реалізувати структуру програмного коду та створення бази даних.

Діаграма абстракції (або, її також називають діаграмою узагальнення) дозволяє представити відношення між більш загальними і більш конкретними сутностями в моделі. У контексті UML це часто реалізується через механізм спадкування, коли базовий клас або інтерфейс задає спільну поведінку або структуру, яку успадковують похідні класи [11]. Така діаграма використовується для демонстрації принципів поліморфізму та інкапсуляції. Вона сприяє зменшенню дублювання коду і покращує повторне використання компонентів. Абстрактні класи, зазвичай, не мають повної реалізації, а лише визначають набір обов’язкових методів для реалізації у підкласах, що дозволяє створювати гнучкі, розширювані системи з чітко визначеними ієрархіями.

Діаграма абстракції відображає основні об’єкти обліку мешканців та їхні ролі (рис 1.5). Шість ключових суб’єктів: мешканець, секретар, голова сільської ради, документ, звіт і інформація обліку мешканців. Мешканець має базові дані (ПІБ, адреса, дата народження) і подає документи на реєстрацію або зміну, а

також отримує документи. Секретар виконує основні дії: приймає документи, шукає дані в архіві, вносить і редагує інформацію, готує документи та формує звіти. Голова сільради формує звіти та дає розпорядження. Інформація обліку мешканців зберігає всі дані в архіві або таблицях та надає інформацію для формування звітів або зміни інформації.

Рис. 1.5 Діаграма абстракції

Діаграма розгортання (Deployment Diagram) відображає фізичне розміщення програмних компонентів на апаратних вузлах системи. Вона корисна на етапах проєктування і реалізації, коли потрібно визначити, як саме система буде працювати у реальному середовищі – на серверах, клієнтських машинах, мобільних пристроях тощо [12]. На діаграмі показуються вузли (сервери, пристрої), з’єднання між ними, а також компоненти програмного забезпечення, які будуть встановлені на кожен з них. Така візуалізація дозволяє оцінити вимоги до інфраструктури, визначити потенційні вузькі місця і спланувати розгортання з урахуванням масштабування та резервування.

Діаграма розгортання описує фізичну модель функціонування програмного забезпечення. У системі беруть участь два користувачі: секретар та голова сільради, які працюють на окремих ПК, підключених до єдиного

SQL-сервера локальної мережі (рис. 1.6). Обмін даними між додатками йде через прямі TCP-з’єднання.

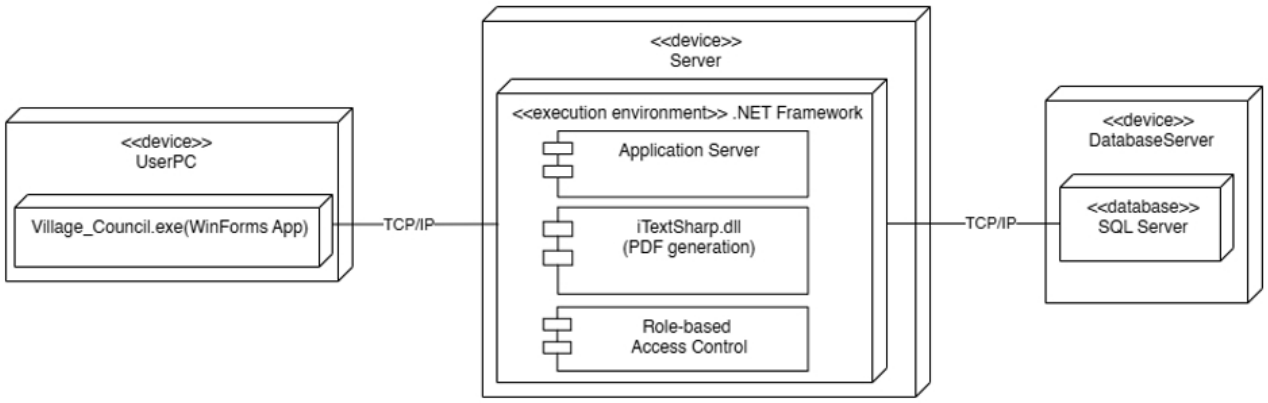


Рис. 1.6 Діаграма розгортання

У представленій конфігурації система використовує трирівневу архітектуру, яка чітко відокремлює клієнтську частину, сервер прикладної логіки та сервер бази даних. Це дозволяє досягти високого рівня масштабованості, підтримуваності та безпеки. На стороні користувача (<<device>> UserPC) розміщується десктопний застосунок Village_Council.exe, розроблений у середовищі Windows Forms. Цей застосунок не містить у собі бізнес-логіки або модулів для безпосереднього доступу до бази даних, а натомість взаємодіє з прикладним сервером через протокол TCP/IP.

Прикладний сервер (<<device>> Server), виконаний на платформі .NET Framework, забезпечує обробку запитів, реалізацію ділової логіки, авторизацію користувачів відповідно до їх ролей, а також генерування звітної документації у форматі PDF за допомогою бібліотеки iTextSharp.dll.

Блок Role-based Access Control відповідає за реалізацію механізму контрольованого доступу до функціональності системи. В залежності від ролі користувача (секретар або голова сільради), накладаються обмеження на перелік доступних функцій, операцій редагування або перегляду конфіденційної інформації. Дана політика гарантує відповідність системи внутрішнім регламентам органів місцевого самоврядування.

Окремим елементом архітектури виступає сервер бази даних (<<device>> DatabaseServer), на якому розгорнуто SQL Server. Цей компонент зберігає всю структуровану інформацію про село, мешканців, підприємства, освітні установи, звіти та логіни користувачів. Комунікація між прикладним сервером і базою даних також здійснюється через захищений TCP/IP канал, що забезпечує стабільність і надійність під час передавання даних.

Модель бази даних фізичного рівня – це опис структури бази даних з урахуванням специфіки обраної СУБД. Вона включає таблиці, стовпці, типи даних, індекси, ключі (первинні та зовнішні), обмеження цілісності, а також взаємозв’язки між таблицями [13]. На відміну від логічної моделі, яка більше зосереджена на концептуальних сутностях, фізична модель враховує реальні технічні параметри зберігання даних, оптимізацію запитів та забезпечення продуктивності. Її створення є критичним кроком для побудови ефективної, масштабованої бази даних, яка відповідатиме як вимогам додатку, так і обмеженням апаратного середовища (рис. 1.7).

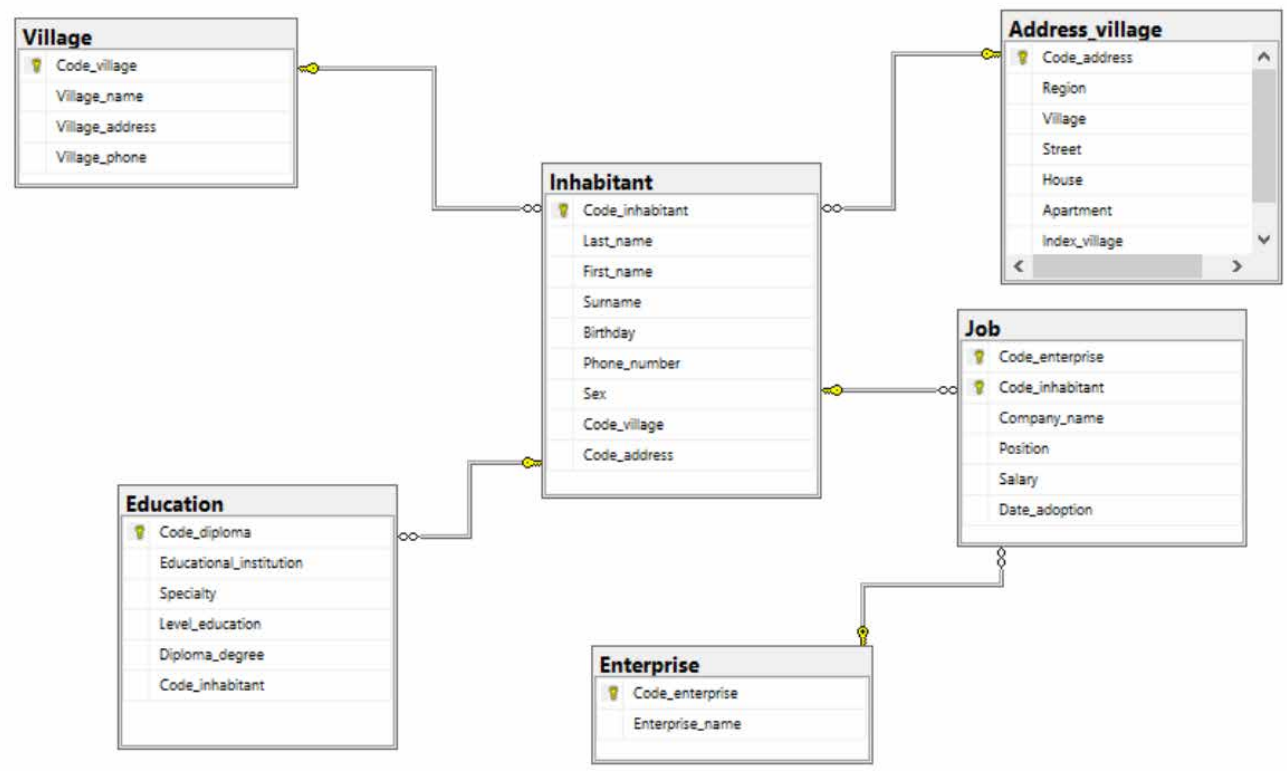


Рис. 1.7 Модель бази даних фізичного рівня

Блок-схема алгоритму – це графічне представлення послідовності дій, що мають бути виконані для вирішення певної задачі. Вона використовується в програмуванні, інженерії та бізнес-аналітиці для документування логіки процесу. Кожен елемент схеми відповідає певному типу дії: прямокутники – для операцій, ромби – для умовних перевірок, овали – для початку та завершення процесу [14]. Блок-схеми забезпечують легке сприйняття навіть складних алгоритмів, допомагають виявити неефективність, зайві кроки або потенційні помилки ще до реалізації програмного коду.

Блок-схема алгоритму входу в систему та роботи користувачів після авторизації демонструє логіку дій залежно від типу облікового запису – секретар або голова (рис. 1.8). Кожен користувач бачить лише той функціонал, який йому дозволено згідно з його роллю. Це дозволяє запобігти помилковому або несанкціонованому доступу до критичних операцій у системі.

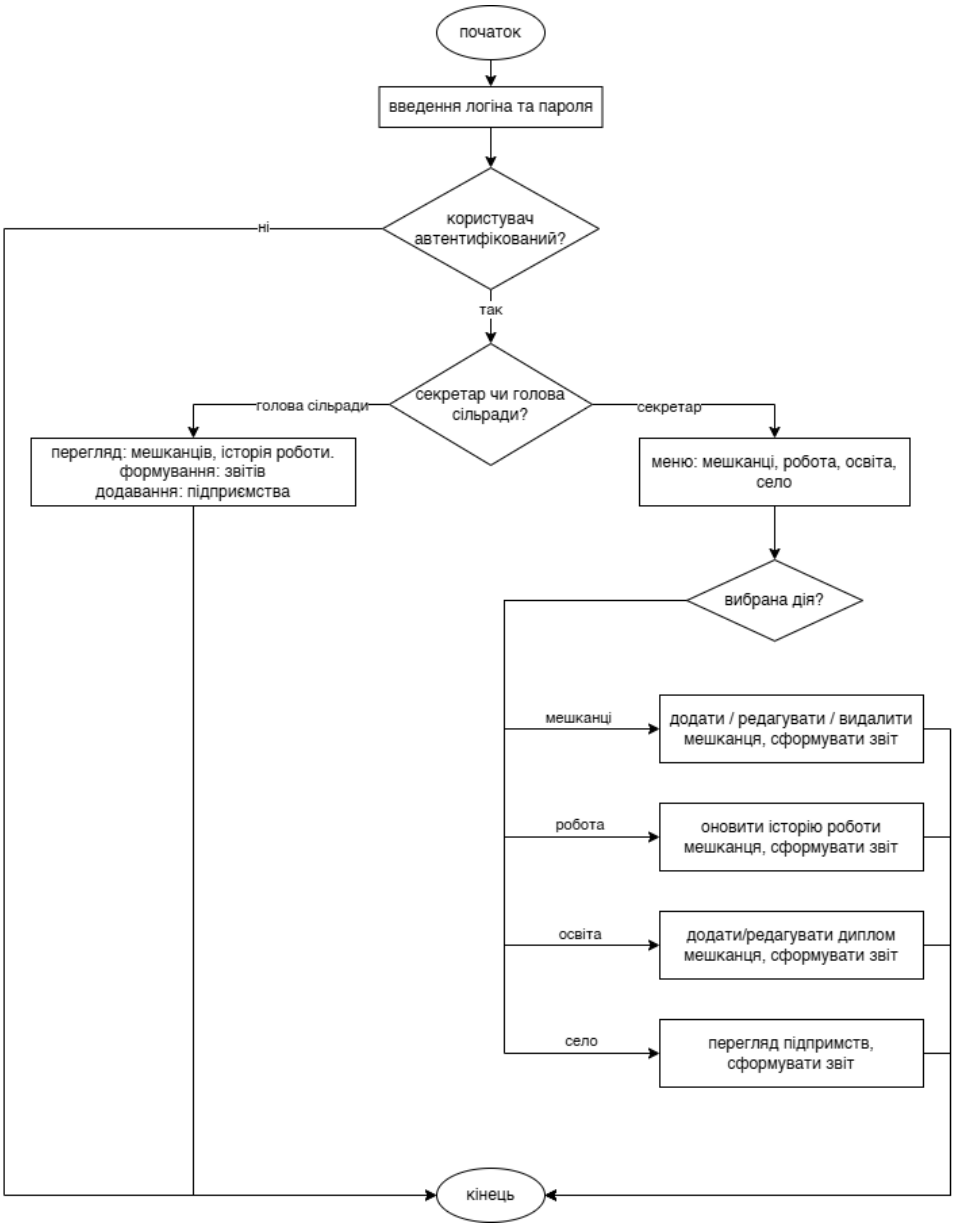


Рис. 1.8 Блох-схема алгоритму

Додатковою перевагою представленої блок-схеми є чітка структуризація сценаріїв використання, яка дозволяє автоматизувати контроль доступу вже на ранньому етапі логіки роботи інтерфейсу. Це забезпечує масштабованість рішення: при додаванні нових ролей (наприклад, «Аналітик» або «Адміністратор архіву») до моделі доступу буде достатньо впровадити відповідну гілку перевірки ролі, не змінюючи основної архітектури системи.

Важливим моментом є те, що система не просто надає або приховує функціональність, а забезпечує контекстну релевантності дій – користувач відразу потрапляє в той розділ, який відповідає його повноваженням. Це

покращує зручність користування ПЗ, скорочує час на пошук потрібних функцій та знижує імовірність помилок при роботі з базами даних.

Для секретаря передбачено декомпозицію доступних дій за логічними модулями: робота з мешканцями, оновлення історії працевлаштування, додавання освітніх даних, створення звітів і перегляд інформації про підприємства в межах села.

Алгоритм також демонструє ефективну реалізацію принципу мінімальних привілеїв, згідно з яким кожному користувачеві надаються лише ті права, які потрібні для виконання його функцій.

Блок-схема закладає основу для реалізації інтелектуального журналювання дій користувачів. Оскільки всі переходи між модулями є чітко структурованими, система може автоматично записувати інформацію про кожну сесію, що дає змогу здійснювати повний аудит усіх змін, запитів і транзакцій у системі.

На рисунку 1.9 представлено алгоритм авторизації користувача та подальшої взаємодії з функціоналом системи обліку мешканців. Алгоритм починається з етапу введення логіна та пароля. Якщо дані введено невірно – користувач отримує повідомлення про помилку. У разі успішної авторизації система перевіряє, хто увійшов – секретар чи голова сільради. Якщо це голова сільради, йому відкривається доступ до перегляду звітів або даних про підприємства. Якщо це секретар, він переходить до головного меню, яке містить пункти: «мешканці», «робота», «освіта», «звіти», «село». Далі, залежно від вибраної дії, реалізується один із варіантів: додавання нового мешканця, редагування диплому або формування звіту. Після вибору конкретної дії система надсилає відповідний запит до бази даних. У результаті користувач отримує на екран або сформований звіт, або повідомлення про виконання дії. Алгоритм завершується логічною крапкою – кінцем процесу.

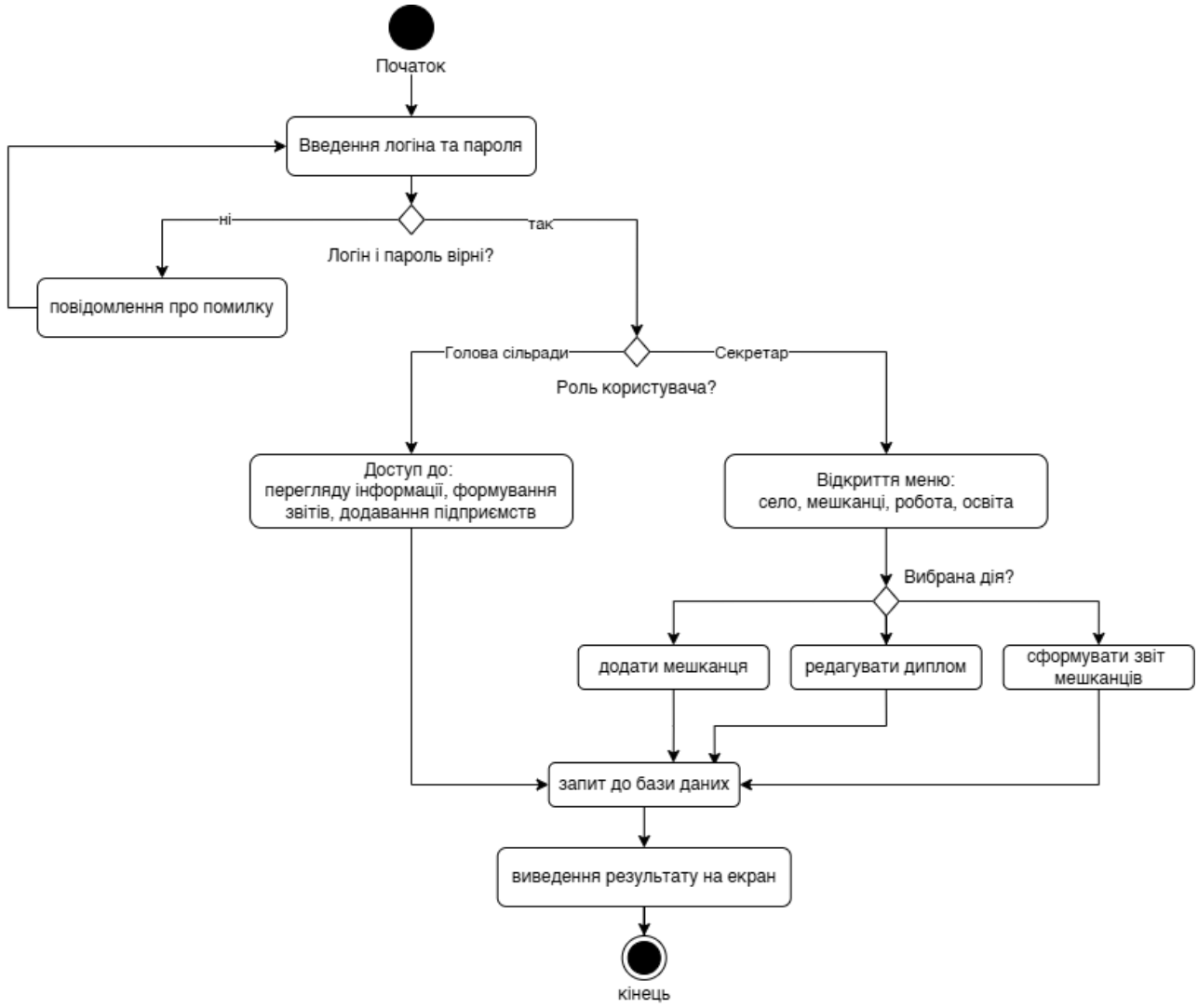


Рис. 1.9 Візуальна реалізація алгоритму

Усі представлені моделі є частинами єдиного комплексу, що описує систему обліку мешканців із різних сторін: від логіки взаємодії до фізичної архітектури. Таке поєднання дає змогу не лише краще зрозуміти, як працюватиме система, але й закласти надійну основу для її реалізації, тестування, впровадження та підтримки. Вони є логічним підґрунтям для розробки технічного завдання, реалізації класів у коді, формування структури бази даних та забезпечення користувацького досвіду, який буде зручним і зрозумілим у практичній роботі працівників місцевого самоврядування.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних

Логічна модель даних є абстрактним представленням структури інформації, яка має бути збережена в інформаційній системі. Вона не прив’язана до конкретної реалізації в базі даних, але відображає всі сутності,

їхні атрибути та зв’язки між ними, які мають значення з точки зору предметної області [15]. Основна мета логічного моделювання полягає в тому, щоб узагальнити інформаційні потреби системи, визначити типи об’єктів, з якими буде працювати користувач, і встановити правила взаємодії між ними. У логічній моделі даних фіксуються первинні та зовнішні ключі, типи атрибутів, обмеження цілісності та унікальності, що дозволяє створити чітке й непротирічне уявлення про структуру даних. Це допомагає запобігти дублюванню інформації, уникнути логічних конфліктів і забезпечити коректне проєктування фізичної моделі. Логічна модель виступає важливим етапом у процесі розробки бази даних, слугуючи мостом між аналізом вимог користувача та технічною реалізацією структури зберігання даних. Вона є основою для створення запитів, забезпечення аналітики та інтеграції різних компонентів системи.

ER-діаграма (Entity-Relationship diagram), або відома як діаграма «сутність-зв’язок», є візуальним інструментом логічного моделювання, що використовується для формального опису структури даних у системі. Вона дозволяє відобразити основні об’єкти предметної області у вигляді сутностей, які мають атрибути, а також встановити між ними зв’язки – асоціації, які можуть бути один-до-одного, один-до-багатьох або багато-до-багатьох [16]. Сутності, зазвичай, представлені прямокутниками, атрибути – еліпсами, а зв’язки – ромбами або лініями з позначенням типу взаємозв’язку. Важливу роль відіграють ключові атрибути – ідентифікатори, що забезпечують унікальність кожного запису. ER-діаграми дають змогу виявити логічні залежності між таблицями бази даних ще до її фізичної реалізації, що мінімізує ймовірність помилок у структурі та забезпечує цілісність інформації. Їх використовують як на етапі аналізу потреб користувачів, так і під час проєктування програмного забезпечення, особливо в контексті систем, де головну роль відіграє облік, управління або збереження великих обсягів даних.

Логічна модель є незалежною від конкретної реалізації та забезпечує повноцінне описання предметної області. На діаграмі, представленій на

рисунку 2.1, зображено основні сутності, які відображають реальні об’єкти системи: мешканець, село, освіту, місце роботи та підприємство.

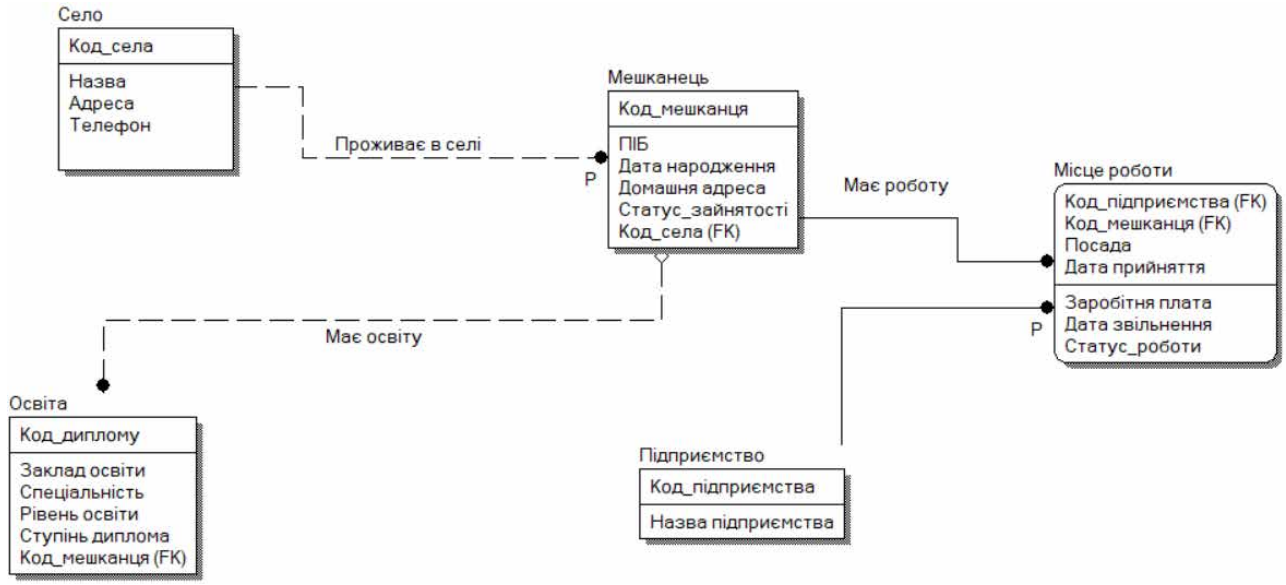


Рис. 2.1 Логічна модель даних

Центральним елементом моделі є сутність мешканець. Вона містить ключову інформацію про кожну особу – повне ім’я, дату народження, адресу, статус зайнятості(працевлаштований або ні) та ідентифікатор села, у якому мешкає особа. Атрибут «Код_села» пов’язує цю таблицю з таблицею Село, де зберігається інформація про назву населеного пункту та його опис. Це дозволяє будувати зв’язок «мешканець – село», що є базовою функцією для формування демографічної структури громади.

Кожен мешканець може мати пов’язані записи у таблицях «освіта» та «місце роботи». Таблиця «освіта» дозволяє зберігати відомості про здобуту освіту, зокрема назву навчального закладу, спеціальність та диплом. Це дає змогу формувати звіти про освітній рівень населення. Таблиця «місце роботи» фіксує історію працевлаштування мешканців, включаючи компанію, період роботи, дату звільнення. У ній реалізовано зовнішній ключ до таблиці «підприємство», яка містить дані про підприємства.

2.2 Вибір системи управління інформаційною базою

З огляду на особливості предметної області, обсяг даних, кількість очікуваних запитів, кількість користувачів та досвід використання в державних і муніципальних структурах, як основу для реалізації фізичної бази даних було обрано систему управління базами даних Microsoft SQL Server.

Microsoft SQL Server – одна з найпотужніших реляційних СУБД, що забезпечує зберігання структурованих даних у вигляді взаємопов’язаних таблиць з підтримкою зовнішніх ключів, індексів, тригерів, унікальних обмежень та всіх інших механізмів, що гарантують цілісність і непротирічність інформації [17]. У контексті мого проєкту це має особливе значення, адже мова йде про зберігання персональних даних мешканців, які повинні бути захищеними, актуальними і доступними в режимі реального часу для затверджених користувачів системи. SQL Server дозволяє реалізувати рольову модель доступу, забезпечити розмежування прав на перегляд, редагування, додавання або видалення записів залежно від рівня авторизації (наприклад, секретар, голова сільради).

SQL Server забезпечує високу продуктивність, підтримує індексацію великих обсягів даних, масштабування у межах локальної мережі, можливість інтеграції з іншими службами Windows Server (зокрема Active Directory), надає засоби моніторингу, аналізу продуктивності та налагодження запитів. Це все дозволяє використовувати систему не лише в навчальних або експериментальних цілях, але й у реальних умовах функціонування сільських та селищних рад.

Вагомим фактором на користь вибору Microsoft SQL Server стала його сумісність із середовищем розробки на платформі .NET, що дозволяє ефективно інтегрувати базу даних з користувацьким інтерфейсом, створеним засобами Windows Forms або WPF. Завдяки підтримці ADO.NET та Entity Framework, розробники мають змогу реалізувати швидкий доступ до даних, автоматично генерувати запити, проводити міграції та керувати схемою бази даних без потреби ручного втручання в SQL-код.

Наявність вбудованих засобів резервного копіювання, журналювання змін та точного відновлення, важливі при роботі з даними підвищеної значущості, зокрема особистими даними громадян. Завдяки цьому система може гарантувати збереження інформації навіть у разі технічних збоїв або людського фактора.

SQL Server дозволяє використовувати розширені функції безпеки, такі як шифрування даних на рівні стовпців (Transparent Data Encryption), використання сертифікатів та політик доступу, що відповідає сучасним стандартам захисту інформації. Це робить його оптимальним вибором для систем, які обробляють конфіденційну інформацію у відповідності до вимог законодавства про захист персональних даних.

У перспективі, при зростанні обсягу інформації та кількості користувачів, SQL Server також дозволяє реалізувати реплікацію бази даних або створити кластер високої доступності, що забезпечить безперервність обслуговування громадян та підвищить надійність всієї системи управління.

2.3 Створення інформаційної бази

З урахуванням архітектури програмного забезпечення, обсягу даних, очікуваної кількості користувачів та технічних можливостей середовища впровадження, було прийнято рішення реалізувати централізовану інформаційну базу, яка функціонує на окремому SQL-сервері в межах локальної мережі сільської ради.

У реалізованій моделі база даних побудована на основі реляційного принципу, із суворим дотриманням нормалізації структури даних. Кожна сутність (мешканці, освіта, місце роботи, підприємства, село, звіти, користувачі системи) представлена окремою таблицею з унікальним первинним ключем і зовнішніми ключами, які реалізують зв’язки типу «один до багатьох». Дана структура дозволяє уникати дублювання інформації, забезпечує простоту оновлення даних, швидкість виконання запитів і збереження логіки

взаємозв’язків між об’єктами. Створення таблиць та обмежень здійснюється за допомогою SQL-скриптів, згенерованих автоматично в середовищі ERWin, яке було використане для формалізації логічної моделі на попередньому етапі.

База даних зберігається на сервері MS SQL Server, до якого мають доступ лише авторизовані користувачі через клієнтський додаток, встановлений на комп’ютерах працівників сільської ради. Секретар та голова сільради мають різні рівні доступу, які регулюються рольовою моделлю в таблиці users. Завдяки цьому голова має можливість лише переглядати інформацію, додавати підприємства та формувати звіти, тоді як секретар має набагато більше прав, наприклад на перегляд, внесення, редагування та формування звітів, окрім додавання підприємств.

Згідно з технічним регламентом, система автоматично створює щоденні резервні копії бази даних, які зберігаються на захищеному фізичному носії або дублюються на віддалений сервер у разі інтеграції з хмарними сервісами. У разі системного збою, збитків унаслідок помилкових дій користувача або втрати частини інформації, адміністратор має можливість швидко відновити стан бази до останньої збереженої контрольної точки. Це забезпечується стандартним механізмом бекапування SQL Server, який дозволяє зберігати як повні, так і диференційні резервні копії, а також журнали транзакцій.

Хоч система і реалізована у вигляді централізованої моделі, її структура спроєктована з урахуванням можливого розширення до розподіленої архітектури. Наприклад, у разі об’єднання кількох населених пунктів в одну адміністративну одиницю або інтеграції з іншими інформаційними системами (державними реєстрами, сервісами Мінцифри), структура бази дозволяє масштабувати зберігання даних без необхідності повної перебудови схеми.

У процесі створення інформаційної бази особливу увагу було приділено нормалізації структури даних. Було забезпечено дотримання принаймні третьої нормальної форми (ЗНФ), що дозволяє уникнути надмірного дублювання інформації, зменшити ймовірність аномалій при оновленні, вставці або видаленні записів, а також забезпечити логічну цілісність зв’язків між

сутностями. На етапі логічного моделювання для кожної таблиці було чітко визначено первинні ключі, встановлено зовнішні ключі з відповідними каскадними обмеженнями, а також розроблено механізми підтримки унікальності даних у критично важливих полях (наприклад, логіни користувачів або номери дипломів).

У рамках реалізації було створено основні таблиці, що зберігають структуровані записи про мешканців, села, освіту, працевлаштування, компанії, звіти та облікові записи користувачів та продумано індексацію найбільш часто використовуваних полів. Зокрема, індекси було накладено на поля VillageID, ResidentID та CompanyID для прискорення виконання запитів, пов’язаних із пошуком, сортуванням і приєднанням таблиць. Такий підхід забезпечує високу швидкодію системи навіть за умов зростання кількості записів.

Важливою частиною побудови інформаційної бази стало створення представлень (views) для реалізації складніших зв’язків та агрегованої інформації.

Створення обмежень на рівні полів (constraints) також мало особливе значення, зокрема CHECK, які унеможливлюють введення некоректних даних (наприклад, заборона введення дати народження, що перевищує поточну дату, або року отримання диплому, що є меншим за рік народження). Це дозволяє контролювати правильність введення даних ще на рівні СУБД, доповнюючи валідацію, реалізовану на рівні інтерфейсу користувача.

Було реалізовано попереднє заповнення бази даних тестовими записами. Було створено спеціальні скрипти, які автоматизують вставку стартових даних для демонстрації функціоналу системи та проведення тестування.

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Організаційна структура програмного забезпечення

Діаграма пакетів – структурний засіб візуалізації програмного забезпечення, який дозволяє логічно організувати систему у вигляді ієрархічних груп – пакетів, що містять класи, інтерфейси або інші пакети. Її основною метою є спрощення сприйняття великої системи, розділення коду на зрозумілі логічні блоки та визначення залежностей між ними [9]. Пакети на діаграмі представлені прямокутниками з вкладеними елементами, які

демонструють структуру системи з точки зору розподілу функціональності. Наприклад, можна виділити окремі пакети для обробки даних, взаємодії з базою даних, інтерфейсу користувача, звітності, адміністративних функцій тощо. Важливим елементом діаграми пакетів є стрілки залежності, які показують, який пакет використовує інший, тим самим вказуючи на напрямок інформаційного або функціонального зв’язку. Завдяки цій діаграмі розробники отримують можливість чітко структурувати архітектуру проєкту, дотримуватись принципів модульності та слабкого зв’язку між компонентами, а також полегшити супровід, масштабування і повторне використання програмних модулів.

Організаційна структура програмного забезпечення побудована на основі модульного підходу, що забезпечує логічне розділення функцій між окремими пакетами (рис. 3.1). Пакет «ui» (user interface) відповідає за взаємодію з користувачем, відображає інтерфейс, приймає дії та передає їх до відповідних модулів. Далі дані передаються до registration, який реалізує логіку внесення, редагування, перегляду інформації про мешканців.

Пакет «archive» зберігає історичні дані, що були змінені або видалені, забезпечуючи збереження архіву. Ядро системи – пакет «db» – обробляє бізнес-логіку, приймає запити з інших модулів і взаємодіє з базою даних. Пакет reporting відповідає за створення, структурування та збереження звітів. Пакет «administration» реалізує функції авторизації, визначення прав доступу, керування ролями користувачів (наприклад, секретар чи голова сільради).

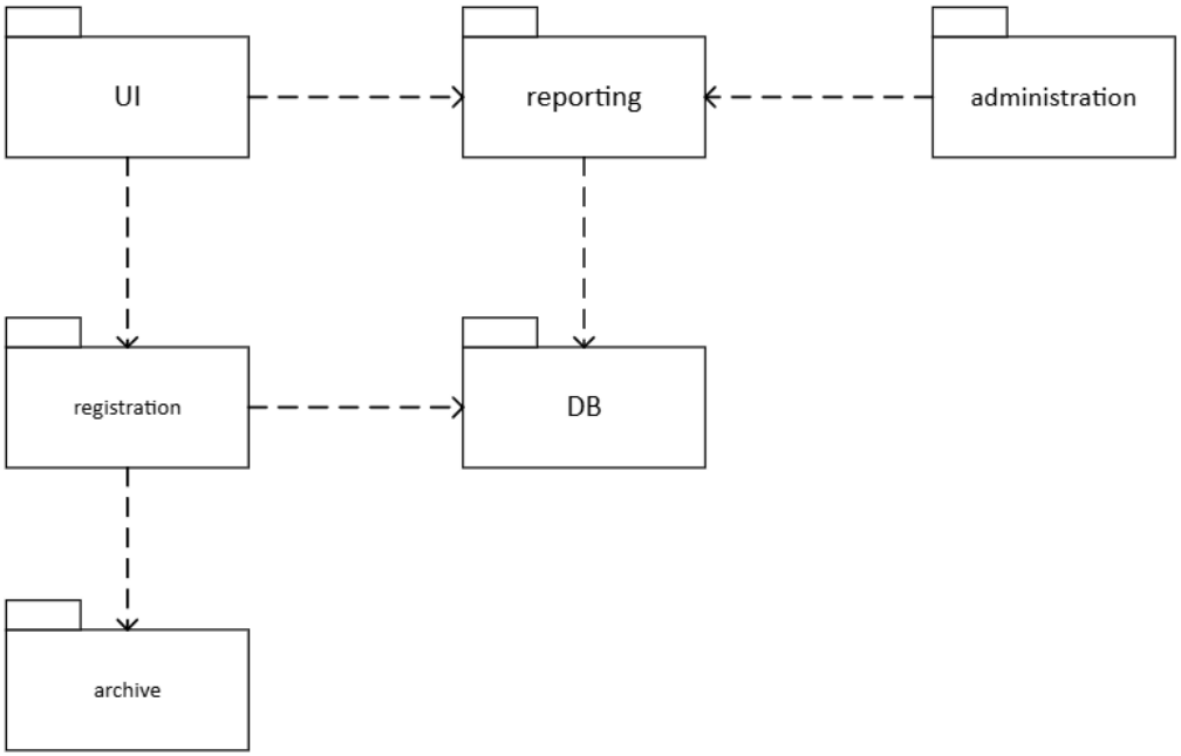


Рис. 3.1 Діаграма пакетів

При побудові організаційної структури було приділено забезпечення ізоляції функціоналу кожного модуля з метою мінімізації залежностей, що відповідає принципам чистої архітектури та сприяє зменшенню впливу змін у одному пакеті на решту системи.

Система підтримує односпрямовану залежність: модулі, які використовуються іншими пакетами (як db або archive), самі не мають зворотної залежності, що дозволяє уникнути циклічних викликів і спрощує тестування кожного пакету окремо. Пакет db виступає у ролі центра взаємодії, приймаючи запити з інших частин системи, і саме в ньому реалізовано основні бізнес-правила, валідацію та обробку критичних даних.

Registration працює з основною інформацією мешканців, а й виконує функції передавання даних до archive, забезпечуючи логічну історичність змін.

Модуль ui залишається ізольованим від внутрішньої логіки системи, спілкуючись лише з зовнішніми сервісами через чітко визначені інтерфейси.

Побудова такої модульної структури відкриває перспективи для впровадження CI/CD-підходу, де кожен пакет може бути зібраний,

протестований і розгорнутий окремо. Завдяки цьому система стає гнучкішою, надійнішою та здатною до швидкого впровадження оновлень без зупинки всього функціоналу.

3.2 Вибір інструментарію для створення ППЗ

У якості мови програмування було обрано C#, як одну з найпотужніших, стабільних та зручних мов для розробки прикладного програмного забезпечення під операційні системи Windows. C# – сучасна, об'єктно-орієнтована мова програмування, розроблена компанією Microsoft у рамках платформи .NET. Вона була створена для того, щоб поєднати потужність мов типу C++ з простотою мов високого рівня, таких як Java. C# дозволяє писати як прості прикладні програми для Windows, так і складні багаторівневі системи, веб-застосунки, ігри, мобільні додатки та хмарі сервіси [19]. C# має строгу типізацію, підтримує інкапсуляцію, успадкування та поліморфізм, що дозволяє створювати гнучкі й масштабовані програми. Вона тісно інтегрується з платформою .NET, забезпечуючи доступ до величезної бібліотеки класів, інструментів і технологій, зокрема для роботи з базами даних, інтерфейсами, мережевими протоколами та безпекою.

Середовище Microsoft Visual Studio виступає основною платформою для створення десктопного застосунку. Вибір саме цього середовища обумовлений високим рівнем інтеграції з іншими продуктами Microsoft, зокрема MS SQL Server, гнучкою підтримкою структури проєктів, широкими можливостями відлагодження, зручним візуальним дизайнером форм, підтримкою Windows Forms, що дає змогу будувати як класичні інтерфейси, так і сучасні адаптивні форми.

Для організації зберігання даних використовується Microsoft SQL Server– це безкоштовна версія потужної системи керування реляційними базами даних, призначена для навчання, розробки, а також невеликих комерційних чи

внутрішніх проєктів, забезпечуючи надійне зберігання, обробку та захист даних [20].

SQL Server дозволяє створювати бази даних з таблицями, зв’язками, запитами, індексами, обмеженнями, підтримує транзакції, збережені процедури, тригери та багато інших функцій, притаманних великим корпоративним СУБД. Її головною перевагою є те, що вона повністю безкоштовна та не потребує ліцензії.

Основними обмеженнями цієї версії є максимальний розмір бази даних до 10 ГБ, обмеження на використання оперативної пам’яті та кількість процесорних ядер. Проте для невеликої інформаційної системи – як, наприклад, система обліку мешканців населеного пункту – цих ресурсів цілком достатньо.

Інтеграція з C# через ADO.NET або Entity Framework дає можливість працювати з базою як на рівні сирих SQL-запитів, так і через об’єктно-орієнтовану модель зменшеної складності [21].

Проєктування структури бази даних, її логічної та фізичної моделі виконувалося в середовищі ERWin Data Modeler, яке дозволяє створювати діаграми зв’язків між сутностями, генерувати SQL-код для створення таблиць, переглядати схеми та забезпечувати цілісність логіки ще до реалізації безпосередньо в коді.

Для створення інтерфейсу та організації взаємодії між модулями у системі використано підхід багаторівневої архітектури. Вона включає рівень представлення (формування інтерфейсу користувача), рівень логіки (обробка дій, бізнес-правила) та рівень доступу до даних (запити до бази, збереження та читання інформації). Завдяки такому розділенню було реалізовано гнучку структуру, яка дозволяє ізолювати окремі функціональні частини, розширювати систему без суттєвого впливу на інші компоненти, а також спрощує тестування.

Для візуалізації моделей, побудови діаграм UML, використано інструмент Drawio. Він надав можливість зручно й чітко спроєктувати архітектуру системи, відобразити взаємозв’язки, процеси, сценарії

використання та поведінку користувачів. Усі моделі виконані відповідно до вимог UML 2.0, що забезпечує стандартизацію опису архітектури й можливість легкого сприйняття її іншими розробниками або технічними спеціалістами.

iTextSharp була обрана для автоматизованого формування звітів і довідок у форматі PDF. Цей інструмент дозволяє динамічно створювати структуровані документи, вставляти таблиці, шрифти, заголовки та навіть зображення, забезпечуючи гнучкість у формуванні вихідної документації відповідно до обраного формату або шаблону. З огляду на те, що документи, які формуються системою, мають офіційний характер і можуть бути передані на паперовому носії або збережені в архіві, підтримка PDF виявилася критично важливою.

У проєкті також застосовано можливості паралельної обробки даних за допомогою async та await, що дозволяє виконувати тривалі операції (наприклад, генерацію PDF або запити до бази даних) у фоновому режимі, не блокуючи головний інтерфейс користувача.

Для налаштування конфігурацій програми (наприклад, підключення до БД, збереження параметрів фільтрації, директорій для збереження звітів) було використано app.config, що дає змогу централізовано керувати параметрами без необхідності зміни коду.

На етапі контролю якості програмного забезпечення активно використовувався вбудований модуль Unit Testing із пакету MSTest, який дозволив автоматизовано перевірити коректність виконання критичних функцій, зокрема обробку введення, збереження даних та формування звітів. Автоматизовані тести зменшують ризик появи помилок після змін у коді та сприяють підтримуваності системи в довгостроковій перспективі.

Для розширення можливостей взаємодії з зовнішніми системами (наприклад, державними реєстрами або сервісами аналітики) передбачено можливість створення REST API на базі ASP.NET, що забезпечує обмін структурованими даними у форматі JSON або XML. Це дозволяє у майбутньому інтегрувати систему в більш широкий цифровий простір громади.

3.3. Алгоритмізація та програмування програмних модулів

Розробка ПЗ передбачає реалізацію низки функціональних модулів, які забезпечують повний життєвий цикл обробки даних: від введення і перевірки до зберігання, редагування та формування звітів. На етапі алгоритмізації та програмування головна мета полягала у побудові зрозумілої, гнучкої та масштабованої структури застосунку, яка б легко адаптувалася до змін вимог користувача.

Основний підхід до реалізації логіки базується на розподілі функцій за окремими формами у середовищі Windows Forms, що дозволяє забезпечити модульність, повторне використання коду та ізоляцію логіки в рамках конкретного компонента. Кожна форма реалізує свою відповідальність: додавання, редагування або перегляд мешканців, підприємств, освіти, роботи тощо. Наприклад, форма AddInhabitant містить алгоритми перевірки правильності введених даних, унікальності коду мешканця, а також забезпечує автоматичне заповнення пов'язаних полів, таких як населені пункти, через запит до бази даних. Аналогічно, AddEducation перевіряє валідність коду програми, зв'язок з обраним мешканцем та формує SQL-запит для збереження інформації в таблиці Education.

Всі операції з базою даних реалізуються через об'єкти SqlConnection, SqlCommand та SqlDataReader, які забезпечують стабільне з’єднання з Microsoft SQL Server і контроль за коректністю виконання транзакцій. У більш складних формах, таких як EditJob, використано словникові структури для збереження зв’язків між повними іменами мешканців і їх кодами, що дозволяє уникнути плутанини та забезпечити точність вибору записів для редагування. Додатково реалізовано механізми оновлення статусу працевлаштування мешканця залежно від змін у таблиці Job.

Програмна логіка передбачає як зворотний зв'язок із користувачем у вигляді повідомлень про помилки або успішні операції, так і обробку можливих виняткових ситуацій.

Впроваджено валідацію даних на рівні інтерфейсу користувача, а також перевірку цілісності даних на рівні бази. За допомогою бібліотеки iTextSharp користувач має змогу формувати текстові або PDF-звіти з урахуванням дати створення та структури таблиці. Така функціональність робить систему інформативною, зручною для аналізу та контролю на рівні керівництва.

Головним компонентом взаємодії користувача із системою є меню, реалізоване у вигляді єдиної форми з панеллю навігації, яка забезпечує динамічне підключення дочірніх форм за потреби, відповідно до ролі авторизованого користувача.

У межах процесу алгоритмізації також було впроваджено принцип розділення відповідальностей (Separation of Concerns), що дозволило чітко відокремити бізнес-логіку, логіку представлення та доступ до даних.

У модулі генерації звітів передбачено динамічне формування SQL-запитів на основі фільтрів, які задаються користувачем – наприклад, за періодом, типом події або статусом мешканця. Це реалізовано через шаблон «Builder», що дозволяє зібрати запит поступово, додаючи лише ті умови, які реально потрібні.

Для зручності користувача також реалізовано автоматичне оновлення інтерфейсу після внесення змін: після додавання або редагування даних, форми автоматично оновлюють вміст таблиць без потреби ручного перезапуску.

Локалізація інтерфейсу дозволяє адаптувати програмне забезпечення під різні мовні середовища. Створено ресурсні файли, які містять переклад усіх повідомлень, заголовків і підписів. Зміна мови виконується автоматично залежно від регіональних налаштувань Windows або вручну через панель керування.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

У процесі створення ПЗ важливим етапом є тестування, яке дозволяє перевірити, наскільки коректно працюють усі функціональні модулі, як взаємодіють між собою окремі компоненти системи та чи відповідає реалізований функціонал вимогам, що були сформульовані на етапі проєктування. Для програмного продукту, який реалізує облік мешканців населеного пункту, тестування охоплювало як окремі блоки введення та редагування інформації, так і загальну інтегровану поведінку системи в цілому.

При запуску програми, першим що бачить користувач це вікно для входу в систему. Щоб отримати доступ до керування даними авторизуємося у ролі секретаря ввівши відповідні дані для входу, такі як логін та пароль, натискаємо кнопку «Увійти», якщо дані введено правильно відкриється нова форма, в іншому разі програма видасть помилку про неправильно введені дані (рис. 4.1).

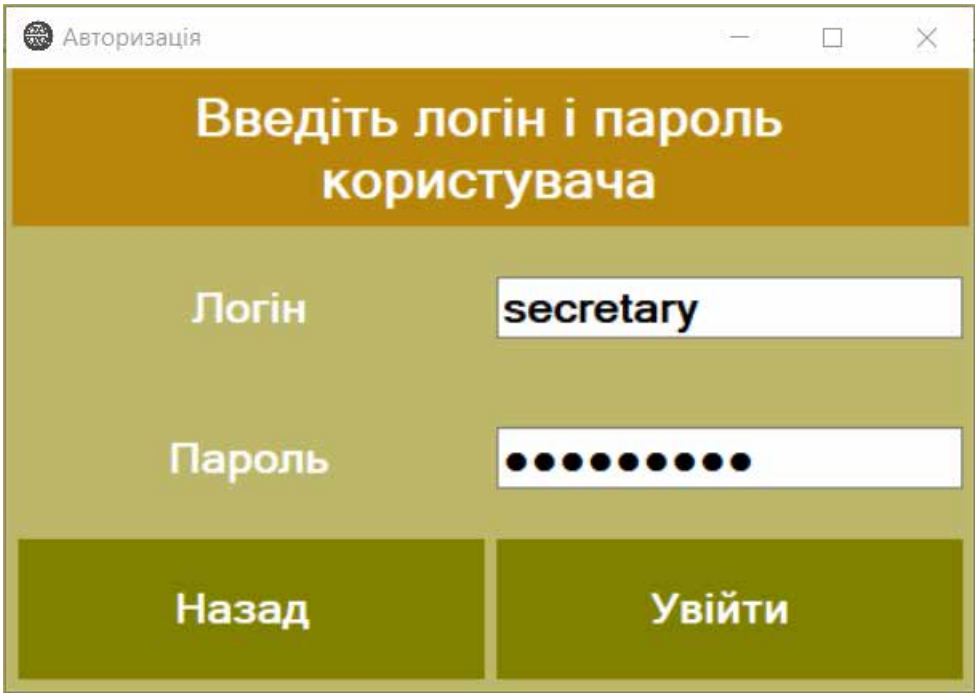


Рис. 4.1 Вікно авторизації

Після успішної авторизації бачимо головну форму «Village Council» де є декілька вкладок підгрупи «Село», які недоступні через обмежені права секретаря. На рисунку 4.2 зображено відкриту вкладку «Переглянути інформацію» з групи «Село». На данній вкладці ми можемо переглянути інформацію та згенерувати звіт натиснувши на кнопку «Згенерувати звіт». На рисунку 4.3 зображено відкритий PDF файл згенерованого звіту.

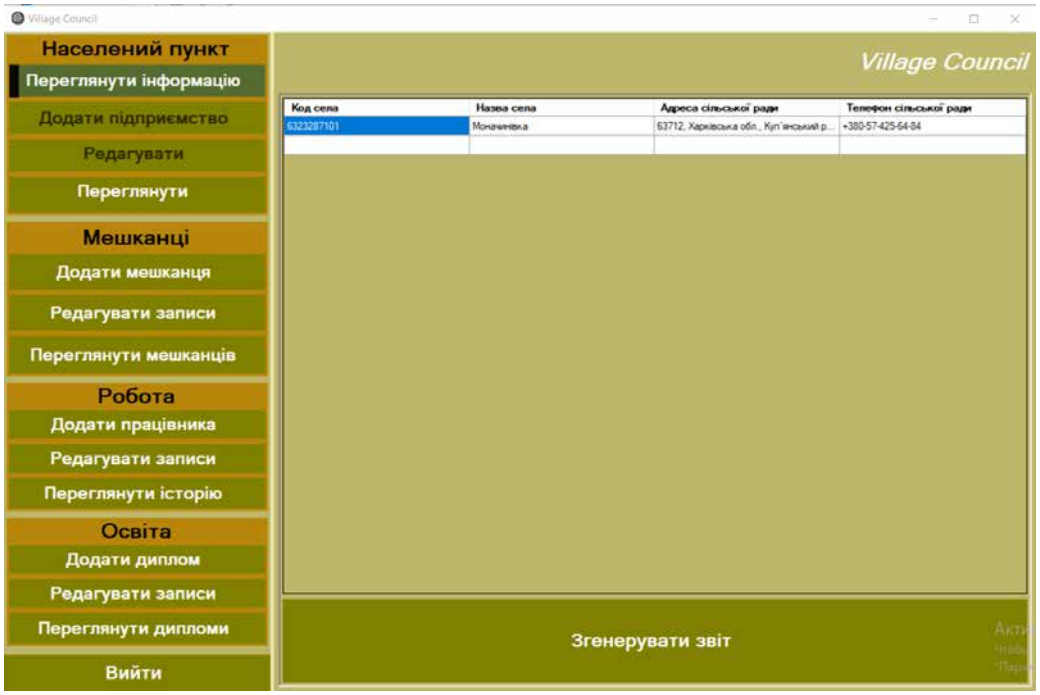


Рис. 4.2 Форма «Переглянути інформацію»

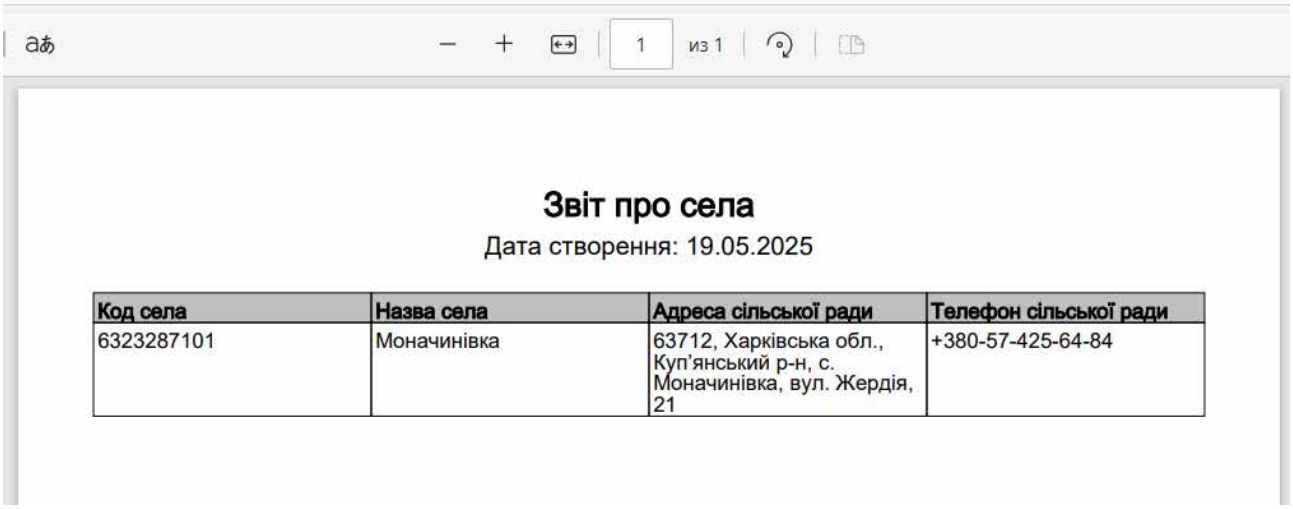


Рис. 4.3 Згенерований звіт про села

Перейдемо на вкладку «Переглянути підприємства», тут ми бачимо перелік підприємств, і, також, можемо згенерувати звіт (рис. 4.4). Натискаємо на кнопку «Згенерувати звіт» (рис. 4.5).

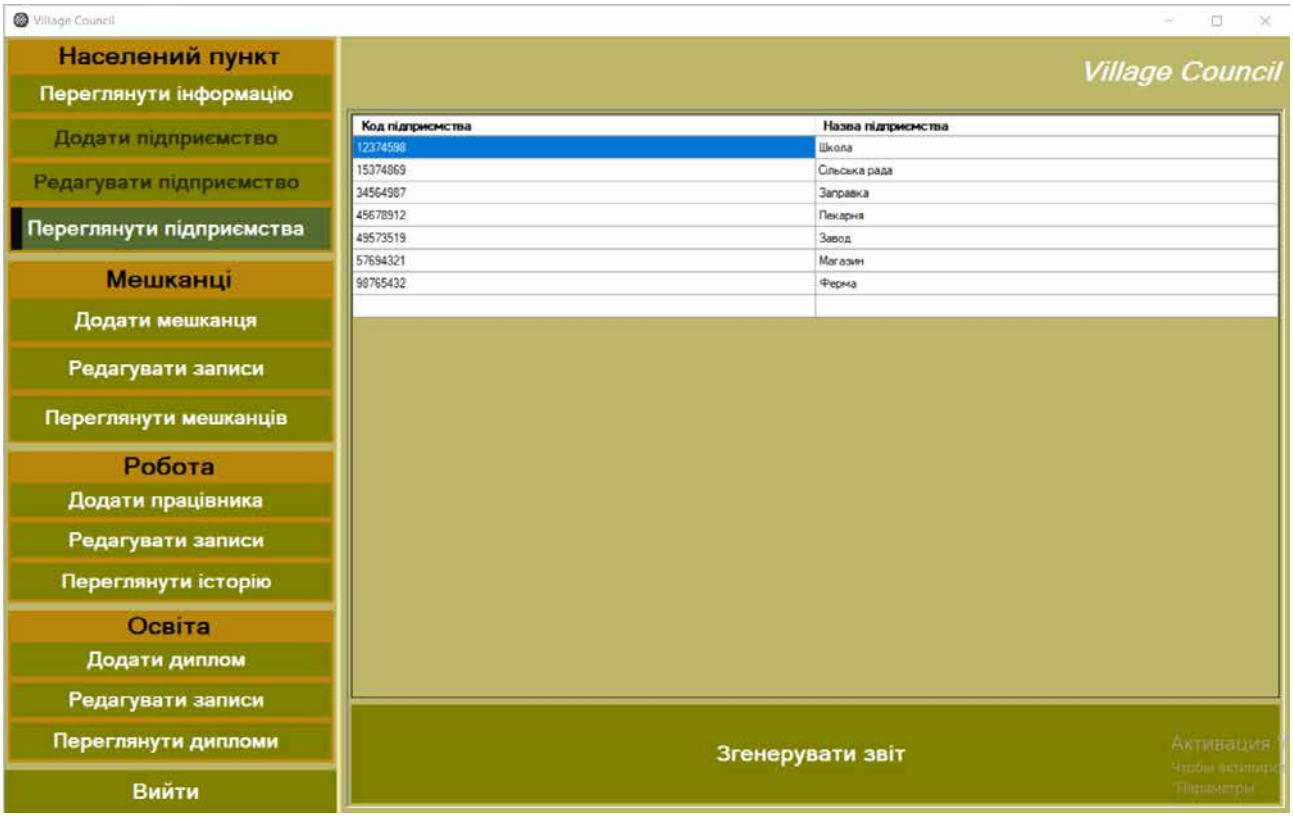


Рис. 4.4 Вкладка «Переглянути підприємства»

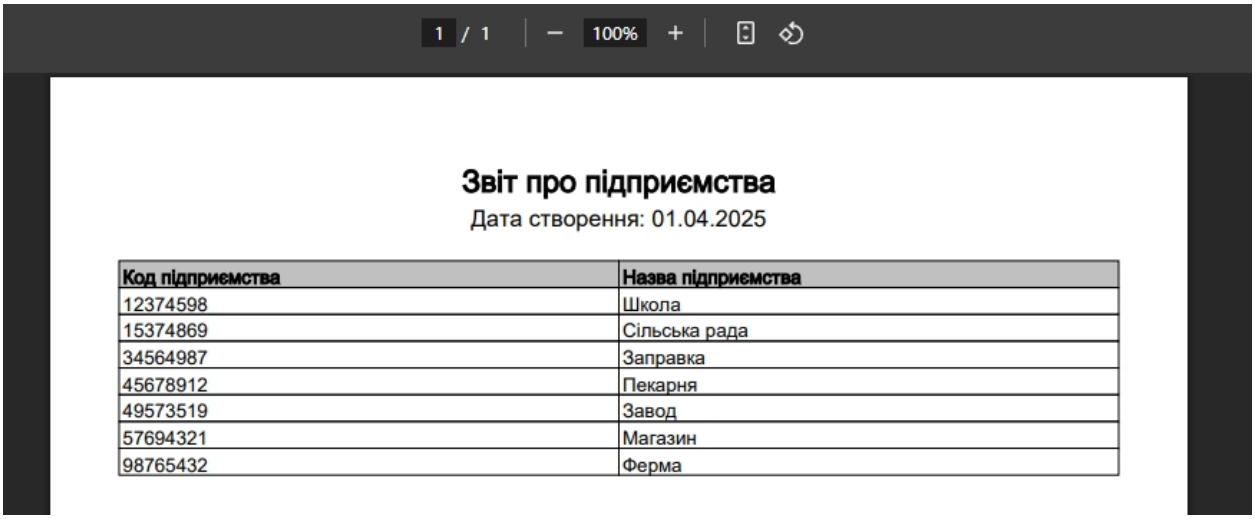


Рис. 4.5 Згенерований звіт про підприємства

Далі можемо додати нового мешканця села, для цього потрібно перейти на вкладку «Додати мешканця», яка знаходиться з лівого боку меню (рис. 4.6). Введемо дані мешканця та натиснемо кнопку «Додати» (рис. 4.7).

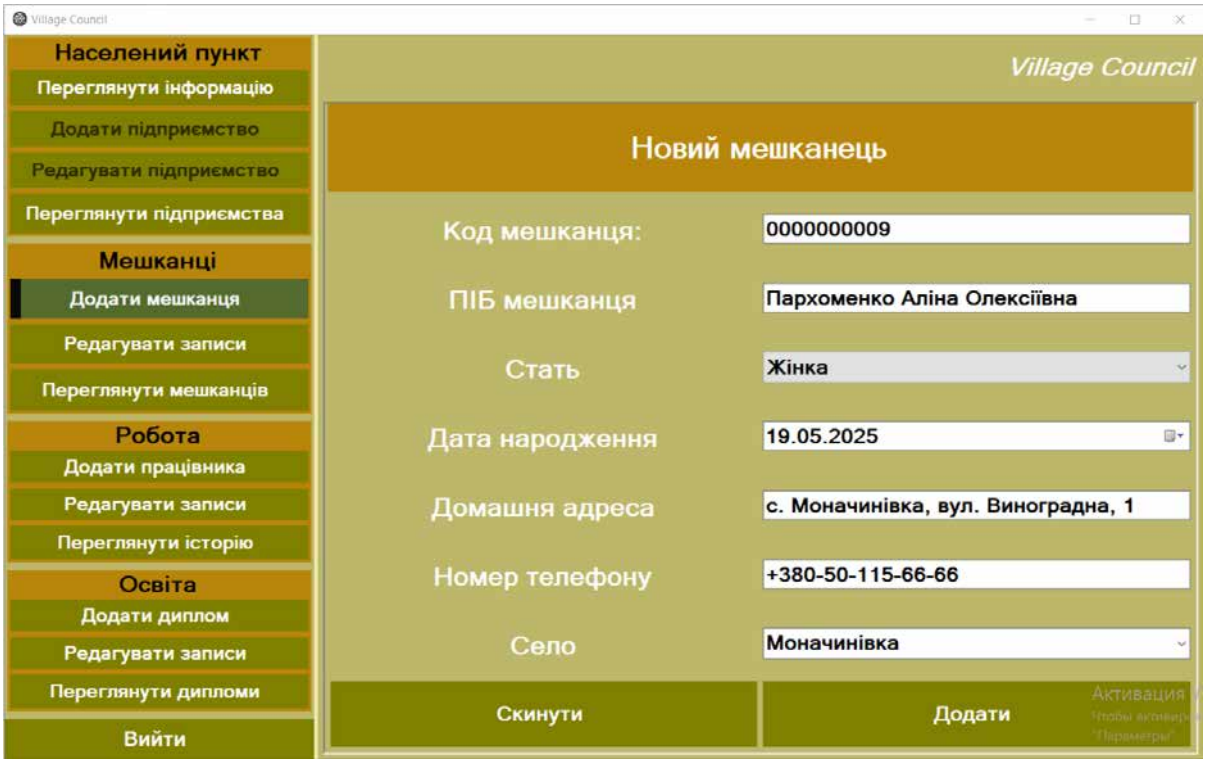


Рис. 4.6 Форма додавання нового мешканця села

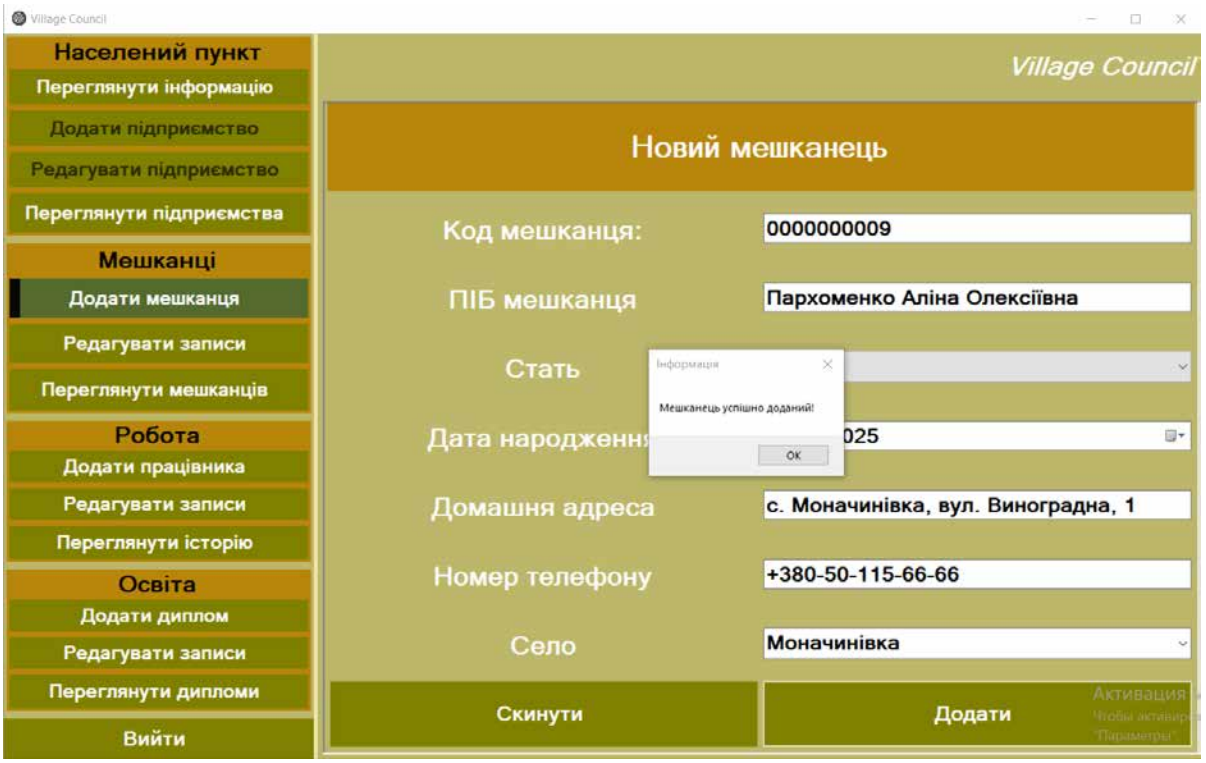


Рис. 4.7 Повідомлення про успішне додавання мешканця

Тепер спробуємо редагувати номер телефону мешканця, який було введено неправильно. Для цього перейдемо на вкладку «Редагувати мешканця» меню (рис. 4.8).

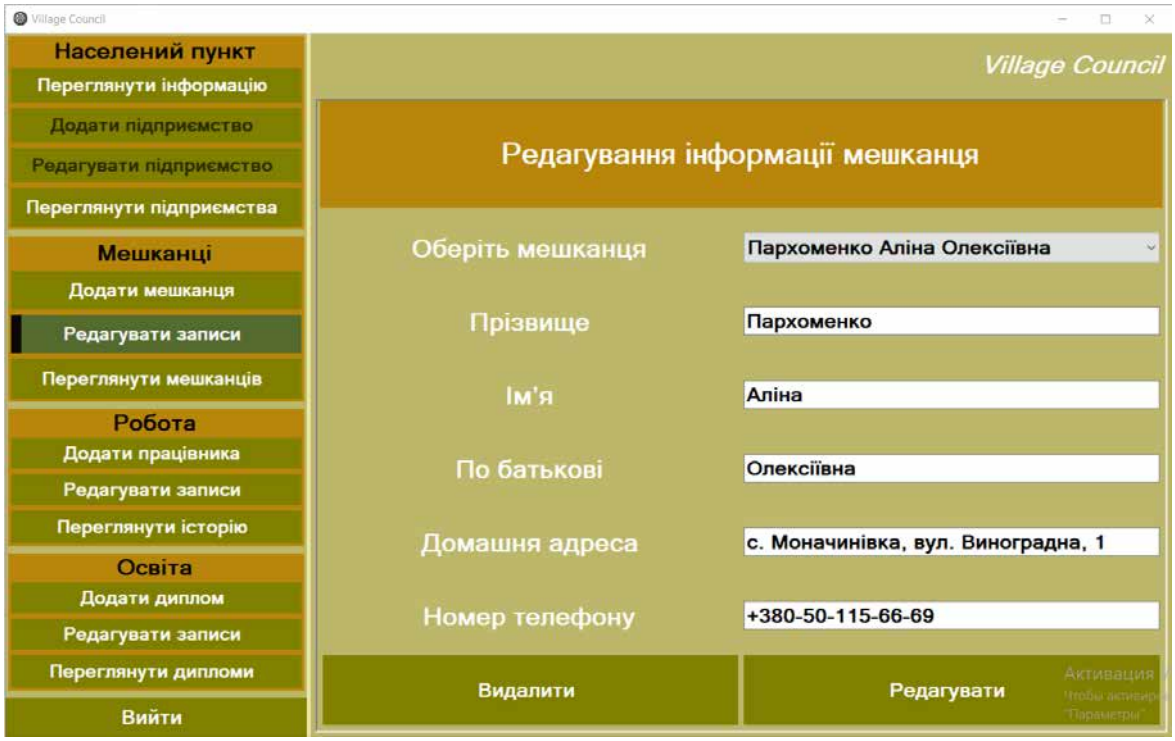


Рис. 4.9 Редагування даних мешканця

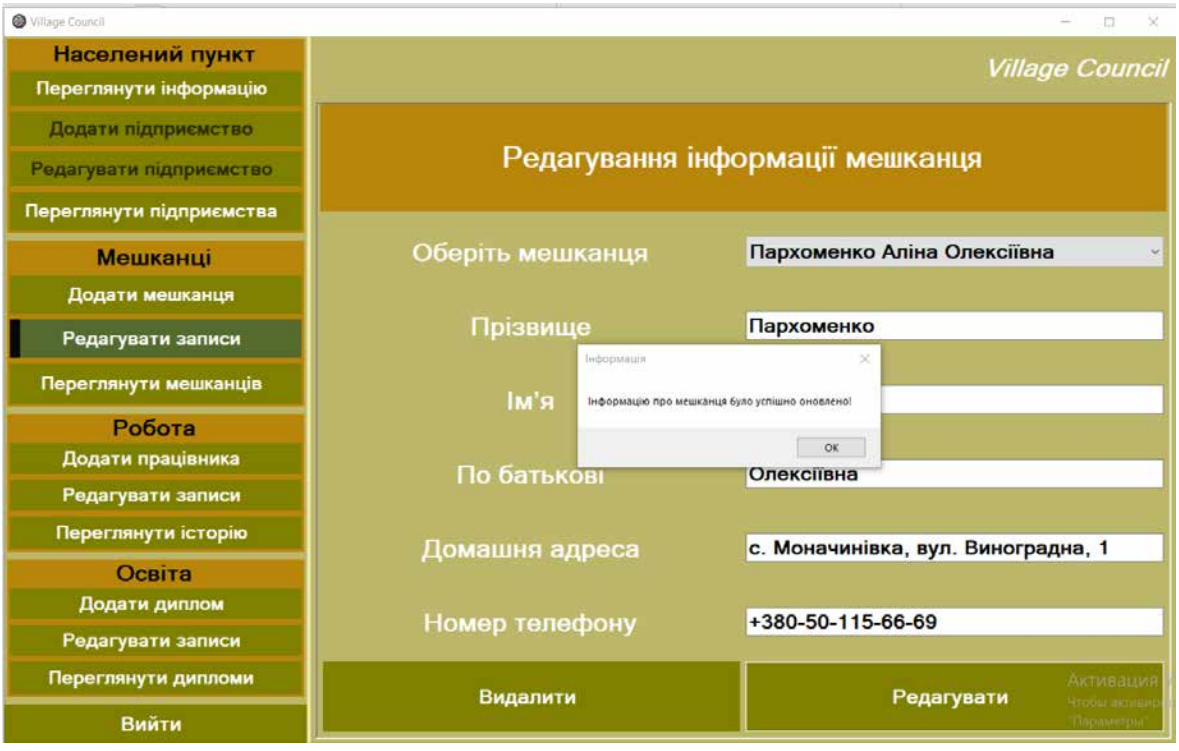


Рис. 4.10 Результат редагування даних мешканця

Тепер коли усі дані мешканців введені правильно, можемо перейти на вкладку «Переглянути мешканців» і згенерувати звіт у PDF форматі (рис. 4.11).

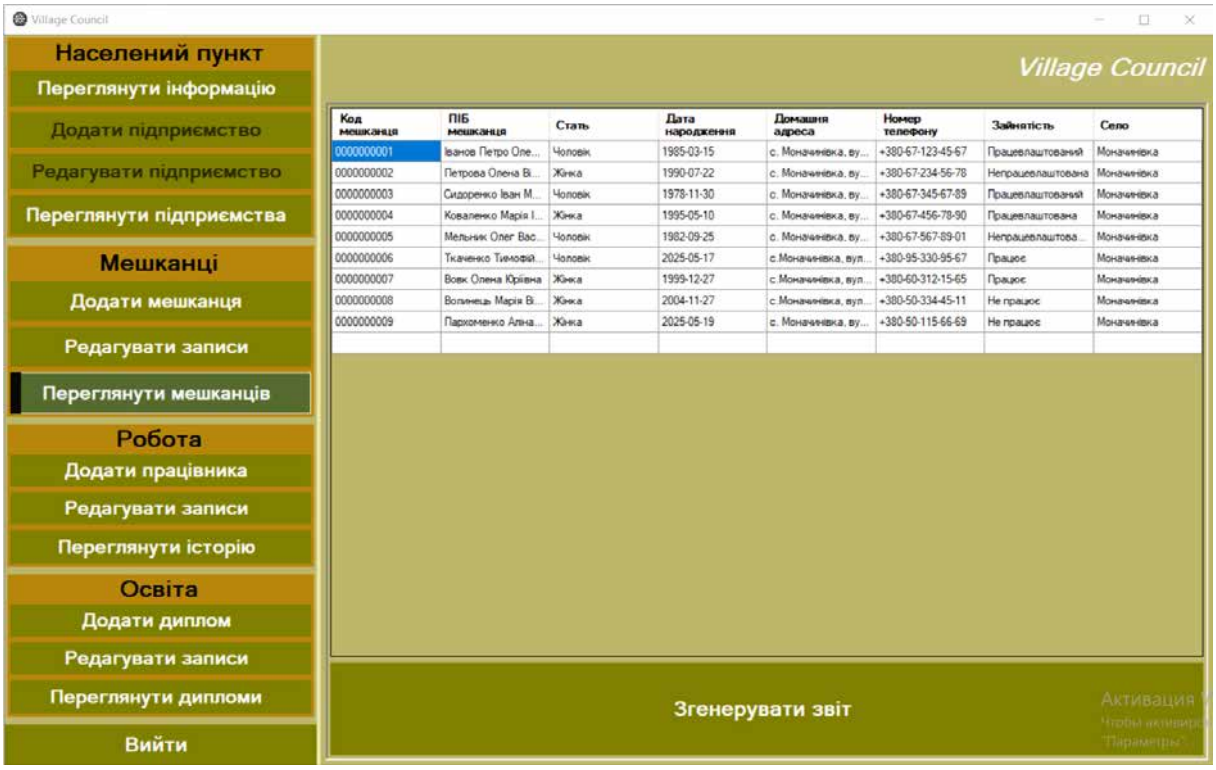
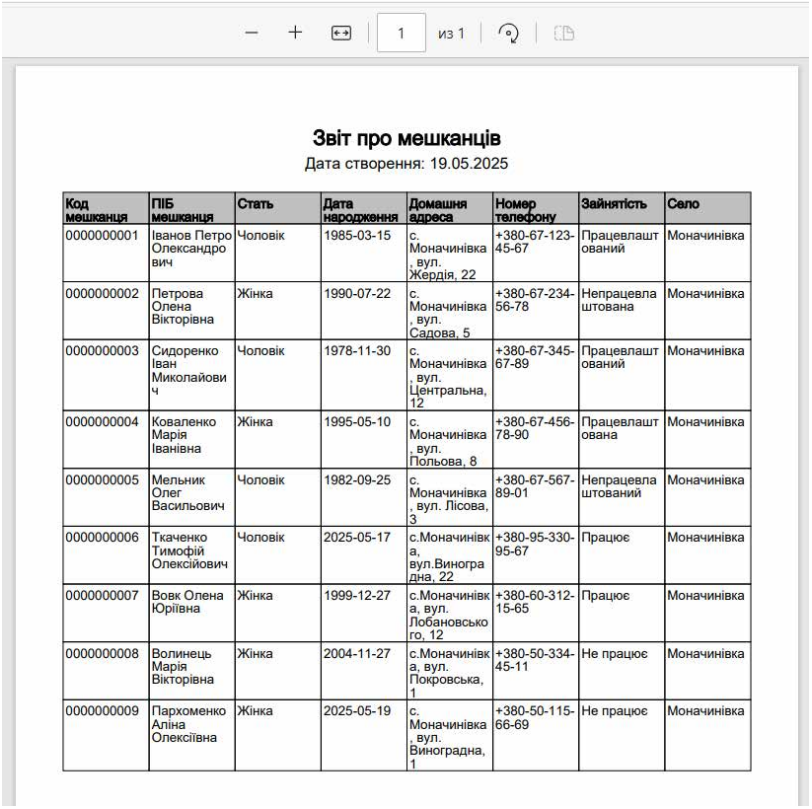


Рис. 4.11 Вкладка «Переглянути мешканців»



1 из 1

Звіт про мешканців

Дата створення: 19.05.2025

Код мешканця	ПІБ мешканця	Стать	Дата народження	Домашня адреса	Номер телефону	Зайнятість	Село
0000000001	Іванов Петро Олександрович	Чоловік	1985-03-15	с. Моначинівка, вул. Жердія, 22	+380-67-123-45-67	Працевлаштований	Моначинівка
0000000002	Петрова Олена Вікторівна	Жінка	1990-07-22	с. Моначинівка, вул. Садова, 5	+380-67-234-56-78	Непрацевлаштована	Моначинівка
0000000003	Сидоренко Іван Миколайович	Чоловік	1978-11-30	с. Моначинівка, вул. Центральна, 12	+380-67-345-67-89	Працевлаштований	Моначинівка
0000000004	Коваленко Марія Іванівна	Жінка	1995-05-10	с. Моначинівка, вул. Польова, 8	+380-67-456-78-90	Працевлаштована	Моначинівка
0000000005	Мельник Олег Васильович	Чоловік	1982-09-25	с. Моначинівка, вул. Лісова, 3	+380-67-567-89-01	Непрацевлаштований	Моначинівка
0000000006	Ткаченко Тимофій Олексійович	Чоловік	2025-05-17	с.Моначинівка, вул.Виноградна, 22	+380-95-330-95-67	Працює	Моначинівка
0000000007	Вовк Олена Юріївна	Жінка	1999-12-27	с.Моначинівка, вул. Лобановського, 12	+380-60-312-15-65	Працює	Моначинівка
0000000008	Волинець Марія Вікторівна	Жінка	2004-11-27	с.Моначинівка, вул. Покровська, 1	+380-50-334-45-11	Не працює	Моначинівка
0000000009	Пархоменко Аліна Олексіївна	Жінка	2025-05-19	с. Моначинівка, вул. Виноградна, 1	+380-50-115-66-69	Не працює	Моначинівка

Рис. 4.12 Згенерований звіт про мешканців

Тепер додамо працівника для підприємства. Заповнюємо поля і натискаємо кнопку «Додати» (рис. 4.13).

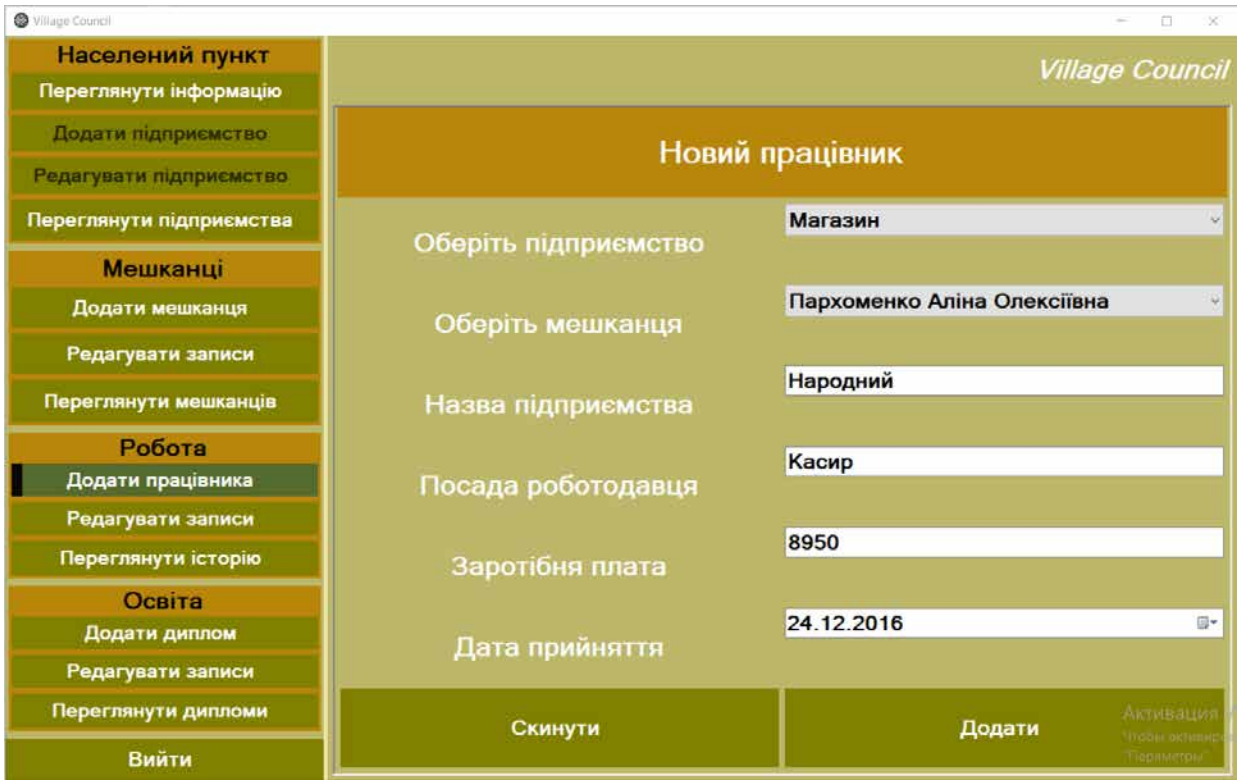


Рис. 4.13 Додавання працівника для підприємства

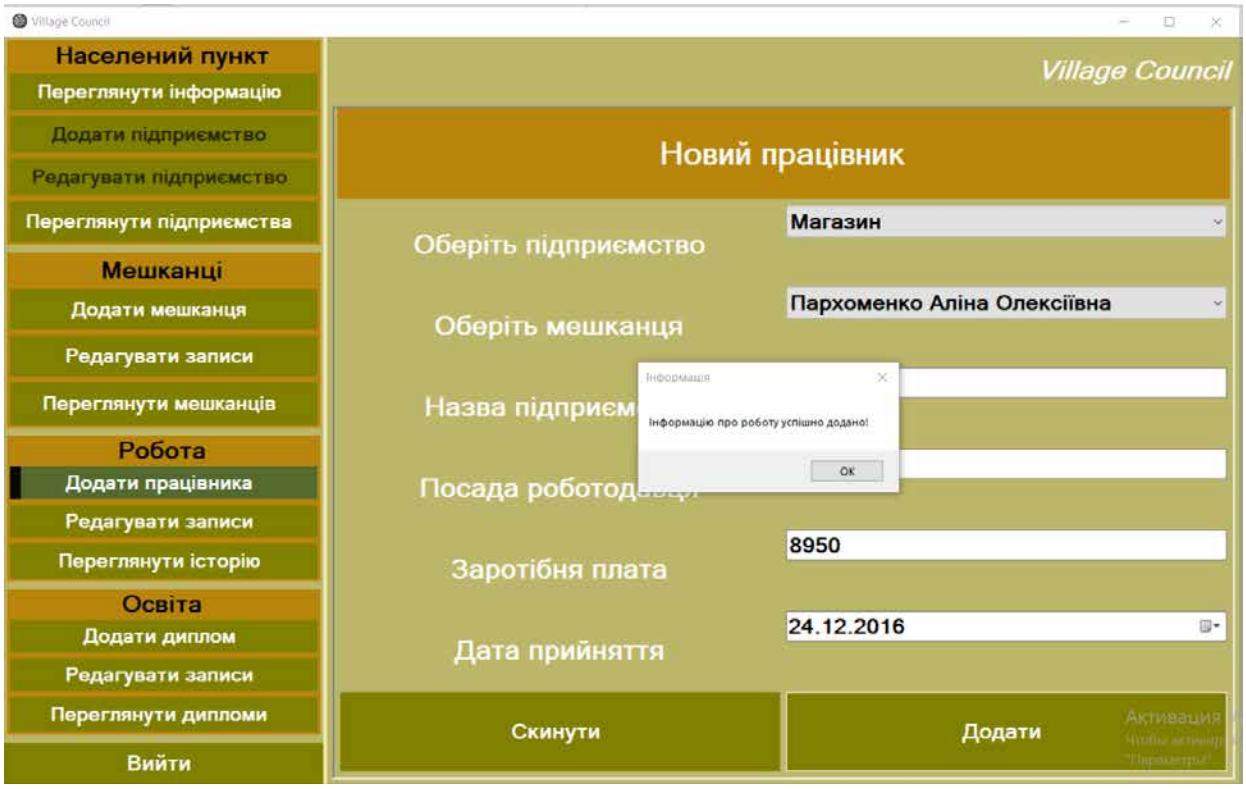


Рис. 4.14 Повідомлення про успішне додавання працівника

На вкладці «Звільнити працівника» можемо звільнити працівника. Обраємо працівника, посаду та дату звільнення, натискаємо кнопку «Звільнити» (Рис. 4.15).

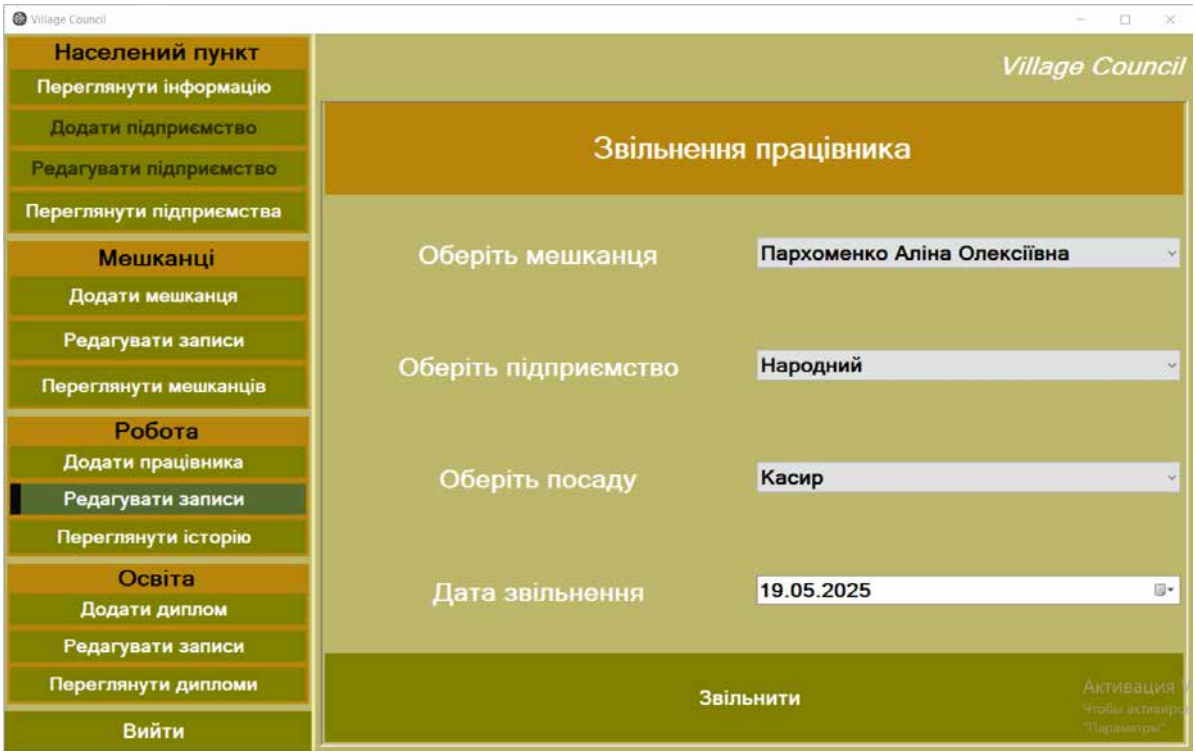


Рис. 4.15 Звільнення праццівника

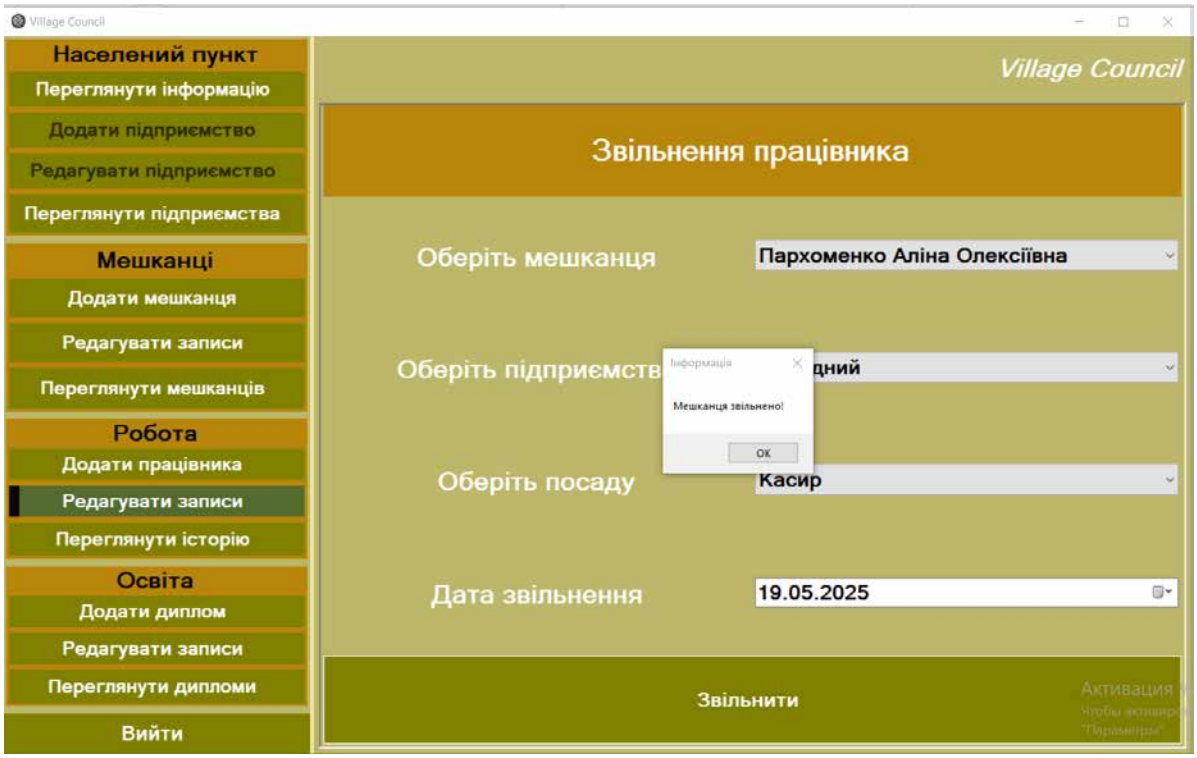


Рис. 4.16 Повідомлення про успішне звільнення працівника

На вкладці «Переглянути історію» можемо преглянути істрої роботи і згенерувати звіт (рис. 4.17). Натискаємо на кнопку «Згенерувати звіт» (рис. 4.18).

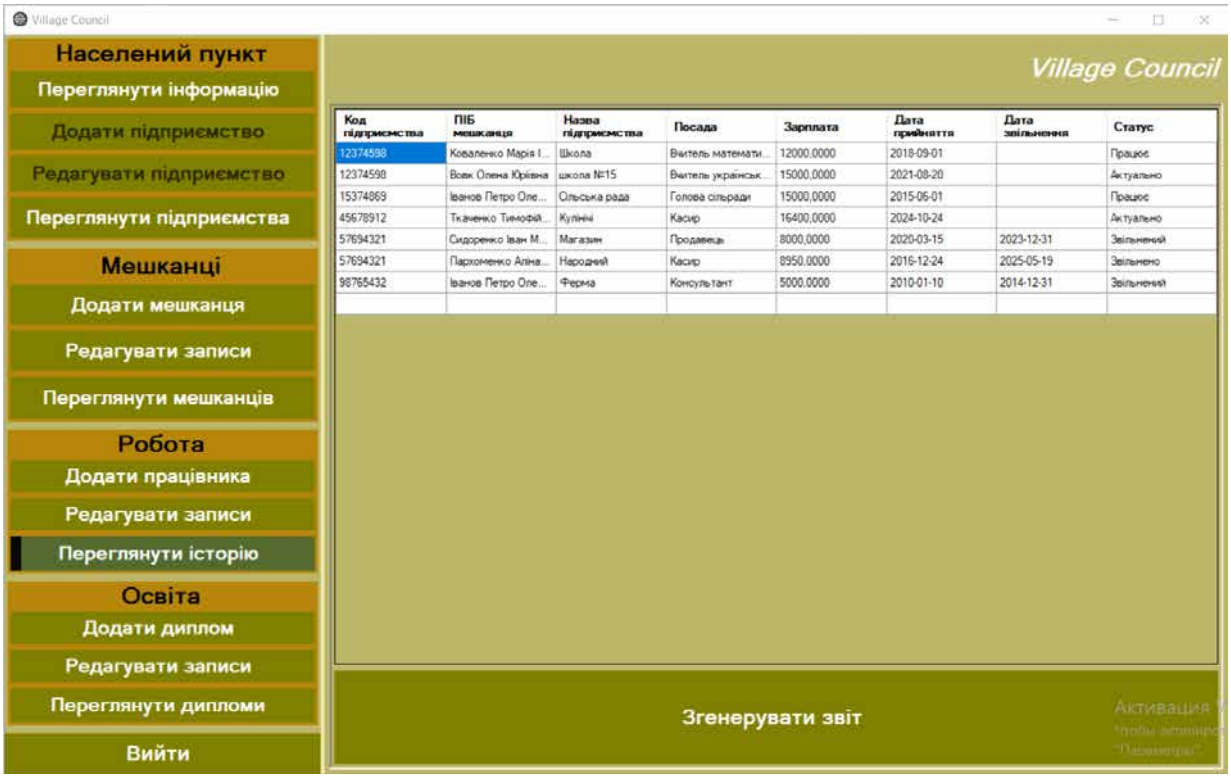
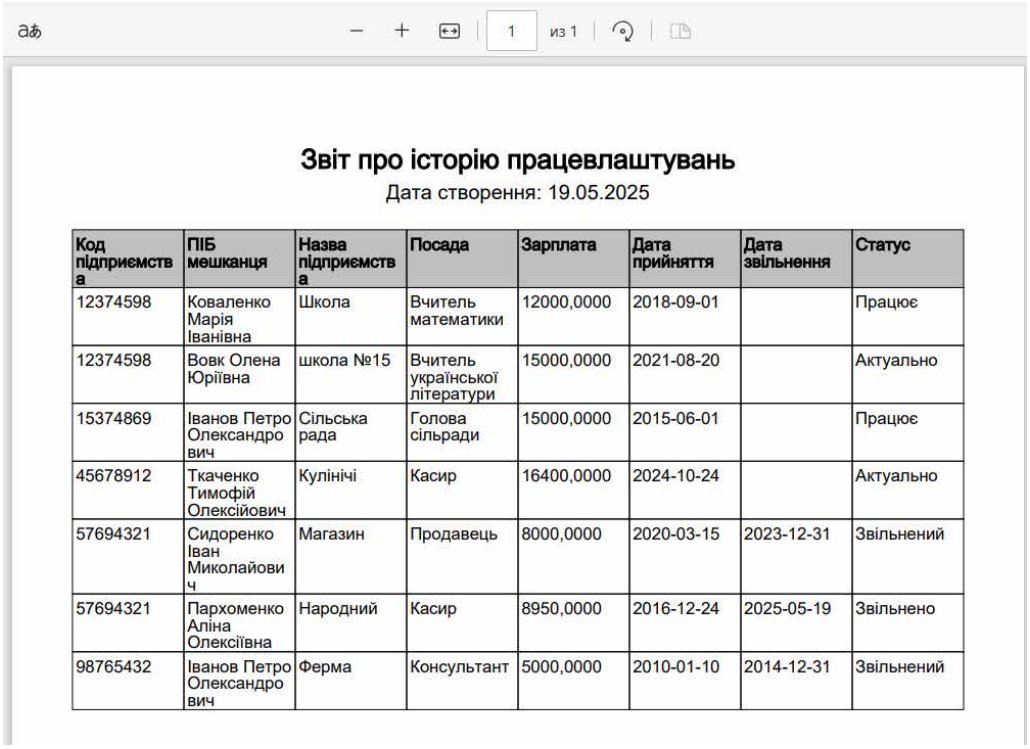


Рис. 4.17 Перегляд історії



Звіт про історію працевлаштувань

Дата створення: 19.05.2025

Код підприємства	ПІБ мешканця	Назва підприємства	Посада	Зарплата	Дата прийняття	Дата звільнення	Статус
12374598	Коваленко Марія Іванівна	Школа	Вчитель математики	12000,0000	2018-09-01		Працює
12374598	Вовк Олена Юріївна	школа №15	Вчитель української літератури	15000,0000	2021-08-20		Актуально
15374869	Іванов Петро Олександрович	Сільська рада	Голова сільради	15000,0000	2015-06-01		Працює
45678912	Ткаченко Тимофій Олексійович	Кулінічі	Касир	16400,0000	2024-10-24		Актуально
57694321	Сидоренко Іван Миколайович	Магазин	Продавець	8000,0000	2020-03-15	2023-12-31	Звільнений
57694321	Пархоменко Аліна Олексіївна	Народний	Касир	8950,0000	2016-12-24	2025-05-19	Звільнено
98765432	Іванов Петро Олександрович	Ферма	Консультант	5000,0000	2010-01-10	2014-12-31	Звільнений

Рис. 4.18 Згенерований звіт історій працевлаштувань

На вкладці «Додати диплом», можемо додати диплом про закінчення освіти (рис. 4.19). Заповнюємо поля і натискаємо на кнопку «Додати» (рис. 4.20).

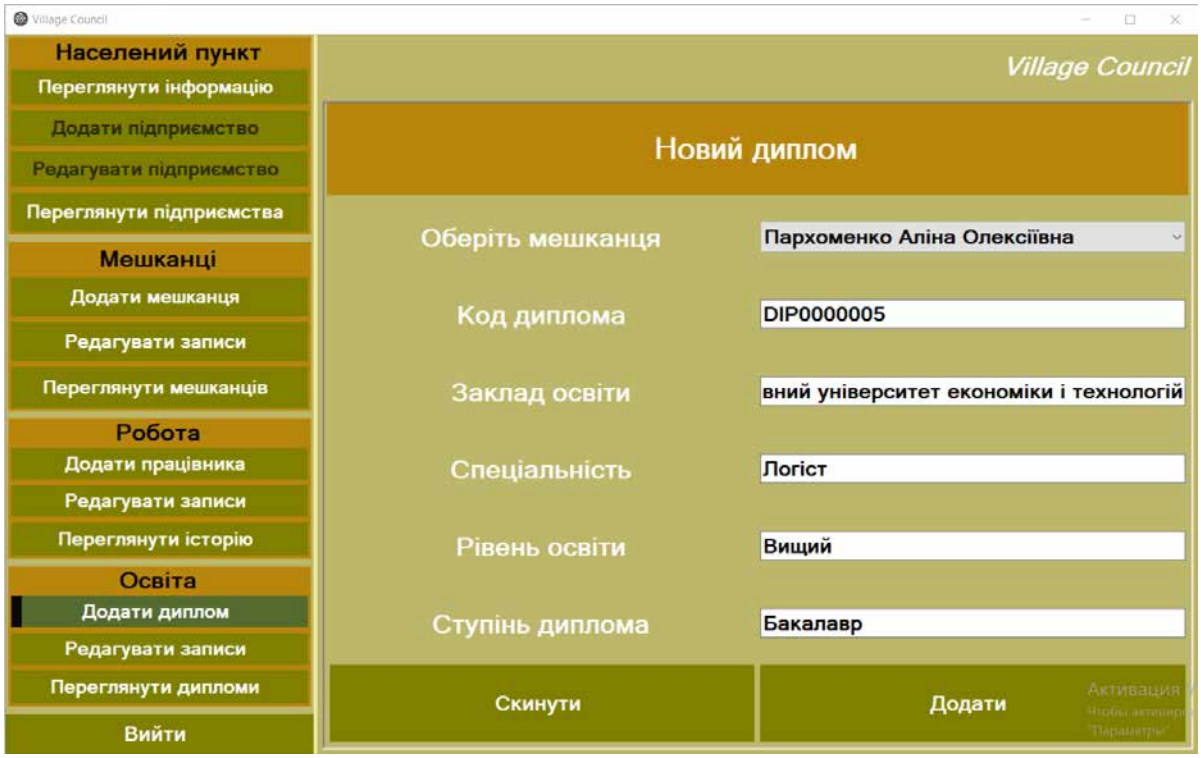


Рис. 4.19 Додавання диплому

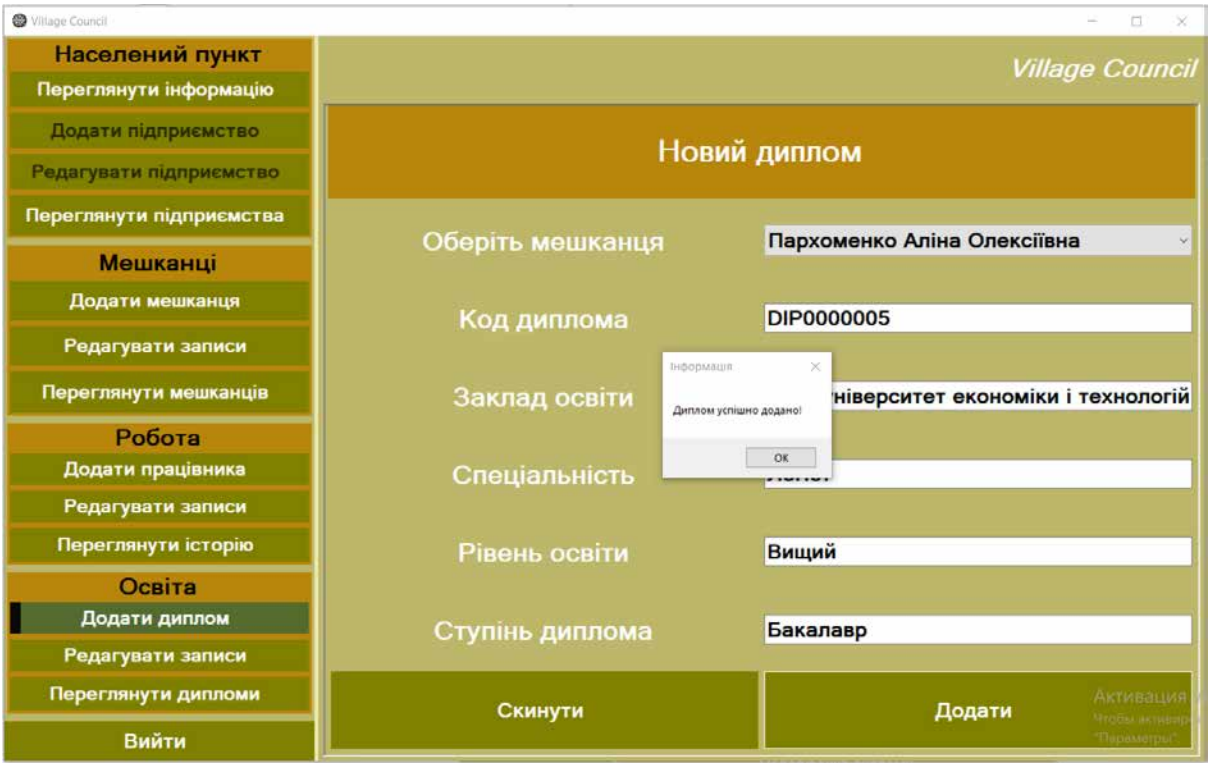


Рис. 4.20 Повідомлення про успішне додавання диплому

Також, можемо редагувати інформацію про дипломи (рис. 4.21).

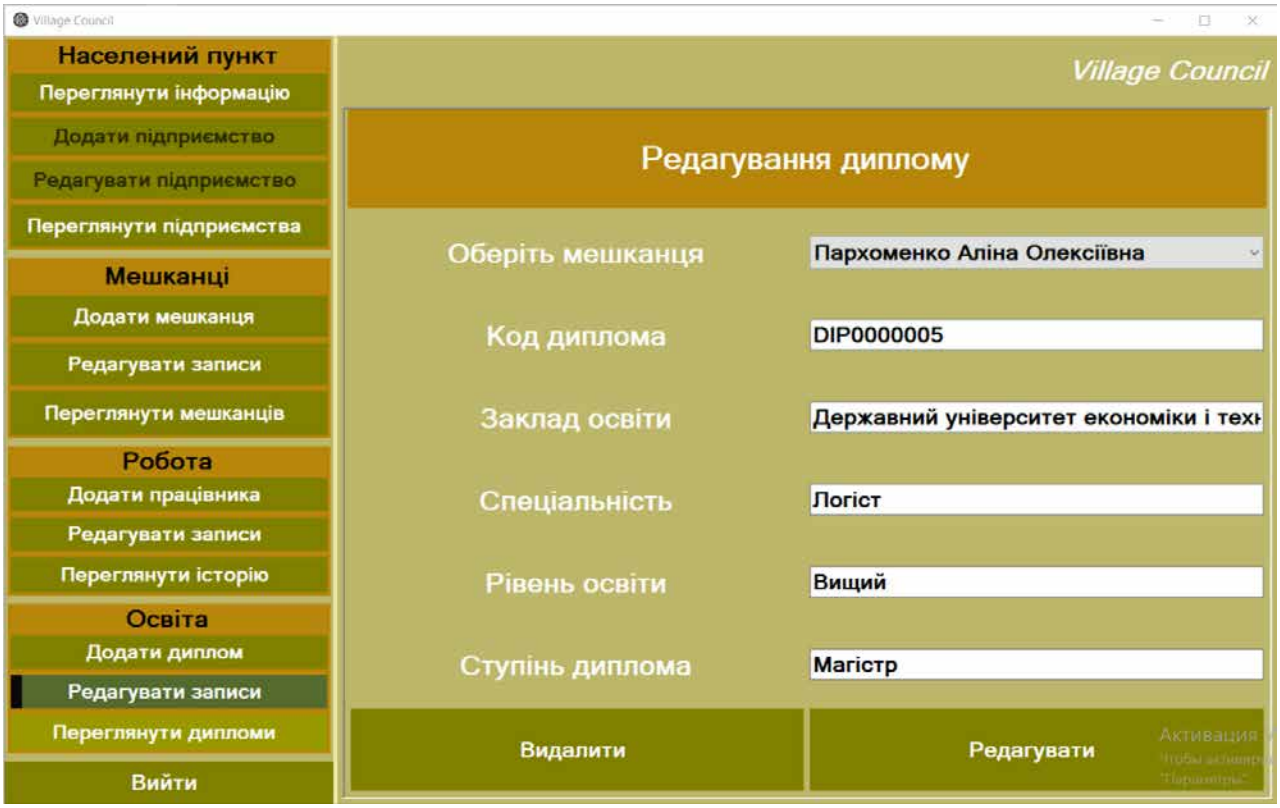


Рис. 4.21 Редагування інформації про диплом

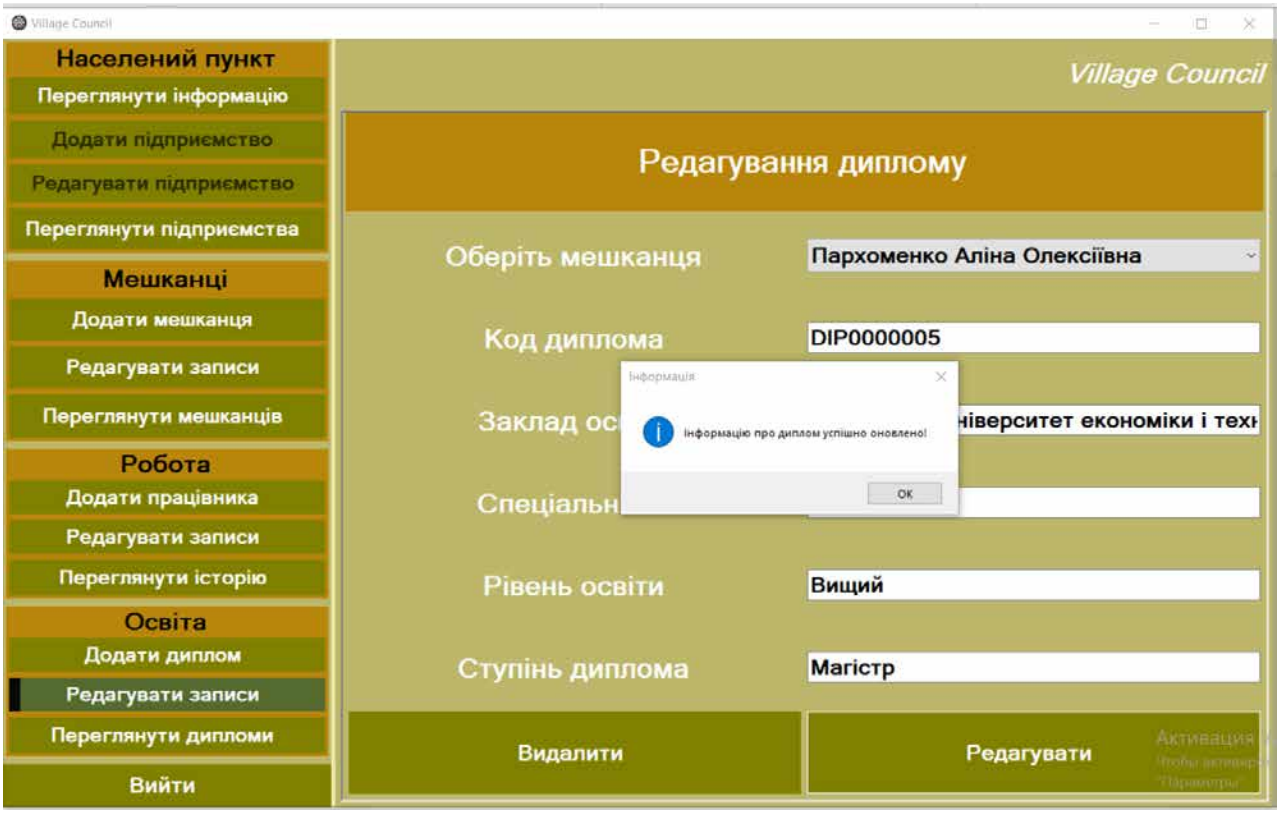


Рис. 4.22 Повідомлення про успішне редагування

На вкладці «Переглянути диплом» можемо переглянути інформацію про дипломи і згенерувати звіт (рис. 4.23). Натискаємо на кнопку «Згенерувати звіт» (рис. 4.24).

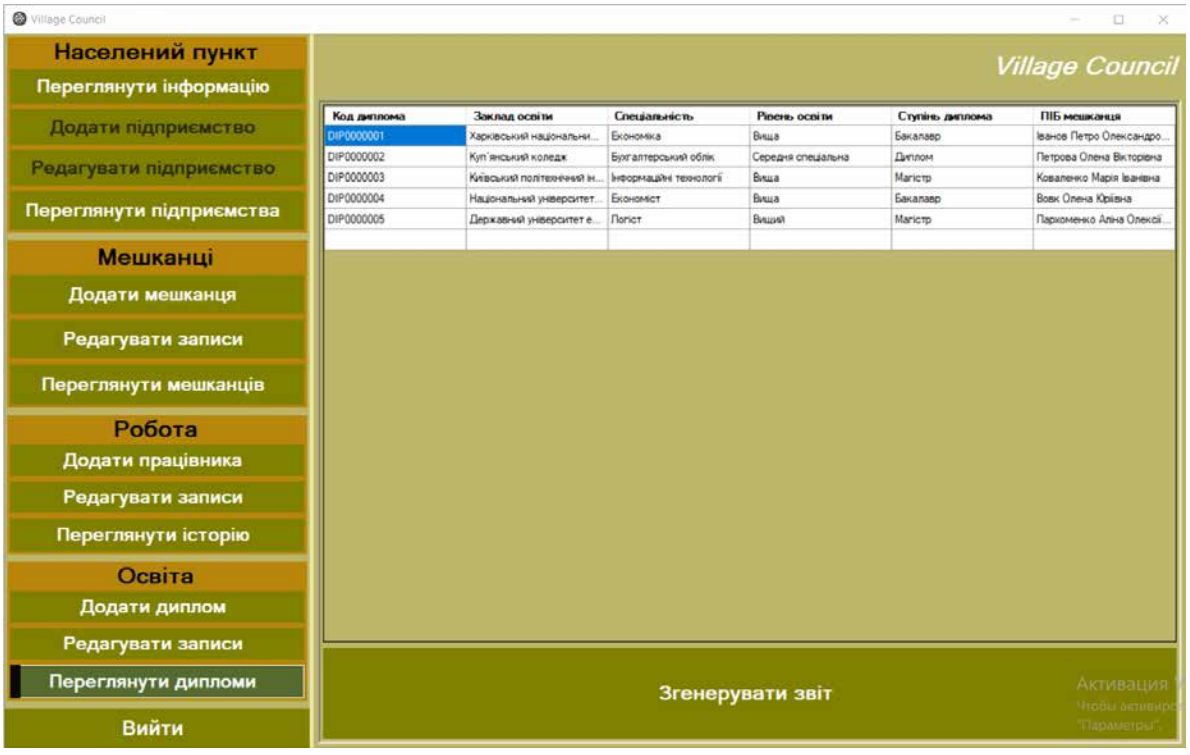
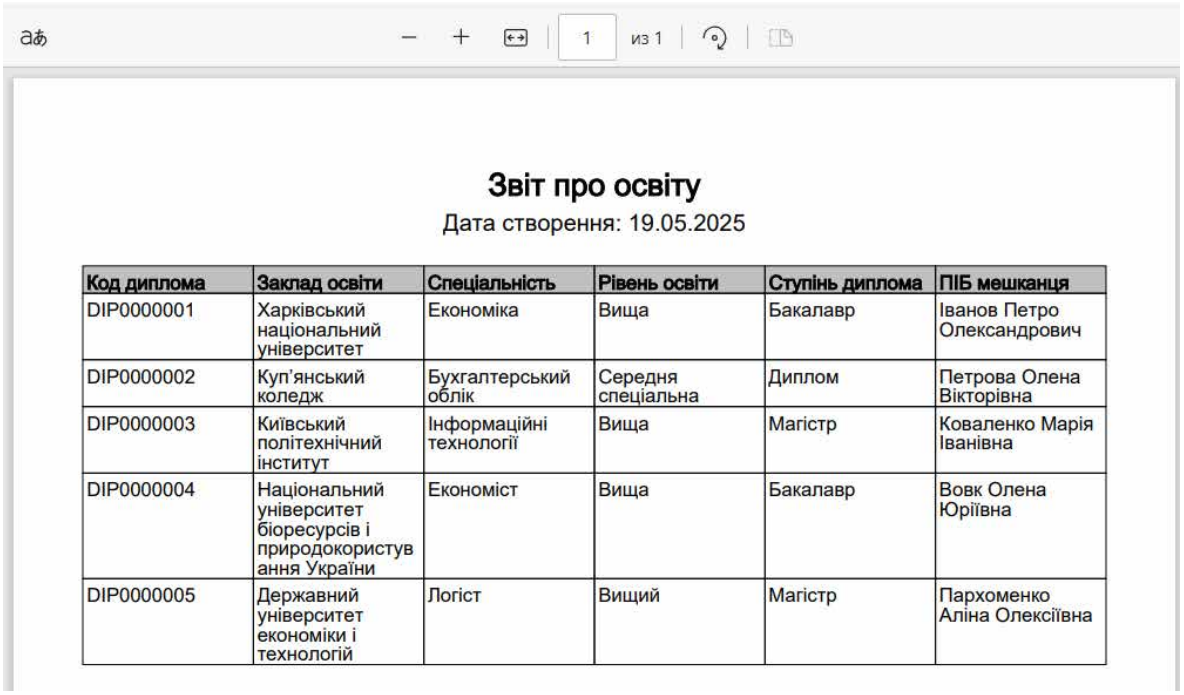


Рис. 4.24 Вкладка «Переглянути диплом»



Звіт про освіту

Дата створення: 19.05.2025

Код диплома	Заклад освіти	Спеціальність	Рівень освіти	Ступінь диплома	ПІБ мешканця
DIP0000001	Харківський національний університет	Економіка	Вища	Бакалавр	Іванов Петро Олександрович
DIP0000002	Куп'янський коледж	Бухгалтерський облік	Середня спеціальна	Диплом	Петрова Олена Вікторівна
DIP0000003	Київський політехнічний інститут	Інформаційні технології	Вища	Магістр	Коваленко Марія Іванівна
DIP0000004	Національний університет біоресурсів і природокористування України	Економіст	Вища	Бакалавр	Вовк Олена Юріївна
DIP0000005	Державний університет економіки і технологій	Логіст	Вищий	Магістр	Пархоменко Аліна Олексіївна

Рис. 4.25 Згенерований звіт про освіту

Ми розглянули усі можливості роботи секретаря, тепер увійдемо в систему у ролі голови села. Вводимо його логін та пароль (рис. 4.26), натискаємо кнопку увійти, якщо данні введено правильно відкриється нова форма.

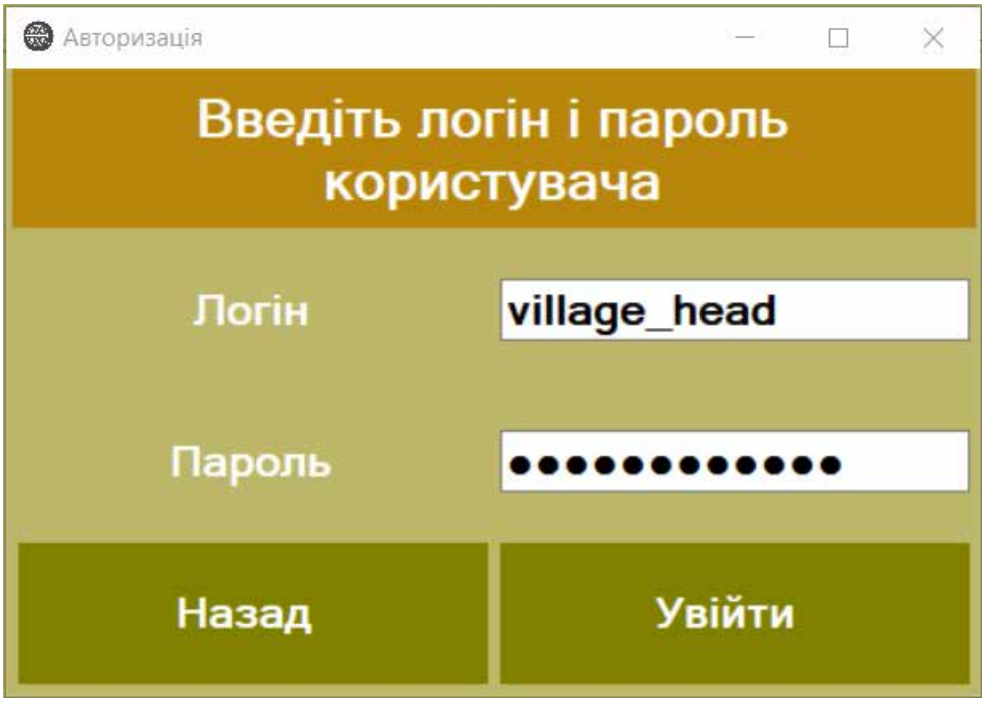


Рис. 4.26 Авторизація у ролі голови села

У сільлського голови є безліч обмежень у користуванні данною програмою. На вкладці «Додати підприємство», можемо додати підприємство. Заповнюємо поля і натискаємо кнопку «Додати» (рис. 4.27).

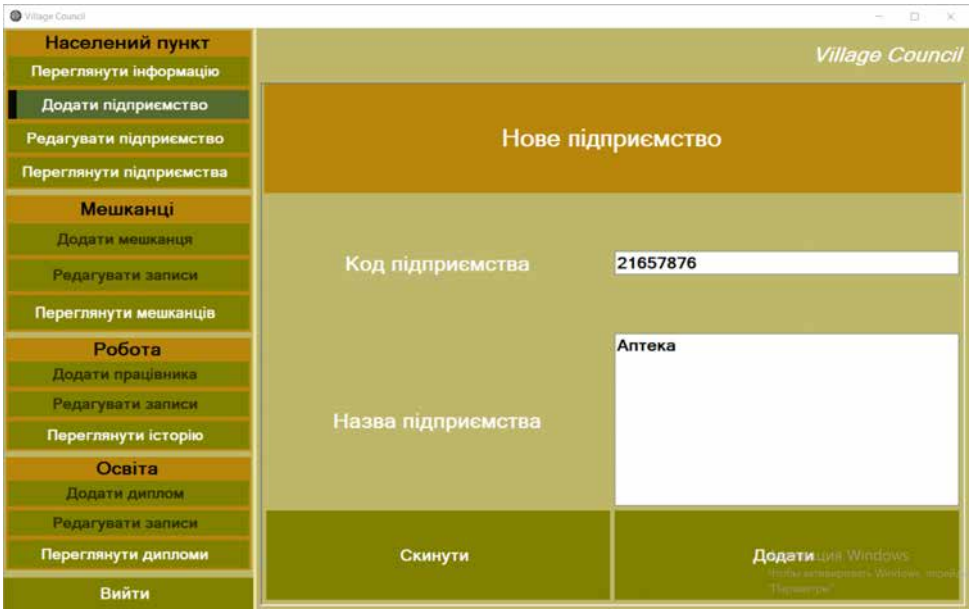


Рис. 4. 27 Форма «Додати підприємство»

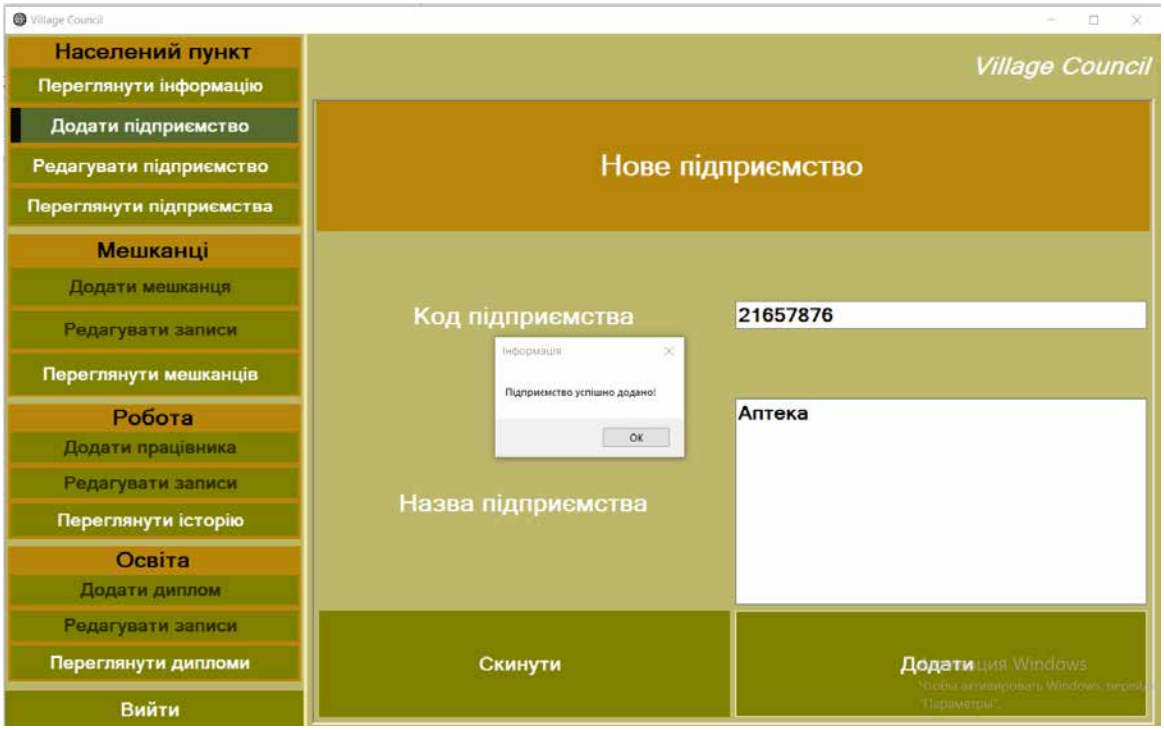


Рис. 4.27 Повідомлення про успішне створення підприємства

На вкладці «Редагувати підприємство», можемо відредагувати назву компанії. Заповнюємо поле новою інформацією і натискаємо на кнопку «Редагувати» (рис. 4.18).

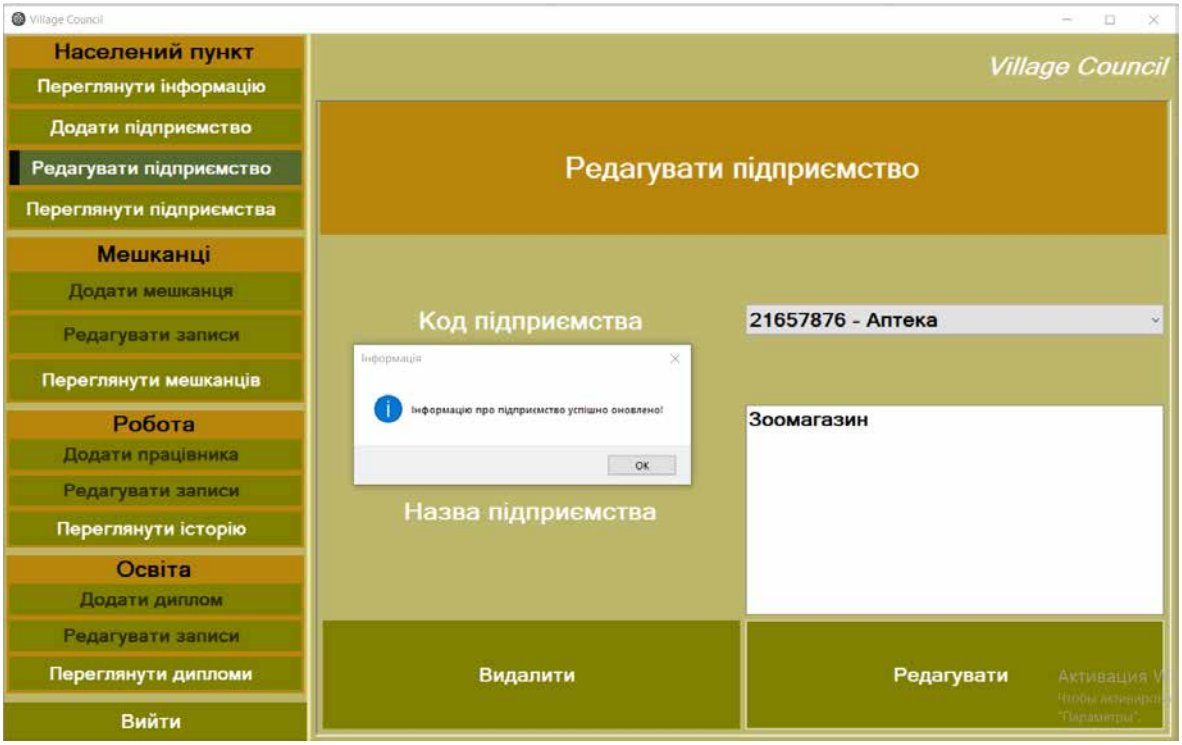


Рис. 4.28 Повідомлення про успішне редагування

Перейдемо на вкладку «Переглянути підприємства» (рис. 4.29), тут можна побачити інформацію про підприємства та згенерувати звіт у PDF форматі.

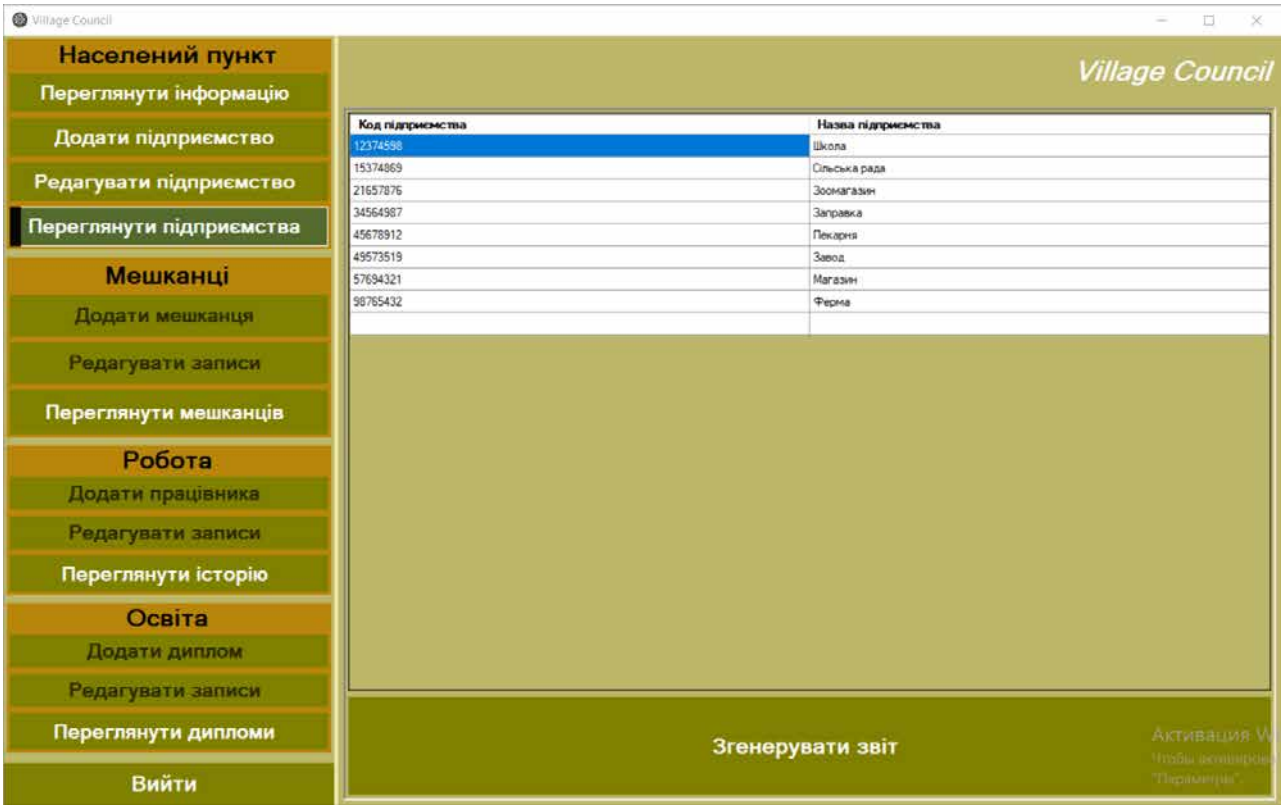


Рис. 4.29 Перегляд підприємства

4.2 Вимоги до апаратного та програмного забезпечення

Для забезпечення повноцінного функціонування ПЗ системи, необхідно чітко визначити як апаратні, так і програмні вимоги, які гарантують стабільність роботи, сумісність з обраними технологіями та зручність користувацької взаємодії. Оскільки система передбачає роботу із базою даних, багатоформовий графічний інтерфейс, доступ до даних у реальному часі та функції авторизації, то мінімальні вимоги мають бути виваженими та обґрунтованими на основі реальних сценаріїв використання. Ураховуючи, що система може бути розгорнута як у невеликій сільській раді, так і в іншій адміністративній установі, де не завжди доступне найсучасніше технічне забезпечення, її реалізація повинна залишатися придатною для експлуатації на комп’ютерах із помірною обчислювальною потужністю.

Для стабільної роботи рекомендується мати комп’ютер із процесором не нижче Intel Core i3 третього покоління або аналогічним за характеристиками AMD, з тактовою частотою не менше 2.0 ГГц. Об’єм оперативної пам’яті

повинен становити щонайменше 4 ГБ, хоча рекомендованим є 8 ГБ і більше, що забезпечить комфортну роботу із кількома одночасно відкритими формами, великим обсягом даних та забезпечить відсутність затримок при виконанні складних операцій. Жорсткий диск має забезпечувати достатній простір для встановлення всієї системи, бази даних та резервних копій, тому мінімальний обсяг вільного місця має становити не менше 2-4 ГБ, а для надійного зберігання історичних даних і звітів – від 10 ГБ і більше. Екран із роздільною здатністю не нижче 1366×768 пікселів є обов’язковим, так як інтерфейс передбачає наявність форм, які повинні вміщуватися без горизонтального прокручування.

З боку ПЗ умовою є встановлення операційної системи Windows 10 або новішої, оскільки розробка здійснювалася з використанням середовища Windows Forms, яке найкраще підтримується в сучасних версіях операційних систем Microsoft. Для коректної роботи програми також необхідно мати встановлений .NET Framework версії не нижче 4.7.2, що забезпечує підтримку всіх функцій, використаних у програмі, зокрема обробку подій, роботу із базою даних через SqlConnection, відображення інтерфейсу користувача та формування звітів. Для запуску системи також необхідна наявність Microsoft SQL Server Express Edition або повної версії SQL Server 2019, де буде розміщена база даних. Бажано, щоб інстанс SQL Server мав можливість запускатися автоматично разом з операційною системою, що дозволить уникнути помилок підключення у випадку неуважності користувача. У разі використання локальної конфігурації допускається встановлення усіх компонентів на одному комп’ютері, однак для розширеної мережевої роботи можливе винесення бази на окремий сервер або централізований комп’ютер, що дозволить обслуговувати декілька робочих місць одночасно.

Для повноцінного функціонування багатокористувацького середовища також доцільно врахувати мережеві вимоги, особливо у випадку розподіленої архітектури системи. Мінімальна пропускна здатність локальної мережі повинна становити не менше 100 Мбіт/с, з бажаним переходом на гігабітне з'єднання для підвищення продуктивності при передачі великих обсягів даних

(наприклад, під час генерації звітів або роботи з архівом). Система має бути адаптована до роботи в умовах DHCP або статичної IP-конфігурації, що полегшує адміністрування мережевої взаємодії між клієнтами та сервером бази даних.

Увагу слід приділити безпеці програмного середовища. Рекомендується встановити антивірусне програмне забезпечення та налаштувати брандмауер, що дозволяє вибірковий доступ до SQL Server з конкретних IP-адрес або підмереж. Для захисту даних користувачів та облікових записів доцільно використовувати облікові записи Windows з обмеженими правами, шифрування конфігураційних файлів програми (наприклад, app.config) та реалізацію журналювання дій користувачів у системі.

З точки зору адміністрування, для зручного обслуговування рекомендується мати доступ до SQL Server Management Studio (SSMS), що дозволяє візуально переглядати структуру бази даних, виконувати запити, резервне копіювання та відновлення, а також надавати або обмежувати доступ для користувачів на основі їхніх ролей у системі. Для забезпечення високої доступності бази даних варто налаштувати періодичне автоматичне резервне копіювання, зберігаючи бекапи окремо від основного пристрою.

Важливо врахувати вимоги до периферійних пристроїв, які можуть знадобитися під час роботи з програмою. Наприклад, для друку довідок або звітів у паперовому вигляді необхідна наявність принтера із підтримкою формату A4. Бажано, щоб принтер підтримував драйвери для Windows 10 і мав USB-інтерфейс для швидкого підключення. Для зберігання архівів на зовнішніх носіях можливе підключення зовнішнього жорсткого диска або флеш-накопичувача.

4.3 Склад інсталяційного пакету

Основним виконуваним файлом, який відповідає за запуск додатку, є Village_Council.exe. Саме він виступає точкою входу у систему і реалізує всю логіку роботи користувача із функціоналом обліку мешканців, редагування даних, генерації звітів та взаємодії з базою даних. Поруч з цим файлом знаходиться файл конфігурації Village_Council.exe.config, який містить параметри підключення до SQL Server, можливі шляхи до ресурсів, а також інші змінні, що регулюють поведінку системи при запуску. Наявність цього конфігураційного файлу дозволяє адаптувати додаток під конкретне середовище без перекомпіляції, зокрема змінювати назву сервера бази даних, тип автентифікації або інші глобальні параметри.

Файл Village_Council.pdb є відладочним модулем, який зберігає символьну інформацію, необхідну для відстеження помилок під час розробки або відлагодження. Він корисний на етапі підтримки системи, оскільки дозволяє локалізувати помилки, які виникають у процесі експлуатації.

Окрім основного виконуваного файлу та пов’язаних із ним службових елементів, до складу інсталяційного пакету входять і зовнішні бібліотеки, які забезпечують підтримку окремих функцій. Наприклад, бібліотека itextsharp.dll реалізує механізми генерації звітів у форматі PDF. Вона дозволяє створювати документ із таблицями, заголовками, форматуванням, українською мовою та з використанням шрифтів, які відповідають вимогам офіційних документів. Файл itextsharp.xml, що знаходиться поруч, містить XML-документацію до функцій бібліотеки, що може бути корисною у разі розширення системи або повторного використання компонентів.

Додатково, до пакету включено бібліотеку BouncyCastle.Crytography.dll та її XML-опис, яка забезпечує функції криптографії, зокрема може використовуватися для шифрування даних, хешування паролів або реалізації цифрових підписів у системі, якщо такі функції передбачені у перспективі розвитку.

ВИСНОВКИ

У результаті проведеного дослідження, розробки та впровадження програмного забезпечення для автоматизованого обліку мешканців населеного пункту було досягнуто поставленої мети – створено повнофункціональну інформаційну систему, яка враховує потреби місцевого самоврядування, забезпечує ефективне управління персоніфікованими даними, формування звітності, ведення історії змін та контроль за актуальністю даних. Усі етапи розробки системи – від аналізу предметної області до розгортання робочого інтерфейсу – були логічно узгоджені між собою, що дозволило досягти високої функціональності, стабільності та відповідності технічним і логічним вимогам.

Здійснене проектування передбачало використання модульної архітектури, що дало змогу структурувати програмний продукт на зрозумілі та взаємопов’язані компоненти: зокрема, реалізовано модулі аутентифікації користувачів, введення і редагування даних мешканців, створення і генерації звітів, ведення бази даних підприємств та закладів освіти, облік працевлаштування і рівня освіти населення. Система побудована з урахуванням можливості масштабування, адаптації під інші регіони або сільські ради, а також подальшого розвитку шляхом інтеграції з іншими електронними сервісами.

Вибір C# та платформи .NET дав змогу реалізувати надійний графічний інтерфейс, використання технології Entity Framework забезпечило зручне управління з’єднанням із базою даних, виключивши необхідність ручного написання SQL-запитів у більшості сценаріїв. Microsoft SQL Server продемонстрував високу стабільність при роботі з великими обсягами даних, надаючи підтримку транзакцій, реплікації, ролей доступу та збережених процедур. Гнучкість, яку надали засоби .NET і Windows Forms, дозволила реалізувати інтерфейс, зрозумілий користувачам з базовим рівнем комп’ютерної підготовки.

Завдяки комплексному підходу до алгоритмізації основних функцій, програма спрощує роботу секретаря або голови сільської ради та виступає

ефективним засобом контролю за змінами даних, ведення архіву, формування звітів на основі заданих фільтрів, експорту інформації у формати для друку та збереження історії взаємодії з системою. Під час тестування системи було виявлено, що усі функціональні модулі працюють стабільно, програма коректно обробляє некоректні дані, генерує підказки для користувача, а система прав доступу обмежує функціонал залежно від ролі користувача.

Реалізоване програмне забезпечення вирішує актуальні проблеми діджиталізації місцевого самоврядування, зокрема – ведення актуального обліку мешканців, покращення обміну інформацією між працівниками адміністрації та забезпечення прозорості управлінських рішень. Запропонована система довела свою ефективність під час апробації, вона готова до подальшого вдосконалення – наприклад, впровадження мобільного додатку, синхронізації з державною базою або створення веб-версії для доступу через браузер.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Інструментарій соціальної політики (за Л. Палом та Т. Ганслі) [Електронний ресурс] – Режим доступу до ресурсу: https://studies.in.ua/socialna-polityka-derzhavy/3457-nstrumentary-socalnoyi-poltiki-za-l-palom-ta-t-gansl.html.

2. Конфіденційна інформація, інформація про особу та персональні дані: співвідношення і регулювання [Електронний ресурс] – Режим доступу до ресурсу: https://cedem.org.ua/analytics/konfidentsijna-informatsiya-informatsiya-pro-osobu-ta-personalni-dani-spivvidnoshennya-i-regulyuvannya/.

3. Про захист персональних даних [Електронний ресурс] – Режим доступу до ресурсу: https://www.rada.gov.ua/.

4. Про місцеве самоврядування в Україні [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/280/97-%D0%B2%D1%80#Text.

5. Єдина інформаційна система соціальної сфери відтепер відкрита для всіх держорганів [Електронний ресурс] – Режим доступу до ресурсу: https://www.kmu.gov.ua/news/yedyna-informatsiina-systema-sotsialnoi-sfery-vidteper-vidkryta-dlia-vsikh-derzhorhaniv.

6. M.E.Doc.Держава [Електронний ресурс] – Режим доступу до ресурсу: https://medoc.ua/.

7. 1С:Підприємство [Електронний ресурс] – Режим доступу до ресурсу: https://eod.ulk.gov.ua/uk_UA/.

8. Як будувати UML-діаграми [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/forums/topic/40575/.

9. Діаграма послідовності (Sequence Diagrams) [Електронний ресурс] – Режим доступу до ресурсу: https://www.maxzosim.com/sequence-diagrams/.

10. Куліков В.М., Рябцев В.В., Паршуков С.С. Об’єктно-орієнтоване програмування для фахівців з кібербезпеки: навч. посіб. / ІСЗЗІ КПІ ім. Ігоря Сікорського. Київ: КПІ ім. Ігоря Сікорського, 2023. 365 с.

11. Для чого потрібні UML діаграми? [Електронний ресурс] – Режим доступу до ресурсу: https://foxminded.ua/uml-diagramy/.

12. Петрик М.Р. Моделювання програмного забезпечення : науковометодичний посібник / М.Р. Петрик, О.Ю. Петрик – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2015. – 200 с.

13. Діаграма розгортання [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/uk/deployment-diagram-uml-example.html

14. Каштан В.Ю., Іванов Д.В. Конспект лекцій з дисципліни “Бази даних в інформаційних системах”. Для студентів галузі знань 12 “Інформаційні технології” спеціальності 126 “Інформаційні системи та технології”. – Д.: НТУ «ДП», 2021. – 58 с.

15. Застосування блок-схем у розробці програм [Електронний ресурс] – Режим доступу до ресурсу: https://foxminded.ua/blok-skhema/.

16. Моделювання даних (Data Modelling) [Електронний ресурс] – Режим доступу до ресурсу: https://www.maxzosim.com/data-modelling/.

17. Модель діаграми зв’язків сутностей (ER) із прикладом СУБД [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/uk/er-diagram-tutorial-dbms.html.

18. Реляційні бази даних усе, що необхідно про них знати [Електронний ресурс] – Режим доступу до ресурсу: https://foxminded.ua/reliatsiini-bazy-danykh/

19. Що таке діаграма компонентів UML в OOAD? [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/uk/component-diagram-uml-example.html.

20. Про мову програмування C# [Електронний ресурс] – Режим доступу до ресурсу: https://foxminded.ua/prohramuvannia-na-si/.

21. Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни "Організація баз даних і знань (ADO.NET)" для студентів напряму підготовки "Комп’ютерні науки" денної форми навчання / укл. М. Ю. Лосєв, О. В. Тарасов, В. В. Федько. – Харків : Вид. ХНЕУ, 2010. – 88 с. (Укр. мов.).

ДОДАТОК А

База даних

Сторінок 7

Київ-2025


```
IF NOT EXISTS (SELECT * FROM sys.databases WHERE name = 'Village_Council')
    CREATE DATABASE Village_Council;
```

```
GO


-- Використання бази даних і створення структури
USE Village_Council
GO


-- Видалення таблиць, якщо вони існують
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Job' AND type_desc =
'USER_TABLE')
   DROP TABLE Job
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Education' AND type_desc =
'USER_TABLE')
   DROP TABLE Education
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Inhabitant' AND type_desc =
'USER_TABLE')
   DROP TABLE Inhabitant
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Enterprise' AND type_desc =
'USER_TABLE')
   DROP TABLE Enterprise
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Village' AND type_desc =
'USER_TABLE')
   DROP TABLE Village
GO


-- Таблиця "Село"
CREATE TABLE Village
(
   Code_village CHAR(10) NOT NULL PRIMARY KEY,
   Village_name VARCHAR(255) NOT NULL,
```



```
   Village_address VARCHAR(255),
   Village_phone VARCHAR(17)
```

```
)
-- Таблиця "Підприємство"
CREATE TABLE Enterprise
(
   Code_enterprise CHAR(8) NOT NULL PRIMARY KEY,
   Enterprise_name VARCHAR(50) NOT NULL
)
-- Таблиця "Мешканець"
CREATE TABLE Inhabitant
(
   Code_inhabitant CHAR(10) NOT NULL PRIMARY KEY,
   Last_name VARCHAR(50) NOT NULL,
   First_name VARCHAR(50) NOT NULL,
   Surname VARCHAR(50) NOT NULL,
   Sex VARCHAR(35),
   Birthday DATE,
   Home_address VARCHAR(255) NOT NULL,
   Phone_number VARCHAR(17),
   Status_employment VARCHAR(50) NOT NULL,
   Code_village CHAR(10) NOT NULL,
   FOREIGN KEY (Code_village) REFERENCES Village(Code_village)
)
-- Таблиця "Освіта"
CREATE TABLE Education
(
   Code_diploma CHAR(10) NOT NULL PRIMARY KEY,
```



```
   Educational_institution VARCHAR(75) NOT NULL,
   Specialty VARCHAR(75) NOT NULL,
```

```
    Level_education VARCHAR(35) NOT NULL,
    Diploma_degree VARCHAR(35) NOT NULL,
    Code_inhabitant CHAR(10) NOT NULL,
    FOREIGN KEY (Code_inhabitant) REFERENCES Inhabitant(Code_inhabitant)
)
-- Таблиця "Місце роботи"
CREATE TABLE Job
(
    Code_enterprise CHAR(8) NOT NULL,
    Code_inhabitant CHAR(10) NOT NULL,
    Company_name VARCHAR(50) NOT NULL,
    Position VARCHAR(50) NOT NULL,
    Salary MONEY NOT NULL,
    Date_adoption DATE NOT NULL,
    Date_release DATE,
    Status_job VARCHAR(50) NOT NULL,
    PRIMARY KEY (Code_enterprise, Code_inhabitant, Position, Date_adoption),
    FOREIGN KEY (Code_enterprise) REFERENCES Enterprise(Code_enterprise),
    FOREIGN KEY (Code_inhabitant) REFERENCES Inhabitant(Code_inhabitant)
)
GO
-- Заповнення таблиці "Село"
BEGIN TRANSACTION vil
INSERT INTO Village (Code_village, Village_name, Village_address, Village_phone) VALUES
('6323287101', 'Моначинівка', '63712, Харківська обл., Куп’янський р-н, с. Моначинівка,
вул. Жердія, 21', '+380-57-425-64-84');
COMMIT
```



```
GO
-- Заповнення таблиці "Підприємство"
```

```
BEGIN TRANSACTION ent

INSERT INTO Enterprise (Code_enterprise, Enterprise_name) VALUES

('15374869', 'Сільська рада'),

('12374598', 'Школа'),

('57694321', 'Магазин'),

('49573519', 'Завод'),

('34564987', 'Заправка'),

('98765432', 'Ферма'),

('45678912', 'Пекарня');

COMMIT

GO

-- Заповнення таблиці "Мешканець"

BEGIN TRANSACTION inh

INSERT INTO Inhabitant (Code_inhabitant, Last_name, First_name, Surname, Sex, Birthday,
Home_address, Phone_number, Status_employment, Code_village) VALUES

('0000000001', 'Іванов', 'Петро', 'Олександрович', 'Чоловік', '1985-03-15', 'с. Моначинівка,
вул. Жердія, 22', '+380-67-123-45-67', 'Працевлаштований', '6323287101'),

('0000000002', 'Петрова', 'Олена', 'Вікторівна', 'Жінка', '1990-07-22', 'с. Моначинівка, вул.
Садова, 5', '+380-67-234-56-78', 'Непрацевлаштована', '6323287101'),

('0000000003', 'Сидоренко', 'Іван', 'Миколайович', 'Чоловік', '1978-11-30', 'с. Моначинівка,
вул. Центральна, 12', '+380-67-345-67-89', 'Працевлаштований', '6323287101'),

('0000000004', 'Коваленко', 'Марія', 'Іванівна', 'Жінка', '1995-05-10', 'с. Моначинівка, вул.
Польова, 8', '+380-67-456-78-90', 'Працевлаштована', '6323287101'),

('0000000005', 'Мельник', 'Олег', 'Васильович', 'Чоловік', '1982-09-25', 'с. Моначинівка, вул.
Лісова, 3', '+380-67-567-89-01', 'Непрацевлаштований', '6323287101');

COMMIT

GO

-- Заповнення таблиці "Освіта"

BEGIN TRANSACTION edu
```



```
INSERT INTO Education (Code_diploma, Educational_institution, Specialty, Level_education,
Diploma_degree, Code_inhabitant) VALUES
```

('DIP0000001', 'Харківський національний університет', 'Економіка', 'Вища', 'Бакалавр', '0000000001'),

('DIP0000002', 'Куп’янський коледж', 'Бухгалтерський облік', 'Середня спеціальна', 'Диплом', '0000000002'),

('DIP0000003', 'Київський політехнічний інститут', 'Інформаційні технології', 'Вища', 'Магістр', '0000000004');

COMMIT

GO

-- Заповнення таблиці "Місце роботи"

BEGIN TRANSACTION job

INSERT INTO Job (Code_enterprise, Code_inhabitant, Company_name, Position, Salary, Date_adoption, Date_release, Status_job) VALUES

('15374869', '0000000001', 'Сільська рада', 'Голова сільради', 15000.00, '2015-06-01', NULL, 'Працює'),

('12374598', '0000000004', 'Школа', 'Вчитель математики', 12000.00, '2018-09-01', NULL, 'Працює'),

('57694321', '0000000003', 'Магазин', 'Продавець', 8000.00, '2020-03-15', '2023-12-31', 'Звільнений'),

('98765432', '0000000001', 'Ферма', 'Консультант', 5000.00, '2010-01-10', '2014-12-31', 'Звільнений');

COMMIT

GO

-- Налаштування логінів і користувачів

IF EXISTS (SELECT name FROM sys.server_principals WHERE name = 'village_head' AND type_desc = 'SQL_LOGIN')

DROP LOGIN village_head

IF EXISTS (SELECT name FROM sys.server_principals WHERE name = 'secretary' AND type_desc = 'SQL_LOGIN')

DROP LOGIN secretary

GO

EXEC sp_addlogin @loginame = 'village_head', @passwd = 'village_head'

EXEC sp_addlogin @loginame = 'secretary', @passwd = 'secretary'

```
GO


USE Village_Council
GO
IF EXISTS (SELECT name FROM sys.database_principals WHERE name = 'village_head' AND type_desc = 'SQL_USER')
BEGIN
   DROP SCHEMA village_head
   DROP USER village_head
END
IF EXISTS (SELECT name FROM sys.database_principals WHERE name = 'secretary' AND type_desc = 'SQL_USER')
BEGIN
   DROP SCHEMA secretary
   DROP USER secretary
END
GO


EXEC sp_adduser @loginame = 'village_head', @name_in_db = 'village_head'
EXEC sp_adduser @loginame = 'secretary', @name_in_db = 'secretary'
GO


-- Права доступу для village_head
GRANT SELECT, INSERT, UPDATE, DELETE ON Village TO village_head
GRANT SELECT, INSERT, UPDATE, DELETE ON Enterprise TO village_head
GRANT SELECT ON Inhabitant TO village_head
GRANT SELECT ON Education TO village_head
```



```
GRANT SELECT ON Job TO village_head
GO
```

```
-- Права доступу для secretary
GRANT SELECT ON Village TO secretary
GRANT SELECT ON Enterprise TO secretary
GRANT SELECT, INSERT, UPDATE, DELETE ON Inhabitant TO secretary
GRANT SELECT, INSERT, UPDATE, DELETE ON Education TO secretary
GRANT SELECT, INSERT, UPDATE, DELETE ON Job TO secretary
GO
```

ДОДАТОК Б

Код програми

Сторінок 23

Київ-2025

AddInhabitant.cs:
```
using System;
using System.Collections.Generic;
```

```
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace Village_Council
{
   public partial class AddInhabitant : Form
   {
public AddInhabitant()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         //this.Close();
         textBox1.Text = "";
         textBox2.Text = "";
         textBox3.Text = "";
         textBox4.Text = "";

      }

      private void AddInhabitant_Load(object sender, EventArgs e)
      {
         comboBox1.Items.Add("Чоловік");
         comboBox1.Items.Add("Жінка");
      }

      private void button2_Click(object sender, EventArgs e)
      {
         // Перевірка на введення всіх даних
         if (string.IsNullOrWhiteSpace(textBox1.Text) ||
string.IsNullOrWhiteSpace(textBox2.Text) || string.IsNullOrWhiteSpace(textBox3.Text) ||
            string.IsNullOrWhiteSpace(textBox4.Text) ||
string.IsNullOrWhiteSpace(comboBox1.Text) || string.IsNullOrWhiteSpace(comboBox2.Text))
         {
            MessageBox.Show("Будь ласка, заповніть усі поля!", "Помилка");
            return;
         }
```



```
         if (textBox1.Text.Length != 10)
         {
```

```
            MessageBox.Show("Код мешканця повинен складатися з 10 символів!",
"Помилка");
            return;
        }

        // Перевірка на правильність введення ПІБ
        if (!CheckName(textBox2.Text))
        {
            MessageBox.Show("Введіть повністю ПІБ мешканця!", "Помилка");
            return;
        }

        // Перевірка на унікальність коду мешканця
        string codeInhabitant = textBox1.Text;
        using (SqlConnection connection = new SqlConnection(Constants.connectionString))
        {
connection.Open();
            string query = "SELECT COUNT(*) FROM Inhabitant WHERE Code_inhabitant =
@codeInhabitant";
            using (SqlCommand command = new SqlCommand(query, connection))
            {
                command.Parameters.AddWithValue("@codeInhabitant", codeInhabitant);
                int count = (int)command.ExecuteScalar();
                if (count > 0)
                {
                    MessageBox.Show("Мешканець з таким ідентифікатором вже існує!",
"Помилка");
                    return;
                }
            }
        }

        // Заповнення значень для вставки в БД
        string[] nameParts = textBox2.Text.Split(' ');
        string lastName = nameParts.Length > 0 ? nameParts[0] : "";
        string firstName = nameParts.Length > 1 ? nameParts[1] : "";
        string surname = nameParts.Length > 2 ? nameParts[2] : "";

        string homeAddress = textBox3.Text;
        string phoneNumber = textBox4.Text;
        string gender = comboBox1.Text;
        string statusEmployment = "Не працює"; // значення за замовчуванням
        // Дата народження
        DateTime birthDate = dateTimePicker1.Value;

        // Вставка даних в БД
        using (SqlConnection connection = new SqlConnection(Constants.connectionString))
        {
            connection.Open();
```

```
            string insertQuery = "INSERT INTO Inhabitant (Code_inhabitant, Last_name,
First_name, Surname, Sex, Birthday, Home_address, Phone_number, Status_employment,
Code_village) " +
                                 "VALUES (@codeInhabitant, @lastName, @firstName, @surname,
@gender, @birthDate, @homeAddress, @phoneNumber, @statusEmployment, @codeVillage)";
            using (SqlCommand insertCommand = new SqlCommand(insertQuery, connection))
            {
                insertCommand.Parameters.AddWithValue("@codeInhabitant", codeInhabitant);
                insertCommand.Parameters.AddWithValue("@lastName", lastName);
                insertCommand.Parameters.AddWithValue("@firstName", firstName);
                insertCommand.Parameters.AddWithValue("@surname", surname);
                insertCommand.Parameters.AddWithValue("@gender", gender);
                insertCommand.Parameters.AddWithValue("@birthDate", birthDate);
                insertCommand.Parameters.AddWithValue("@homeAddress", homeAddress);
                insertCommand.Parameters.AddWithValue("@phoneNumber", phoneNumber);
                insertCommand.Parameters.AddWithValue("@statusEmployment",
statusEmployment);
                insertCommand.Parameters.AddWithValue("@codeVillage",
GetVillageCode(comboBox2.Text));

                insertCommand.ExecuteNonQuery();
                MessageBox.Show("Мешканець успішно доданий!", "Інформація");

                // Очистка полів
                textBox1.Text = "";
                textBox2.Text = "";
                textBox3.Text = "";
                textBox4.Text = "";
                comboBox1.Text = "";
                comboBox2.Text = "";
                dateTimePicker1.Value = DateTime.Now;
            }
        }
    }

    private bool CheckName(string inhabitant)
    {
        string[] words = inhabitant.Split(' ');
        return words.Length >= 3;
    }

    private void FillVillageComboBox()
    {
        using (SqlConnection connection = new SqlConnection(Constants.connectionString))
        {
            connection.Open();
            string query = "SELECT Village_name FROM Village";
            using (SqlCommand command = new SqlCommand(query, connection))
            {
```

```
                using (SqlDataReader reader = command.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        string villageName = reader["Village_name"].ToString();
                        comboBox2.Items.Add(villageName);
                    }
                }
            }
        }

        private string GetVillageCode(string villageName)
        {
            string code = "";
            using (SqlConnection connection = new SqlConnection(Constants.connectionString))
            {
 connection.Open();
                string query = $"SELECT Code_village FROM Village WHERE Village_name =
'{villageName}'";
                SqlCommand command = new SqlCommand(query, connection);
                var result = command.ExecuteScalar();
                if (result != null)
                {
                    code = result.ToString();
                }
            }
            return code;
        }

        private void AddInhabitant_VisibleChanged(object sender, EventArgs e)
        {
            if (this.Visible)
            {
                try
                {
                    comboBox2.Items.Clear();
                }
                catch { }

                FillVillageComboBox();
                dateTimePicker1.Value = DateTime.Now;
            }
        }
    }
}
```

Constants.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Village_Council
{
   public static class Constants
   {
 //public const string connectionString = "Data Source=localhost;Initial
Catalog=Village_Council;Integrated Security=True;";
       public const string connectionString =
"Server=localhost\\MsServer;Database=Village_Council;Integrated Security=True;";

   }
}
```

EditEnterprise.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Village_Council.Entities;
using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace Village_Council
{
   public partial class EditInhabitant : Form
   {

      public EditInhabitant()
      {
         InitializeComponent();
      }

      // Delete from database
      private void button1_Click(object sender, EventArgs e)
      {
         if (comboBox1.SelectedItem == null)
         {
```

```
            MessageBox.Show("Будь ласка, оберіть мешканця для видалення!", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
```



```
            return;
        }

        InhabitantItem selectedInhabitant = (InhabitantItem)comboBox1.SelectedItem;
        string codeInhabitant = selectedInhabitant.CodeInhabitant;

        using (SqlConnection connection = new SqlConnection(Constants.connectionString))
        {
            connection.Open();

            // Видаляємо пов’язані записи з таблиці Job
            string deleteJobQuery = "DELETE FROM Job WHERE Code_inhabitant =
@CodeInhabitant";
            using (SqlCommand deleteJobCommand = new SqlCommand(deleteJobQuery,
connection))
            {
                deleteJobCommand.Parameters.AddWithValue("@CodeInhabitant", codeInhabitant);
                deleteJobCommand.ExecuteNonQuery();
            }

            // Видаляємо пов’язані записи з таблиці Education
            string deleteEducationQuery = "DELETE FROM Education WHERE Code_inhabitant
= @CodeInhabitant";
            using (SqlCommand deleteEducationCommand = new
SqlCommand(deleteEducationQuery, connection))
            {
                deleteEducationCommand.Parameters.AddWithValue("@CodeInhabitant",
codeInhabitant);
 deleteEducationCommand.ExecuteNonQuery();
            }

            // Видаляємо мешканця з таблиці Inhabitant
            string deleteInhabitantQuery = "DELETE FROM Inhabitant WHERE Code_inhabitant
= @CodeInhabitant";
            using (SqlCommand deleteInhabitantCommand = new
SqlCommand(deleteInhabitantQuery, connection))
            {
                deleteInhabitantCommand.Parameters.AddWithValue("@CodeInhabitant",
codeInhabitant);
                int rowsAffected = deleteInhabitantCommand.ExecuteNonQuery();

                if (rowsAffected > 0)
                {
                    MessageBox.Show("Мешканця успішно видалено!", "Інформація",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                    comboBox1.Items.Clear();
                    FillComboBox();
```

```
                comboBox1.Text = "";
                // Очищаємо текстові поля
                textBox2.Text = "";
```



```
                textBox3.Text = "";
                textBox4.Text = "";
                textBox5.Text = "";
                textBox6.Text = "";
            }
            else
            {
                MessageBox.Show("Не вдалося видалити мешканця!", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
    }
}

private void FillComboBox()
{
    comboBox1.Items.Clear(); // Очищаємо перед заповненням
    using (SqlConnection connection = new SqlConnection(Constants.connectionString))
    {
        string query = "SELECT Code_inhabitant, Last_name, First_name, Surname FROM
Inhabitant";
        SqlCommand command = new SqlCommand(query, connection);
        connection.Open();
        SqlDataReader inhabitant = command.ExecuteReader();
        while (inhabitant.Read())
        {
            string codeInhabitant = inhabitant["Code_inhabitant"].ToString();
 string fullName = $"{inhabitant["Last_name"]} {inhabitant["First_name"]}
{inhabitant["Surname"]}";
            comboBox1.Items.Add(new InhabitantItem(codeInhabitant, fullName));
        }
        inhabitant.Close();
    }
}

private void button2_Click(object sender, EventArgs e)
{
    if (comboBox1.SelectedItem == null)
    {
        MessageBox.Show("Будь ласка, оберіть мешканця для редагування!", "Помилка");
    }
    else
    {
        string selectedInhabitant = comboBox1.SelectedItem.ToString();
        string[] selectedInhabitantParts = selectedInhabitant.Split(' ');
        string lastName = textBox2.Text;
```

```
            string firstName = textBox3.Text;
            string surname = textBox4.Text;
            string homeAddress = textBox5.Text;
            string phoneNumber = textBox6.Text;
```



```
            using (SqlConnection connection = new SqlConnection(Constants.connectionString))
            {
                connection.Open();
                string updateQuery = "UPDATE Inhabitant SET ";
                List<SqlParameter> parameters = new List<SqlParameter>();

                if (!string.IsNullOrWhiteSpace(lastName))
                {
                    updateQuery += "Last_name = @LastName, ";
                    parameters.Add(new SqlParameter("@LastName", lastName));
                }

                if (!string.IsNullOrWhiteSpace(firstName))
                {
                    updateQuery += "First_name = @FirstName, ";
                    parameters.Add(new SqlParameter("@FirstName", firstName));
                }

                if (!string.IsNullOrWhiteSpace(surname))
                {
                    updateQuery += "Surname = @Surname, ";
                    parameters.Add(new SqlParameter("@Surname", surname));
                }

                if (!string.IsNullOrWhiteSpace(homeAddress))
                {
updateQuery += "Home_address = @HomeAddress, ";
                    parameters.Add(new SqlParameter("@HomeAddress", homeAddress));
                }

                if (!string.IsNullOrWhiteSpace(phoneNumber))
                {
                    updateQuery += "Phone_number = @PhoneNumber, ";
                    parameters.Add(new SqlParameter("@PhoneNumber", phoneNumber));
                }
                updateQuery = updateQuery.TrimEnd(' ', ',');
                updateQuery += " WHERE Last_name = @SelectedLastName AND First_name =
@SelectedFirstName AND Surname = @SelectedSurname";
                parameters.Add(new SqlParameter("@SelectedLastName",
selectedInhabitantParts[0]));
                parameters.Add(new SqlParameter("@SelectedFirstName",
selectedInhabitantParts[1]));
                parameters.Add(new SqlParameter("@SelectedSurname",
selectedInhabitantParts[2]));
```

```
            SqlCommand command = new SqlCommand(updateQuery, connection);
            command.Parameters.AddRange(parameters.ToArray());
            int rowsAffected = command.ExecuteNonQuery();
            if (rowsAffected > 0)
            {
```



```
                MessageBox.Show("Інформацію про мешканця було успішно оновлено!",
"Інформація");
                textBox2.Text = "";
                textBox3.Text = "";
                textBox4.Text = "";
                textBox5.Text = "";
                textBox6.Text = "";

                comboBox1.Items.Clear();
                comboBox1.Text = "";

                FillComboBox();
            }
            else
            {
                MessageBox.Show("Не вдалося знайти вказаного мешканця для
редагування!", "Помилка");
            }
        }
    }
}

private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    if (comboBox1.SelectedItem == null)
    {
// Якщо нічого не вибрано, очищаємо поля
        textBox2.Text = "";
        textBox3.Text = "";
        textBox4.Text = "";
        textBox5.Text = "";
        textBox6.Text = "";
        return;
    }

    InhabitantItem selectedInhabitant = (InhabitantItem)comboBox1.SelectedItem;
    string codeInhabitant = selectedInhabitant.CodeInhabitant;

    using (SqlConnection connection = new SqlConnection(Constants.connectionString))
    {
        string query = "SELECT Last_name, First_name, Surname, Home_address,
Phone_number FROM Inhabitant WHERE Code_inhabitant = @CodeInhabitant";
        SqlCommand command = new SqlCommand(query, connection);
        command.Parameters.AddWithValue("@CodeInhabitant", codeInhabitant);
```

```
            connection.Open();
            SqlDataReader reader = command.ExecuteReader();
            if (reader.Read())
            {
               textBox2.Text = reader["Last_name"].ToString();
               textBox3.Text = reader["First_name"].ToString();
```



```
               textBox4.Text = reader["Surname"].ToString();
               textBox5.Text = reader["Home_address"].ToString();
               textBox6.Text = reader["Phone_number"].ToString();
            }
            reader.Close();
         }
      }

      private void EditInhabitant_VisibleChanged(object sender, EventArgs e)
      {
         if (this.Visible)
         {
            try
            {
               comboBox1.Items.Clear();
            }
            catch { }

            FillComboBox();
         }
      }
   }
}
```

Login.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Village_Council
{
   public partial class Login : Form
   {
      // village_head
      SqlConnection connection;
```

```
    Menu menu;

    public Login()
    {
       InitializeComponent();
```



```
       connection = new SqlConnection(Constants.connectionString);

       textBox1.Text = "village_head";
       textBox2.Text = "village_head";

       //textBox1.Text = "secretary";
       //textBox2.Text = "secretary";

    }

    private void Menu_FormClosed(object sender, FormClosedEventArgs e)
    {

this.Close();
    }

    private void Login_Load(object sender, EventArgs e)
    {
       //label1.Parent = pictureBox1;
       //label2.Parent = pictureBox1;
       //label3.Parent = pictureBox1;
    }

    private void button1_Click(object sender, EventArgs e)
    {
       this.Close();
    }

    private void button2_Click(object sender, EventArgs e)
    {
       string username = textBox1.Text;
       string password = textBox2.Text;



       using (SqlConnection connection = new SqlConnection(Constants.connectionString))
       {
          connection.Open();
          string query = "SELECT COUNT(*) FROM sys.database_principals WHERE name =
@username AND type_desc = 'SQL_USER' AND default_schema_name = @password";
          using (SqlCommand command = new SqlCommand(query, connection))
          {
```

```
                command.Parameters.AddWithValue("@username", username);
                command.Parameters.AddWithValue("@password", password);
                int count = (int)command.ExecuteScalar();
                if (count > 0)
                {
                    menu = new Menu(username);

                    menu.FormClosed += Menu_FormClosed;
```



```
                    menu.Show();

                    textBox1.Text = "";
                    textBox2.Text = "";

                    this.Hide();


                    //Menu form2 = new Menu(username);
                    //form2.Show();

                    //textBox1.Text = "";
//textBox2.Text = "";

                    //this.Enabled = false;

                    //form2.FormClosed += (s, args) => { this.Enabled = true; };
                }
                else
                {
                    MessageBox.Show("Ім'я користувача або пароль введено невірно!",
"Помилка");

                }
            }
        }


        //menu.Show();

        //textBox1.Text = "";
        //textBox2.Text = "";

        //this.Hide();

        //this.Enabled = false;

        //form2.FormClosed += (s, args) => { this.Enabled = true; };
    }

    private void Login_FormClosed(object sender, FormClosedEventArgs e)
```

```
        {
            if (connection != null && connection.State == ConnectionState.Open)
            {
                connection.Close();
            }
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {
```



```

        }

        private void label1_Click(object sender, EventArgs e)
        {

        }

        private void label2_Click(object sender, EventArgs e)
        {

        }

        private void label3_Click(object sender, EventArgs e)

{

        }
    }
}
```

Menu.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Village_Council
{
    public partial class Menu : Form
    {
        private string loggedInUser;

        ShowVillage showVillage;
        ShowJob showJob;
        ShowInhabitant showInhabitant;
        ShowEnterprise showEnterprise;
```

```
ShowEducation showEducation;
EditEducation editEducation;
EditEnterprise editEnterprise;
EditJob editJob;
EditInhabitant editInhabitant;
AddJob addJob;
AddInhabitant addInhabitant;
AddEnterprise addEnterprise;
AddEducation addEducation;
```



```
Panel_Navigator pnlNav_class;

public Menu(string loggedInUser)
{
   InitializeComponent();
   this.loggedInUser = loggedInUser;
   //this.loggedInUser = "Admin";
}


private void button1_Click(object sender, EventArgs e)

{
   this.Close();
}

private void toolStripMenuItem12_Click(object sender, EventArgs e)
{
   pnlNav_class.pnl_logic((Button)sender);

   this.pnlLoadForm.Controls.Clear();

   showVillage.FormBorderStyle = FormBorderStyle.None;
   this.pnlLoadForm.Controls.Add(showVillage);
   showVillage.Show();
}

private void toolStripMenuItem13_Click(object sender, EventArgs e)
{
   pnlNav_class.pnl_logic((Button)sender);

   this.pnlLoadForm.Controls.Clear();

   addEnterprise.FormBorderStyle = FormBorderStyle.None;
   this.pnlLoadForm.Controls.Add(addEnterprise);
   addEnterprise.Show();
}

private void toolStripMenuItem14_Click(object sender, EventArgs e)
```

```
        {
            pnlNav_class.pnl_logic((Button)sender);

            this.pnlLoadForm.Controls.Clear();

            showEnterprise.FormBorderStyle = FormBorderStyle.None;
            this.pnlLoadForm.Controls.Add(showEnterprise);
            showEnterprise.Show();
        }

        private void toolStripMenuItem5_Click(object sender, EventArgs e)
```



```
        {
            pnlNav_class.pnl_logic((Button)sender);

            this.pnlLoadForm.Controls.Clear();

            addInhabitant.FormBorderStyle = FormBorderStyle.None;
            this.pnlLoadForm.Controls.Add(addInhabitant);
            addInhabitant.Show();
        }

        private void toolStripMenuItem6_Click(object sender, EventArgs e)
        {


pnlNav_class.pnl_logic((Button)sender);

            this.pnlLoadForm.Controls.Clear();
            editInhabitant.FormBorderStyle = FormBorderStyle.None;
            this.pnlLoadForm.Controls.Add(editInhabitant);
            editInhabitant.Show();
        }

        private void toolStripMenuItem7_Click(object sender, EventArgs e)
        {
            pnlNav_class.pnl_logic((Button)sender);

            this.pnlLoadForm.Controls.Clear();

            showInhabitant.FormBorderStyle = FormBorderStyle.None;
            this.pnlLoadForm.Controls.Add(showInhabitant);
            showInhabitant.Show();
        }

        private void toolStripMenuItem8_Click(object sender, EventArgs e)
        {
            pnlNav_class.pnl_logic((Button)sender);

            this.pnlLoadForm.Controls.Clear();
```

```
            addJob.FormBorderStyle = FormBorderStyle.None;
            this.pnlLoadForm.Controls.Add(addJob);
            addJob.Show();
        }

        private void toolStripMenuItem9_Click(object sender, EventArgs e)
        {
            pnlNav_class.pnl_logic((Button)sender);

            this.pnlLoadForm.Controls.Clear();

            editJob.FormBorderStyle = FormBorderStyle.None;
```



```
            this.pnlLoadForm.Controls.Add(editJob);
            editJob.Show();
        }

        private void toolStripMenuItem10_Click(object sender, EventArgs e)
        {
            pnlNav_class.pnl_logic((Button)sender);

            this.pnlLoadForm.Controls.Clear();

            showJob.FormBorderStyle = FormBorderStyle.None;
            this.pnlLoadForm.Controls.Add(showJob);

showJob.Show();
        }
        private void toolStripMenuItem11_Click(object sender, EventArgs e)
        {
            pnlNav_class.pnl_logic((Button)sender);

            this.pnlLoadForm.Controls.Clear();

            addEducation.FormBorderStyle = FormBorderStyle.None;
            this.pnlLoadForm.Controls.Add(addEducation);
            addEducation.Show();
        }

        private void toolStripMenuItem15_Click(object sender, EventArgs e)
        {
            pnlNav_class.pnl_logic((Button)sender);

            this.pnlLoadForm.Controls.Clear();

            showEducation.FormBorderStyle = FormBorderStyle.None;
            this.pnlLoadForm.Controls.Add(showEducation);
            showEducation.Show();
        }
```

```
    private void button13_Click(object sender, EventArgs e)
    {
        pnlNav_class.pnl_logic((Button)sender);

        this.pnlLoadForm.Controls.Clear();

        editEducation.FormBorderStyle = FormBorderStyle.None;
        this.pnlLoadForm.Controls.Add(editEducation);
        editEducation.Show();
    }

    private void button14_Click(object sender, EventArgs e)
    {
```



```
        pnlNav_class.pnl_logic((Button)sender);

        this.pnlLoadForm.Controls.Clear();

        editEnterprise.FormBorderStyle = FormBorderStyle.None;
        this.pnlLoadForm.Controls.Add(editEnterprise);
        editEnterprise.Show();
    }


    private void Menu_Load(object sender, EventArgs e)
    {

showVillage = new ShowVillage() { Dock = DockStyle.Fill, TopLevel = false, TopMost = true };
        showJob = new ShowJob() { Dock = DockStyle.Fill, TopLevel = false, TopMost = true };
        showInhabitant = new ShowInhabitant() { Dock = DockStyle.Fill, TopLevel = false,
TopMost = true };
        showEnterprise = new ShowEnterprise() { Dock = DockStyle.Fill, TopLevel = false,
TopMost = true };
        showEducation = new ShowEducation() { Dock = DockStyle.Fill, TopLevel = false,
TopMost = true };
        editJob = new EditJob() { Dock = DockStyle.Fill, TopLevel = false, TopMost = true };
        editInhabitant = new EditInhabitant() { Dock = DockStyle.Fill, TopLevel = false,
TopMost = true };
        addJob = new AddJob() { Dock = DockStyle.Fill, TopLevel = false, TopMost = true };
        addInhabitant = new AddInhabitant() { Dock = DockStyle.Fill, TopLevel = false,
TopMost = true };
        addEnterprise = new AddEnterprise() { Dock = DockStyle.Fill, TopLevel = false,
TopMost = true };
        addEducation = new AddEducation() { Dock = DockStyle.Fill, TopLevel = false,
TopMost = true };
        editEducation = new EditEducation() { Dock = DockStyle.Fill, TopLevel = false,
TopMost = true };
        editEnterprise = new EditEnterprise() { Dock = DockStyle.Fill, TopLevel = false,
TopMost = true };
```

```
        pnlNav_class = new Panel_Navigator(pnlNav, ref button2, button2.BackColor,
Color.DarkOliveGreen);
        toolStripMenuItem12_Click(button2, EventArgs.Empty);


        if (loggedInUser == "village_head")
        {
           button6.Enabled = false;
           button5.Enabled = false;
           button7.Enabled = true;
           button9.Enabled = false;
           button8.Enabled = false;
           button10.Enabled = true;
           button12.Enabled = false;
```



```
           button2.Enabled = true;
           button3.Enabled = true;
           button4.Enabled = true;
           button11.Enabled = true;
           button14.Enabled = true;
           button13.Enabled = false;
        }
        else if (loggedInUser == "secretary")
        {
           button6.Enabled = true;
           button5.Enabled = true;
           button7.Enabled = true;
           button9.Enabled = true;
           button8.Enabled = true;
           button10.Enabled = true;
           button12.Enabled = true;
           button2.Enabled = true;
           button3.Enabled = false;
           button4.Enabled = true;
           button11.Enabled = true;
           button13.Enabled = true;
           button14.Enabled = false;
        }
        else
        {
           tableLayoutPanel3.Enabled = false;
           button6.Enabled = false;
           button5.Enabled = false;
           button7.Enabled = false;
           button9.Enabled = false;
           button8.Enabled = false;
           button10.Enabled = false;
           button12.Enabled = false;
           button2.Enabled = false;
           button3.Enabled = false;
```

```
            button4.Enabled = false;
            button11.Enabled = false;
            button14.Enabled = false;
            button13.Enabled = false;
        }
    }

    private void pnlLoadForm_Paint(object sender, PaintEventArgs e)
    {

    }

  }
}
```


Panel_Navigator.cs:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Village_Council
{
  public class Panel_Navigator
{
    private Panel pnlNav;

    private Button prevBtn;

    Color StartColor;
    Color EndColor;

    public Panel_Navigator(Panel pnlNav, ref Button defoultBtn, Color StartColor, Color
EndColor)
    {
        this.pnlNav = pnlNav;
        this.prevBtn = defoultBtn;

        this.StartColor = StartColor;
        this.EndColor = EndColor;
    }

    public void pnl_logic(Button btn)
    {
        if (prevBtn != null)
        {
```

```
            prevBtn.BackColor = StartColor;//Color.FromArgb(56, 26, 26);
        }
        //pnlNav.Height = btn.Height;
        //pnlNav.Top = btn.Top;
        //pnlNav.Left = btn.Left;

        Point screenPoint = btn.PointToScreen(Point.Empty);
        Point formPoint = pnlNav.Parent.PointToClient(screenPoint);

        pnlNav.Height = btn.Height;
        pnlNav.Top = formPoint.Y;
        pnlNav.Left = formPoint.X;

        btn.BackColor = EndColor;//Color.FromArgb(98, 48, 48);
        prevBtn = btn;
    }
```

Сторінка 20

```
  }
}
```

Program.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Village_Council
{
internal static class Program
  {
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main()
    {
      Application.EnableVisualStyles();
      Application.SetCompatibleTextRenderingDefault(false);
      Application.Run(new Login());
    }
  }
}
```

ReportGen.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
using iTextSharp.text;
using iTextSharp.text.pdf;
using System.IO;
using System.Windows.Forms;

namespace Village_Council
{
   public class ReportGen
   {
      readonly string reportTitle;
      readonly DataGridView dataGrid;

      public ReportGen(string reportTitle, DataGridView dataGrid) {

         this.reportTitle = reportTitle;
         this.dataGrid = dataGrid;
```



```
      }

      public void GenerateTextReport(int size)
      {
         StringBuilder report = new StringBuilder();
         report.AppendLine(reportTitle);
         report.AppendLine($"Дата створення: {DateTime.Now:dd.MM.yyyy}");
         report.AppendLine();

         for (int i = 0; i < dataGrid.Columns.Count; i++)
         {
            report.Append(dataGrid.Columns[i].HeaderText.PadRight(size));

}
         report.AppendLine();
         report.AppendLine(new string('-', dataGrid.Columns.Count * size));

         foreach (DataGridViewRow row in dataGrid.Rows)
         {
            if (row.IsNewRow) continue;
            for (int i = 0; i < dataGrid.Columns.Count; i++)
            {
               string cellValue = row.Cells[i].Value?.ToString() ?? "";
               report.Append(cellValue.PadRight(size));
            }
            report.AppendLine();
         }

         SaveFileDialog saveFileDialog = new SaveFileDialog();
         saveFileDialog.Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
         saveFileDialog.FileName = $"{reportTitle.Replace(" ",
"_")}_{DateTime.Now:yyyyMMdd_HHmmss}.txt";
```

```
        saveFileDialog.Title = "Оберіть місце для збереження звіту";

        if (saveFileDialog.ShowDialog() == DialogResult.OK)
        {
            try
            {
                File.WriteAllText(saveFileDialog.FileName, report.ToString());
                MessageBox.Show($"Звіт збережено у файл: {saveFileDialog.FileName}",
"Інформація", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Помилка при збереженні звіту: {ex.Message}", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
    }
```



```
    public void GeneratePdfReport()
    {
        SaveFileDialog saveFileDialog = new SaveFileDialog();
        saveFileDialog.Filter = "PDF files (*.pdf)|*.pdf|All files (*.*)|*.*";
        saveFileDialog.FileName = $"{reportTitle.Replace(" ",
"_")}_{DateTime.Now:yyyyMMdd_HHmmss}.pdf";
        saveFileDialog.Title = "Оберіть місце для збереження звіту";

        if (saveFileDialog.ShowDialog() == DialogResult.OK)
        {
            try
            {
                Document document = new Document(PageSize.A4);
PdfWriter.GetInstance(document, new FileStream(saveFileDialog.FileName, FileMode.Create));
                document.Open();

                BaseFont baseFont = BaseFont.CreateFont("C:\\Windows\\Fonts\\arial.ttf",
BaseFont.IDENTITY_H, BaseFont.EMBEDDED);
                Font titleFont = new Font(baseFont, 16, Font.BOLD);
                Font dateFont = new Font(baseFont, 12, Font.NORMAL);
                Font headerFont = new Font(baseFont, 10, Font.BOLD);
                Font cellFont = new Font(baseFont, 10, Font.NORMAL);

                Paragraph title = new Paragraph(reportTitle, titleFont);
                title.Alignment = Element.ALIGN_CENTER;
                document.Add(title);

                Paragraph date = new Paragraph($"Дата створення:
{DateTime.Now:dd.MM.yyyy}", dateFont);
                date.Alignment = Element.ALIGN_CENTER;
                document.Add(date);
```

```
                document.Add(new Paragraph("\n"));

                PdfPTable table = new PdfPTable(dataGrid.Columns.Count);
                table.WidthPercentage = 100;

                for (int i = 0; i < dataGrid.Columns.Count; i++)
                {
                    PdfPCell cell = new PdfPCell(new Phrase(dataGrid.Columns[i].HeaderText,
headerFont));
                    cell.BackgroundColor = BaseColor.LIGHT_GRAY;
                    table.AddCell(cell);
                }

                foreach (DataGridViewRow row in dataGrid.Rows)
                {
                    if (row.IsNewRow) continue;
                    for (int i = 0; i < dataGrid.Columns.Count; i++)
                    {
                        string cellValue = row.Cells[i].Value?.ToString() ?? "";
```



```
                        PdfPCell cell = new PdfPCell(new Phrase(cellValue, cellFont));
                        table.AddCell(cell);
                    }
                }

                document.Add(table);
                document.Close();

                MessageBox.Show($"Звіт збережено у файл: {saveFileDialog.FileName}",
"Інформація", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            catch (Exception ex)
            {
MessageBox.Show($"Помилка при збереженні звіту: {ex.Message}", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
    }

  }
}
```