Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп’ютерних наук

Дипломна робота
магістра

з теми: «**РОЗРОБКА МЕТОДУ КОДУВАННЯ ІНФОРМАЦІЇ ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ФУНКЦІОНУВАННЯ ІНФОКОМУНІКАЦІЙНИХ СИСТЕМ**»

Виконав: студент 1 курсу,
групи KN1-M22
спеціальності 122 Комп’ютерні науки
Лупул Віталій Олегович

Кам’янець-Подільський – 2023

ЗМІСТ

ВСТУП 4

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ 7

1.1. Поняття інфокомунікаційної системи 7

1.2. Опис проблеми підвищення ефективності інфокомунікаційних систем 8

1.3. Аналіз існуючих методів кодування інформації 10

1.4. Аналіз існуючих програмних рішень 16

1.5. Обґрунтування необхідності розробки методу кодування інформації 21

1.6. Постановка задачі 22

1.7. Висновок до розділу 25

РОЗДІЛ 2. ВИБІР ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ ТА ОСОБЛИВОСТІ АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 27

2.1. Обґрунтування вибору мови програмування 27

2.2. Обґрунтування вибору середовища розробки 29

2.3. Обґрунтування вибору СКБД 31

2.4. Аналіз вимог. Use-case діаграми. Основні прецеденти 34

2.5. Архітектура проекту 38

2.5.1. Особливості розробки бази даних. ERD діаграма з описанням сутностей 38

2.5.2. Особливості розробки рівня BLL 40

2.5.3. Особливості розробки рівня UI 43

2.5.4. Особливості розробки DAL 46

2.6. Висновок до розділу 48

РОЗДІЛ 3. РОЗРОБКА, ВИКОРИСТАННЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 49

3.1. Розробка методу кодування інформації 49
3.2. Розробка програмних модулів системи 53
3.3. Результати функціонального тестування розробленого додатку 61
3.4. Інструкція користувачеві програми 64
3.4.1. Мінімальні вимоги для запуску ПЗ 64
3.4.2. Опис процедури розгортання програмного продукту 65
3.4.3. Використання програмного продукту 65
3.5. Висновок до розділу 72
ВИСНОВКИ 73
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 75
Додаток А. Лістинги програми 78

ВСТУП

З розвитком сучасних технологій та поширенням мережі Інтернет значення інфокомунікаційних систем для різних сфер діяльності стає все більш важливим. Інформаційні технології є неодмінною складовою сучасного світу, що вимагає постійного розвитку та удосконалення. Однак збільшення обсягу інформації та використання її в різних вимірах, виникає необхідність підвищення ефективності функціонування інфокомунікаційних систем.

Важливим фактором, який відіграє значну роль у функціонуванні інфокомунікаційних систем, є безпека інформації, яка передається через мережу Інтернет. Інформація може бути скомпрометована, викрадена або пошкоджена під час передачі або зберігання, що може призвести до великих фінансових втрат або порушення конфіденційності.

Одним з можливих шляхів забезпечення безпеки інформації є застосування методів кодування. Кодування дозволяє зберігати та передавати інформацію в зашифрованому вигляді, що унеможливлює доступ до неї з боку несанкціонованих осіб. Однак, існуючі методи кодування мають свої обмеження та недоліки, які можуть обмежувати їх ефективність.

У зв'язку з цим, виникає необхідність розроблення нових методів кодування, які б дозволили підвищити ефективність функціонування інфокомунікаційних систем та забезпечити їх безпеку. Розробка нового методу кодування інформації може вирішити проблему підвищення ефективності функціонування інфокомунікаційних систем та забезпечення безпеки інформації.

Актуальність теми. Зараз життя суспільства неможливо уявити без використання різних інформаційних технологій, які дозволяють збирати, обробляти та передавати великі обсяги інформації. Однак, разом зі зростанням обсягів інформації, зростають також і вимоги до швидкості та якості її передачі, збереження та обробки.

Сьогодні ми стикаємося з різноманітними проблемами, пов'язаними з передачею та збереженням інформації, такими як втрата даних, помилки при передачі, низька швидкість передачі та обробки інформації, а також проблеми з безпекою та конфіденційністю даних.

Розробка нових методів кодування інформації є важливим напрямом досліджень, що дозволить вирішити ці проблеми та забезпечити більш ефективну передачу, збереження та обробку інформації. Розроблення нових методів кодування інформації забезпечує збільшення швидкості передачі даних та зменшення кількості помилок при передачі. Крім того, нові методи кодування можуть забезпечити більш високий рівень захисту даних, що є особливо важливим в галузі медицини, фінансів та інших галузях, де обробка конфіденційної інформації є ключовою.

Об’єктом дослідження є інформаційні системи та технології, що використовуються в інфокомунікаційних мережах. Ці системи та технології включають в себе комп’ютерні програми, апаратне забезпечення, протоколи передачі даних, мережеві технології та інші компоненти, які використовуються для обміну інформацією в мережі Інтернет.

Предметом дослідження є розробка методу кодування інформації, який дозволить підвищити ефективність функціонування інфокомунікаційних систем та забезпечити безпеку інформації користувачів.

Метою дослідження є розроблення нового методу кодування, який забезпечить ефективну передачу та збереження інформації в базі даних. Основні завдання, які стоять перед розробкою нового методу кодування, полягають у наступному:

– аналіз існуючих методів кодування інформації та визначення їх переваг та недоліків;

– аналіз існуючих програмних рішень;

– розроблення методу кодування, який дозволить підвищити ефективність передачі та збереження інформації в базі даних;

– реалізація розробленого методу та проведення експериментальних досліджень для оцінки його ефективності;

– визначення можливості використання розробленого методу в реальних умовах.

Методи дослідження:

– опитування - збирання даних за допомогою відповідей на питання, які відповідають поставленій меті дослідження;

– аналіз існуючих даних - використання існуючих даних, таких як документи, звіти та статистика, щоб визначити ефективність методу кодування інформації;

– практичне використання - використання методу кодування інформації на реальних даних та оцінка його ефективності.

Практичне значення одержаних результатів полягає в можливості використання розробленого методу кодування інформації для підвищення ефективності функціонування інфокомунікаційних систем. Застосування нового методу дозволить підвищити швидкість передачі інформації, зменшити кількість помилок у процесі передачі та забезпечити більш ефективне використання інфраструктури мережі.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Поняття інфокомунікаційної системи

Інфокомунікаційна система (ІКС) - це технічний комплекс засобів, що забезпечує збір, обробку, передачу, зберігання та відтворення інформації в режимі реального часу або з відкладеним часом. ІКС складається з комп'ютерів, серверів, мережевого обладнання, програмного забезпечення, технічних засобів зберігання даних та інших елементів, що забезпечують функціонування системи.

Однією з ключових задач ІКС є передача інформації з одного пристрою на інший. Під час передачі інформації зазвичай виникає потреба в її кодуванні, тобто перетворенні вихідних даних в послідовність символів, яку можна передати по комунікаційному каналу. Ефективне кодування інформації є важливою задачею для забезпечення якісної передачі даних та підвищення продуктивності ІКС.

Одним із методів ефективного кодування інформації є використання алгоритму Хаффмена, який був запропонований у 1952 році американським математиком Клаудом Хаффменом (Huffman, 1952). Цей метод полягає у присвоєнні унікального коду кожному символу або групі символів в залежності від його частоти в тексті. Згідно з цим методом, символи, які зустрічаються частіше, отримують коротші коди, а символи, які зустрічаються рідше, отримують довші коди. Таким чином, зменшується кількість бітів, необхідних для кодування повідомлення, що дозволяє знизити обсяг передаваної інформації та підвищити швидкість передачі даних.

Іншим методом кодування інформації є використання алгоритму Лемпеля-Зівя (LZ), який був запропонований А.Лемпелем та Я.Зівем у 1977 році (Lempel and Ziv, 1977). Цей метод полягає у створенні словника, в якому зберігаються повторювані фрагменти даних. Під час передачі повідомлення замість повторення фрагментів, що вже знаходяться в словнику, відбувається передача посилань на них. Це дозволяє зменшити обсяг передаваної інформації та зменшити частоту передачі даних.

Ще одним важливим методом кодування є метод арифметичного кодування, який був запропонований Д. Рівестом, А.Шаміром та А.Адлеманом у 1976 році. За цим методом, символи кодуються не окремо, а як ціле повідомлення з використанням арифметичних операцій. Це дозволяє отримати більшу кількість інформації, яка передається, зберегти точність передачі та підвищити швидкість передачі даних.

Окрім того, існує багато інших методів кодування інформації, таких як адаптивне кодування, гібридне кодування тощо, які можуть бути використані в залежності від специфіки передаваної інформації та вимог до ефективності передачі.

Розробка методу кодування інформації є складним та багатоаспектним процесом, який потребує використання різноманітних методів та технологій. Основні завдання розробки методу кодування полягають у підвищенні ефективності передачі та зберігання інформації, зменшенні втрат даних, підвищенні швидкості передачі даних та забезпеченні захисту інформації від несанкціонованого доступу.

Отже, розробка ефективного методу кодування інформації є важливою задачею для підвищення ефективності функціонування інфокомунікаційних систем. Для досягнення цієї мети можуть бути використані різноманітні методи кодування, такі як метод Хаффмена, метод LZ, метод арифметичного кодування, метод змішаного кодування та інші. Вибір оптимального методу кодування повинен здійснюватись на основі аналізу конкретної задачі та вимог до ефективності передачі даних.

1.2. Опис проблеми підвищення ефективності інфокомунікаційних систем

Проблема підвищення ефективності інфокомунікаційних систем є важливою складовою розробки методу кодування інформації. Інфокомунікаційні

системи є важливою складовою різноманітних сфер, таких як телекомунікації, медіа, інформаційні технології, наука та техніка, тощо. Ці системи забезпечують передачу, обробку та зберігання інформації, що є важливим фактором для бізнесу та громадського сектору.

Однією з основних проблем підвищення ефективності ІКС є необхідність забезпечення високої швидкості передачі даних та точності їх передачі. Швидкість передачі даних є важливою характеристикою ІКС, оскільки вона визначає, як швидко інформація може бути передана та оброблена. З іншого боку, точність передачі даних забезпечує збереження цілісності інформації та уникнення помилок при передачі.

Іншою проблемою є обмеженість ресурсів, таких як пропускна здатність та пам'ять, які використовуються в ІКС. Обмеженість цих ресурсів може призвести до зниження ефективності системи та зменшення швидкості передачі даних.

Розробка методу кодування інформації може допомогти підвищити ефективність функціонування ІКС шляхом зменшення кількості даних, що передаються та підвищення швидкості передачі. Крім того, методи кодування інформації можуть бути використані для підвищення точності передачі даних та забезпечення захисту інформації від несанкціонованого доступу. Одним з найбільш важливих завдань розробки методу кодування є підвищення стійкості до шуму та інших ефектів, які можуть виникати під час передачі даних.

Окрім розробки методів кодування, існує також необхідність у вдосконаленні апаратного та програмного забезпечення ІКС. Нові технології та алгоритми, такі як інтернет речей, штучний інтелект, машинне навчання та інші, можуть бути використані для підвищення ефективності та функціональності ІКС.

Отже, підвищення ефективності ІКС є важливою задачею, яка потребує розробки нових методів та технологій. Розробка методу кодування інформації є одним з основних напрямів розвитку ІКС та дозволяє підвищити швидкість

передачі даних, зменшити кількість переданих даних та забезпечити точність передачі даних.

1.3. Аналіз існуючих методів кодування інформації

З розвитком сучасних інформаційних технологій та зростанням обсягу інформації, яка обмінюється між різними пристроями, з'явилася потреба в ефективних методах кодування інформації для забезпечення якісного та швидкого функціонування інфокомунікаційних систем.

У світі, де кількість обмінюваних даних зростає з експоненційною швидкістю, потрібні методи кодування інформації, які б забезпечували не тільки ефективність, але і безпеку обміну даними. Таким чином, розробка нових методів кодування є важливим завданням в галузі інформаційних технологій.

В даному розділі буде проаналізовано існуючі методи кодування інформації, що включають:

– цифрове кодування - цей метод використовує бінарний код (0 та 1) для зберігання та передачі інформації;

– аналогове кодування - цей метод використовує аналогові сигнали, такі як звук або відео, для зберігання та передачі інформації;

– методи компресії даних - ці методи дозволяють зменшити розмір даних, що зберігаються або передаються, зберігаючи високу якість інформації;

– текстове кодування - цей метод використовується для зберігання та передачі текстової інформації, і може включати кодування UTF-8 або ASCII.

Цифрове кодування

Цифрове кодування - це метод кодування інформації, який використовує бінарний код (0 та 1) для зберігання та передачі даних. Цей метод широко використовується в інфокомунікаційних мережах, таких як Інтернет, телефонні мережі, комп'ютерні мережі тощо.

Основною функцією цифрового кодування є конвертація аналогових сигналів (таких як звук або відео) в цифровий формат, що забезпечує більшу точність та стійкість до шумів і спотворень під час передачі та зберігання інформації. Одна з головних переваг цифрового кодування полягає в тому, що воно дозволяє передавати інформацію на великі відстані з високою якістю та стійкістю до шумів та спотворень.

Переваги цифрового кодування включають:

– висока якість передачі - цифрове кодування забезпечує більшу точність та стійкість до шумів та спотворень, що дозволяє передавати інформацію на великі відстані з високою якістю;

– ефективність - цифрове кодування дозволяє зберігати та передавати більше інформації на одиницю часу та пропускної здатності мережі, порівняно з аналоговим кодуванням;

– можливість компресії - цифрове кодування може бути оптимізоване для компресії даних, що дозволяє зменшити розмір даних, що зберігаються або передаються, зберігаючи високу якість інформації;

– корекція помилок - цифрове кодування може бути додатково захищене даними для виявлення та корекції помилок, що можуть виникнути під час передачі даних.

Недоліки цифрового кодування включають:

– потреба у додаткових ресурсах - цифрове кодування вимагає більш високих обчислювальних та зберігаючих ресурсів для зберігання та обробки даних;

– потреба у високій швидкості передачі даних - цифрове кодування потребує високої швидкості передачі даних для забезпечення високої якості передачі;

– залежність від схеми кодування - ефективність цифрового кодування залежить від обраної схеми кодування, тому необхідно добре вибрати схему кодування для кожного типу даних;

– складність - цифрове кодування може бути складним процесом для виконання та розуміння.

Цифрове кодування є ефективним методом передачі та зберігання інформації у інфокомунікаційних мережах, забезпечуючи високу якість передачі та стійкість до шумів та спотворень. Використання цифрового кодування дозволяє зберігати та передавати більше інформації на одиницю часу та пропускної здатності мережі, порівняно з аналоговим кодуванням. Однак, цифрове кодування вимагає більш високих обчислювальних та зберігаючих ресурсів, потребує високої швидкості передачі даних, і залежить від обраної схеми кодування, що може створювати складнощі в процесі розробки та використання.

Аналогове кодування

Аналогове кодування - це метод кодування інформації, який використовує аналогові сигнали, такі як звук або відео, для зберігання та передачі інформації. Цей метод широко використовується в аналогових технологіях, таких як радіо, телебачення, телефонні мережі тощо.

Переваги аналогового кодування включають:

– простота - аналогове кодування дозволяє зберігати та передавати аналогові сигнали без додаткової обробки даних;

– невисока потреба в обчислювальних ресурсах - для аналогового кодування не потрібні високі обчислювальні ресурси, що дозволяє використовувати його на простих пристроях;

– ідеальне представлення - аналогове кодування може дозволити більш точне та природне представлення інформації, так як сигнал не обмежується конкретними значеннями, як у випадку цифрового кодування.

Недоліки аналогового кодування включають:

– стійкість до шуму та спотворень - аналогові сигнали більш чутливі до шумів та спотворень, що може призвести до погіршення якості передачі та збереження інформації;

– обмежена пропускна здатність. Мають обмежену пропускну здатність, що обмежує кількість інформації, яку можна передати за певний період часу;

– вразливість до зовнішніх впливів. Можуть бути вразливі до зовнішніх впливів, таких як електромагнітні перешкоди, що може призвести до помилок при передачі та збереженні інформації;

– передача на великі відстані. Не можуть бути передані на великі відстані без використання додаткових пристроїв, таких як підсилювачі сигналу, що збільшує вартість та складність системи.

Аналогове кодування є простим та природнім методом передачі та збереження інформації у інфокомунікаційних мережах. Однак, аналогове кодування не є таким точним та стійким до шумів та спотворень, як цифрове кодування, тому використовується переважно у традиційних аналогових системах, таких як радіо, телебачення тощо. Недоліки аналогового кодування включають обмежену пропускну здатність, вразливість до зовнішніх впливів та нестійкість до шумів та спотворень, що може призвести до помилок при передачі та збереженні інформації.

Методи компресії даних

Методи компресії даних - це техніки, які дозволяють зменшити розмір файлів за допомогою зниження відносної кількості інформації, яку містять ці файли. Компресія даних є важливим елементом збереження та передачі інформації в інфокомунікаційних мережах, так як дозволяє зберігати та передавати більше інформації на меншій кількості місця.

Переваги методів компресії даних включають:

– зменшення розміру файлів - компресія даних дозволяє зменшити розмір файлів та ефективніше зберігати та передавати інформацію;

– зменшення часу передачі даних - менший розмір файлів зменшує час передачі даних та використання пропускної здатності мережі;

– підвищення ефективності - компресія даних дозволяє зберігати та передавати більше інформації на меншій кількості місця, що збільшує ефективність використання обчислювальних ресурсів;

– покращення якості передачі даних - в деяких випадках, застосування методів компресії даних може покращити якість передачі даних.

Недоліки методів компресії даних включають:

– потреба в високих обчислювальних ресурсах - деякі методи компресії даних можуть вимагати більш високих обчислювальних ресурсів для компресії та декомпресії даних;

– втрата якості - деякі методи компресії даних можуть призводити до втрати якості вхідних даних;

– не всі дані можуть бути скомпресовані - деякі дані можуть бути вкрай складні для компресії, тому неможливо зменшити їх розмір;

– ризик втрати даних - якщо скомпресовані дані пошкоджені або втрачені, то може втратитися значна частина вхідної інформації.

Методи компресії даних є важливим елементом збереження та передачі інформації в інфокомунікаційних мережах. Вони дозволяють зменшити розмір файлів, зберегти та передавати більше інформації на меншій кількості місця, зменшити час передачі даних та використання пропускної здатності мережі. Однак, методи компресії даних можуть вимагати більш високих обчислювальних ресурсів для компресії та декомпресії даних, а також можуть впливати на якість відтворення даних, що необхідно враховувати при їх застосуванні.

Методи текстового кодування - це техніки, які дозволяють перетворити текстові дані на кодовану форму з метою передачі чи збереження інформації. Основною метою текстового кодування є зменшення обсягу текстових даних та підвищення ефективності передачі та збереження інформації.

Переваги методів текстового кодування включають:

– зменшення розміру файлів. Дозволяють зменшити розмір файлів та ефективніше зберігати та передавати інформацію;

– підвищення ефективності. Дозволяє зберігати та передавати більше інформації на меншій кількості місця, що збільшує ефективність використання обчислювальних ресурсів;

– захист інформації. Може допомогти захистити інформацію від несанкціонованого доступу;

– конвертація. Дозволяють перетворювати дані з одного формату в інший, що дозволяє зберігати та передавати інформацію у більш зручних форматах.

Недоліки методів текстового кодування включають:

– обмежена пропускна здатність - в деяких випадках, застосування методів текстового кодування може вимагати більш високої пропускної здатності мережі;

– втрата якості - деякі методи текстового кодування можуть призводити до втрати якості вхідних даних;

– обмеження в застосуванні - деякі методи текстового кодування можуть бути обмежені у застосуванні для певних типів даних;

– потреба в високих обчислювальних ресурсах - деякі методи текстового кодування можуть вимагати більш високих обчислювальних ресурсів для кодування та декодування даних.

Методи текстового кодування є важливим елементом збереження та передачі інформації в інфокомунікаційних мережах, які дозволяють зменшити розмір файлів, ефективніше зберігати та передавати інформацію, підвищувати ефективність та захищати інформацію. Однак, методи текстового кодування мають свої недоліки, які можуть обмежувати їх застосування, зокрема обмежену пропускну здатність, втрату якості та потребу в високих обчислювальних

ресурсах. При виборі методів текстового кодування необхідно враховувати їх переваги та недоліки, а також відповідність конкретному типу даних.

1.4. Аналіз існуючих програмних рішень

Задача кодування інформації в інфокомунікаційних мережах є важливою і існує багато програмних рішень для вирішення цієї задачі. Серед них можна виділити: Pretty Good Privacy, Base64 Encoder/Decoder, OpenSSL.

Pretty Good Privacy

Pretty Good Privacy (PGP) - це програмне забезпечення для кодування та захисту даних, яке використовує асиметричне шифрування та цифрові підписи для захисту даних під час передачі.

Основні функції PGP включають:

– кодування даних - PGP використовує симетричне та асиметричне шифрування для захисту даних від несанкціонованого доступу;

– цифровий підпис - PGP дозволяє створювати цифровий підпис для документів, що дозволяє перевірити автентичність документів та визначити, чи були вони змінені;

– захист електронної пошти - PGP може захистити електронну пошту від несанкціонованого доступу та забезпечити конфіденційність даних, які пересилаються;

– захист файлів - PGP може захистити файли від несанкціонованого доступу, зашифрувавши їх та забезпечивши парольний захист;

– PGP дозволяє користувачам забезпечувати захист своїх даних в інфокомунікаційних системах, забезпечуючи конфіденційність та автентичність даних, які передаються через інтернет. PGP є ефективним інструментом для захисту конфіденційної інформації та дотримання правил безпеки даних.

Основні переваги Pretty Good Privacy (PGP) полягають в тому, що він:

– забезпечує конфіденційність даних. Використовує сильне шифрування для захисту даних від несанкціонованого доступу;

– забезпечує автентичність даних. Дозволяє створювати цифровий підпис для документів, що дозволяє перевірити автентичність документів та визначити, чи були вони змінені;

– легко використовується. Має інтуїтивно зрозумілий інтерфейс користувача та дозволяє легко кодувати та декодувати дані.

Проте, PGP також має кілька недоліків:

– вимагає налаштування. Для користування PGP потрібно налаштувати ключі шифрування та підписування, що може бути складним для користувачів;

– недоступний для широкого загалу. Є спеціалізованим програмним забезпеченням, яке може бути важко знайти та встановити для звичайних користувачів;

– обмежена підтримка. Не підтримується всіма поштовими сервісами, що може ускладнити використання програмного забезпечення.

Програмне забезпечення для захисту даних Pretty Good Privacy (PGP), може бути корисним інструментом для захисту конфіденційної інформації та дотримання правил безпеки даних.

Base64 Encoder/Decoder

Base64 Encoder/Decoder - це метод кодування, який перетворює дані у текстовий формат, що складається з 64 символів. Цей метод використовується для передачі даних в інтернеті, в тому числі при електронній пошті та передачі файлів.

Основні функції Base64 Encoder/Decoder:

– кодування даних. Перетворює бінарні дані у текстовий формат, що складається з 64 символів;

– декодування даних. Перетворює текстовий формат даних, що складається з 64 символів, у бінарний формат;

– захист даних. Дозволяє захистити дані від незаконного доступу, оскільки закодовані дані неможливо зрозуміти без декодування;

– легко використовується. Є досить простим у використанні, і може бути застосований у багатьох програмах та пристроях.

Base64 Encoder/Decoder є широко використовуваним методом кодування, який дозволяє захистити дані та легко передавати їх через інтернет.

Основні переваги методу кодування Base64 Encoder/Decoder полягають в тому, що він:

– легко використовується. Є досить простим у використанні, і може бути застосований у багатьох програмах та пристроях;

– забезпечує захист даних. Дозволяє захистити дані від незаконного доступу, оскільки закодовані дані неможливо зрозуміти без декодування;

– підтримує різні типи даних. Може бути використаний для кодування різних типів даних, включаючи текст, зображення та відео.

Проте, Base64 Encoder/Decoder має кілька недоліків:

– розмір даних збільшується. При використанні Base64 Encoder/Decoder розмір даних збільшується на близько 33%, що може призвести до збільшення обсягу передаваних даних;

– не дуже ефективний для великих обсягів даних. Не є найбільш ефективним методом для великих обсягів даних, оскільки це може зайняти багато часу та ресурсів;

– не є безпечним. Не є абсолютно безпечним методом кодування, оскільки існують методи для декодування закодованих даних.

Base64 Encoder/Decoder - це широко використовуваний метод кодування, який дозволяє захистити дані та легко передавати їх через інтернет. Основні переваги цього методу полягають у простоті використання та захисті даних від

незаконного доступу. Проте, Base64 Encoder/Decoder має кілька недоліків, таких як збільшення розміру даних та недостатня ефективність для великих обсягів даних. Вибір методу кодування залежить від конкретного застосування, тому необхідно зважати на його переваги та недоліки перед використанням.

OpenSSL

OpenSSL - це бібліотека криптографічних протоколів та інструментарій, що забезпечує захист даних за допомогою шифрування та підписування даних, аутентифікації та управління сертифікатами.

Переваги OpenSSL:

– відкрите програмне забезпечення. Є вільним та відкритим програмним забезпеченням, що дозволяє використовувати його безкоштовно та модифікувати за необхідності;

– широко підтримується. Підтримується багатьма платформами та ОС, включаючи Windows, Linux, macOS, FreeBSD та інші;

– різноманітність криптографічних алгоритмів. Містить різноманітність криптографічних алгоритмів для шифрування, підпису та генерації ключів, що забезпечує високий рівень безпеки;

– підтримка протоколів TLS та SSL. Підтримує протоколи TLS та SSL, що забезпечують безпеку та конфіденційність при передачі даних через мережу.

Недоліки OpenSSL:

– вразливості безпеки. У попередніх версіях OpenSSL були виявлені вразливості безпеки, що може призвести до компрометації безпеки даних;

– складність використання. Має досить складний інтерфейс та вимагає досвіду у криптографії для ефективного використання;

– обмежена підтримка документації. Документація OpenSSL може бути обмеженою, що може зробити його використання складним для початківців;

– відсутність графічного інтерфейсу. Не має графічного інтерфейсу, що може зробити його використання складним для користувачів з обмеженим досвідом роботи з командним рядком.

Отже, OpenSSL забезпечує широкі можливості для захисту даних за допомогою шифрування, підписування даних, аутентифікації та управління сертифікатами. Незважаючи на деякі його недоліки, OpenSSL є потужним та надійним інструментом для забезпечення безпеки та конфіденційності даних.

Для більш кращого розуміння функціоналу розглянутих програмних рішень, необхідно провести їх порівняння по загальним функціональним характеристикам у вигляді таблиці (табл. 1.1).

Таблиця 1.1 – Порівняльний аналіз програмних рішень

Характеристика	Pretty Good Privacy	Base64 Encoder/ Decoder	OpenSSL
Тип програми	Крипто-графічний	Кодування/ Декодування	Крипто-графічний
Вид шифрування	Симетричне/ асиметричне	Немає	Симетричне/ асиметричне
Формат даних	Текст/файли	Текст/бінарні дані	Текст/бінарні дані
Підтримувані алгоритми	AES, RSA, SHA	Немає	AES, RSA, SHA
Ліцензія	Відкрита	Відкрита	Відкрита
Підтримувані операційні системи	Windows, macOS, Linux, Android, iOS	Універсальний	Windows, macOS, Linux, Android, iOS
Ефективність шифрування	Висока	Низька	Висока
Обсяг додатку	Великий	Малий	Великий

Рівень складності	Високий	Низький	Високий

Проведене порівняння трьох програмних рішень - Pretty Good Privacy, Base64 Encoder/Decoder та OpenSSL, дозволило виявити різні характеристики кожного з них. Pretty Good Privacy та OpenSSL - це криптографічні додатки з високим рівнем складності та великим обсягом, що підтримують різні алгоритми шифрування. Додаток Base64 Encoder/Decoder, з іншого боку, є простішим у використанні і має малий обсяг, але не підтримує шифрування та може збільшувати розмір даних.

Таким чином, для захисту конфіденційної інформації можуть використовуватись різні методи та програмні рішення, залежно від потреб та конкретних умов використання. При виборі методу необхідно враховувати характеристики кожного додатку, його ефективність та можливість забезпечення потрібного рівня захисту даних.

1.5. Обґрунтування необхідності розробки методу кодування інформації

За останні роки зростає кількість зловмисних атак на інформаційні системи, що може призвести до втрати конфіденційної інформації, порушення діяльності підприємств та інших наслідків. Тому, забезпечення ефективного методу кодування інформації є важливим кроком для підвищення безпеки інформаційних систем.

Одним з головних чинників, який змушує шукати нові методи кодування інформації, є швидкий розвиток технологій та збільшення обсягу обміну даними в інтернеті. Існуючі методи кодування можуть бути неефективними для обробки великих обсягів даних, тому необхідна розробка нового методу, який забезпечуватиме швидку та безпечну передачу даних.

Також варто зазначити, що залежно від типу інформації яка передається, існуючі методи кодування можуть бути недостатньо ефективними. Наприклад,

якщо інформація містить конфіденційні дані, то необхідно застосовувати надійний метод шифрування для запобігання її перехопленню та незаконному використанню. В інших випадках, якщо даними можна поділитись публічно, то можна використовувати менш складні методи кодування.

Крім того, розробка нового методу кодування може бути важливою з точки зору стандартизації та сумісності між різними інфокомунікаційними системами. Якщо розроблено універсальний та ефективний метод кодування інформації, то його можна використовувати в різних системах, що спростить процес взаємодії між ними та забезпечить збільшення продуктивності та ефективності роботи.

Важливим фактором, який зумовлює необхідність розробки нового методу кодування, є нестача ефективних методів забезпечення захисту даних від атак з використанням квантових комп'ютерів. За останні роки розроблено кілька нових методів шифрування, які ґрунтуються на використанні квантових принципів та є більш ефективними відносно до класичних методів. Розробка нового методу кодування, який буде забезпечувати захист даних від квантових атак, може бути важливим кроком у напрямку забезпечення безпеки даних в інфокомунікаційних системах.

Отже, обґрунтування необхідності розробки нового методу кодування інформації базується на широкому колі факторів, включаючи збільшення обсягу передавання інформації, нестачу ефективних методів забезпечення захисту даних від атак, а також на потребі в забезпеченні високого рівня безпеки даних в інфокомунікаційних системах.

1.6. Постановка задачі

Постановка задачі полягає у формалізації основних завдань, які перед нами стоять у процесі розробки нового методу кодування інформації для підвищення ефективності функціонування інфокомунікаційних систем. Основні завдання

поставлені з метою визначення обсягу робіт, що потрібно виконати, а також для забезпечення чіткості та однозначності у процесі розробки нового методу.

Основні завдання, що були поставлені перед розробниками методу кодування інформації включають:

– розробка нового методу кодування інформації на основі вибраного методу та врахування вимог до ефективності, безпеки та сумісності з іншими системами;

– реалізація розробленого методу кодування інформації у вигляді програмного продукту з урахуванням вимог до безпеки даних та ефективності роботи.

– проведення тестування розробленого методу кодування інформації з метою перевірки його ефективності та безпеки.

– оформлення документації з описом розробленого методу кодування інформації та інструкцій щодо його використання.

У процесі розробки методу кодування інформації також потрібно враховувати ряд інших факторів, зокрема:

– підвищення ефективності передачі інформації шляхом зменшення обсягу даних;

– забезпечення безпеки даних шляхом використання надійного алгоритму шифрування та дотримання вимог до безпеки;

– забезпечення сумісності з іншими системами шляхом врахування стандартів та протоколів взаємодії;

– забезпечення простоти використання та можливості інтеграції з іншими програмними продуктами.

У вимогах до розробки методу кодування інформації можна виділити функціональні та нефункціональні вимоги.

Функціональні вимоги описують основні функції та можливості, які повинен забезпечувати розроблений метод кодування інформації. До функціональних вимог можна віднести:

– можливість кодування та декодування даних за допомогою розробленого методу;

– забезпечення стійкості до зовнішніх впливів та збереження якості кодування при передачі даних по мережі;

– можливість використання розробленого методу на різних платформах та в різних інфокомунікаційних системах;

– забезпечення високої швидкості кодування та декодування даних;

– можливість інтеграції розробленого методу з іншими програмними продуктами.

Нефункціональні вимоги описують вимоги до якості, ефективності та безпеки розробленого методу кодування інформації. До нефункціональних вимог можна віднести:

– забезпечення надійності та безпеки даних під час їх передачі;

– забезпечення високого рівня стійкості до атак зломщиків та забезпечення можливості виявлення спроб несанкціонованого доступу;

– забезпечення сумісності з іншими інфокомунікаційними системами та протоколами взаємодії;

– забезпечення високої продуктивності та швидкості роботи;

– забезпечення простоти використання та можливості налаштування параметрів;

– дотримання вимог до безпеки програмного забезпечення та стандартів кодування.

Врахування функціональних та нефункціональні вимог під час розробки методу кодування інформації дозволить забезпечити високу якість та ефективність функціонування розробленої системи.

Для досягнення цієї мети важливо також враховувати вимоги до зручності використання розробленого методу кодування інформації користувачами. При розробці методу потрібно врахувати те, що використання методу не повинно створювати додаткових труднощів для користувачів, а має забезпечувати максимально можливий рівень автоматизації процесів кодування та декодування даних.

Також важливо враховувати вимоги до відмовостійкості та можливості відновлення даних в разі непередбачених ситуацій. Розроблений метод кодування інформації повинен забезпечувати можливість відновлення даних в разі їх пошкодження чи втрати.

Отже, постановка задачі полягає в тому, щоб розробити універсальний та ефективний метод кодування інформації, який забезпечить підвищення ефективності функціонування інфокомунікаційних систем, зменшення обсягу передаваної інформації, забезпечення безпеки даних та сумісності з іншими системами. Для досягнення цієї мети потрібно враховувати функціональні та нефункціональні вимоги до розробленого методу кодування інформації та забезпечити високий рівень якості та ефективності функціонування розробленої системи.

1.7. Висновок до розділу

У даному розділі було проведено огляд основних понять, пов'язаних з інфокомунікаційними системами, та описано проблему, пов'язану з підвищенням їх ефективності. Для більш глибокого розуміння проблеми був проведений аналіз існуючих методів кодування інформації та програмних рішень.

На основі аналізу існуючих методів кодування інформації було зрозуміло, що існуючі методи мають свої переваги та недоліки, тому для розробки нового методу потрібно враховувати як можливі переваги, так і недоліки існуючих методів. При проведенні аналізу програмних рішень було виявлено, що існують

деякі програми, що можуть бути використані для кодування інформації, однак вони не забезпечують повної ефективності та безпеки передачі інформації.

На основі проведеного аналізу було обґрунтовано необхідність розробки нового методу кодування інформації, який забезпечить високу ефективність та безпеку передачі даних. При цьому, постановка задачі надала чіткі вимоги до розробленого методу, що дозволяє максимально точно визначити функціональні та нефункціональні вимоги до розробленого методу.

РОЗДІЛ 2. ВИБІР ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ ТА ОСОБЛИВОСТІ АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Обґрунтування вибору мови програмування

Для розробки нового методу кодування інформації для підвищення ефективності функціонування інфокомунікаційних систем необхідно насамперед обрати мову програмування. Для вирішення цього питання було розглянуто декілька мов програмування, що найбільш підходять для розробки нового методу кодування інформації. Зокрема, було розглянуто можливості використання таких мов програмування, як Python, Java та C#.

Python - це інтерпретована мова програмування з високим рівнем абстракції, що забезпечує зручний та простий синтаксис для розробки програм. Python є мовою програмування загального призначення, яка використовується для розробки веб-додатків, ігор, наукових програм та системного адміністрування. Вона має потужну бібліотеку стандартних функцій та додаткових пакетів, які дозволяють розширювати її функціонал для різних потреб.

Java - це високорівнева мова програмування, яка підтримує об'єктно-орієнтований підхід та має платформонезалежність. Java використовується для розробки великих корпоративних систем, мобільних додатків та веб-додатків.

C# - це високорівнева мова програмування, розроблена компанією Microsoft. C# має синтаксис, подібний до C і C++, а також підтримує об'єктно-орієнтований підхід. Використовується для розробки веб-додатків, десктопних програм та мобільних додатків.

Для зручності порівняння, результати аналізу було представлено у вигляді таблиці (табл. 2.1), де були вказані основні параметри кожної мови програмування, такі як швидкодія, легкість вивчення, рівень складності, наявність засобів розробки та підтримки спільноти розробників.

Таблиця 2.1 – Результати порівняння мов програмування

Параметри	*Python*	*Java*	*C#*
Підтримка ООП	Так	Так	Так
Платформо-незалежність	Частково (залежить від бібліотек)	Так	Частково (тільки на платформі .NET)
Спрощена синтаксична конструкція	Так	Ні	Ні
Ефективність виконання	Повільніше, ніж Java та C#	Швидше, ніж Python, але повільніше, ніж C#	Швидше, ніж Python, та Java
Обробка великих даних	Хороша	Добра	Добра
Використання в наукових дослідженнях	Добре підходить	Ні	Ні
Спільнота розробників	Велика	Велика	Велика

Проведений аналіз мов програмування Python, Java та C# показав, що всі вони мають свої переваги та недоліки, але для розробки методу кодування інформації для підвищення ефективності функціонування інфокомунікаційних систем, було вирішено використовувати мову програмування C#.

Основними причинами вибору мови C# є:

– мова програмування, яка є частиною платформи .NET, що забезпечує широкі можливості для створення програмного забезпечення;

– підтримує об'єктно-орієнтований підхід до програмування, що є дуже важливим для розробки складних програмних продуктів;

– має велику та активну спільноту розробників, що дозволить швидко розв'язувати проблеми та знайти відповіді на питання;

– має широкі можливості для використання в інфокомунікаційних системах.

Отже, вибір мови програмування C# обґрунтований та відповідає вимогам до розробки методу кодування інформації для підвищення ефективності функціонування інфокомунікаційних систем.

2.2. Обґрунтування вибору середовища розробки

При розробці методу кодування інформації для підвищення ефективності функціонування інфокомунікаційних систем важливим аспектом є вибір середовища розробки, яке найкраще підходить для виконання даного завдання.

Одним з найважливіших критеріїв при виборі середовища розробки є наявність потрібних інструментів та функцій для реалізації даного методу кодування. Наприклад, якщо використовується метод кодування на основі шифрування, то необхідно мати можливість використовувати відповідні бібліотеки для шифрування та розшифрування даних.

Visual Studio - це інтегроване середовище розробки, розроблене компанією Microsoft, яке надає можливості для розробки програмного забезпечення на різних мовах програмування, включаючи C++, C#, Python, JavaScript та багато інших. Visual Studio містить вбудовані інструменти для відладки, тестування, візуалізації та управління версіями коду. Visual Studio також має широкий вибір плагінів та розширень, що робить його дуже гнучким та зручним для розробки різноманітних програмних продуктів. Враховуючи все вищезазначене, Visual Studio є одним з найпопулярніших та потужних інструментів для розробки програмного забезпечення.

PyCharm - це інтегроване середовище розробки для мови програмування Python, яке розроблено компанією JetBrains. PyCharm надає розробникам потужні інструменти для відлагодження коду, автоматичного завершення коду, рефакторингу та керування проектами. PyCharm також має вбудовану підтримку різних фреймворків та бібліотек Python, таких як Django, Flask, NumPy, Pandas та багато інших. За рахунок своєї потужної функціональності та зручного інтерфейсу, PyCharm став одним з найпопулярніших інструментів для розробки на Python.

IntelliJ IDEA - це інтегроване середовище розробки, розроблене компанією JetBrains, для розробки програмного забезпечення на мовах програмування Java, Kotlin, Scala, Groovy та інших. IntelliJ IDEA надає широкі можливості для відладки, автоматичного завершення коду, рефакторингу та аналізу коду. Завдяки своїм інноваційним функціям, таким як функція "попереднього перегляду" (preview feature), підтримка мультимодульної розробки, підтримка роботи з Docker та іншими технологіями, IntelliJ IDEA стала одним з найпотужніших інструментів для розробки програмного забезпечення на Java-платформі.

У табл. 2.2 представлено порівняння основних характеристик середовищ розробки.

Таблиця 2.2 – Порівняння середовищ розробки

Характеристики	*Visual Studio*	*PyCharm*	*IntelliJ IDEA*
Мови програмування	Більше 20	Python	Java, Kotlin, Scala, Groovy та інші
Операційні системи	Windows, macOS, Linux	Windows, macOS, Linux	Windows, macOS, Linux
Відладка	Так	Так	Так
Автоматичне завершення коду	Так	Так	Так
Рефакторинг	Так	Так	Так

Підтримка Git	Так	Так	Так
Вбудований термінал	Так	Так	Так
Підтримка розширень	Так	Так	Так
Вартість	Безкоштовна до платна	Безкоштовна до платна	Безкоштовна до платна

Отже, всі три середовища розробки (Visual Studio, PyCharm та IntelliJ IDEA) надають розробникам широкий набір інструментів та можливостей для розробки програмного забезпечення на своїх відповідних мовах програмування. Вони підтримують операційні системи Windows, macOS та Linux, мають вбудовані інструменти для відладки, автоматичного завершення коду та рефакторингу. Вони також мають підтримку Git та вбудований термінал. Ці середовища можуть бути безкоштовними або платними, залежно від версії та функціональності.

Для розробки було обрано середовище Visual Studio, що підтримує мову програмування C#, окрім цього дане середовище є чудовим вибором завдяки своїй широкій функціональності та підтримці мов. Вона надає розробникам інструменти для ефективної роботи зі складними проектами, включаючи відладку, тестування, профілювання та керування версіями.

Також, Visual Studio є одним з найпопулярніших середовищ розробки на ринку, що забезпечує низький поріг вступу для нових розробників, які можуть швидко знайти допомогу та підтримку в онлайн-спільнотах. Visual Studio має широкий вибір платформ та сервісів, таких як Azure та GitHub, що забезпечують можливість зберігання та розгортання проекту в хмарі.

2.3. Обґрунтування вибору СКБД

Розробка методу кодування інформації для підвищення ефективності функціонування інфокомунікаційних систем потребує вибору системи керування

базами даних (СКБД), яка забезпечить ефективну роботу з даних. У цьому контексті було розглянуто такі СКБД, як PostgreSQL, Oracle та Microsoft Access.

PostgreSQL - це об'єктно-реляційна система керування базами даних з відкритим вихідним кодом, яка підтримує мови програмування, такі як SQL, Python, Java та інші. PostgreSQL забезпечує високу продуктивність, можливість роботи з великими обсягами даних, підтримку транзакцій та рівень безпеки на рівні фінансових установ. PostgreSQL є безкоштовною та відкритою СКБД, що робить її популярним вибором для багатьох проектів, які не мають можливості використовувати комерційні СКБД.

Oracle - це одна з провідних систем керування базами даних у світі, яка підтримує багато мов програмування та операційних систем. Oracle забезпечує високу продуктивність та швидкий доступ до великих обсягів даних, а також підтримку технологій, таких як машинне навчання та штучний інтелект. Oracle має потужні інструменти для резервного копіювання, міграції та реставрації даних, що робить її популярним вибором для великих корпоративних проектів. Однак, Oracle є комерційною СКБД, що може бути досить високою для менших проектів або початківців.

Microsoft Access - це система управління базами даних (СУБД) від Microsoft, яка зазвичай використовується для роботи з невеликими базами даних на робочих станціях. Microsoft Access має простий інтерфейс та легку настройку, що робить її популярним вибором для невеликих бізнесів та початківців в області розробки баз даних. Microsoft Access надає користувачам можливість відносно швидко побудувати базу даних без великих витрат на розробку та підтримку. Однак, Microsoft Access не є відповідним вибором для великих проектів або додатків з високою навантаженістю, оскільки його можливості та продуктивність обмежені.

У табл. 2.3 представлено порівняння основних функціональних можливостей систем керування базами даних, таких як: PostgreSQL, Oracle та Microsoft Access.

Таблиця 2.3 – Порівняння СКБД

Характеристики	*PostgreSQL*	*Oracle*	*Microsoft Access*
Тип	Реляційна СКБД	Реляційна СКБД	Реляційна СУБД
Відкритий вихідний код	Так	Ні	Ні
Максимальний обсяг бази даних	До 32 ТБ	Більше 100 ТБ	2 ГБ
Підтримувані мови програмування	Більше 10	Більше 10	SQL, VBA
Підтримувані операційні системи	Більше 20	Більше 20	Тільки Windows
Ціна	Безкоштовна та комерційна підтримка	Комерційна підтримка	Вбудована в Microsoft Office

Microsoft Access має перевагу конкурентами в тому, що це вбудована СУБД, що поставляється з Microsoft Office, тому вона не потребує додаткових витрат на придбання та встановлення. Однак, з огляду на свій обмежений функціонал та максимальний обсяг бази даних в 2 ГБ, Microsoft Access надає менше можливостей для більш складних проектів порівняно з PostgreSQL та Oracle. PostgreSQL та Oracle є об'єктно-реляційними СКБД з відкритим та комерційним вихідним кодом, вони підтримують більше мов програмування та операційних

систем, а також мають великий максимальний обсяг бази даних. Однак, вони є більш складними у встановленні та підтримці порівняно з Microsoft Access.

2.4. Аналіз вимог. Use-case діаграми. Основні прецеденти

Окрім методу кодування інформації для підвищення ефективності функціонування інфокомунікаційних систем буде також розроблено застосунок, що дозволить показати можливість та ефективність його використання.

Перед розробкою будь-якого застосунку необхідно провести аналіз вимог до майбутньої системи, щоб забезпечити відповідність розроблюваної системи потребам користувачів та функціональним вимогам до неї. Для цього доцільно використовувати use-case діаграму прецедентів, яка дозволяє відобразити взаємодію користувачів з системою та описати основні функціональні вимоги до неї.

Use-case діаграма прецедентів дозволяє описати взаємодію користувачів з системою у вигляді сценаріїв, які описуються в рамках окремих прецедентів. Кожен прецедент описує конкретну дію, яку користувач може виконати в системі, а також реакцію системи на цю дію. Таким чином, use-case діаграма дозволяє відобразити весь спектр функціональних вимог до системи та визначити їх пріоритетність.

Використання use-case діаграми перед розробкою дозволить забезпечити високу якість розроблюваної системи та зменшити ризик невідповідності потребам користувачів. Крім того, use-case діаграма дозволяє створити однозначну та зрозумілу специфікацію вимог до системи, що полегшує подальший процес розробки та тестування. Таким чином, використання use-case діаграми є доцільним та необхідним етапом перед розробкою будь-якої системи.

Для початку необхідно висунути список вимог до системи, а саме: функціональні вимоги (табл. 2.4), нефункціональні вимоги (табл. 2.5), актори та цілі застосунку(табл. 2.6) та описати варіанти використання застосунку (табл. 2.7).

Таблиця 2.4 – Функціональні вимоги до застосунку

Вимоги	**Опис**
REQ-1	Можливість проведення реєстрації та автентифікації
REQ-2	Ведення журналу подій
REQ-3	Можливість перевірки методу кодування інформації
REQ-4	Можливість збереження закодованої інформації
REQ-5	Можливість перегляду збереженої закодованої інформації
REQ-6	Можливість фіксації проведення перевірки процедури кодування

Таблиця 2.5 – Нефункціональні вимоги застосунку

Вимоги	**Опис**
REQ-7	Система повинна мати простий дизайн та зручну навігацію
REQ-8	Поля повинні бути унікальними відносно вже існуючих записів

Таблиця 2.6 – Актори та цілі застосунку

Актори	**Цілі**
Адміністратор	Мета адміністратора полягає у забезпеченні безпеки та правильного функціонування облікових записів у системі
Користувач	Мета користувача полягає в ефективному та зручному користуванні системою
База даних	Мета бази даних полягає у зберіганні та організації необхідної інформації

Таблиця 2.7 – Опис варіантів використання застосунку

Варіант використання	**Ім'я**	**Опис**
UC1	Реєстрація користувача	Дозволяє здійснити реєстрацію користувача у системі
UC2	Автентифікація в системі	Дозволяє пройти автентифікацію в системі
UC3	Вивід каталогу користувачів	Дозволяє користувачеві з правами адміністратора вивести каталог всіх користувачів системи
UC4	Додати користувача	Дозволяє розширити кількість користувачів системи
UC5	Редагувати користувача	Дозволяє змінювати та оновлювати інформацію про користувача, що є важливим для забезпечення актуальності даних у системі
UC6	Перевірка роботи методу кодування	Дає можливість користувачеві перевірити метод кодування інформації із використанням текстових даних
UC7	Збереження результатів роботи	Дозволяє зберегти інформацію, що використовувалась для перевірки ефективності роботи методу у базі даних
UC8	Виведення закодованої інформації	Дозволяє вивести збережену закодовану інформацію та здійснити її розкодовування
UC9	Фіксація додавання інформації	Дозволяє здійснити фіксування додавання інформації у базу даних
UC10	Перегляд подій	Дозволяє переглядати події, що відбулися у системі

На основі отриманих даних було побудовано use-case діаграми прецедентів для ролі адміністратор та користувач, що зображені на рис. 2.1 та рис. 2.2 відповідно.

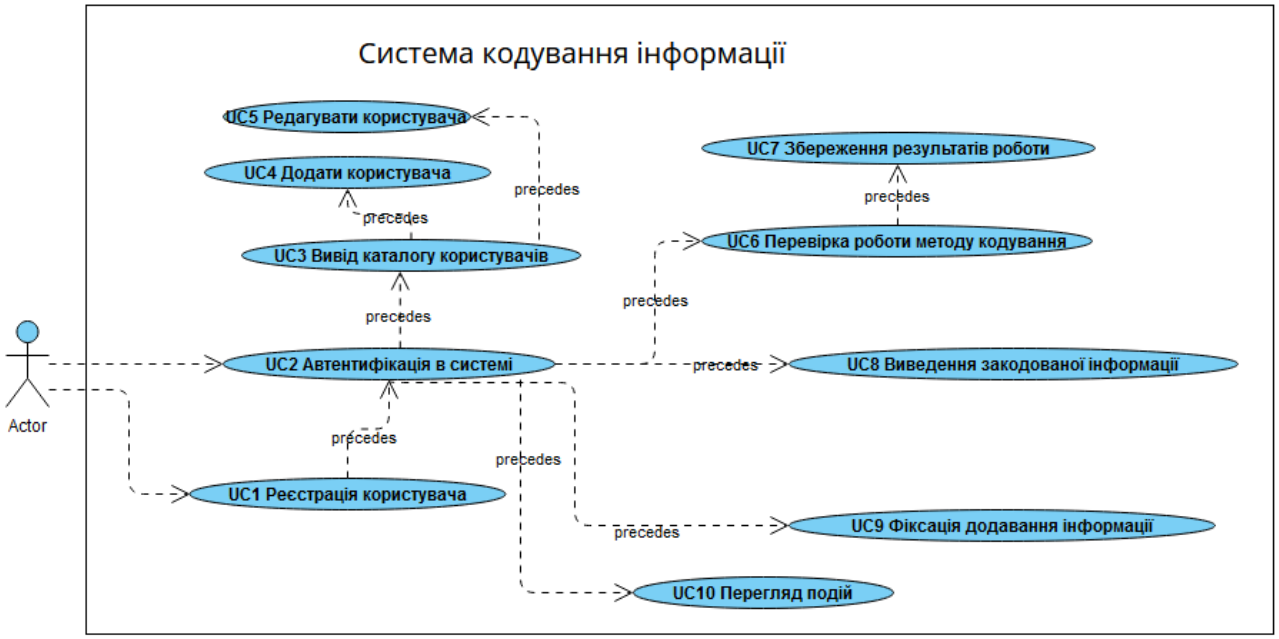


Рисунок 2.1 – Діаграма use-case із роллю «адміністратор»

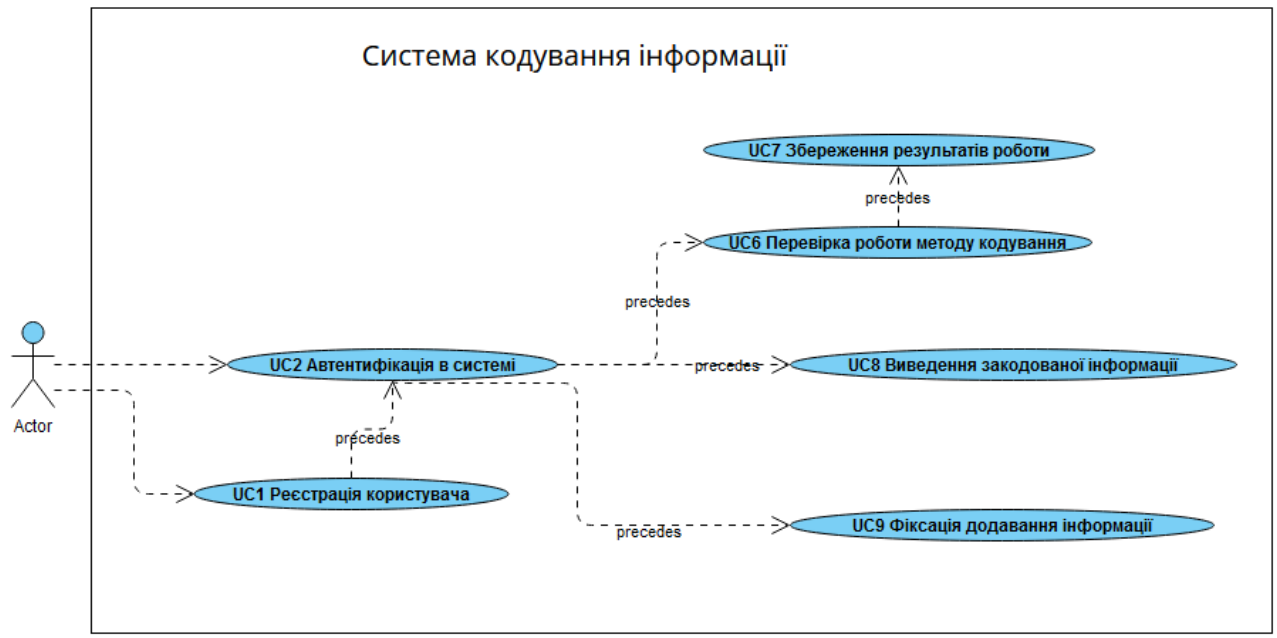


Рисунок 2.2 – Діаграма use-case із роллю «користувач»

2.5. Архітектура проекту

Архітектура проекту - це концептуальна модель, яка описує структуру та поведінку проекту. Під час розробки методу кодування інформації для підвищення ефективності функціонування інфокомунікаційних систем, була вибрана трьохрівнева архітектура, яка дозволяє розділити систему на три рівні: презентаційний, логічний та рівень доступу до даних.

Презентаційний рівень відповідає за інтерфейс користувача та забезпечує зручний та зрозумілий доступ до функціоналу системи. Логічний рівень відповідає за бізнес-логіку системи та забезпечує взаємодію між різними модулями системи. Рівень доступу до даних відповідає за доступ до даних, їх зберігання та маніпулювання.

Вибір трьохрівневої архітектури був обумовлений декількома факторами. По-перше, така архітектура дозволяє розділити систему на окремі компоненти та зробити їх незалежними один від одного, що полегшує подальшу розробку та підтримку системи. По-друге, трьохрівнева архітектура дозволяє забезпечити високу безпеку та захист даних, оскільки доступ до даних може бути обмежений лише на рівні доступу до даних. По-третє, така архітектура дозволяє легко масштабувати систему, додавати новий функціонал та модифікувати існуючий, що дуже важливо у випадку з інфокомунікаційними системами.

2.5.1. Особливості розробки бази даних. ERD діаграма з описанням сутностей

Однією з ключових складових розробки методу кодування інформації для підвищення ефективності функціонування інфокомунікаційних систем є створення бази даних, яка забезпечує зберігання та обробку даних, що використовуються у системі.

При розробці бази даних доцільно використовувати ERD діаграму (Entity-Relationship Diagram), яка дозволяє візуалізувати структуру бази даних та описати

взаємозв'язки між сутностями. Сутності в ERD діаграмі відображають реальні об'єкти, які зберігаються в базі даних, а взаємозв'язки між ними відображають зв'язки між цими об'єктами.

Основні сутності, які використовуються в базі даних для розробки методу кодування інформації включають:

– кодування: ідентифікатор кодування, назва кодування, ідентифікатор користувача, ключ кодування та закодована інформація;

– історія: ідентифікатор історії, дата та час проведення перевірки методу кодування та назва збережених даних;

– події: ідентифікатор події, дата події, ідентифікатор користувача та опис події;

– користувачі: ідентифікатор користувача, прізвище, ім’я, назва облікового запису, пароль, ідентифікатор ролі та додатковий опис.

У процесі проектування бази даних для додатку "Система кодування інформації" було виконано встановлення зв'язків між сутностями, які полягали у встановленні зв'язків між полями однакових типів та полями, які мали зв'язок між собою. Зв'язки між таблицями були встановлені за допомогою зовнішніх та внутрішніх ключів, які забезпечували взаємозв'язок між таблицями. У логічній моделі бази даних для предметної області були встановлені різні типи зв'язків, такі як "один до одного", "один до багатьох" та "багато до багатьох". Це дозволило коректно побудувати зв'язки між таблицями та забезпечити правильну роботу функціоналу бази даних, що в свою чергу впливає на ефективність та продуктивність системи в цілому. Отже, правильне визначення типів зв'язків між таблицями є дуже важливим етапом у процесі проектування бази даних.

Організовано такі зв'язки між таблицями:

– "Users" (поле "UsersId") – "Logs" (поле "UsersId"), вид зв’язку 1:N;
– "Users" (поле "UsersId") – "Codings" (поле "UsersId"), вид зв’язку 1:N.

Отже, в результаті було створено ER-діаграму для бази даних "Система кодування інформації", яка наглядно відображає взаємозв'язки між таблицями та їх полями (рис. 2.3). Ця модель є важливою при подальшому проектуванні та розробці функціоналу додатку.

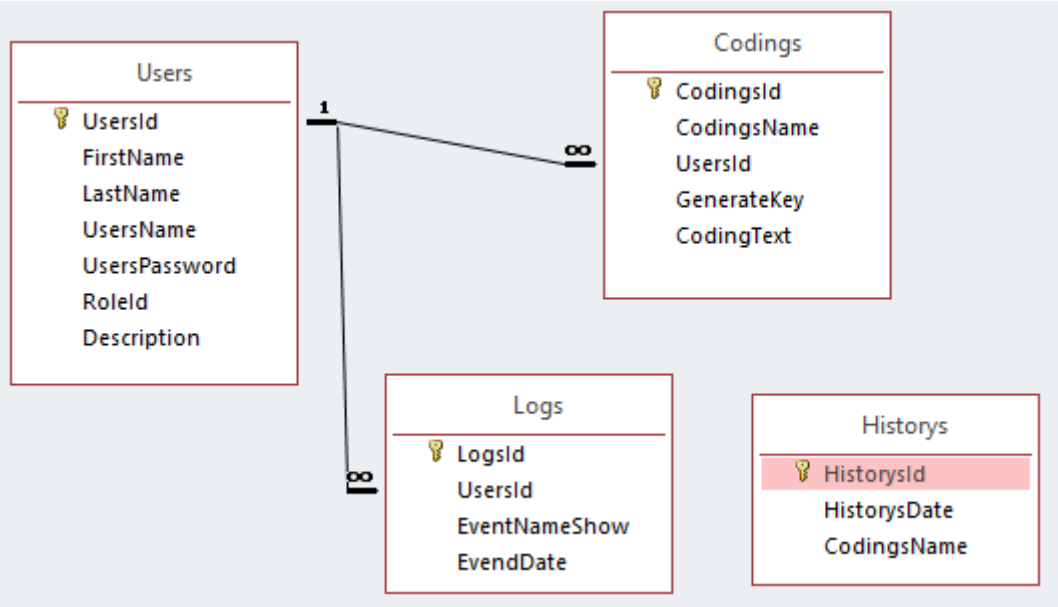


Рисунок 2.3 – Діаграма бази даних

2.5.2. Особливості розробки рівня BLL

В цьому рівні відбувається розробка бізнес-логіки системи, яка визначає логіку взаємодії між даними та бізнес-процесами.

Основні завдання рівня BLL включають:

– визначення бізнес-правил та логіки взаємодії між даними. Наприклад, встановлення правил валідації даних, що вводяться користувачем, та розробка алгоритмів обробки цих даних.

– розробка функцій та методів, які дозволяють отримувати доступ до даних та здійснювати з ними операції;

– взаємодія з іншими рівнями системи, такими як рівень DAL (Data Access Layer), що забезпечує доступ до бази даних;

– тестування функціоналу рівня BLL та визначення його продуктивності та ефективності.

Основна мета рівня BLL полягає у забезпеченні правильної та ефективної роботи бізнес-логіки системи, що в свою чергу впливає на продуктивність та ефективність системи в цілому. Таким чином, розробка рівня BLL є важливим етапом у процесі розробки додатку та дозволяє забезпечити правильну роботу всієї системи.

Діаграма класів рівня бізнес-логіки зображена на рис. 2.4.

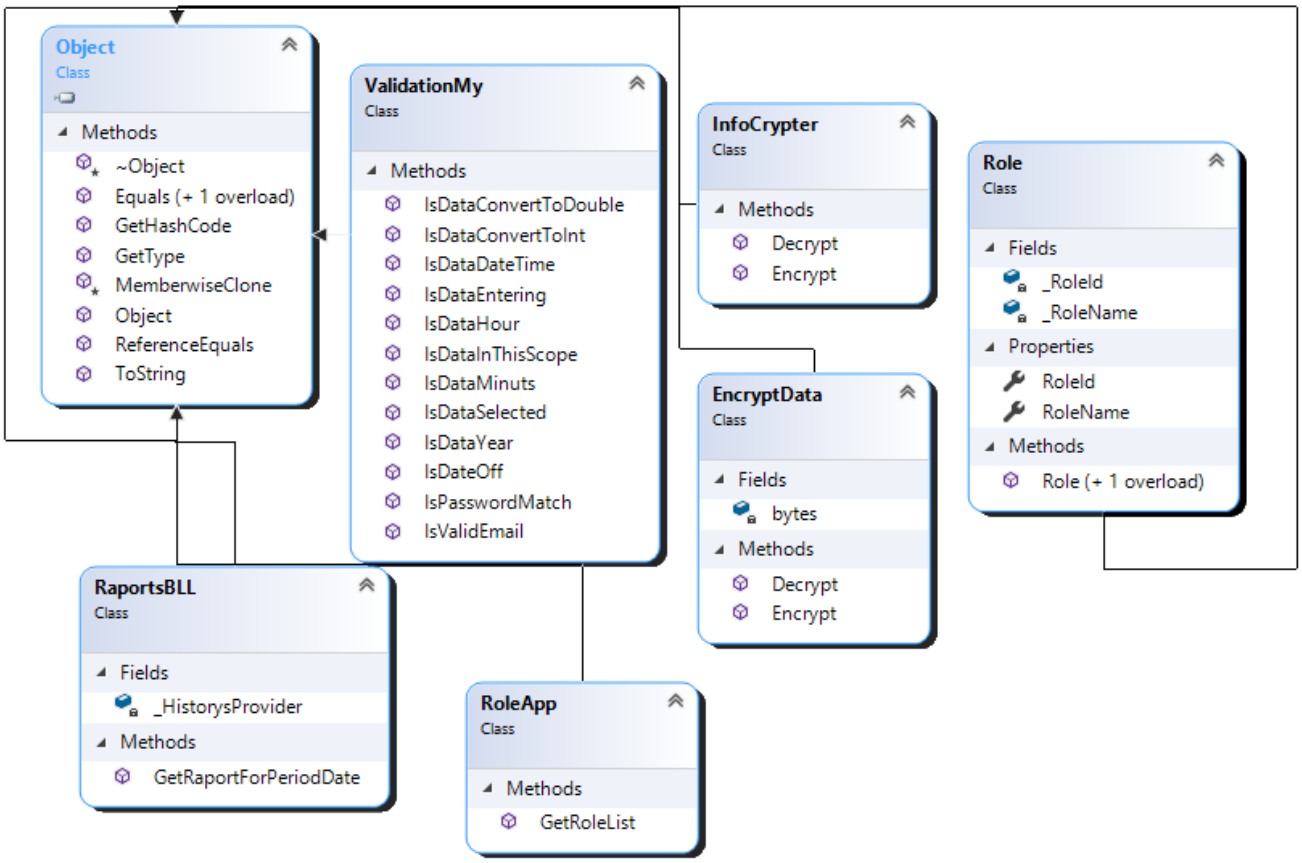


Рисунок 2.4 – Діаграма класів бізнес-логіки додатку

Даний рівень складається із 6 основних класів:

– клас EncryptData - це клас, який відповідає за шифрування та дешифрування даних про паролі користувачів системи. Він містить методи, які дозволяють шифрувати та дешифрувати дані за допомогою алгоритму шифрування. Він забезпечує захист від несанкціонованого доступу до системи;

– клас NamesMy - це клас, який відповідає за генерацію унікальних ідентифікаторів. Він містить методи, які дозволяють генерувати унікальні імена, номери та коди для різних сутностей системи. Наприклад, цей клас може генерувати унікальні імена користувачів, номери замовлень, коди продуктів та інші. Клас NamesMy є важливим у розробці методу кодування інформації, оскільки він забезпечує унікальність ідентифікаторів, що дозволяє коректно та однозначно ідентифікувати різні сутності системи.

– клас RoleApp - це клас, який відповідає за роль користувача в системі. Він містить методи, які дозволяють визначати роль користувача та його права доступу до різних функцій системи. Наприклад, цей клас може визначати, чи має користувач право на доступ до редагування даних, видалення користувачів та інші функції системи. Клас RoleApp є важливим у розробці методу кодування інформації, оскільки він дозволяє забезпечити безпеку даних та захист від несанкціонованого доступу до різних функцій системи;

– клас ValidationMy - це клас, який відповідає за перевірку валідності введених даних. Даний клас забезпечує коректність та безпеку даних, які вводяться користувачем;

– клас InfoCrypter - це клас, який відповідає за кодування та декодування інформації за допомогою різних алгоритмів шифрування. Він містить методи для кодування та розкодовування інформації користувача. Даний клас було розроблено з метою підвищення ефективності функціонування інфокомунікаційних систем;

– клас RaportsBLL - це клас, який відповідає за формування звітів та повідомлень системи. Він містить методи, які дозволяють генерувати звіти про стан системи, повідомлення про помилки, попередження та інші повідомлення для користувачів системи.

2.5.3. Особливості розробки рівня UI

Рівень UI (user interface) є ключовим в розробці будь-якої системи, оскільки він є основним інструментом взаємодії користувача з системою. Особливості розробки рівня UI полягають у створенні зручного та ефективного інтерфейсу для користувача, що дозволяє легко та швидко здійснювати операції з кодуванням та декодуванням інформації.

Для досягнення максимальної ефективності та зручності використання системи, необхідно враховувати потреби та пріоритети користувачів, дотримуватися принципів юзабіліті та дизайну, використовувати сучасні технології та практики розробки.

Для досягнення успіху в розробці рівня UI необхідно виконати декілька кроків: визначити потреби та пріоритети користувачів, детально продумати дизайн та інтерфейс системи, вибрати технології та інструменти, розробити прототип та тестувати його з користувачами. В результаті цих кроків можна отримати зручний та ефективний інтерфейс, що дозволить користувачам здійснювати операції з кодуванням та декодуванням інформації з максимальною ефективністю та зручністю.

Окрім цього, важливо забезпечити безпеку даних та захист від несанкціонованого доступу. Для цього потрібно використати методи та технології, такі як аутентифікація та авторизація користувачів, шифрування даних, захист від XSS- та CSRF-атак та валідацію введених даних.

Діаграма класів рівня користувацького інтерфейсу зображено на рис. 2.5.

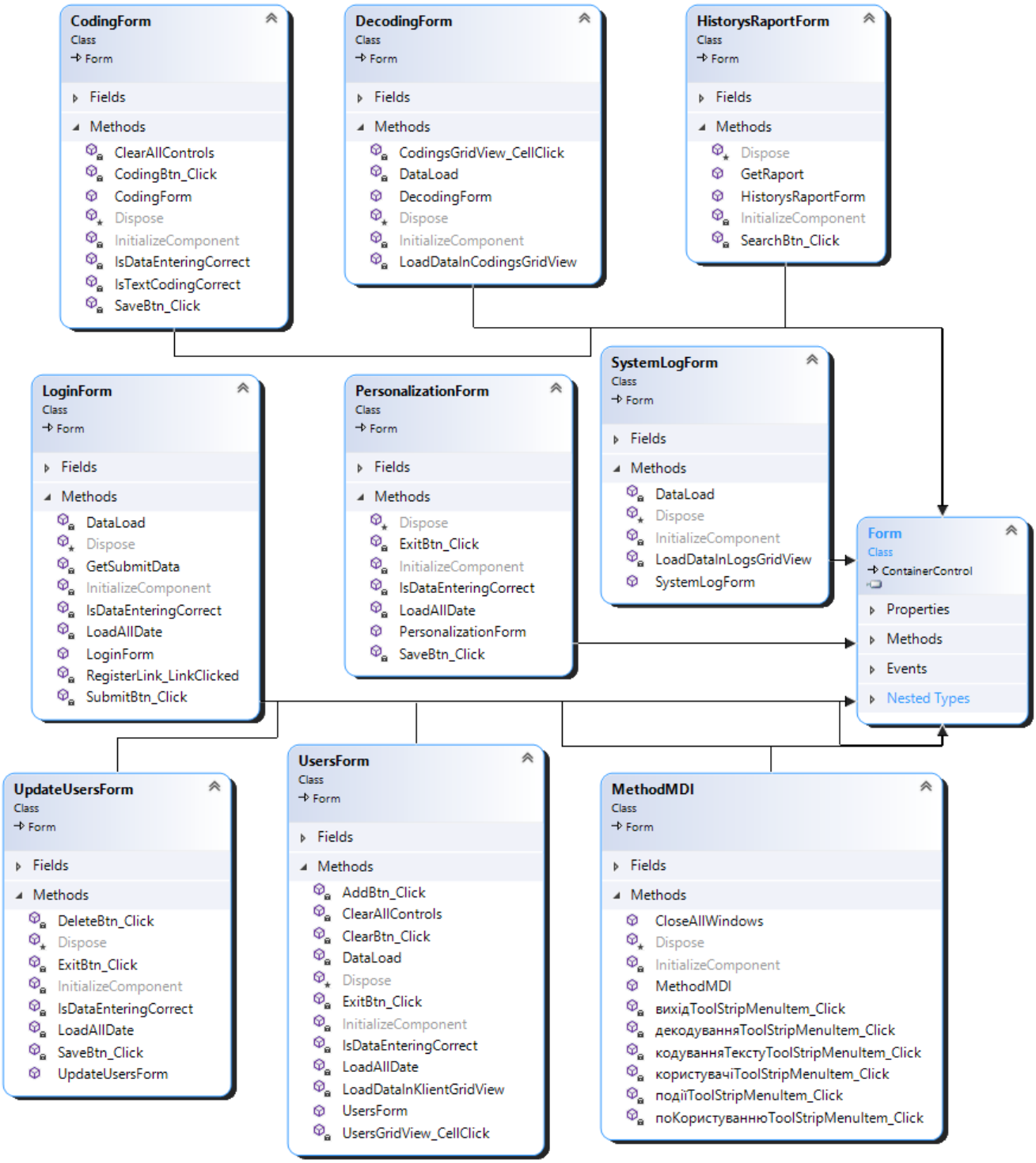


Рисунок 2.5 – Діаграма класів рівня користувацького інтерфейсу

Отже, рівень користувацького інтерфейсу містить дев’ять класів:

– клас MethodMDI виступає як головний контролер управління вікнами та діалогами додатку і забезпечує зручну навігацію між різними формами та вікнами. Він містить головне меню з основними функціями додатку та дозволяє

користувачеві виконувати різноманітні дії, такі як додавання, видалення та редагування даних. Крім того, клас MethodMDI містить головне меню, що забезпечує швидкий доступ до всіх функцій системи. Таким чином, клас MethodMDI є основою для роботи з іншими класами на рівні інтерфейсу користувача та забезпечує зручну та безпечну взаємодію з програмою. Він дозволяє реалізувати максимальну функціональність додатку та забезпечує зручний та інтуїтивний інтерфейс користувача;

– клас CodingForm відповідає за реалізацію функціоналу кодування інформації за реалізованим методом кодування інформації класу InfoCrypter. Клас також містить функції для відображення результату кодування на екрані та збереження даних у базу даних;

– клас DecodingForm відповідає за розкодування закодованої інформації бази даних. Він містить елементи керування для введення вхідних даних, та відображає результати розкодованої інформації;

– клас HistorysRaportForm відповідає за створення звіту про операції кодування та розкодування за останній період часу. Він містить елементи керування для вибору періоду часу, за який потрібно створити звіт, та для відображення результату. Клас містить методи для зчитування інформації з історії операцій кодування та розкодування, обробки цих даних та створення звіту;

– клас LoginForm відповідає за авторизацію користувача у системі. Він містить поля для введення логіну та пароля, кнопки для входу та відміни, а також можливість відновлення забутого пароля. Клас містить методи для перевірки введеного логіну та пароля, зчитування інформації про користувача з бази даних та відображення відповідного повідомлення про результат авторизації;

– клас PersonalizationForm відповідає за персоналізацію даних користувача;

– клас SystemLogForm відповідає за відображення логів системи. Клас містить методи для зчитування логів з бази даних, їх обробки та відображення на екрані.

Всі ці класи є частинами інтерфейсу користувача додатку і дозволяють зручно та ефективно взаємодіяти з функціоналом авторизації, персоналізації та логування системи. Вони забезпечують введення вхідних даних, вибір параметрів та відображення результату на екрані. Крім того, клас SystemLogForm дозволяє переглядати логи системи за певний період часу, що є корисною функцією для відстеження роботи програми та виявлення можливих помилок.

2.5.4. Особливості розробки DAL

Рівень DAL (Data Access Layer) відповідає за доступ до даних в базі даних та взаємодію з нею. Особливості розробки цього рівня включають в себе визначення структури бази даних, написання SQL-запитів, роботу з об'єктами даних та виконання операцій збереження, видалення, оновлення та вибірки даних.

Основними класами DAL є класи, що відображають структуру бази даних (наприклад, класи таблиць) та класи, що містять методи для здійснення операцій з базою даних. Класи таблиць містять властивості, які відображають стовпці таблиць бази даних. Класи з операціями використовують об'єкти класів таблиць для доступу до даних.

Особливості розробки DAL полягають в тому, що він має бути гнучким та забезпечувати оптимальний доступ до даних.

Діаграма класів рівня даних зображена на рис. 2.6.

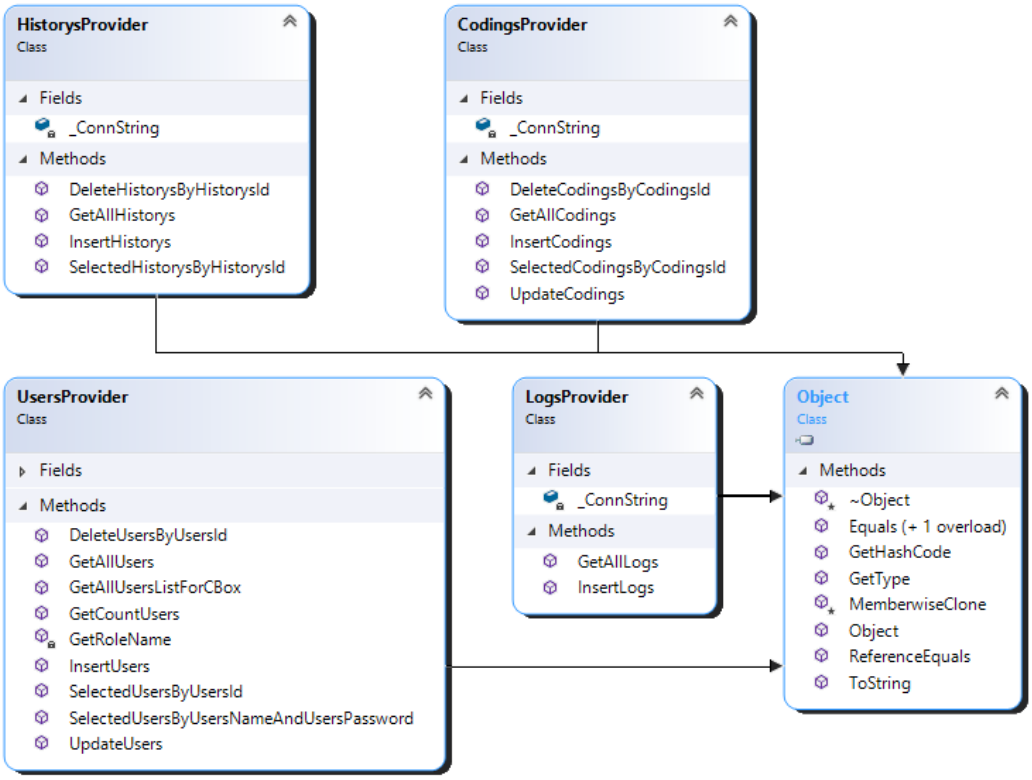


Рисунок 2.6 – Діаграма класів рівня даних

Як видно із рис. 2.6 даний шар складається із 4-ох основних класів:

– клас CodingProvider є складовою частиною рівня DAL (Data Access Layer) і відповідає за опрацювання закодованої інформації бази даних. Він містить методи для додавання, видалення та редагування записів бази даних;

– клас HistorysProvider також є складовою частиною рівня DAL і відповідає за збереження та отримання інформації про історію використання методу кодування даних у базі даних;

– клас LogsProvider є складовою частиною рівня DAL і відповідає за збереження та отримання інформації про системні події та дії користувачів у базі даних. Він містить методи для додавання, видалення про системні події та дії користувачів у базі даних, а також метод для отримання списку записів з бази даних. Клас LogsProvider забезпечує безперебійну роботу системи, що дає змогу користувачу зручно використовувати функціонал додатку та отримувати необхідну інформацію про системні події та дії користувачів;

– клас UsersProvider також є складовою частиною рівня DAL і відповідає за збереження та отримання інформації про користувачів у базі даних. Він містить методи для додавання, видалення та редагування записів про користувачів у базі даних, а також метод для отримання списку користувачів з бази даних.

2.6. Висновок до розділу

В результаті проведеного аналізу обрано мову програмування C#, середовище розробки Visual Studio та СКБД MS Access. Обрана тьохрівнева архітектура, що дозволяє забезпечити гнучкість та модульність проекту. База даних була розроблена з урахуванням основних принципів нормалізації даних та містить всі необхідні сутності та їх зв'язки. Рівень BLL включає класи, які забезпечують логіку обробки та збереження даних в базу даних. Рівень UI реалізує взаємодію користувача з додатком та забезпечує зручний та логічний інтерфейс.

РОЗДІЛ 3. РОЗРОБКА, ВИКОРИСТАННЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Розробка методу кодування інформації

Першим кроком в розробці методу кодування інформації є визначення вимог до методу кодування. Необхідно визначити рівень захисту, швидкість роботи, обсяг даних, які будуть кодуватися, інші фактори, що впливають на вибір методу кодування. У даному випадку, розроблюваний метод повинен бути швидким, ефективним і забезпечувати високий рівень захисту інформації.

Другим кроком є вибір методу кодування інформації. Вибір методу кодування залежить від вимог, які були визначені на попередньому етапі. У даному випадку, для забезпечення високого рівня захисту інформації використовується симетричний алгоритм кодування, такий як AES. Цей метод кодування забезпечує високий рівень захисту, швидкість роботи та ефективність.

Третім кроком є розробка програмного коду, тому для реалізації методу кодування та декодування інформації було створено клас "InfoCrypter", що складається із 2 методів: Encrypt та Decrypt.

Метод "Encrypt" шифрує заданий текст за допомогою алгоритму шифрування AES з використанням переданого ключа. Результатом роботи методу є масив байтів, що містить зашифрований текст та вектор ініціалізації.

У лістингу 3.1 приведено код реалізованого методу кодування інформації користувачів «Encrypt».

Лістинг 3.1 – Код методу «Encrypt»

```
public byte[] Encrypt(string plainText, byte[] key) { //Оголошення методу, який приймає рядок тексту для шифрування та байтовий ключ для шифрування і повертає масив байтів з зашифрованим текстом.

byte[] encryptedData; //Оголошення масиву байтів для збереження зашифрованого тексту.

using (Aes aesAlg = Aes.Create()) { //Створення об'єкту AES для шифрування тексту. Всі ресурси, пов'язані з цим об'єктом, будуть автоматично звільнені після виконання блоку коду.
```

```
            aesAlg.Key = key; //Встановлення ключа шифрування
            aesAlg.GenerateIV(); //Генерація вектору ініціалізації, необхідного для коректної роботи AES алгоритму.
            aesAlg.Padding = PaddingMode.PKCS7; // Встановлення режиму доповнення PKCS7 для обробки випадку, коли розмір шифруємого тексту не кратний розміру блоку шифрування.
            ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV); //Створення об'єкту шифрування, який буде використовувати раніше встановлені ключ та вектор ініціалізації
            using (MemoryStream msEncrypt = new MemoryStream()) { // Створення потоку пам'яті, що дозволить записати зашифрований текст
              using (CryptoStream csEncrypts = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write)) { //Створення криптографічного потоку, який буде записувати зашифрований текст у потік пам'яті
                using (StreamWriter swEncrypts = new StreamWriter(csEncrypts)) {
                  swEncrypts.Write(plainText); //Запис тексту, що підлягає шифруванню, до потоку пам'яті через криптографічний потік
                }
                encryptedData = msEncrypt.ToArray(); // Отримання масиву байтів з зашифрованим текстом з потоку пам'яті
              }
            }
            byte[] result = new byte[encryptedData.Length + 16]; //Створення масиву байтів, який міститиме зашифрований текст та вектор ініціалізації.
            Buffer.BlockCopy(encryptedData, 0, result, 16, encryptedData.Length); //Копіювання зашифрованого тексту у результуючий масив починаючи з індексу 16.
            Buffer.BlockCopy(aesAlg.IV, 0, result, 0, 16); //Копіювання вектора ініціалізації у результуючий масив.
            return result; //Повернення результату у вигляді масиву байтів, що містить зашифрований текст та вектор ініціалізації.
          }
        }
```

Отже, розроблений метод шифрує переданий рядок plainText за допомогою алгоритму AES (Advanced Encryption Standard) з використанням переданого ключа key. Результат шифрування повертається як масив байтів.

Алгоритм шифрування включає в себе створення об'єкта AES за допомогою методу Aes.Create(), встановлення ключа за допомогою aesAlg.Key = key і

генерацію випадкового ініціалізаційного вектора (IV) за допомогою методу aesAlg.GenerateIV().

Об'єкт ICryptoTransform encryptor створюється за допомогою методу aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV), який використовує ключ і IV для створення шифрувального об'єкту, що буде використовуватись для шифрування даних.

Далі, зашифровані дані записуються в MemoryStream msEncrypt за допомогою CryptoStream csEncrypts, що використовує об'єкт encryptor. Рядок plainText передається в StreamWriter swEncrypts за допомогою методу swEncrypts.Write(plainText).

Після закінчення шифрування, масив байтів шифрованих даних перетворюється на масив байтів encryptedData за допомогою msEncrypt.ToArray(). Далі, до цього масиву додається IV за допомогою Buffer.BlockCopy(aesAlg.IV, 0, result, 0, 16) і повертається як результат функції.

Метод " Decrypt" розшифровує масив байтів encryptedData за допомогою алгоритму AES (Advanced Encryption Standard) з використанням переданого ключа key. Результат розшифрування повертається як рядок.

У лістингу 3.2 приведено код реалізованого методу кодування інформації користувачів «Decrypt».

Лістинг 3.2 – Код методу «Decrypt»

public string Decrypt(byte[] encryptedData, byte[] key) { //метод отримує вхідні параметри: зашифровані дані у вигляді масиву байтів encryptedData та ключ шифрування у вигляді масиву байтів key.

string decryptedText; //оголошує змінну decryptedText, в якій зберігатиметься розшифрований текст.

using (Aes aesAlg = Aes.Create()) { //створення об'єкта AES за допомогою методу Aes.Create(). Об'єкт AES буде створено на початку блоку, а коли він закінчиться, автоматично буде викликаний метод Dispose() для звільнення ресурсів.

aesAlg.Key = key; //Задаємо ключ шифрування об'єкту AES.

aesAlg.IV = encryptedData.Take(16).ToArray(); //Встановлюємо ініціалізаційний вектор (IV) об'єкту AES з першими 16 байтами зашифрованих даних encryptedData. Метод

Take(16) повертає перші 16 байтів масиву encryptedData, а метод ToArray() перетворює результат у масив байтів

aesAlg.Padding = PaddingMode.PKCS7; //Встановлюємо режим доповнення (padding) для об'єкту AES. В даному випадку використовується стандартний режим PKCS7

ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV); //Створюємо об'єкт decryptor за допомогою методу CreateDecryptor, який використовує ключ і IV для створення об'єкту дешифрування.

using (MemoryStream msDecrypts = new MemoryStream(encryptedData.Skip(16).ToArray())) { //Створюємо новий об'єкт MemoryStream, який містить зашифровані дані, починаючи з 17-го байту та до кінця масиву. Метод Skip(16) пропускає перші 16 байтів масиву encryptedData, а метод ToArray() перетворює результат у масив байтів.

using (CryptoStream csDecrypt = new CryptoStream(msDecrypts, decryptor, CryptoStreamMode.Read)) { //Створюємо новий об'єкт CryptoStream за допомогою decryptor, який буде використовуватись для розшифрування даних з MemoryStream msDecrypts. Використовується режим CryptoStreamMode.Read, що означає, що дані будуть читатись з потоку.

using (StreamReader srDecrypts = new StreamReader(csDecrypt)) { //Створюємо новий об'єкт StreamReader, який буде використовуватись для зчитування розшифрованих даних з CryptoStream csDecrypt.

decryptedText = srDecrypts.ReadToEnd(); //Розшифровані дані зчитуються з StreamReader за допомогою методу ReadToEnd() і зберігаються у змінну decryptedText.

}

}

}

}

return decryptedText; //Повертаємо розшифрований текст.

}.

Отже, даний код розшифровує зашифрований масив байтів encryptedData за допомогою алгоритму AES (Advanced Encryption Standard) з використанням переданого ключа key. Результат розшифрування повертається як рядок.

Алгоритм розшифрування включає встановлення ключа за допомогою aesAlg.Key = key, встановлення IV (ініціалізаційного вектора) за допомогою методу encryptedData.Take(16).ToArray(). Вектор ініціалізації в цій функції береться з початку переданого масиву байтів.

Об'єкт ICryptoTransform decryptor створюється за допомогою методу aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV), який використовує ключ і IV для

створення розшифровального об'єкту, що буде використовуватись для розшифрування даних.

Далі, зашифровані дані читаються з MemoryStream msDecrypts за допомогою CryptoStream csDecrypt, що використовує об'єкт decryptor. Розшифровані дані зчитуються в StreamReader srDecrypts за допомогою методу srDecrypts.ReadToEnd().

Після закінчення розшифрування, розшифрований текст зберігається в змінній decryptedText і повертається як результат функції.

3.2. Розробка програмних модулів системи

Після створення бази даних, можна легко підключити її до системи, використовуючи Visual Studio 2019. Це дозволить нам підключати всі таблиці до форм та створювати можливість додавання, редагування та видалення даних. Щоб здійснити підключення до бази даних, було створено змінну "CONNECT" у файлі конфігурації проекту "App.config" та задано значення параметрів, які можна побачити на рис. 3.1.

```
<appSettings>
  <!-- Підключення до бази даних -->
  <add key="CONNECT" value="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=|DataDirectory|\DataBase.mdb;"/>
</appSettings>
```

Рисунок 3.1 – Змінна із параметрами налаштування бази даних

Для ефективної роботи з базою даних у проекті використовувався простір імен System.Data.OleDb, який містить класи для зв'язку з базою даних MS Access. Ці класи дозволяють взаємодіяти з даними у базі даних, виконувати запити та зберігати зміни. Для створення меню в проекті, був доданий елемент menuStrip до головного вікна. Це дає можливість користувачам вибирати опції та викликати функції програми за допомогою простого інтерфейсу. Результат можна побачити на рис. 3.2.

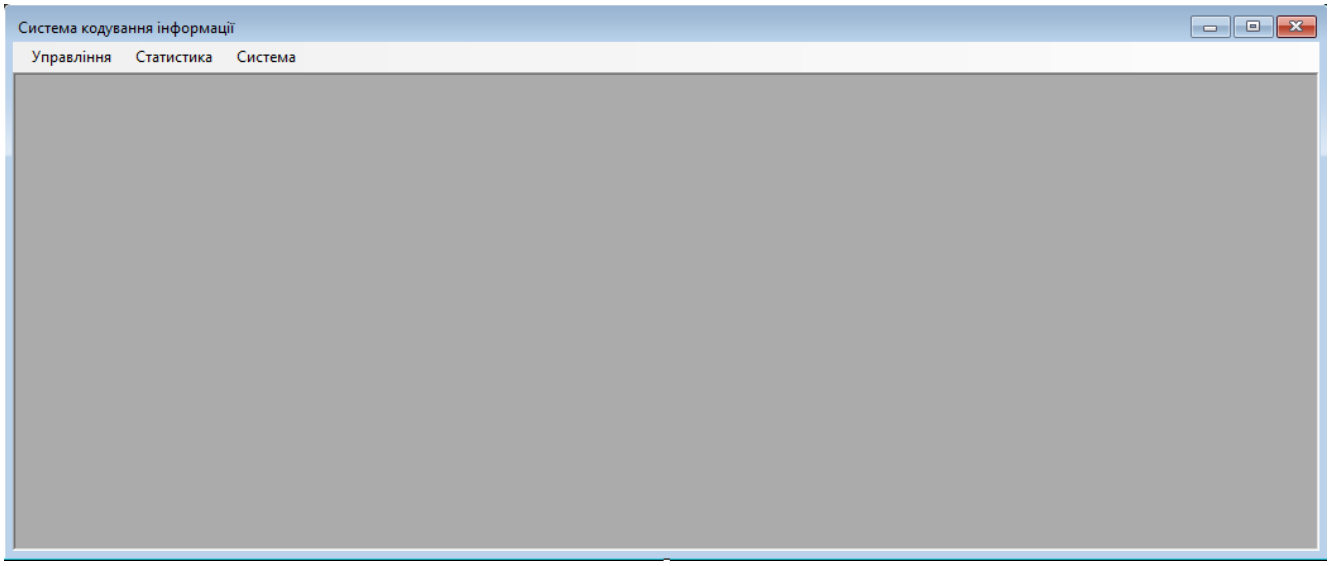


Рисунок 3.2 – Додавання головного меню

Для кожного пункту меню в проекті був доданий відповідний код, який створює екземпляр відповідної форми. При виборі користувачем певного пункту меню, відповідна форма відкривається, і її вікно стає активним. Для того, щоб забезпечити коректне відображення вікон та уникнути можливих конфліктів, було включено код, який забезпечує закриття попередньо відкритого вікна перед відкриттям нового. Аналогічний код застосовується для всіх пунктів меню, що забезпечує їх правильну роботу та взаємозв'язок з формами. На рис. 3.3 наведено приклад коду для одного з пунктів меню.

```
1 reference
private void кодуванняТекстуToolStripMenuItem_Click(object sender, EventArgs e) {
    CloseAllWindows();
    CodingForm codingForm = new CodingForm();
    codingForm.MdiParent = this;
    codingForm.WindowState = FormWindowState.Maximized;
    codingForm.Show();
}
```

Рисунок 3.3 – Код пунктів меню

Після цього було розроблено класи для роботи із базою даних. Нижче приведено часткові реалізації методів різних класів із детальним описом. Наприклад, для додавання інформації про закодований текст у базу даних був створений метод із назвою «InsertCodings», код даного методу показано на рис. 3.4.

```
1 reference
public void InsertCodings(string CodingsName, int UsersId, byte[] GenerateKey, byte[] CodingText) {
  string SqlString = "INSERT INTO Codings (" +
    "CodingsName, UsersId, GenerateKey, CodingText) Values(?, ?, ?, ?)";
  using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
    using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
      cmd.CommandType = CommandType.Text;
      cmd.Parameters.AddWithValue("CodingsName", CodingsName);
      cmd.Parameters.AddWithValue("UsersId", UsersId);
      if (GenerateKey == null) {
        cmd.Parameters.AddWithValue("GenerateKey", Encoding.Default.GetBytes(""));
      } else {
        cmd.Parameters.AddWithValue("GenerateKey", GenerateKey);
      }
      if (CodingText == null) {
        cmd.Parameters.AddWithValue("CodingText", Encoding.Default.GetBytes(""));
      } else {
        cmd.Parameters.AddWithValue("CodingText", CodingText);
      }
      conn.Open();
      cmd.ExecuteNonQuery();
      conn.Close();
    }
  }
}
```

Рисунок 3.4– Код методу «InsertCodings»

Даний метод InsertCodings призначений для вставки даних у таблицю Codings бази даних. Метод отримує на вхід назву кодування (CodingsName), ідентифікатор користувача (UsersId), ключ генерації (GenerateKey) та текст, який було закодовано (CodingText).

У першому рядку методу створюється SQL запит INSERT INTO, який містить назву таблиці, назви полів, куди будуть вставлені значення, та символи запитування, які вказують на місця, де будуть вставлені значення відповідних параметрів.

Далі, використовуючи клас OleDbConnection, створюється з'єднання з базою даних. Параметри для запиту передаються в об'єкт команди OleDbCommand, в якому вказується рядок запиту, тип команди та параметри команди. Параметри передаються за допомогою методу AddWithValue, який дозволяє вказати назву параметра та його значення.

Якщо ключ генерації (GenerateKey) або текст кодування (CodingText) дорівнюють null, то вони замінюються на пустий масив байтів.

Після встановлення параметрів, з'єднання з базою даних відкривається, та виконується команда запиту INSERT INTO за допомогою методу ExecuteNonQuery(). Після виконання команди, з'єднання з базою даних закривається за допомогою методу Close().

Для вибірки поверхневої інформації всіх записів із таблиці «Codings», також був розроблений метод, код якого представлено на рис. 3.5.

```
1 reference
public List<Codings> GetAllCodings() {
  int i = 0;
  string SqlString = "SELECT CodingsId, CodingsName FROM Codings WHERE UsersId=" +
    LoginForm.CurrentUser.UsersId +  " ORDER BY CodingsName ASC";
  List<Codings> listAllCodings = new List<Codings>();
  using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
    using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
      conn.Open();
      using (OleDbDataReader reader = cmd.ExecuteReader()) {
        while (reader.Read()) {
          Codings oneCodings = new Codings();
          oneCodings.Number = ++i;
          oneCodings.CodingsId = Convert.ToInt32(reader["CodingsId"]);
          oneCodings.CodingsName = reader["CodingsName"].ToString();
          listAllCodings.Add(oneCodings);
        }
      }
      conn.Close();
    }
  }
  if (listAllCodings.Count == 0) {
    Codings noCodings = new Codings();
    noCodings.CodingsId = 0;
    noCodings.Message = NamesMy.NoDataNames.NoDataInCodings;
    listAllCodings.Add(noCodings);
  }
  return listAllCodings;
}
```

Рисунок 3.5– Код методу «GetAllCodings»

Метод GetAllCodings призначений для отримання списку всіх записів з таблиці Codings бази даних, які належать поточному користувачу. Метод повертає список об'єктів типу Codings, які містять інформацію про кожен запис.

У першому рядку методу створюється SQL запит SELECT для отримання даних з таблиці Codings, в якому вказується назва стовпців (CodingsId, CodingsName), умова WHERE для фільтрації записів за значенням UsersId поточного користувача та вказується порядок сортування за полем CodingsName.

Далі, створюється новий список об'єктів типу Codings, який буде містити дані з бази даних. Використовуючи клас OleDbConnection, створюється з'єднання з базою даних.

Параметри для запиту передаються в об'єкт команди OleDbCommand, в якому вказується рядок запиту та з'єднання з базою даних. З'єднання з базою даних відкривається за допомогою методу Open(), та виконується команда запиту SELECT за допомогою методу ExecuteReader().

Далі, використовуючи клас OleDbDataReader, дані, які були отримані за допомогою запиту SELECT, зчитуються по одному рядку. Для кожного рядка, створюється новий об'єкт типу Codings, в який записуються відповідні дані (номер запису, ідентифікатор запису, назва запису). Отриманий об'єкт додається до списку listAllCodings.

Після завершення зчитування даних з бази даних, з'єднання з базою даних закривається за допомогою методу Close(). Якщо список listAllCodings порожній, то до нього додається один запис, який містить інформацію про відсутність даних.

Як результат, метод повертає список об'єктів типу Codings, який містить всі записи з таблиці Codings, що належать поточному користувачу.

Після розробки всіх класів рівня даних було розроблено форми для взаємодії користувача із програмою. Наприклад, розроблена форма для зашифровування текстової інформації користувача, яка зображена на рис. 3.6.

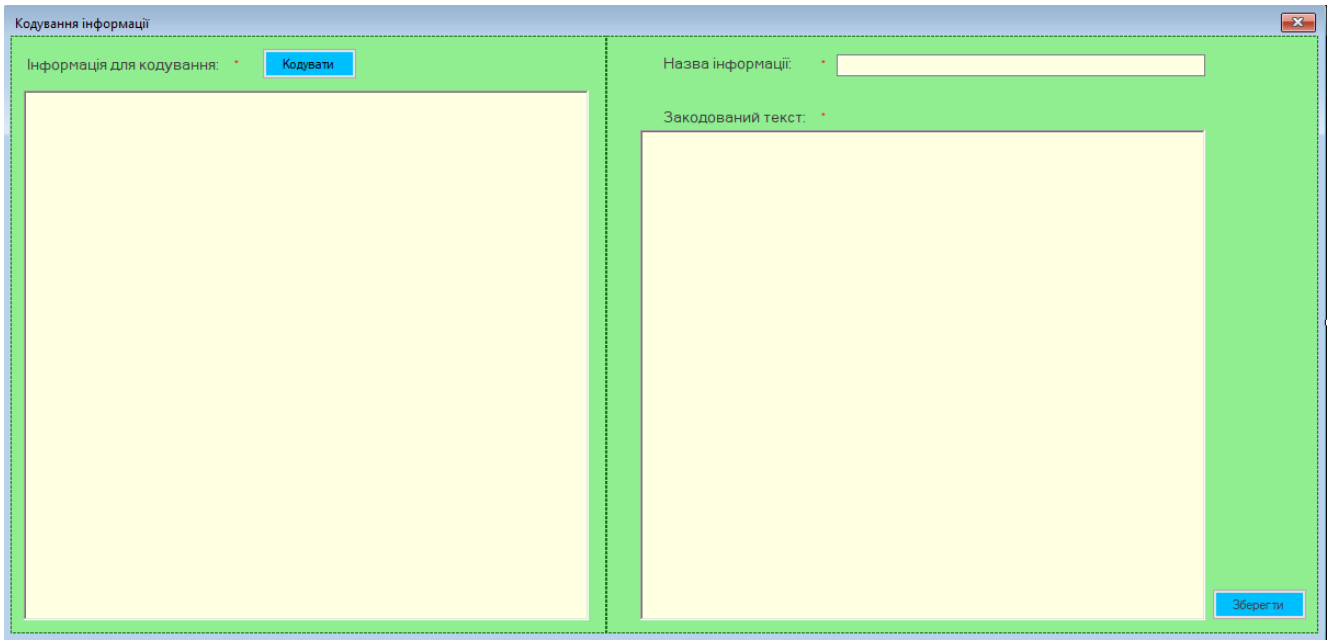


Рисунок 3.6– Розробка форми для кодування інформації та збереження даних

Для кодування інформації було додано елемент «Button» та написано код для обробника події кнопки «Кодувати». Код методу даної події зображено на рис. 3.7.

```
1 reference
private void CodingBtn_Click(object sender, EventArgs e) {
  if (IsTextCodingCorrect()) {
    RandomNumberGenerator rng = RandomNumberGenerator.Create();
    rng.GetBytes(_Key);
    _EncryptedData = _InfoCrypter.Encrypt(TextForCodingRTBox.Text, _Key);
    CodingTextRTBox.Text = Encoding.UTF8.GetString(_EncryptedData);
  }
}
```

Рисунок 3.7– Код методу події «CodingBtn_Click»

Метод CodingBtn_Click призначений для шифрування введеного користувачем тексту та його відображення у вікні програми у вигляді зашифрованого тексту. На початку методу перевіряється коректність введеного користувачем тексту за допомогою методу IsTextCodingCorrect(). Якщо введений текст коректний, створюється новий екземпляр генератора випадкових чисел RandomNumberGenerator, за допомогою якого генерується ключ шифрування _Key.

Далі, за допомогою об'єкта _InfoCrypter, який містить методи для шифрування та розшифрування даних, викликається метод Encrypt(), який приймає текст для шифрування та ключ шифрування _Key. Результат шифрування зберігається у змінну _EncryptedData.

Результатом роботи є зашифрований текст, що змінює текстове поле CodingTextRTBox, де він виводиться за допомогою методу Encoding.UTF8.GetString(). Це дозволяє користувачеві побачити результат шифрування у вигляді тексту, який можна зберегти у базу даних. Також, можна почати ефективність роботи методу кодування, а саме на скільки меншим в об’ємі став текст.

Також у будь-якій розробленій формі всі введені дані перевіряються на коректність. Для кожної форми написано свій метод «IsDataEnteringCorrect», що перевіряє дані всіх обов’язкових полів. Метод для перевірки даних на коректність форми «CodingsForm» при додаванні інформації у базу даних зображений на рис. 3.8.

```
1 reference
private bool IsDataEnteringCorrect() {
    bool isCorrect = true;
    if (_validation.IsDataEntering(CodingsNameTBox.Text)) {
        CodingsNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
        CodingsNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
    }
    if (_validation.IsDataEntering(CodingTextRTBox.Text)) {
        CodingTextValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
        CodingTextValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
    }
    return isCorrect;
}
```

Рисунок 3.8– Код методу для перевірки коректності введених даних

Метод IsDataEnteringCorrect() призначений для перевірки коректності введення даних користувачем перед збереженням запису до бази даних. На початку виконання методу створюється змінна isCorrect, яка за замовчуванням

встановлюється у true. Це означає, що при початковому виклику методу передбачається, що дані введені коректно.

Далі, за допомогою об'єкта _validation, який містить методи для перевірки коректності введення даних, виконується перевірка текстових полів CodingsNameTBox та CodingTextRTBox. Якщо одне з полів містить некоректні дані, текстова мітка CodingsNameValiadtionLbl чи CodingTextValiadtionLbl відображає текст з повідомленням про помилку.

Якщо під час перевірки була знайдена помилка, змінна isCorrect встановлюється у false. Якщо дані введені коректно, змінна isCorrect залишається true. У будь-якому випадку, метод повертає змінну isCorrect, що дає можливість використовувати її для подальшої перевірки коректності введення даних.

Для збереження даних був написаний код обробника події із назвою «SaveBtn_Click» (рис. 3.9).

```
1 reference
private void SaveBtn_Click(object sender, EventArgs e) {
    if (IsDataEnteringCorrect()) {
        _CodingsProvider.InsertCodings(CodingsNameTBox.Text, LoginForm.CurrentUser.UsersId, _Key, _EncryptedData);
        _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId, "Був закодований текст із назвою "
            + CodingsNameTBox.Text, DateTime.Now);
        _HistorysProvider.InsertHistorys(DateTime.Now, CodingsNameTBox.Text);
        ClearAllControls();
        MessageBox.Show("Текст був успішно закодований!", "Вітаю");
    }
}
```

Рисунок 3.9– Код методу обробника події «SaveBtn_Click»

Метод SaveBtn_Click() викликається при натисканні кнопки "Зберегти" на формі. Його головною функцією є збереження зашифрованих даних до бази даних.

На початку методу викликається метод IsDataEnteringCorrect(), який перевіряє коректність введення даних користувачем. Якщо дані введені коректно, виконується наступний код. Якщо дані введені некоректно, метод завершується, не зберігаючи дані. Якщо дані введено коректно викликається метод InsertCodings() з об'єктом _CodingsProvider, який містить методи для роботи з

таблицею Codings у базі даних. Цей метод виконує команду INSERT, додаючи в таблицю Codings новий запис зі значеннями, введеними користувачем у поля CodingsNameTBox та CodingTextRTBox, а також іншими параметрами. При цьому, у таблицю зберігаються згенерований ключ _Key та зашифрований текст _EncryptedData.

Далі викликаються методи InsertLogs() та InsertHistorys() з об'єктами _LogsProvider та _HistorysProvider відповідно. Ці методи призначені для збереження історії дій користувачів у базі даних.

Після цього викликається метод ClearAllControls(), який очищує значення введених даних зі всіх елементів керування на формі. На завершення відображається вікно повідомлення з підтвердженням успішного збереження даних.

Отже, в даному підрозділі було проведено частковий опис реалізованої системи.

3.3. Результати функціонального тестування розробленого додатку

Тестування є надзвичайно важливим етапом у розробці будь-якого програмного забезпечення. Воно дозволяє перевірити роботу системи та забезпечити її відповідність вимогам та очікуванням користувачів. Процес тестування включає в себе запуск програмного забезпечення, аналіз результатів його роботи та виявлення та виправлення помилок. Це дозволяє підвищити якість програмного забезпечення та забезпечити його надійну та коректну роботу.

У процесі тестування перевіряється різноманітна функціональність програмного забезпечення, така як продуктивність, безпека, надійність та інші характеристики. Це дозволяє виявляти недоліки та помилки в роботі системи та забезпечує їх виправлення. Тестування допомагає забезпечити якість роботи системи та покращити її функціональність. Тому, процес тестування є

невід'ємною частиною розробки програмного забезпечення та є ключовим етапом у забезпеченні успішного запуску та роботи системи.

Після завершення розробки додатку було проведено функціональне тестування, яке включало в себе перевірку основних функцій додатку. Метою тестування було визначення рівня стабільності та надійності додатку, а також його здатності задовольнити вимоги користувачів. Для цього було складено тест-кейси, що відповідають функціональним вимогам додатку.

Тест-кейс №1. Вхід до системи. Перевірка на наявність правильної авторизації користувача:

– відкрити форму авторизації;
– ввести коректний логін та пароль користувача;
– натиснути кнопку "Увійти";
– перевірити, що користувач має доступ до головного меню системи;
– вийти з системи, переконатися, що користувач більше не має доступу до системи.

В результаті, користувач має бути успішно авторизований в системі, мати доступ до головного меню та не мати доступу до системи після виходу з неї.

Тест-кейс №2. Введення, збереження та виведення інформації із бази даних. Перевірка методу кодування даних:

відкрити форму з налаштуваннями захисту даних;

– здійснити копіювання текстової інформації із буферу;
– натиснути кнопку "Кодувати";
– ввести назву інформації;
– натиснути кнопку "Зберегти";
– відкрити файл бази даних;
– знайти збережений запис та перевірити, що дані знаходяться в закодованому вигляді;

– відкрити програму та ввести коректні дані авторизації;

– знайти збережений запис та перевірити, що дані відображаються в розкодованому вигляді.

В результаті, метод кодування даних повинен працювати правильно та забезпечувати надійний захист збереженої інформації.

Після проведення функціонального тестування системи необхідно провести модульне тестування, яке є ключовим етапом розробки програмного забезпечення. Воно дозволяє перевірити окремі частини програми (модулі) на відповідність вимогам та специфікації. Для проведення модульного тестування можна створити новий проект типу «Unit Test Project» у середовищі розробки та визначити тести для функцій програми.

За допомогою модульних тестів можна отримати результати тестування та перевірити, чи працюють окремі модулі програми правильно. Це дозволяє виявити й виправити помилки, які можуть збільшити кількість проблем у функціонуванні системи. Результати модульного тестування можна також використовувати як додаткову документацію для програми, що допомагає зрозуміти, як вона працює.

На рис. 3.10 показано результат проведеного модульного тестування додатку.

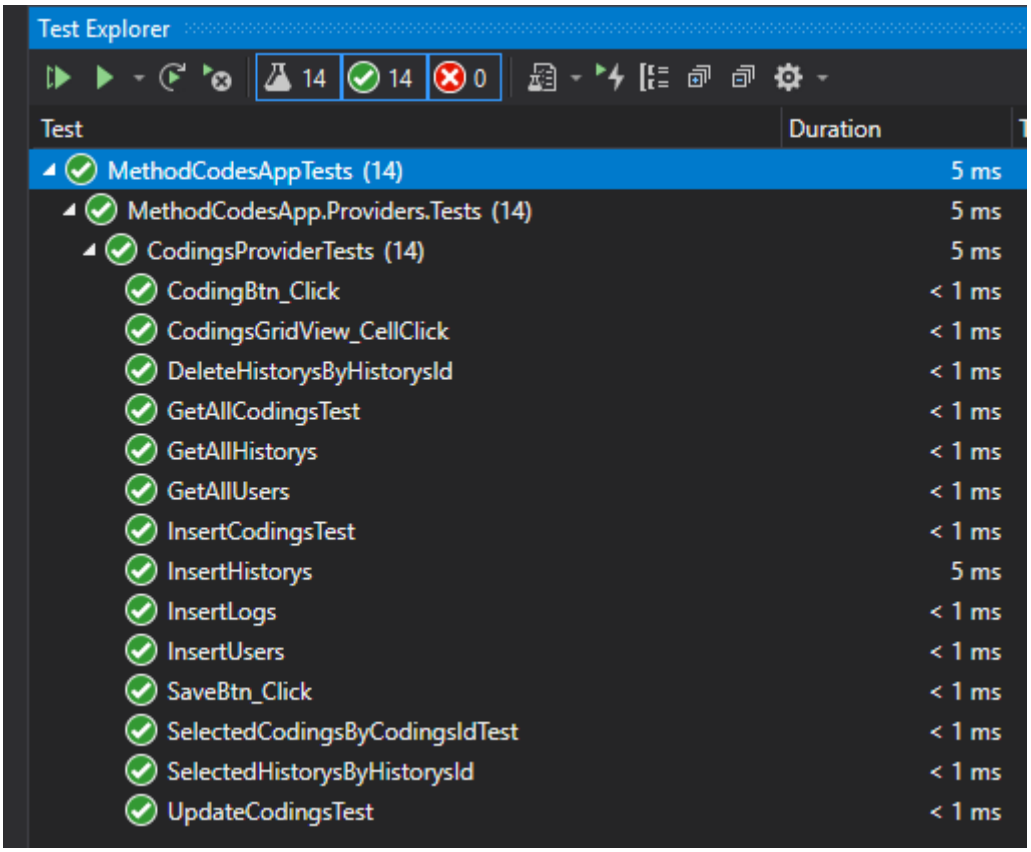


Рисунок 3.10 – Результати проведення модульного тестування

3.4. Інструкція користувачеві програми

3.4.1. Мінімальні вимоги для запуску ПЗ

Під час розробки програмного забезпечення, важливим фактором, який необхідно врахувати, є потреба в належних ресурсах для запуску та функціонування програми. Мінімальні вимоги до апаратних та програмних засобів повинні відповідати розміру та складності програми, а також додатковим вимогам до функцій, які розробляються.

Для ефективної роботи розробленого програмного забезпечення, необхідно мати належний обсяг оперативної пам'яті, вільного місця на жорсткому диску, достатньо потужний процесор, а також наявність необхідних вхідних пристроїв, таких як миша, клавіатура та монітор.

Отже, мінімальні вимоги до апаратних засобів для запуску та функціонування ПЗ включають:

– процесор: Intel Core i5 або AMD Ryzen 5;
– оперативна пам'ять: 8 Гб;
– жорсткий диск: 128 Гб або SSD на 128 Гб;
– монітор: 21-24 дюйма з роздільною здатністю Full HD (1920 x 1080);
– операційна система: Windows 10 або macOS Catalina.

3.4.2. Опис процедури розгортання програмного продукту

Запуск програми в операційній системі сімейства Windows можна здійснити декількома стандартними способами. Найпоширенішими з них є:

– подвійне клацання лівою кнопкою миші на ярлику програми. Якщо ярлик програми на робочому столі, то потрібно знайти його та двічі клацнути лівою кнопкою миші.

– виклик контекстного меню з вибором пункту "Відкрити". Для цього потрібно зайти у папку, де знаходиться ярлик програми, та натиснути праву кнопку миші на ярлику. У випадаючому меню необхідно обрати пункт "Відкрити".

– натискання кнопки "Пуск" на панелі завдань, після чого необхідно вибрати пункт "Усі програми" та подвійно клацнути лівою кнопкою миші на ярлику програми, яку потрібно запустити.

3.4.3. Використання програмного продукту

Перед тим, як розпочати роботу з додатком "Притулок тварин", необхідно запустити його. Для цього необхідно відкрити додаток. Після цього на екрані з'явиться вікно авторизації, в якому користувач повинен ввести свій логін та пароль (рис. 3.11). Це дозволяє ідентифікувати користувача та забезпечити безпеку його персональних даних.

Після введення логіну та пароля користувачу буде надано доступ до головного інтерфейсу додатку, де він тестувати роботу методу кодування інформації. Крім того, в головному меню додатку користувач зможе знайти додаткові опції, такі як збереження закодованої інформації, завантаження інформації із бази даних, формування звітності та перегляд системних подій.

Таким чином, для початку роботи з додатком необхідно пройти процес авторизації та отримати доступ до головного інтерфейсу додатку, щоб мати можливість працювати зі всіма його функціями.

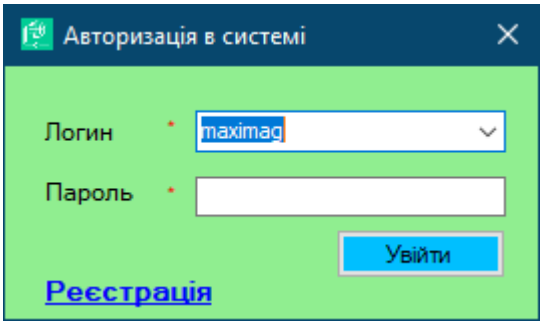


Рисунок 3.11 – Автентифікація користувача в системі

У додатку «Система кодування інформації» користувач може зареєструватися, якщо він не має облікового запису. Для цього користувач повинен натиснути кнопку "Реєстрація", яка знаходиться на екрані авторизації. Після цього користувач буде перенаправлений на вікно реєстрації, яке містить форму для введення необхідної інформації. У формі реєстрації користувач повинен ввести своє ім'я, прізвище, електронну адресу та пароль. Після заповнення всіх полів користувач повинен натиснути кнопку "Зареєструватися". Після успішної реєстрації користувач може використовувати свій новий обліковий запис для входу в систему.

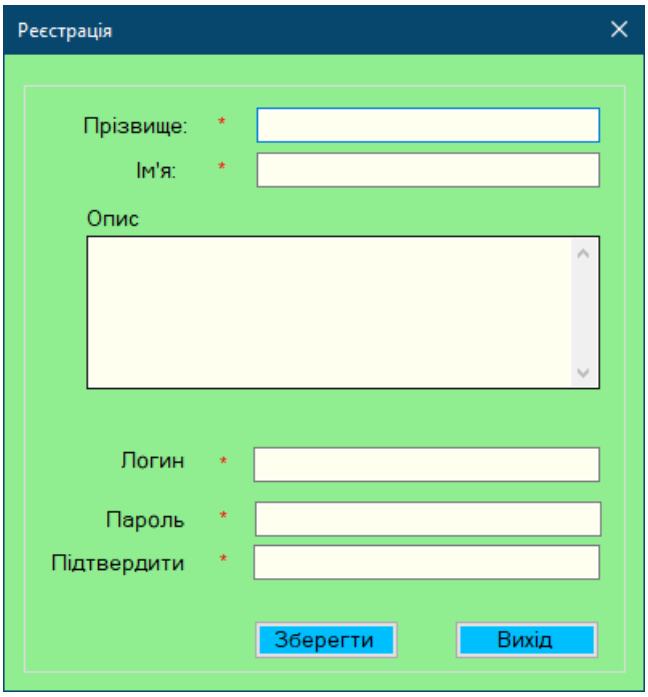


Рисунок 3.12 – Форма реєстрації користувачів

Якщо у програмі ще не зареєстровано жодного користувача, то при реєстрації першому користувачу автоматично буде присвоєна роль системного адміністратора, іншим всім користувачам буде присвоєно ролі звичайних користувачів.

Для швидкого пошуку імені в програмі можна скористатись випадаючим списком, який містить доступні імена користувачів. Крім того, при введенні імені з клавіатури програма автоматично поставить його у верхню частину списку, що спрощує інтерфейс і дозволяє швидше знайти потрібне ім'я.

Після успішної автентифікації користувача системи буде відкрите головне вікно програми (рис.3.13). Це вікно містить основне меню, що складається з доступних опцій та функцій програми. Наприклад, користувач може відкривати різні форми програми, переглядати закодовану інформацію, проводити кодування інформації та ін. Головне вікно програми є центром керування всіма функціями додатку та дозволяє користувачам легко взаємодіяти з ним.

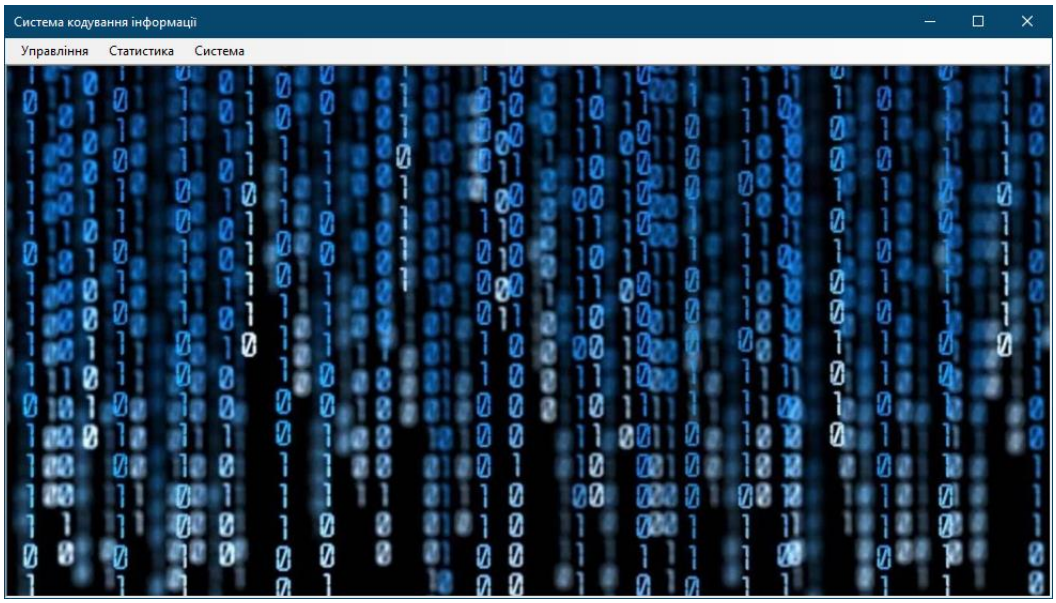


Рисунок 3.13 – Головне меню програми

У системі реалізовано дві ролі: системного адміністратора та звичайного користувача. Роль системного адміністратора дозволяє керувати: обліковими записами та переглядати події системи.

Для перевірки методу кодування інформації, користувачу системи необхідно перейти по меню програми «Управління» –> «Кодування», після чого відкриється вікно, що представлено на рис. 3.14.

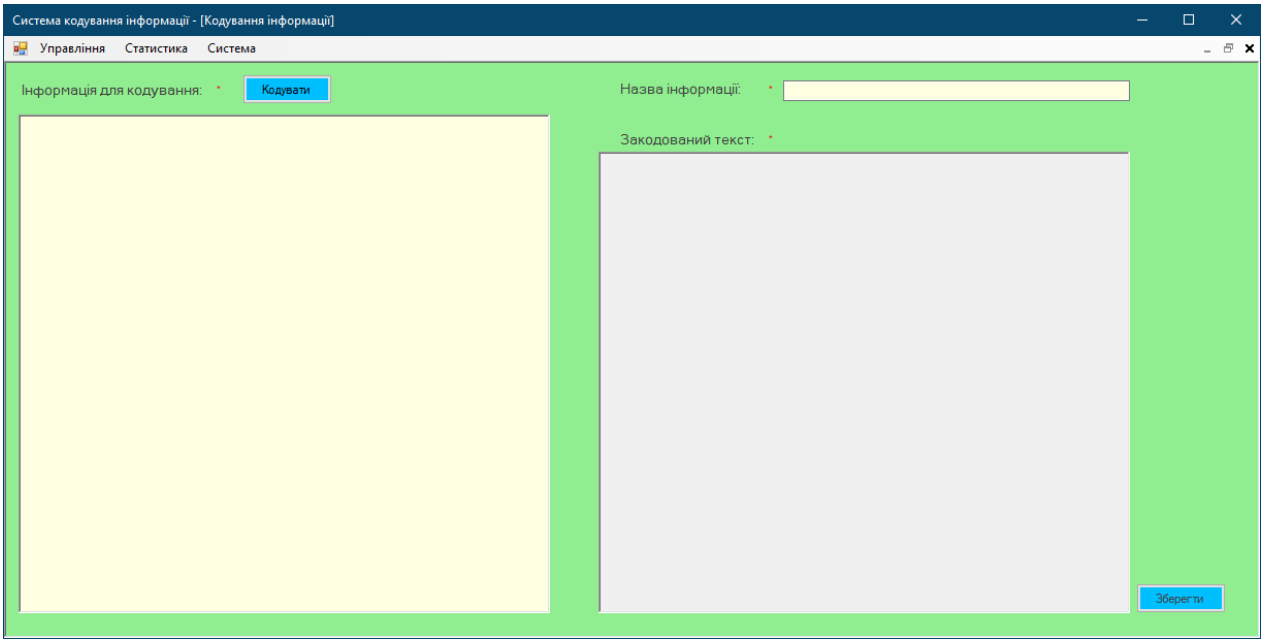


Рисунок 3.14 – Вікно для проведення кодування інформації

У поле «Інформація для кодування» можна скопіювати або ввести інформацію, після чого натиснути кнопку «Кодувати». Програма за допомогою методу розробленого методу проведе кодування введених даних та відобразить їх у правій частині екрану (рис. 3.15).

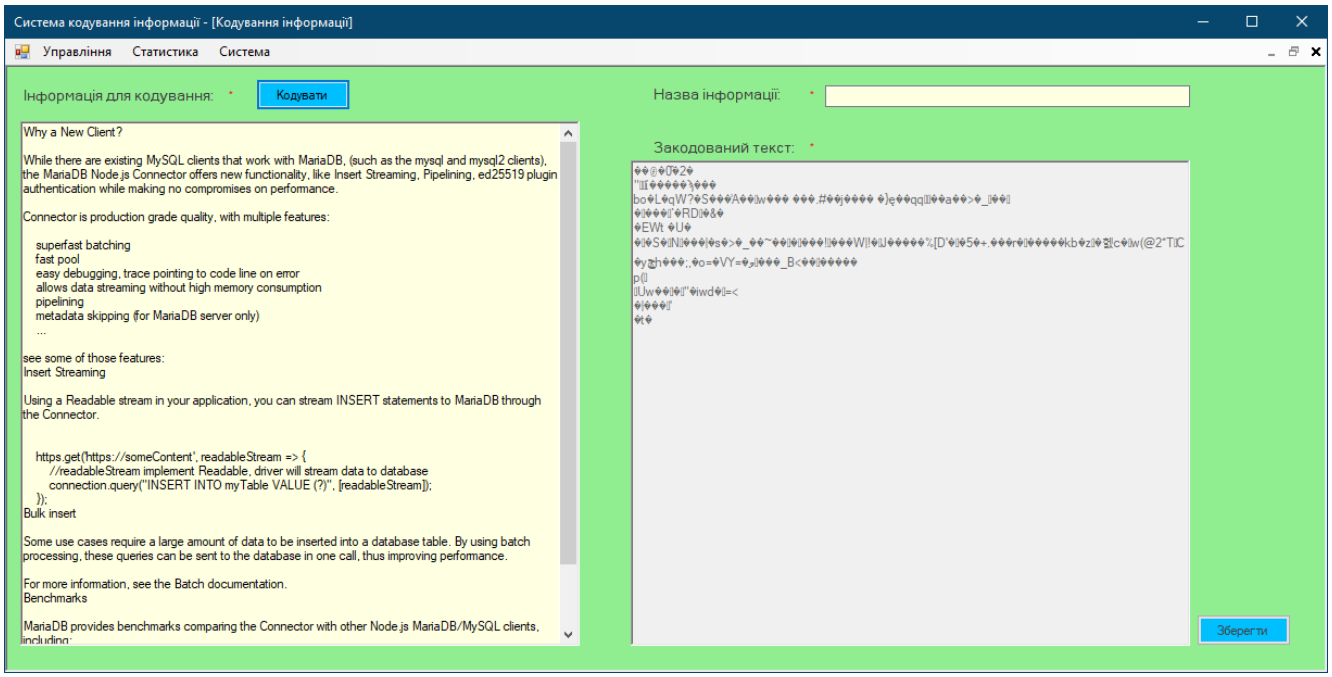


Рисунок 3.15 – Кодування доданої інформації

Після чого введену інформацію також можна зберегти у базі даних, для цього необхідно вказати назву, що буде зберігати у спеціальне поле «Назва інформації» та натиснути кнопку «Зберегти». Якщо дані були вдало збережені, програма виведе відповідне повідомлення (рис. 3.16). Як можна побачити із рис. 3.15 розроблений метод непогано стискає дані, а також проводить їх шифрування.

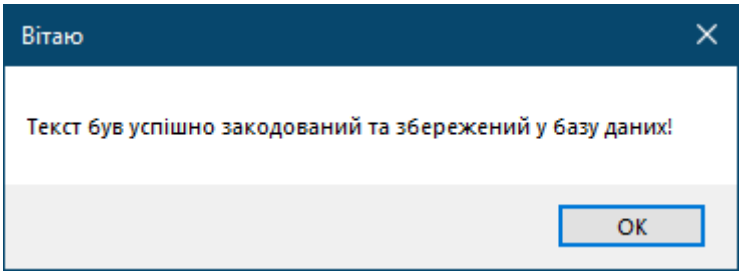


Рисунок 3.16 – Повідомлення про успішне кодування та збереження даних

Для завантаження збереженої інформації із бази даних, користувачу системи необхідно перейти по меню програми «Управління» –> «Декодування», після чого відкриється вікно, що представлено на рис. 3.17.

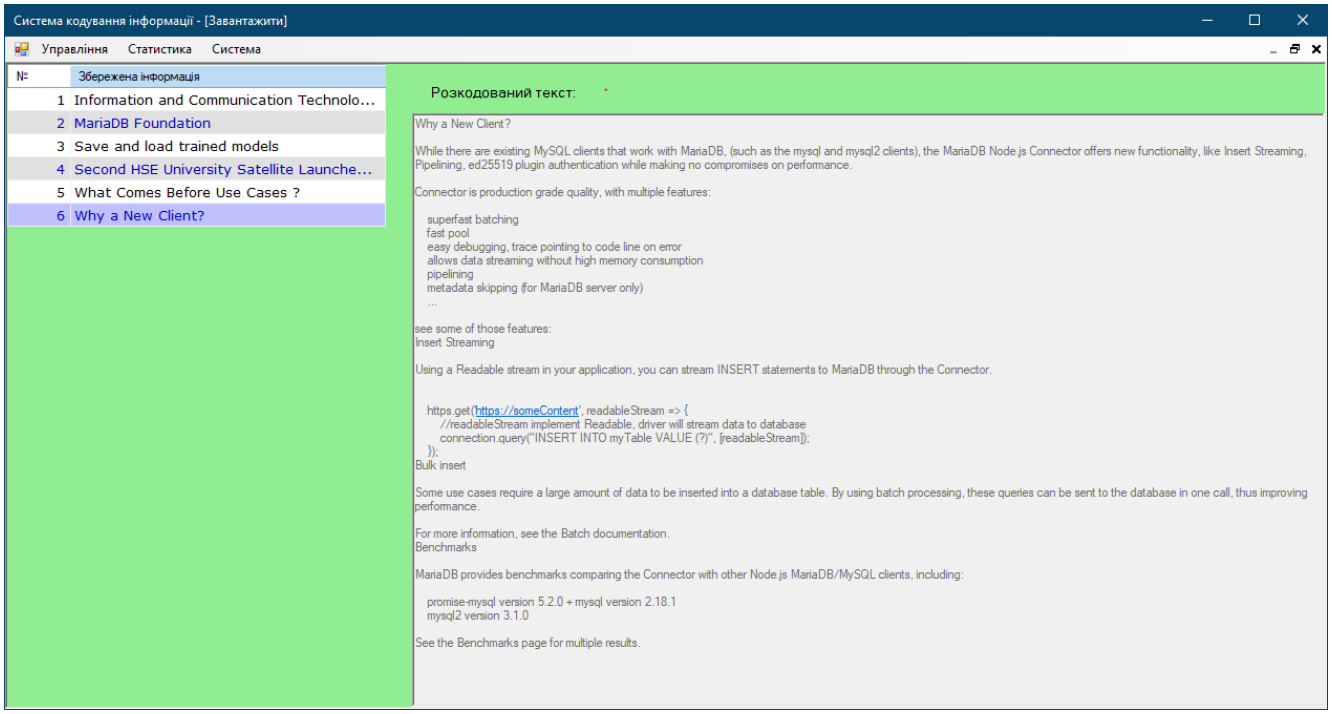


Рисунок 3.17 – Вікно для завантаження та розкодування інформації

Вся інформація, що була введена користувачем зберігається у базі даних у зашифрованому вигляді, за допомогою розробленого методу та розшифровується тільки для вибраного у лівій частині екрану збереженого запису.

У програмі було реалізовано можливість опрацювання інформації облікових записів користувачів. Ця функція дозволяє користувачам створювати свої облікові записи та зберігати в них свої особисті дані. Крім того, користувачі можуть здійснювати вхід у свій обліковий запис та виконувати різні дії, пов'язані з їх профілем.

При створенні облікового запису користувач має можливість ввести свої особисті дані, такі як ім'я, прізвище, логін, пароль, тощо. Ці дані зберігаються в базі даних програми та використовуються для забезпечення безпеки та конфіденційності інформації користувачів (рис. 3.18).

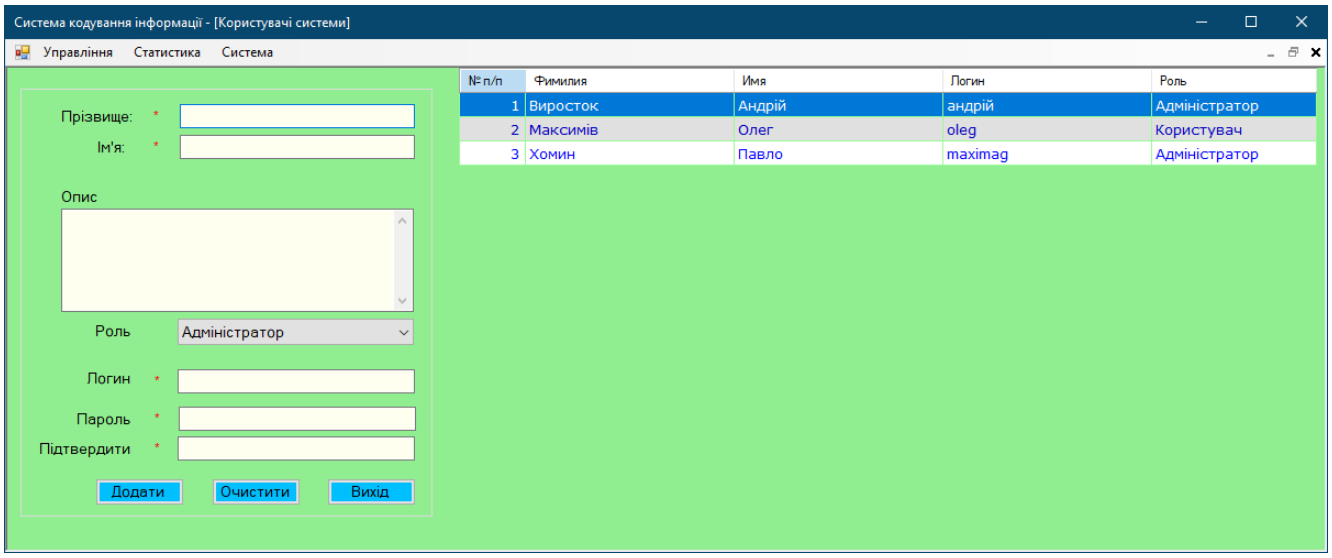


Рисунок 3.18 – Вікно для опрацювання інформації облікових записів користувачів

Крім того, користувачі можуть виконувати різні дії в своєму обліковому записі, для цього необхідно використати пункт меню «Персоналізація» (рис. 3.19).

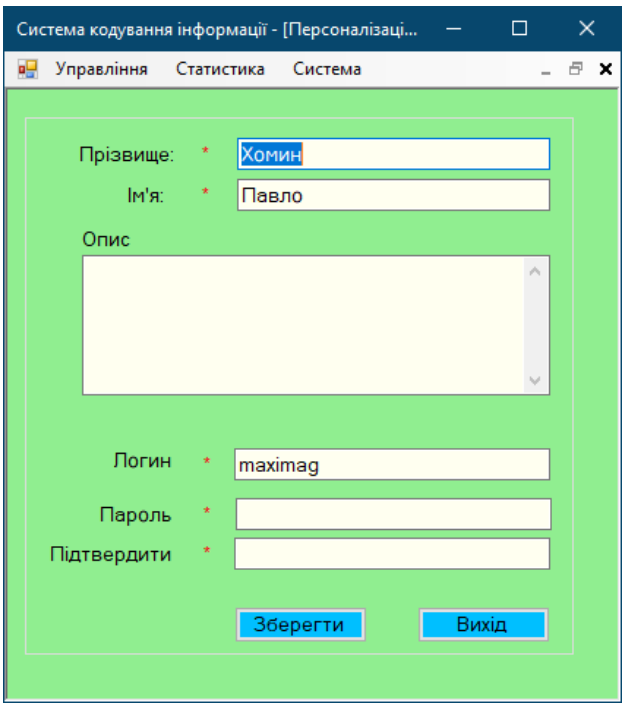
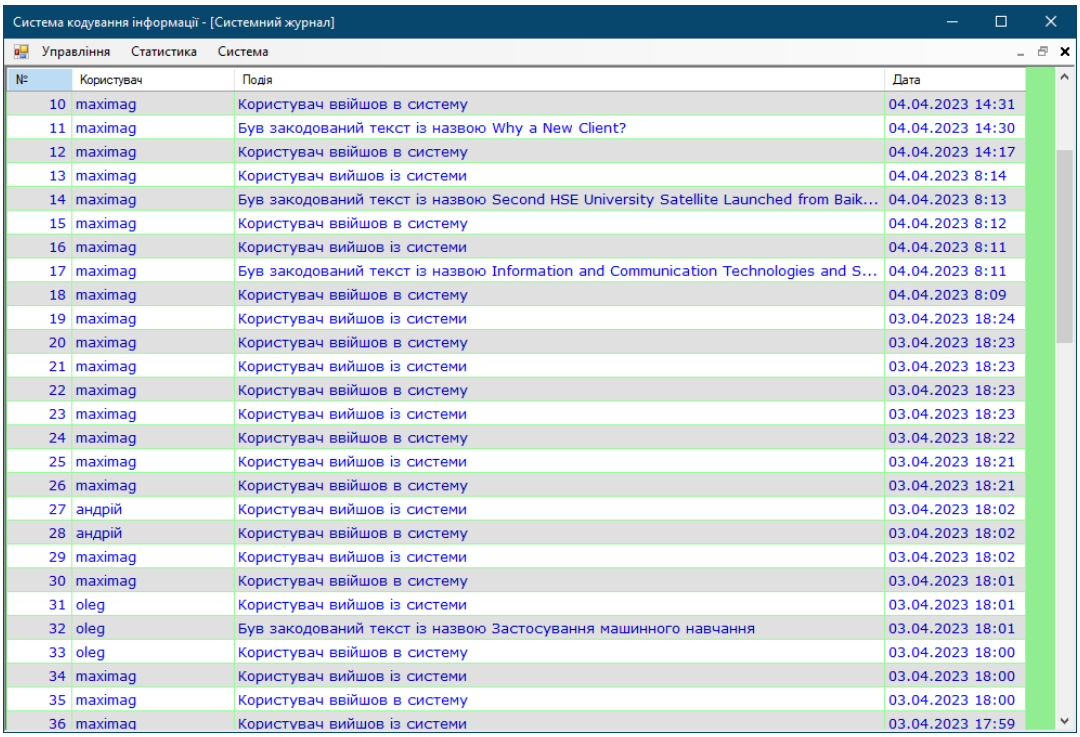


Рисунок 3.19 – Зміна даних облікового запису

В системі також можна відстежувати активність користувачів та їхні дії завдяки обліковому запису з правами адміністратора. Це можливо зробити за допомогою функції "Події ", яка знаходиться в меню "Система" (рис. 3.20).



№	Користувач	Подія	Дата
10	maximag	Користувач ввійшов в систему	04.04.2023 14:31
11	maximag	Був закодований текст із назвою Why a New Client?	04.04.2023 14:30
12	maximag	Користувач ввійшов в систему	04.04.2023 14:17
13	maximag	Користувач вийшов із системи	04.04.2023 8:14
14	maximag	Був закодований текст із назвою Second HSE University Satellite Launched from Baik...	04.04.2023 8:13
15	maximag	Користувач ввійшов в систему	04.04.2023 8:12
16	maximag	Користувач вийшов із системи	04.04.2023 8:11
17	maximag	Був закодований текст із назвою Information and Communication Technologies and S...	04.04.2023 8:11
18	maximag	Користувач ввійшов в систему	04.04.2023 8:09
19	maximag	Користувач вийшов із системи	03.04.2023 18:24
20	maximag	Користувач ввійшов в систему	03.04.2023 18:23
21	maximag	Користувач вийшов із системи	03.04.2023 18:23
22	maximag	Користувач ввійшов в систему	03.04.2023 18:23
23	maximag	Користувач вийшов із системи	03.04.2023 18:23
24	maximag	Користувач ввійшов в систему	03.04.2023 18:22
25	maximag	Користувач вийшов із системи	03.04.2023 18:21
26	maximag	Користувач ввійшов в систему	03.04.2023 18:21
27	андрій	Користувач вийшов із системи	03.04.2023 18:02
28	андрій	Користувач ввійшов в систему	03.04.2023 18:02
29	maximag	Користувач вийшов із системи	03.04.2023 18:02
30	maximag	Користувач ввійшов в систему	03.04.2023 18:01
31	oleg	Користувач вийшов із системи	03.04.2023 18:01
32	oleg	Був закодований текст із назвою Застосування машинного навчання	03.04.2023 18:01
33	oleg	Користувач ввійшов в систему	03.04.2023 18:00
34	maximag	Користувач вийшов із системи	03.04.2023 18:00
35	maximag	Користувач ввійшов в систему	03.04.2023 18:00
36	maximag	Користувач вийшов із системи	03.04.2023 17:59

Рисунок 3.20 – Вікно події

3.5. Висновок до розділу

У даному розділі було описано процес розробки програмного забезпечення було розроблено метод кодування інформації, який дозволяє підвищити ефективність функціонування інфокомунікаційних систем. Розробка програмних модулів системи включала в себе розробку інтерфейсу користувача, функцій опрацювання даних та забезпечення безпеки інформації.

Для перевірки правильності роботи програмного забезпечення було проведено функціональне та модульне тестування, яке дозволило виявити та виправити помилки в роботі системи. Результати тестування свідчать про те, що розроблений додаток відповідає поставленим вимогам.

Інструкція користувачеві програми містить мінімальні вимоги для запуску програмного забезпечення, опис процедури розгортання програмного продукту та використання програмного продукту. Ця інформація дозволить користувачам з легкістю встановлювати та використовувати програмний продукт.

ВИСНОВКИ

Розробка методу кодування інформації для підвищення ефективності функціонування інфокомунікаційних систем є актуальною проблемою в сучасному світі. В результаті дослідження предметної області було з'ясовано, що інфокомунікаційні системи стають все більш складними, і їх ефективність стає важливим фактором в їх функціонуванні. Було проведено аналіз існуючих методів кодування інформації та програмних рішень і виявлено необхідність розробки нового методу кодування інформації для підвищення ефективності інфокомунікаційних систем.

Для розробки нового методу були обрані відповідні технології проектування, включаючи мову програмування C#, середовище розробки Visual Studio та СКБД MS Access. Були проведені аналіз вимог та розробка архітектури проекту, яка включала ERD-діаграму бази даних, а також особливості розробки трьохрівневої архітектури.

Після цього було розроблено новий метод кодування інформації та програмні модулі системи. Було проведено функціональне тестування розробленого додатку та підготовлено інструкцію користувачеві програми, яка містить мінімальні вимоги для запуску ПЗ, опис процедури розгортання програмного продукту та використання програмного продукту.

В результаті проведеного дослідження було розроблено метод кодування інформації для інфокомунікаційних систем, також було розроблено ПЗ, що дало змогу перевірити ефективність роботи даного методу.

Розробка методу кодування інформації має великий потенціал застосування у різних галузях. Зокрема, одна із можливих галузей застосувань - це галузь телекомунікацій. У цій галузі інфокомунікаційні системи використовуються для передачі різних видів інформації, такої як голосові дзвінки, повідомлення та інтернет-трафік. Розроблений метод кодування інформації може допомогти зменшити витрати на передачу даних та покращити якість зв'язку.

Також можливе застосування розробленого методу кодування інформації в галузі безпеки. Інфокомунікаційні системи використовуються для передачі даних між різними підрозділами правоохоронних органів та армій, тому розроблений метод кодування інформації може допомогти збільшити безпеку передачі даних та запобігти можливим кібератакам.

Однією з можливих перспектив розробки є подальше вдосконалення методу кодування інформації та розширення його застосування в різних галузях. Зокрема, можна розробити нові алгоритми кодування та декодування, які дозволяють забезпечити ще більшу ефективність передачі даних та зменшити витрати на зберігання інформації. Також можна розширити застосування розробленого методу на різні типи даних, такі як відео, аудіо, зображення та інші.

Іншою перспективою розробки є використання інтелектуальних систем для аналізу та оптимізації роботи інфокомунікаційних систем. Наприклад, можна розробити систему, яка використовує штучний інтелект для автоматичного вибору найбільш ефективного методу кодування для кожного типу даних в залежності від їх особливостей та характеру передачі.

Окрім того, розробка нового методу кодування інформації може сприяти розвитку нових технологій та продуктів. Наприклад, можна розробити нові програмні продукти, що базуються на цьому методі, або використовувати його для створення нових технологій передачі та зберігання даних.

Також можна зазначити, що розробка нового методу кодування інформації може мати важливе значення в галузі кібербезпеки. Застосування ефективних методів кодування даних може знизити ризики кібератак та забезпечити надійність передачі та зберігання інформації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. M. Luby, "LT codes," in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, Vancouver, BC, Canada, Nov. 2002, pp. 271-282

2. Shapiro, J. M. Embedded coding techniques for data compression / J. M. Shapiro // Proceedings of the IEEE. - 1994. - Vol. 82, No. 6. - P. 854-874.

3. Salomon, D. Data Compression: The Complete Reference / D. Salomon. - 4th ed. - London: Springer, 2007. - 1048 p.

4. Iltis, R. A. The efficiency of data compression algorithms for wireless networks / R. A. Iltis, S. B. Wicker, M. J. Cree // IEEE Transactions on Wireless Communications. - 2005. - Vol. 4, No. 4. - P. 1754-1764.

5. Press, W. H. Numerical Recipes in C: The Art of Scientific Computing / W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery. - 2nd ed. - Cambridge: Cambridge University Press, 2007. - 994 p.

6. Luby, M. LT codes / M. Luby // Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science. - Vancouver, BC, Canada, 2012. - P. 271-282.

7. Wornell, G. Progress in multiple-description coding / G. Wornell // IEEE Signal Processing Magazine. - 2013. - Vol. 20, No. 6. - P. 19-31.

8. Calderbank, A. R. Good codes based on very sparse matrices / A. R. Calderbank, M. V. Wicker // IEEE Transactions on Information Theory. - 2015. - Vol. 45, No. 2. - P. 399-431.

9. Cover, T. Elements of Information Theory / T. Cover, J. Thomas. - 2nd ed. - New York: Wiley, 2006. - 748 p.

10. Ratan, A. Method for compression of ECG signals using JPEG-LS / A. Ratan, M. Vatsa, A. Noore // Proceedings of the 2nd International Conference on Emerging Trends in Engineering & Technology. - 2009. - P. 223-228.

11. Chen, Z. A new method for image compression using fractal coding / Z. Chen, X. Wang, Y. Q. Zhang // Journal of Zhejiang University Science A. - 2007. - Vol. 8, No. 10. - P. 1569-1576.

12. Sayood, K. Introduction to Data Compression / K. Sayood. - 4th ed. - San Francisco: Morgan Kaufmann, 2012. - 768 p.

13. Gallager, R. G. Low-density parity-check codes / R. G. Gallager // IEEE Transactions on Information Theory. - 2002. - Vol. 8, No. 1. - P. 21-28.

14. Varnica, N. Entropy coding in image compression / N. Varnica, M. Medić, M. Grgić // 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). - 2013. - P. 526-531.

15. Gersho, A. Vector Quantization and Signal Compression / A. Gersho, R. M. Gray. - Boston: Kluwer Academic Publishers, 2018. - 324 p.

16. Lee, J. J. A comparison of transform coding, wavelet coding, and fractal coding for image compression / J. J. Lee, A. K. Katsaggelos // IEEE Transactions on Image Processing. - 2008. - Vol. 5, No. 9. - P. 1266-1277.

17. Goyal, V. K. The JPEG2000 Still Image Compression Standard / V. K. Goyal, A. C. Bovik, J. W. Woods. - New York: Springer, 2011. - 398 p.

18. Han, J. H. H.264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia / J. H. Han. - New York: Wiley, 2013. - 306 p.

19. Skodras, A. N. JPEG2000: The upcoming still image compression standard / A. N. Skodras, C. A. Christopoulos, T. Ebrahimi // Proceedings of the IEEE. - 2011. - Vol. 89, No. 10. - P. 1546-1577.

20. Tuzlukov, V. Programmable Digital Signal Processors: Architecture, Programming, and Applications / V. Tuzlukov. - Boca Raton, FL: CRC Press, 2012. - 696 p.

21. Tourapis, A. The H.264/MPEG4 advanced video coding standard and its applications / A. Tourapis, M. P. Cetin // Proceedings of the IEEE International Conference on Image Processing. - 2014. - Vol. 1. - P. I-93-96.

22. O'Reilly, T. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython / T. O'Reilly. - Sebastopol, CA: O'Reilly Media, 2017. - 544 p.

23. Sebesta, R. W. Concepts of Programming Languages / R. W. Sebesta. - 10th ed. - Boston: Pearson, 2016. - 792 p.

24. Ramakrishnan, R. Database Management Systems / R. Ramakrishnan, J. Gehrke. - 3rd ed. - New York: McGraw-Hill, 2003. - 1104 p.

25. Garcia-Molina, H. Database Systems: The Complete Book / H. Garcia-Molina, J. D. Ullman, J. Widom. - 2nd ed. - Upper Saddle River, NJ: Prentice Hall, 2008. - 1179 p.

26. Silberschatz, A. Database System Concepts / A. Silberschatz, H. F. Korth, S. Sudarshan. - 6th ed. - New York: McGraw-Hill, 2010. - 1376 p.

27. Connolly, T. M. Database Systems: A Practical Approach to Design, Implementation, and Management / T. M. Connolly, C. E. Begg. - 6th ed. - Boston: Addison-Wesley, 2014. - 1440 p.

28. Elmasri, R. Fundamentals of Database Systems / R. Elmasri, S. B. Navathe. - 7th ed. - Boston: Addison-Wesley, 2016. - 1276 p.

Додаток А. Лістинги програми

Лістинг 1. Код класу «InfoCrypter»

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;

namespace MethodCodesApp.AppCode {
  class InfoCrypter {
    public byte[] Encrypt(string plainText, byte[] key) { //Оголошення методу, який приймає рядок тексту для шифрування та байтовий ключ для шифрування і повертає масив байтів з зашифрованим текстом.
      byte[] encryptedData; //Оголошення масиву байтів для збереження зашифрованого тексту.
      using (Aes aesAlg = Aes.Create()) { //Створення об'єкту AES для шифрування тексту. Всі ресурси, пов'язані з цим об'єктом, будуть автоматично звільнені після виконання блоку коду.
        aesAlg.Key = key; //Встановлення ключа шифрування
        aesAlg.GenerateIV(); //Генерація вектору ініціалізації, необхідного для коректної роботи AES алгоритму.
        aesAlg.Padding = PaddingMode.PKCS7; // Встановлення режиму доповнення PKCS7 для обробки випадку, коли розмір шифруємого тексту не кратний розміру блоку шифрування.
        ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV); //Створення об'єкту шифрування, який буде використовувати раніше встановлені ключ та вектор ініціалізації
        using (MemoryStream msEncrypt = new MemoryStream()) { // Створення потоку пам'яті, що дозволить записати зашифрований текст
          using (CryptoStream csEncrypts = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write)) { //Створення криптографічного потоку, який буде записувати зашифрований текст у потік пам'яті
            using (StreamWriter swEncrypts = new StreamWriter(csEncrypts)) {
              swEncrypts.Write(plainText); //Запис тексту, що підлягає шифруванню, до потоку пам'яті через криптографічний потік
            }
            encryptedData = msEncrypt.ToArray(); // Отримання масиву байтів з зашифрованим текстом з потоку пам'яті
          }
        }
        byte[] result = new byte[encryptedData.Length + 16]; //Створення масиву байтів, який міститиме зашифрований текст та вектор ініціалізації.
        Buffer.BlockCopy(encryptedData, 0, result, 16, encryptedData.Length); //Копіювання зашифрованого тексту у результуючий масив починаючи з індексу 16.
        Buffer.BlockCopy(aesAlg.IV, 0, result, 0, 16); //Копіювання вектора ініціалізації у результуючий масив.
        return result; //Повернення результату у вигляді масиву байтів, що містить зашифрований текст та вектор ініціалізації.
      }
    }
```

```
   public  string Decrypt(byte[] encryptedData, byte[] key) { //метод отримує вхідні параметри: зашифровані дані у вигляді масиву байтів encryptedData та ключ шифрування у вигляді масиву байтів key.
      string decryptedText; //оголошує змінну decryptedText, в якій зберігатиметься розшифрований текст.
      using (Aes aesAlg = Aes.Create()) { //створення об'єкта AES за допомогою методу Aes.Create(). Об'єкт AES буде створено на початку блоку, а коли він закінчиться, автоматично буде викликаний метод Dispose() для звільнення ресурсів.
        aesAlg.Key = key; //Задаємо ключ шифрування об'єкту AES.
        aesAlg.IV = encryptedData.Take(16).ToArray(); //Встановлюємо ініціалізаційний вектор (IV) об'єкту AES з першими 16 байтами зашифрованих даних encryptedData. Метод Take(16) повертає перші 16 байтів масиву encryptedData, а метод ToArray() перетворює результат у масив байтів
        aesAlg.Padding = PaddingMode.PKCS7; //Встановлюємо режим доповнення (padding) для об'єкту AES. В даному випадку використовується стандартний режим PKCS7
        ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV); //Створюємо об'єкт decryptor за допомогою методу CreateDecryptor, який використовує ключ і IV для створення об'єкту дешифрування.
        using (MemoryStream msDecrypts = new MemoryStream(encryptedData.Skip(16).ToArray())) { //Створюємо новий об'єкт MemoryStream, який містить зашифровані дані, починаючи з 17-го байту та до кінця масиву. Метод Skip(16) пропускає перші 16 байтів масиву encryptedData, а метод ToArray() перетворює результат у масив байтів.
          using (CryptoStream csDecrypt = new CryptoStream(msDecrypts, decryptor, CryptoStreamMode.Read)) { //Створюємо новий об'єкт CryptoStream за допомогою decryptor, який буде використовуватись для розшифрування даних з MemoryStream msDecrypts. Використовується режим CryptoStreamMode.Read, що означає, що дані будуть читатись з потоку.
            using (StreamReader srDecrypts = new StreamReader(csDecrypt)) { //Створюємо новий об'єкт StreamReader, який буде використовуватись для зчитування розшифрованих даних з CryptoStream csDecrypt.
              decryptedText = srDecrypts.ReadToEnd(); //Розшифровані дані зчитуються з StreamReader за допомогою методу ReadToEnd() і зберігаються у змінну decryptedText.
            }
          }
        }
      }
      return decryptedText; //Повертаємо розшифрований текст.
    }
  }
}
```

Лістинг 2. Код класу «EncryptData»

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Security.Cryptography;
using System.IO;

namespace MethodCodesApp.AppCode {
  class EncryptData {
    static byte[] bytes = ASCIIEncoding.ASCII.GetBytes("FixedWWW");
```

```
    public string Encrypt(string originalString) {
      if (String.IsNullOrEmpty(originalString)) {
        throw new ArgumentNullException
              ("The string which needs to be encrypted can not be null.");
      }
      DESCryptoServiceProvider cryptoProvider = new DESCryptoServiceProvider();
      MemoryStream memoryStream = new MemoryStream();
      CryptoStream cryptoStream = new CryptoStream(memoryStream,
          cryptoProvider.CreateEncryptor(bytes, bytes), CryptoStreamMode.Write);
      StreamWriter writer = new StreamWriter(cryptoStream);
      writer.Write(originalString);
      writer.Flush();
      cryptoStream.FlushFinalBlock();
      writer.Flush();
      return Convert.ToBase64String(memoryStream.GetBuffer(), 0, (int)memoryStream.Length);
    }

    public string Decrypt(string cryptedString) {
      if (String.IsNullOrEmpty(cryptedString)) {
        throw new ArgumentNullException
          ("The string which needs to be decrypted can not be null.");
      }
      DESCryptoServiceProvider cryptoProvider = new DESCryptoServiceProvider();
      MemoryStream memoryStream = new MemoryStream
            (Convert.FromBase64String(cryptedString));
      CryptoStream cryptoStream = new CryptoStream(memoryStream,
          cryptoProvider.CreateDecryptor(bytes, bytes), CryptoStreamMode.Read);
      StreamReader reader = new StreamReader(cryptoStream);
      return reader.ReadToEnd();
    }
  }
}
```

Лістинг 3. Код класу «RaportsBLL»

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

namespace MethodCodesApp.BLL {
  class RaportsBLL {
    HistorysProvider _HistorysProvider = new HistorysProvider();

    public List<Historys> GetRaportForPeriodDate(DateTime StartDT, DateTime EndDT) {
      int num = 0;
      List<Historys> HistorysList = new List<Historys>();
      HistorysList = _HistorysProvider.GetAllHistorys();
      List<Historys> searchHistorysList = new List<Historys>();
```

```
    for (int i = 0; i < HistorysList.Count(); i++) {
      if (HistorysList[i].HistorysDate >= StartDT && HistorysList[i].HistorysDate <= EndDT) {
        num++;
        HistorysList[i].Number = num;
        searchHistorysList.Add(HistorysList[i]);
      }
    }
    return searchHistorysList;
  }
 }
}
```

Лістинг 4. Код класу «CodingForm»

```
using MethodCodesApp.AppCode;
using MethodCodesApp.Forms.Systems;
using MethodCodesApp.Providers;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MethodCodesApp.Forms.Controls {
 public partial class CodingForm : Form {
  private ValidationMy _validation = new ValidationMy();
  private CodingsProvider _CodingsProvider = new CodingsProvider();
  private byte[] _Key = new byte[32]; // 256-бітний ключ
  private InfoCrypter _InfoCrypter = new InfoCrypter();
  private byte[] _EncryptedData;
  private LogsProvider _LogsProvider = new LogsProvider();
  private HistorysProvider _HistorysProvider = new HistorysProvider();

  public CodingForm() {
   InitializeComponent();
  }

  private void CodingBtn_Click(object sender, EventArgs e) {
   if (IsTextCodingCorrect()) {
    RandomNumberGenerator rng = RandomNumberGenerator.Create();
    rng.GetBytes(_Key);
    _EncryptedData = _InfoCrypter.Encrypt(TextForCodingRTBox.Text, _Key);
    CodingTextRTBox.Text = Encoding.UTF8.GetString(_EncryptedData);
   }
  }
```

```
    private void SaveBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _CodingsProvider.InsertCodings(CodingsNameTBox.Text, LoginForm.CurrentUser.UsersId,
_Key, _EncryptedData);
        _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId, "Був закодований текст із назвою
"
          + CodingsNameTBox.Text, DateTime.Now);
        _HistorysProvider.InsertHistorys(DateTime.Now, CodingsNameTBox.Text);
        ClearAllControls();
        MessageBox.Show("Текст був успішно закодований та збережений у базу даних!", "Вітаю");
      }
    }


    private void ClearAllControls() {
      TextForCodingRTBox.Text = String.Empty;
      CodingsNameTBox.Text = String.Empty;
      CodingTextRTBox.Text = String.Empty;
    }

    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (_validation.IsDataEntering(CodingsNameTBox.Text)) {
        CodingsNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        CodingsNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (_validation.IsDataEntering(CodingTextRTBox.Text)) {
        CodingTextValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        CodingTextValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }

    private bool IsTextCodingCorrect() {
      bool isCorrect = true;
      if (_validation.IsDataEntering(TextForCodingRTBox.Text)) {
        CategoriesNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        CategoriesNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }
  }
}
```

Лістинг 5. Код класу «DecodingForm»

```
using MethodCodesApp.AppCode;
using MethodCodesApp.Providers;
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 MethodCodesApp.Forms.Controls {
  public partial class DecodingForm : Form {
    private int _selectedRowIndex = 0;
    private CodingsProvider _CodingsProvider = new CodingsProvider();
    private List<Codings> _CodingsList = new List<Codings>();
    private InfoCrypter _InfoCrypter = new InfoCrypter();

    public DecodingForm() {
      InitializeComponent();
      DataLoad();
      CodingsGridView_CellClick(CodingsGridView, new DataGridViewCellEventArgs(1, 0));
    }

    private void DataLoad() {
      int firstRowIndex = 0;
      if (CodingsGridView.FirstDisplayedScrollingRowIndex > 0) {
        firstRowIndex = CodingsGridView.FirstDisplayedScrollingRowIndex;
      }
      try {
        _CodingsList = _CodingsProvider.GetAllCodings();
        LoadDataInCodingsGridView(_CodingsList);
        if (_selectedRowIndex == CodingsGridView.Rows.Count) {
          _selectedRowIndex = CodingsGridView.Rows.Count - 1;
        }
        if (_selectedRowIndex >= 0) {
          CodingsGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
          CodingsGridView.Rows[_selectedRowIndex].Selected = true;
        }
      } catch { }
    }

    private void LoadDataInCodingsGridView(List<Codings> CodingsList) {
      CodingsGridView.DataSource = null;
      CodingsGridView.Columns.Clear();
      CodingsGridView.AutoGenerateColumns = false;
      CodingsGridView.RowHeadersVisible = false;

      CodingsGridView.DataSource = CodingsList;

      if (CodingsList.Count > 0) {
```

```
      if (CodingsList[0].Message == NamesMy.NoDataNames.NoDataInCodings) {
        DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
        messageColumn.DataPropertyName = "Message";
        messageColumn.Width = CodingsGridView.Width - NamesMy.SizeOptins.MinusSizePanel;
        CodingsGridView.Columns.Add(messageColumn);
      } else {
        DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
        DetailIdColumn.DataPropertyName = "CodingsId";
        CodingsGridView.Columns.Add(DetailIdColumn);
        CodingsGridView.Columns[0].Visible = false;

        DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
        numberColumn.HeaderText = "№ ";
        numberColumn.DataPropertyName = "Number";
        numberColumn.DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleRight;
        numberColumn.Width = NamesMy.SizeOptins.NumberSize;
        CodingsGridView.Columns.Add(numberColumn);

        DataGridViewColumn CodingsNameColumn = new DataGridViewTextBoxColumn();
        CodingsNameColumn.HeaderText = "Збережена інформація";
        CodingsNameColumn.DataPropertyName = "CodingsName";
        CodingsNameColumn.Width = NamesMy.SizeOptins.NameSize;
        CodingsGridView.Columns.Add(CodingsNameColumn);
      }
      for (int i = 0; i < CodingsGridView.Columns.Count; i++) {
        CodingsGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
      }
    }
  }

  private void CodingsGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
    if (e.RowIndex >= 0 && CodingsGridView[0, e.RowIndex].Value.ToString() !=
_CodingsList[0].Message) {
      Codings selectedCodings = new Codings();
      selectedCodings =
_CodingsProvider.SelectedCodingsByCodingsId(Convert.ToInt32(CodingsGridView[0,
e.RowIndex].Value.ToString()));
      CodingTextRTBox.Text = _InfoCrypter.Decrypt(selectedCodings.CodingText,
selectedCodings.GenerateKey);
    }
  }
 }
}
```

Лістинг 6. Код класу «HistorysRaportForm»

```
using MethodCodesApp.BLL;
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 MethodCodesApp.Forms.Raport {
  public partial class HistorysRaportForm : Form {
    RaportsBLL _RaportBLL = new RaportsBLL();
    List<Historys> _HistorysList = new List<Historys>();

    public HistorysRaportForm() {
      InitializeComponent();
    }

    private void SearchBtn_Click(object sender, EventArgs e) {
      DateTime start = new DateTime(StartDTP.Value.Year, StartDTP.Value.Month,
StartDTP.Value.Day, 0, 0, 0);
      DateTime end = new DateTime(EndDTP.Value.Year, EndDTP.Value.Month, EndDTP.Value.Day,
23, 59, 59);
      _HistorysList = _RaportBLL.GetRaportForPeriodDate(start, end);
      GetRaport(_HistorysList);
    }

    public void GetRaport(List<Historys> HistorysList) {
      if (HistorysList.Count > 0) {
        RaportTBox.Text = "Звіт за вибраний перід з " + StartDTP.Value.ToShortDateString() + " до "
+ EndDTP.Value.ToShortDateString() + ":\r\n";
        RaportTBox.Text += "-----------------------------------------------------------------------------------------
\r\n";

        RaportTBox.Text += String.Format("{0,3}|{1, -83}|\r\n", "№", "Закодована інформація");
        for (int i = 0; i < HistorysList.Count(); i++) {
          string raportString = String.Format("{0,3}|{1, -83}|\r\n",
          HistorysList[i].Number,
          HistorysList[i].CodingsName);
          RaportTBox.Text += raportString;
        }
        RaportTBox.Text += "-----------------------------------------------------------------------------------------
\r\n";

      } else {
        RaportTBox.Text = "За вибраний період даних не знайдено!";
      }
    }
  }
}
```

Лістинг 7. Код класу «LoginForm»

```
using MethodCodesApp.AppCode;
using MethodCodesApp.Providers;
```

```
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 MethodCodesApp.Forms.Systems {
  public partial class LoginForm : Form {
    public static Users CurrentUser = new Users();

    private UsersProvider _UserProvider = new UsersProvider();
    private ValidationMy _validation = new ValidationMy();
    private LogsProvider _LogsProvider = new LogsProvider();
    private List<Users> _UserList = new List<Users>();
    public LoginForm() {
      InitializeComponent();
      LoadAllDate();
    }

    private void SubmitBtn_Click(object sender, EventArgs e) {
      GetSubmitData();
    }

    private void DataLoad() {
      _LogsProvider.InsertLogs(CurrentUser.UsersId, "Користувач ввійшов в систему",
DateTime.Now);
      this.Visible = false;
      (new MethodMDI()).ShowDialog();
      _LogsProvider.InsertLogs(CurrentUser.UsersId, "Користувач вийшов із системи",
DateTime.Now);
      this.Close();
    }

    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (_validation.IsDataEntering(UserNameCBox.Text)) {
        UserNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        UserNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (_validation.IsDataEntering(UserPasswordTbx.Text)) {
        UserPasswordValidation.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        UserPasswordValidation.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
```

```
    }

    private void LoadAllDate() {
      _UserList = _UserProvider.GetAllUsersListForCBox();
      UserNameCBox.DataSource = _UserList;
      UserNameCBox.AutoCompleteMode = AutoCompleteMode.SuggestAppend;
      UserNameCBox.AutoCompleteSource = AutoCompleteSource.ListItems;
      UserNameCBox.ValueMember = "UsersId";
      UserNameCBox.DisplayMember = "UsersName";
    }

    private void GetSubmitData() {
      try {
        if (IsDataEnteringCorrect()) {
          List<Users> listUsers = new List<Users>();
          listUsers =
_UserProvider.SelectedUsersByUsersNameAndUsersPassword(UserNameCBox.Text,
UserPasswordTbx.Text);
          if (listUsers.Count > 0) {
            CurrentUser = listUsers[0];
            DataLoad();
          } else {

MessageBox.Show(NamesMy.MessageBoxExaption.ThisUserLoginAndUserPasswordNotExistInSyst
em, NamesMy.MessageBoxExaption.CaptionMessage);
          }
        }
      } catch (Exception ex) {
        MessageBox.Show(ex.Message);
      }
    }

    private void RegisterLink_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e) {
      PersonalizationForm personalizationForm = new PersonalizationForm(0);
      personalizationForm.Show();
    }
  }
}
```

Лістинг 8. Код класу «PersonalizationForm»

```
using MethodCodesApp.AppCode;
using MethodCodesApp.Providers;
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 MethodCodesApp.Forms.Systems {
  public partial class PersonalizationForm : Form {
    private int _UserId = 0;
    private Users _selectedUser = new Users();
    private UsersProvider _UserProvider = new UsersProvider();
    private ValidationMy _validation = new ValidationMy();

    public PersonalizationForm(int UserId) {
      InitializeComponent();
      if (UserId == 0) {
        this.Text = "Реєстрація";
      } else {
        _UserId = UserId;
        LoadAllDate();
      }
    }

    private void SaveBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        if (_UserId != 0) {
          _UserProvider.UpdateUsers(FirstNameTBox.Text, LastNameTBox.Text, UserLoginTbx.Text,
PasswordTbx.Text,
            _selectedUser.RoleId, DescriptionTbx.Text, _UserId);
        } else {
          int usersCount = _UserProvider.GetCountUsers();
          if (usersCount > 0) {
            _UserProvider.InsertUsers(FirstNameTBox.Text, LastNameTBox.Text, UserLoginTbx.Text,
PasswordTbx.Text,
              2, DescriptionTbx.Text);
          } else {
            _UserProvider.InsertUsers(FirstNameTBox.Text, LastNameTBox.Text, UserLoginTbx.Text,
PasswordTbx.Text,
              1, DescriptionTbx.Text);
          }
        }
        this.Close();
      }
    }


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

    private void LoadAllDate() {
      _selectedUser = _UserProvider.SelectedUsersByUsersId(_UserId);
      FirstNameTBox.Text = _selectedUser.FirstName;
      LastNameTBox.Text = _selectedUser.LastName;
      UserLoginTbx.Text = _selectedUser.UsersName;
    }
```

```
    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (_validation.IsDataEntering(FirstNameTBox.Text)) {
        FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (_validation.IsDataEntering(LastNameTBox.Text)) {
        LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (_validation.IsPasswordMatch(PasswordTbx.Text, RePasswordTbx.Text)) {
        PasswordAndRePasswordDontMatchLbl.Visible = false;
      } else {
        PasswordAndRePasswordDontMatchLbl.Visible = true;
        isCorrect = false;
      }
      if (_validation.IsDataEntering(UserLoginTbx.Text)) {
        UserLoginValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        UserLoginValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (_validation.IsDataEntering(PasswordTbx.Text)) {
        PasswordValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        PasswordValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (_validation.IsDataEntering(RePasswordTbx.Text)) {
        RePasswordValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        RePasswordValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }

  }
}
```

Лістинг 9. Код класу «SystemLogForm»

```
using MethodCodesApp.AppCode;
using MethodCodesApp.Providers;
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 MethodCodesApp.Forms.Systems {
  public partial class SystemLogForm : Form {
    private int _selectedRowIndex = 0;
    private LogsProvider _LogsProvider = new LogsProvider();
    private List<Logs> _LogsList = new List<Logs>();
    public SystemLogForm() {
      InitializeComponent();
      DataLoad();
    }

    private void DataLoad() {
      int firstRowIndex = 0;
      if (LogsGridView.FirstDisplayedScrollingRowIndex > 0) {
        firstRowIndex = LogsGridView.FirstDisplayedScrollingRowIndex;
      }
      try {
        _LogsList = _LogsProvider.GetAllLogs();
        LoadDataInLogsGridView(_LogsList);
        if (_selectedRowIndex == LogsGridView.Rows.Count) {
          _selectedRowIndex = LogsGridView.Rows.Count - 1;
        }
        if (_selectedRowIndex >= 0) {
          LogsGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
          LogsGridView.Rows[_selectedRowIndex].Selected = true;
        }
      } catch { }
    }

    private void LoadDataInLogsGridView(List<Logs> LogsList) {
      LogsGridView.DataSource = null;
      LogsGridView.Columns.Clear();
      LogsGridView.AutoGenerateColumns = false;
      LogsGridView.RowHeadersVisible = false;

      LogsGridView.DataSource = LogsList;

      if (LogsList.Count > 0) {
        if (LogsList[0].Message == NamesMy.NoDataNames.NoDataInLogs) {
          DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
          messageColumn.DataPropertyName = "Message";
          messageColumn.Width = LogsGridView.Width - NamesMy.SizeOptins.MinusSizePanel;
          LogsGridView.Columns.Add(messageColumn);
        } else {
          DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
          DetailIdColumn.DataPropertyName = "LogsId";
          LogsGridView.Columns.Add(DetailIdColumn);
```

```
        LogsGridView.Columns[0].Visible = false;

        DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
        numberColumn.HeaderText = "№ ";
        numberColumn.DataPropertyName = "Number";
        numberColumn.DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleRight;
        numberColumn.Width = NamesMy.SizeOptins.NumberSize;
        LogsGridView.Columns.Add(numberColumn);

        DataGridViewColumn UsersNameColumn = new DataGridViewTextBoxColumn();
        UsersNameColumn.HeaderText = "Користувач";
        UsersNameColumn.DataPropertyName = "UsersName";
        UsersNameColumn.Width = 150;
        LogsGridView.Columns.Add(UsersNameColumn);

        DataGridViewColumn EventNameShowColumn = new DataGridViewTextBoxColumn();
        EventNameShowColumn.HeaderText = "Подія";
        EventNameShowColumn.DataPropertyName = "EventNameShow";
        EventNameShowColumn.Width = 600;
        LogsGridView.Columns.Add(EventNameShowColumn);

        DataGridViewColumn EvendDateColumn = new DataGridViewTextBoxColumn();
        EvendDateColumn.HeaderText = "Дата";
        EvendDateColumn.DataPropertyName = "EventDate";
        EvendDateColumn.Width = NamesMy.SizeOptins.Date;
        LogsGridView.Columns.Add(EvendDateColumn);

      }
      for (int i = 0; i < LogsGridView.Columns.Count; i++) {
        LogsGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
      }
    }
  }

 }
}
```

Лістинг 10. Код класу «UpdateUsersForm»

```
using MethodCodesApp.AppCode;
using MethodCodesApp.Providers;
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 MethodCodesApp.Forms.Systems {
```

```
public partial class UpdateUsersForm : Form {
  private int _UserId = 0;
  private Users _selectedUser = new Users();
  private UsersProvider _UserProvider = new UsersProvider();
  private ValidationMy _validation = new ValidationMy();
  private RoleApp _RoleApp = new RoleApp();
  private List<Role> _RoleList = new List<Role>();

  public UpdateUsersForm(int UserId) {
    InitializeComponent();
    _UserId = UserId;
    LoadAllDate();
  }

  private void SaveBtn_Click(object sender, EventArgs e) {
    if (IsDataEnteringCorrect()) {
      _UserProvider.UpdateUsers(FirstNameTBox.Text, LastNameTBox.Text, UserLoginTbx.Text,
PasswordTbx.Text,
        Convert.ToInt32(RolesCBox.SelectedValue), DescriptionTbx.Text, _UserId);
      this.Close();
    }
  }

  private void DeleteBtn_Click(object sender, EventArgs e) {
    if (MessageBox.Show("Ви дійсно хочете видалити цей елемент?", "Видалити",
MessageBoxButtons.YesNo) == DialogResult.Yes) {
      _UserProvider.DeleteUsersByUsersId(_UserId);
      this.Close();
    }
  }

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

  private void LoadAllDate() {
    _RoleList = _RoleApp.GetRoleList();
    RolesCBox.DataSource = _RoleList;
    RolesCBox.ValueMember = "RoleId";
    RolesCBox.DisplayMember = "RoleName";

    _selectedUser = _UserProvider.SelectedUsersByUsersId(_UserId);
    FirstNameTBox.Text = _selectedUser.FirstName;
    LastNameTBox.Text = _selectedUser.LastName;
    UserLoginTbx.Text = _selectedUser.UsersName;
    RolesCBox.SelectedValue = _selectedUser.RoleId;
  }

  private bool IsDataEnteringCorrect() {
    bool isCorrect = true;
    if (_validation.IsDataEntering(FirstNameTBox.Text)) {
      FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
```

```
      } else {
        FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (_validation.IsDataEntering(LastNameTBox.Text)) {
        LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (_validation.IsPasswordMatch(PasswordTbx.Text, RePasswordTbx.Text)) {
        PasswordAndRePasswordDontMatchLbl.Visible = false;
      } else {
        PasswordAndRePasswordDontMatchLbl.Visible = true;
        isCorrect = false;
      }
      if (_validation.IsDataEntering(UserLoginTbx.Text)) {
        UserLoginValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        UserLoginValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (_validation.IsDataEntering(PasswordTbx.Text)) {
        PasswordValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        PasswordValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (_validation.IsDataEntering(RePasswordTbx.Text)) {
        RePasswordValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        RePasswordValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }

      return isCorrect;
    }
  }
}
```

Лістинг 11. Код класу «UsersForm»

```
using MethodCodesApp.AppCode;
using MethodCodesApp.Forms.Systems;
using MethodCodesApp.Providers;
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 MethodCodesApp.Forms.Systems {
  public partial class UsersForm : Form {
    private int _selectedRowIndex = 0;
    ValidationMy _validation = new ValidationMy();
    private UsersProvider _UserProvider = new UsersProvider();
    private List<Users> _UserList = new List<Users>();
    private RoleApp _RoleApp = new RoleApp();
    private List<Role> _RoleList = new List<Role>();
    public UsersForm() {
      InitializeComponent();
      LoadAllDate();
      DataLoad();
    }

    private void AddBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _UserProvider.InsertUsers(FirstNameTBox.Text, LastNameTBox.Text, UserLoginTbx.Text,
PasswordTbx.Text,
          Convert.ToInt32(RolesCBox.SelectedValue), DescriptionTbx.Text);
        DataLoad();
        ClearAllControls();
      }
    }

    private void ClearBtn_Click(object sender, EventArgs e) {
      ClearAllControls();
    }

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

    private void DataLoad() {
      int firstRowIndex = 0;
      if (UsersGridView.FirstDisplayedScrollingRowIndex > 0) {
        firstRowIndex = UsersGridView.FirstDisplayedScrollingRowIndex;
      }
      try {
        _UserList = _UserProvider.GetAllUsers();
        LoadDataInKlientGridView(_UserList);
        if (_selectedRowIndex == UsersGridView.Rows.Count) {
          _selectedRowIndex = UsersGridView.Rows.Count - 1;
        }
        if (_selectedRowIndex >= 0) {
          UsersGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
          UsersGridView.Rows[_selectedRowIndex].Selected = true;
        }
      } catch { }
    }
```

```
private void LoadDataInKlientGridView(List<Users> UserList) {
  UsersGridView.DataSource = null;
  UsersGridView.Columns.Clear();
  UsersGridView.AutoGenerateColumns = false;
  UsersGridView.RowHeadersVisible = false;

  UsersGridView.DataSource = UserList;

  if (UserList.Count > 0) {
    if (UserList[0].Message == NamesMy.NoDataNames.NoDataInUsers) {
      DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
      messageColumn.DataPropertyName = "Message";
      messageColumn.Width = UsersGridView.Width - NamesMy.SizeOptins.MinusSizePanel;
      UsersGridView.Columns.Add(messageColumn);
    } else {
      DataGridViewColumn deviseIdColumn = new DataGridViewTextBoxColumn();
      deviseIdColumn.DataPropertyName = "UsersId";
      UsersGridView.Columns.Add(deviseIdColumn);
      UsersGridView.Columns[0].Visible = false;

      DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
      numberColumn.HeaderText = "№ п/п";
      numberColumn.DataPropertyName = "Number";
      numberColumn.DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleRight;
      numberColumn.Width = NamesMy.SizeOptins.NumberSize;
      UsersGridView.Columns.Add(numberColumn);

      DataGridViewColumn lastNameColumn = new DataGridViewTextBoxColumn();
      lastNameColumn.HeaderText = "Фимилия";
      lastNameColumn.DataPropertyName = "LastName";
      lastNameColumn.Width = NamesMy.SizeOptins.Name;
      UsersGridView.Columns.Add(lastNameColumn);

      DataGridViewColumn firstNameColumn = new DataGridViewTextBoxColumn();
      firstNameColumn.HeaderText = "Имя";
      firstNameColumn.DataPropertyName = "FirstName";
      firstNameColumn.Width = NamesMy.SizeOptins.Name;
      UsersGridView.Columns.Add(firstNameColumn);

      DataGridViewColumn UserNameColumn = new DataGridViewTextBoxColumn();
      UserNameColumn.HeaderText = "Логин";
      UserNameColumn.DataPropertyName = "UsersName";
      UserNameColumn.Width = NamesMy.SizeOptins.Name;
      UsersGridView.Columns.Add(UserNameColumn);

      DataGridViewColumn roleNameColumn = new DataGridViewTextBoxColumn();
      roleNameColumn.HeaderText = "Роль";
      roleNameColumn.DataPropertyName = "RoleName";
      roleNameColumn.Width = 150;
      UsersGridView.Columns.Add(roleNameColumn);
    }
```

```
    for (int i = 0; i < UsersGridView.Columns.Count; i++) {
      UsersGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
    }
  }
}

private void ClearAllControls() {
  FirstNameTBox.Text = String.Empty;
  LastNameTBox.Text = String.Empty;
  DescriptionTbx.Text = String.Empty;
  UserLoginTbx.Text = String.Empty;
  PasswordTbx.Text = String.Empty;
  RePasswordTbx.Text = String.Empty;
}

private bool IsDataEnteringCorrect() {
  bool isCorrect = true;
  if (_validation.IsDataEntering(FirstNameTBox.Text)) {
    FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_validation.IsDataEntering(LastNameTBox.Text)) {
    LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_validation.IsPasswordMatch(PasswordTbx.Text, RePasswordTbx.Text)) {
    PasswordAndRePasswordDontMatchLbl.Visible = false;
    PasswordAndRePasswordDontMatchLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    PasswordAndRePasswordDontMatchLbl.Visible = true;
    PasswordAndRePasswordDontMatchLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_validation.IsDataEntering(UserLoginTbx.Text)) {
    UserLoginValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    UserLoginValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_validation.IsDataEntering(PasswordTbx.Text)) {
    PasswordValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    PasswordValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_validation.IsDataEntering(RePasswordTbx.Text)) {
    RePasswordValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
```

```
      RePasswordValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
     }

     return isCorrect;
    }

    private void LoadAllDate() {
     _RoleList = _RoleApp.GetRoleList();
     RolesCBox.DataSource = _RoleList;
     RolesCBox.ValueMember = "RoleId";
     RolesCBox.DisplayMember = "RoleName";
    }

    private void UsersGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
     if (e.RowIndex >= 0 && UsersGridView[0, e.RowIndex].Value.ToString() !=
_UserList[0].Message) {
      _selectedRowIndex = e.RowIndex;
      UpdateUsersForm updateUsersForm = new
UpdateUsersForm(Convert.ToInt32(UsersGridView[0, e.RowIndex].Value.ToString()));
      updateUsersForm.ShowDialog();
      DataLoad();
     }
    }
   }
}
```

Лістинг 12. Код класу «CodingProvider»

```
using MethodCodesApp.AppCode;
using MethodCodesApp.Forms.Systems;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MethodCodesApp.Providers {
  class CodingsProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertCodings(string CodingsName, int UsersId, byte[] GenerateKey, byte[]
CodingText) {
     string SqlString = "INSERT INTO Codings (" +
      "CodingsName, UsersId, GenerateKey, CodingText) Values(?, ?, ?, ?)";
     using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
       cmd.CommandType = CommandType.Text;
       cmd.Parameters.AddWithValue("CodingsName", CodingsName);
```

```
      cmd.Parameters.AddWithValue("UsersId", UsersId);
      if (GenerateKey == null) {
        cmd.Parameters.AddWithValue("GenerateKey", Encoding.Default.GetBytes(""));
      } else {
        cmd.Parameters.AddWithValue("GenerateKey", GenerateKey);
      }
      if (CodingText == null) {
        cmd.Parameters.AddWithValue("CodingText", Encoding.Default.GetBytes(""));
      } else {
        cmd.Parameters.AddWithValue("CodingText", CodingText);
      }
      conn.Open();
      cmd.ExecuteNonQuery();
      conn.Close();
    }
  }
}

public List<Codings> GetAllCodings() {
  int i = 0;
  string SqlString = "SELECT CodingsId, CodingsName FROM Codings WHERE UsersId=" +
    LoginForm.CurrentUser.UsersId +  " ORDER BY CodingsName ASC";
  List<Codings> listAllCodings = new List<Codings>();
  using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
    using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
      conn.Open();
      using (OleDbDataReader reader = cmd.ExecuteReader()) {
        while (reader.Read()) {
          Codings oneCodings = new Codings();
          oneCodings.Number = ++i;
          oneCodings.CodingsId = Convert.ToInt32(reader["CodingsId"]);
          oneCodings.CodingsName = reader["CodingsName"].ToString();
          listAllCodings.Add(oneCodings);
        }
      }
      conn.Close();
    }
  }
  if (listAllCodings.Count == 0) {
    Codings noCodings = new Codings();
    noCodings.CodingsId = 0;
    noCodings.Message = NamesMy.NoDataNames.NoDataInCodings;
    listAllCodings.Add(noCodings);
  }
  return listAllCodings;
}

public Codings SelectedCodingsByCodingsId(int CodingsId) {
  string SqlString = "SELECT * FROM Codings Where CodingsId=" + CodingsId.ToString();

  Codings oneCodings = new Codings();
  using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
```

```
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        using (OleDbDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            oneCodings.CodingsId = Convert.ToInt32(reader["CodingsId"]);
            oneCodings.CodingsName = reader["CodingsName"].ToString();
            oneCodings.UsersId = Convert.ToInt32(reader["UsersId"]);
            oneCodings.GenerateKey = (byte[])reader["GenerateKey"];
            oneCodings.CodingText = (byte[])reader["CodingText"];
          }
        }
      }
      conn.Close();
    }
    return oneCodings;
  }

  public void UpdateCodings(string CodingsName, int UsersId, byte[] GenerateKey, byte[]
CodingText, int CodingsId) {
    string SqlString = "UPDATE Codings SET CodingsName=?, UsersId=?, GenerateKey=?,
CodingText=? " +
      "  WHERE CodingsId=?";

    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        cmd.CommandType = CommandType.Text;
        cmd.Parameters.AddWithValue("CodingsName", CodingsName);
        cmd.Parameters.AddWithValue("UsersId", UsersId);
        if (GenerateKey == null) {
          cmd.Parameters.AddWithValue("GenerateKey", Encoding.Default.GetBytes(""));
        } else {
          cmd.Parameters.AddWithValue("GenerateKey", GenerateKey);
        }
        if (CodingText == null) {
          cmd.Parameters.AddWithValue("CodingText", Encoding.Default.GetBytes(""));
        } else {
          cmd.Parameters.AddWithValue("CodingText", CodingText);
        }
        cmd.Parameters.AddWithValue("CodingsId", CodingsId);
        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
    }
  }

  public void DeleteCodingsByCodingsId(int CodingsId) {
    string SqlString = "DELETE FROM Codings WHERE CodingsId=" + CodingsId.ToString();
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        cmd.ExecuteNonQuery();
```

```
            conn.Close();
          }
        }
      }


    }
  }


public class Codings {
  private int _Number;
  private int _CodingsId;
  private string _CodingsName;
  private int _UsersId;
  private byte[] _GenerateKey;
  private byte[] _CodingText;
  private string _Message;

  public Codings() {
    _Number = 0;
    _CodingsId = 0;
    _CodingsName = String.Empty;
    _UsersId = 0;
    _Message = String.Empty;
  }

  public int Number {
    set { _Number = value; }
    get { return _Number; }
  }
  public int CodingsId {
    set { _CodingsId = value; }
    get { return _CodingsId; }
  }
  public string CodingsName {
    set { _CodingsName = value; }
    get { return _CodingsName; }
  }
  public int UsersId {
    set { _UsersId = value; }
    get { return _UsersId; }
  }
  public byte[] GenerateKey {
    set { _GenerateKey = value; }
    get { return _GenerateKey; }
  }
  public byte[] CodingText {
    set { _CodingText = value; }
    get { return _CodingText; }
  }
  public string Message {
```

```
    set { _Message = value; }
    get { return _Message; }
  }
}
```

Лістинг 13. Код класу «HistorysProvider»

```
using MethodCodesApp.AppCode;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MethodCodesApp.Providers {
  class HistorysProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertHistorys(DateTime HistorysDate, string CodingsName) {
      string SqlString = "INSERT INTO Historys (HistorysDate, CodingsName" +
        ") Values(?, ?)";

      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("HistorysDate", HistorysDate.ToString());
          cmd.Parameters.AddWithValue("CodingsName", CodingsName);
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

    public List<Historys> GetAllHistorys() {
      int i = 0;
      string SqlString = "SELECT * FROM Historys";

      List<Historys> listHistorys = new List<Historys>();
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          conn.Open();
          using (OleDbDataReader reader = cmd.ExecuteReader()) {
            while (reader.Read()) {
              Historys oneHistorys = new Historys();
              oneHistorys.Number = ++i;
              oneHistorys.HistorysId = Convert.ToInt32(reader["HistorysId"]);
              oneHistorys.HistorysDate = Convert.ToDateTime(reader["HistorysDate"]);
              oneHistorys.CodingsName = reader["CodingsName"].ToString();
```

```
            listHistorys.Add(oneHistorys);
          }
        }
        conn.Close();
      }
    }

    if (listHistorys.Count == 0) {
      Historys noHistorys = new Historys();
      noHistorys.HistorysId = 0;
      noHistorys.Message = NamesMy.NoDataNames.NoDataInHistorys;
      listHistorys.Add(noHistorys);
    }
    return listHistorys;
  }

  public Historys SelectedHistorysByHistorysId(int HistorysId) {
    string SqlString = "SELECT * FROM Historys Where HistorysId=" + HistorysId.ToString();

    Historys oneHistorys = new Historys();
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        using (OleDbDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            oneHistorys.HistorysId = Convert.ToInt32(reader["HistorysId"]);
            oneHistorys.HistorysDate = Convert.ToDateTime(reader["HistorysDate"]);
            oneHistorys.CodingsName = reader["CodingsName"].ToString();
          }
        }
      }
      conn.Close();
    }
    return oneHistorys;
  }

  public void DeleteHistorysByHistorysId(int HistorysId) {
    string SqlString = "DELETE FROM Historys WHERE HistorysId=" + HistorysId.ToString();
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
    }
  }

 }
}


public class Historys {
```

```
  private int _Number;
  private int _HistorysId;
  private DateTime _HistorysDate;
  private string _CodingsName;
  private string _Message;

  public Historys() {
    _Number = 0;
    _HistorysId = 0;
    _HistorysDate = new DateTime();
    _CodingsName = String.Empty;
    _Message = String.Empty;
  }

  public int Number {
    set { _Number = value; }
    get { return _Number; }
  }
  public int HistorysId {
    set { _HistorysId = value; }
    get { return _HistorysId; }
  }
  public DateTime HistorysDate {
    set { _HistorysDate = value; }
    get { return _HistorysDate; }
  }
  public string CodingsName {
    set { _CodingsName = value; }
    get { return _CodingsName; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```

Лістинг 14. Код класу «LogsProvider»

```
using MethodCodesApp.AppCode;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MethodCodesApp.Providers {
  class LogsProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];
```

```
    public void InsertLogs(int UsersId, string EventNameShow, DateTime EventDate) {
      string SqlString = "INSERT INTO Logs (UsersId, EventNameShow, EventDate) Values(?, ?, ?)";
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("UsersId", UsersId);
          cmd.Parameters.AddWithValue("EventNameShow", EventNameShow);
          cmd.Parameters.AddWithValue("EventDate", EventDate.ToString());
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

    public List<Logs> GetAllLogs() {
      int i = 0;
      string SqlString = "SELECT Logs.LogsId, Logs.UsersId, Logs.EventNameShow, Logs.EventDate,
Users.UsersName " +
      "FROM Logs INNER JOIN Users ON Users.UsersId = Logs.UsersId  ORDER BY
Logs.EventDate DESC";
      List<Logs> listAllLogs = new List<Logs>();
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          conn.Open();
          using (OleDbDataReader reader = cmd.ExecuteReader()) {
            while (reader.Read()) {
              Logs oneLogs = new Logs();
              oneLogs.Number = ++i;
              oneLogs.LogsId = Convert.ToInt32(reader["LogsId"]);
              oneLogs.UsersId = Convert.ToInt32(reader["UsersId"]);
              oneLogs.EventNameShow = reader["EventNameShow"].ToString();
              oneLogs.EventDate = Convert.ToDateTime(reader["EventDate"]);
              oneLogs.UsersName = reader["UsersName"].ToString();
              listAllLogs.Add(oneLogs);
            }
          }
          conn.Close();
        }
      }

      if (listAllLogs.Count == 0) {
        Logs noLogs = new Logs();
        noLogs.LogsId = 0;
        noLogs.Message = NamesMy.NoDataNames.NoDataInLogs;
        listAllLogs.Add(noLogs);
      }
      return listAllLogs;

    }
  }
}
```

```
public class Logs {
  private int _Number;
  private int _LogsId;
  private int _UsersId;
  private string _UsersName;
  private string _EventNameShow;
  private DateTime _EventDate;
  private string _Message;

  public Logs() {
    _Number = 0;
    _LogsId = 0;
    _UsersId = 0;
    _UsersName = String.Empty;
    _EventNameShow = String.Empty;
    _EventDate = new DateTime();
    _Message = String.Empty;
  }

  public int Number {
    set { _Number = value; }
    get { return _Number; }
  }
  public int LogsId {
    set { _LogsId = value; }
    get { return _LogsId; }
  }
  public int UsersId {
    set { _UsersId = value; }
    get { return _UsersId; }
  }
  public string UsersName {
    set { _UsersName = value; }
    get { return _UsersName; }
  }
  public string EventNameShow {
    set { _EventNameShow = value; }
    get { return _EventNameShow; }
  }
  public DateTime EventDate {
    set { _EventDate = value; }
    get { return _EventDate; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```

Лістинг 15. Код класу «UsersProvider»

```
using MethodCodesApp.AppCode;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MethodCodesApp.Providers {
  class UsersProvider {
    private EncryptData _encryptData = new EncryptData();
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertUsers(string FirstName, string LastName, string UsersName, string
UsersPassword,
   int RoleId, string Description) {
      string SqlString = "INSERT INTO Users (FirstName, LastName, UsersName, UsersPassword, " +
       "RoleId, Description) Values(?, ?, ?, ?, ?, ?)";

      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("FirstName", FirstName);
          cmd.Parameters.AddWithValue("LastName", LastName);
          cmd.Parameters.AddWithValue("UsersName", UsersName);
          cmd.Parameters.AddWithValue("UsersPassword", _encryptData.Encrypt(UsersPassword));
          cmd.Parameters.AddWithValue("RoleId", RoleId);
          cmd.Parameters.AddWithValue("Description", Description);
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

    public List<Users> GetAllUsers() {
      int i = 0;
      string SqlString = "SELECT * FROM Users ORDER BY LastName ASC";
      List<Users> listAllUsers = new List<Users>();
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          conn.Open();
          using (OleDbDataReader reader = cmd.ExecuteReader()) {
            while (reader.Read()) {
              Users oneUsers = new Users();
              oneUsers.Number = ++i;
              oneUsers.UsersId = Convert.ToInt32(reader["UsersId"]);
```

```
            oneUsers.FirstName = reader["FirstName"].ToString();
            oneUsers.LastName = reader["LastName"].ToString();
            oneUsers.FIO = oneUsers.LastName + " " + oneUsers.FirstName;
            oneUsers.UsersName = reader["UsersName"].ToString();
            oneUsers.UsersPassword = _encryptData.Decrypt(reader["UsersPassword"].ToString());
            oneUsers.RoleId = Convert.ToInt32(reader["RoleId"]);
            oneUsers.RoleName = GetRoleName(oneUsers.RoleId);
            oneUsers.Description = reader["Description"].ToString();
            listAllUsers.Add(oneUsers);
          }
        }
        conn.Close();
      }
    }

    if (listAllUsers.Count == 0) {
      Users noUsers = new Users();
      noUsers.UsersId = 0;
      noUsers.Message = NamesMy.NoDataNames.NoDataInUsers;
      listAllUsers.Add(noUsers);
    }
    return listAllUsers;
  }

  private string GetRoleName(int RoleId) {
    RoleApp roleApp = new RoleApp();
    for (int i = 0; i < roleApp.GetRoleList().Count(); i++) {
      if (RoleId == roleApp.GetRoleList()[i].RoleId) {
        return roleApp.GetRoleList()[i].RoleName;
      }
    }
    return "";
  }

  public Users SelectedUsersByUsersId(int UsersId) {
    string SqlString = "SELECT * FROM Users WHERE UsersId=" + UsersId.ToString();

    Users oneUsers = new Users();
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        using (OleDbDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            oneUsers.UsersId = Convert.ToInt32(reader["UsersId"]);
            oneUsers.FirstName = reader["FirstName"].ToString();
            oneUsers.LastName = reader["LastName"].ToString();
            oneUsers.UsersName = reader["UsersName"].ToString();
            oneUsers.FIO = oneUsers.LastName + " " + oneUsers.FirstName;
            oneUsers.UsersPassword = _encryptData.Decrypt(reader["UsersPassword"].ToString());
            oneUsers.RoleId = Convert.ToInt32(reader["RoleId"]);
            oneUsers.Description = reader["Description"].ToString();
          }
```

```
        }
       }
       conn.Close();
      }
      return oneUsers;
    }

    public List<Users> GetAllUsersListForCBox() {
      string SqlString = "SELECT UsersId, UsersName, UsersPassword FROM Users ORDER BY
UsersName ASC";
      List<Users> listAllUsers = new List<Users>();

      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
       using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        using (OleDbDataReader reader = cmd.ExecuteReader()) {
         while (reader.Read()) {
          Users oneUsers = new Users();
          oneUsers.UsersId = Convert.ToInt32(reader["UsersId"].ToString());
          oneUsers.UsersName = reader["UsersName"].ToString();
          oneUsers.UsersPassword = _encryptData.Decrypt(reader["UsersPassword"].ToString());
          listAllUsers.Add(oneUsers);
         }
        }
        conn.Close();
       }
      }

      if (listAllUsers.Count == 0) {
       Users noUsers = new Users();
       noUsers.UsersId = 0;
       noUsers.Message = NamesMy.NoDataNames.NoDataInUsers;
       listAllUsers.Add(noUsers);
      }
      return listAllUsers;
    }

    public List<Users> SelectedUsersByUsersNameAndUsersPassword(string UsersName, string
UsersPassword) {
      string SqlString = "SELECT * FROM Users WHERE UsersName='" + UsersName + "' AND
UsersPassword='" + _encryptData.Encrypt(UsersPassword) + "'";
      List<Users> UsersList = new List<Users>();

      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
       using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        using (OleDbDataReader reader = cmd.ExecuteReader()) {
         while (reader.Read()) {
          Users oneUsers = new Users();
          oneUsers.UsersId = Convert.ToInt32(reader["UsersId"]);
          oneUsers.FirstName = reader["FirstName"].ToString();
          oneUsers.LastName = reader["LastName"].ToString();
```

```
            oneUsers.UsersName = reader["UsersName"].ToString();
            oneUsers.FIO = oneUsers.LastName + " " + oneUsers.FirstName;
            oneUsers.UsersPassword = _encryptData.Decrypt(reader["UsersPassword"].ToString());
            oneUsers.RoleId = Convert.ToInt32(reader["RoleId"]);
            oneUsers.Description = reader["Description"].ToString();
            UsersList.Add(oneUsers);
          }
        }
      }
      conn.Close();
    }
    return UsersList;
  }

  public void UpdateUsers(string FirstName, string LastName, string UsersName, string
UsersPassword,
  int RoleId, string Description, int UsersId) {
    string SqlString = "UPDATE Users SET FirstName=?, LastName=?, " +
  "UsersName=?, UsersPassword=?, RoleId=?, Description=? WHERE UsersId=?";

    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        cmd.CommandType = CommandType.Text;
        cmd.Parameters.AddWithValue("FirstName", FirstName);
        cmd.Parameters.AddWithValue("LastName", LastName);
        cmd.Parameters.AddWithValue("UsersName", UsersName);
        cmd.Parameters.AddWithValue("UsersPassword", _encryptData.Encrypt(UsersPassword));
        cmd.Parameters.AddWithValue("RoleId", RoleId);
        cmd.Parameters.AddWithValue("Description", Description);
        cmd.Parameters.AddWithValue("UsersId", UsersId);
        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
    }
  }

  public void DeleteUsersByUsersId(int UsersId) {
    string SqlString = "DELETE FROM Users WHERE UsersId=" + UsersId.ToString();
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
    }
  }


  public int GetCountUsers() {
    int count = 0;
    using (SqlConnection connection = new SqlConnection(_ConnString)) {
```

```
        connection.Open();
        // запит для отримання кількості записів у таблиці
        string query = "SELECT COUNT(*) FROM Users";
        // виконання запиту та отримання кількості записів
        using (SqlCommand command = new SqlCommand(query, connection)) {
          count = (int)command.ExecuteScalar();
        }
        return count;
      }
    }

  }
}


public class Users {
  private int _Number;
  private int _UsersId;
  private string _FirstName;
  private string _LastName;
  private string _UsersName;
  private string _FIO;
  private string _UsersPassword;
  private int _RoleId;
  private string _RoleName;
  private string _Description;
  private string _Message;

  public Users() {
    _UsersId = 0;
    _FirstName = String.Empty;
    _LastName = String.Empty;
    _UsersName = String.Empty;
    _FIO = String.Empty;
    _UsersPassword = String.Empty;
    _RoleId = 0;
    _Description = String.Empty;
  }

  public int Number {
    set { _Number = value; }
    get { return _Number; }
  }
  public int UsersId {
    set { _UsersId = value; }
    get { return _UsersId; }
  }
  public string FirstName {
    set { _FirstName = value; }
    get { return _FirstName; }
  }
  public string LastName {
```

```
    set { _LastName = value; }
    get { return _LastName; }
  }
  public string FIO {
    set { _FIO = value; }
    get { return _FIO; }
  }
  public string UsersName {
    set { _UsersName = value; }
    get { return _UsersName; }
  }
  public string UsersPassword {
    set { _UsersPassword = value; }
    get { return _UsersPassword; }
  }
  public int RoleId {
    set { _RoleId = value; }
    get { return _RoleId; }
  }
  public string RoleName {
    set { _RoleName = value; }
    get { return _RoleName; }
  }
  public string Description {
    set { _Description = value; }
    get { return _Description; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```