МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
ДЕРЖАВНОГО НЕКОМЕРЦІЙНОГО ПІДПРИЄМСТВА
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»
Циклова комісія *комп'ютерних систем та мереж*
(повна назва циклової комісії)

Допустити до захисту
Голова випускової циклової комісії
комп'ютерних систем та мереж
(повна назва циклової комісії)
________ *Ірина КРАВЧУК*
(підпис) (ім'я, ПРІЗВИЩЕ)
« *10* » *06* 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬО-ПРОФЕСІЙНОГО СТУПЕНЯ ФАХОВИЙ МОЛОДШИЙ БАКАЛАВР

Тема: *Програмна система уніфікації звітних даних для державних установ з використанням C# та DBF-баз даних*

Група: *3-011* Спеціальність: *123 «Комп'ютерна інженерія»*

Здобувач освіти ________ *Владислав ПАВЛЮК*
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи ________ *Світлана ТЕРЬОШИНА*
(підпис) (ім'я, ПРІЗВИЩЕ)

Консультант з оформлення пояснювальної записки ________ *Оксана ОСАДЧА*
(підпис) (ім'я, ПРІЗВИЩЕ)

Кривий Ріг 2025 р.

КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
ДЕРЖАВНОГО НЕКОМЕРЦІЙНОГО ПІДПРИЄМСТВА
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

Відділення *комп'ютерної та програмної інженерії*
Циклова комісія *комп'ютерних систем та мереж*
Освітній ступінь *фаховий молодший бакалавр*
Спеціальність *123 «Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Голова випускової циклової комісії
комп'ютерних систем та мереж
(повна назва циклової комісії)
________ *Ірина КРАВЧУК*
(підпис) (ім'я, ПРІЗВИЩЕ)
« *01* » *03* 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ОСВІТИ

Павлюка Владислава Сергійовича
(прізвище, ім'я, по батькові)

1.Тема роботи *Програмна система уніфікації звітних даних для державних використанням C# та DBF-баз даних*

Керівник роботи *Терьошина Світлана Сергіївна, викладач вищої категорії*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по коледжу від « *04* » *04* 2025 року № *51-ст*

2. Строк подання здобувачем освіти роботи з *19.05.2025* по *13.06.2025*

3. Вихідні дані до роботи *Розробка програмної системи уніфікації звітних використанням C# та DBF-баз даних*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз інформаційного забезпечення предметної області. Розробка структури. Визначення структури і функціональне призначення модулів системи, їх взаємозв'язок. Розробка програмних модулів системи. Визначення вимог до технічних засобів

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація Microsoft PowerPoint

6. Консультанти розділів роботи (проекту)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ____________________

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	*Узгодження технічного завдання з керівником кваліфікаційної роботи*	*17.03.2025-21.03.2025*	*виконано*
2	*Підбір та вивчення науково-технічної літератури за темою кваліфікаційної роботи*	*24.03.2025-28.03.2025*	*виконано*
3	*Обґрунтування вибору програмних засобів*	*31.03.2025-04.04.2025*	*виконано*
4	*Опис компонентів. Обґрунтування їх вибору.*	*07.04.2025-09.04.2025*	*виконано*
5	*Розробка програмного забезпечення*	*10.04.2025-28.04.2025*	*виконано*
6	*Дослідження ефективності реалізованих методів.*	*29.04.2025-02.05.2025*	*виконано*
7	*Написання пояснювальної записки*	*12.05.2025-23.05.2025*	*виконано*
8	*Перевірка на плагіат пояснювальної записки*	*02.06.2025-06.06.2025*	*виконано*
9	*Попередній захист кваліфікаційної роботи*	*26.05.2025-30.05.2025*	*виконано*
10	*Захист кваліфікаційної роботи*		

Здобувач освіти ____________ *Владислав Павлюк*
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи ____________ *Світлана Терьошина*
(підпис) (ім'я, ПРІЗВИЩЕ)

StrikePlagiarism.com

Дата звіту 6/9/2025
Дата редагування ---

Звіт подібності

метадані

Назва організації
Ukrainian national aviation university

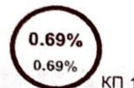
Заголовок
Кваліфікаційна_робота Павлюк 3011

Автор / Науковий керівник / Експерт
ПавлюкТерьошина С.С

підрозділ
Криворізький Фаховий коледж

Обсяг знайдених подібностей

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



КП 1

0.74%
0.74%
КЦ

25
Довжина фрази для коефіцієнта подібності 2

12992
Кількість слів

99149
Кількість символів

Тривога

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

Заміна букв	ß	1
Інтервали	A→	0
Мікропробіли		0
Білі знаки	ß	0
Парафрази (SmartMarks)	a	6

Подібності за списком джерел

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

10 найдовших фраз

Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://ela.kpi.ua/bitstreams/1322c304-fd94-414e-a9a5-256e21b3b90a/download	16 0.12 %
2	https://manager.bobrodobro.ru/97599	14 0.11 %
3	https://ela.kpi.ua/bitstream/123456789/58548/1/Dolid_bakalavr.pdf	13 0.10 %

РЕФЕРАТ

Пояснювальна записка «Програмна система уніфікації звітних даних для державних установ з використанням C# та DBF-баз даних» містить: 63 сторінки, 1 додаток, 11 використаних джерел .

Сучасні технології: C#, .NET Framework (VFPOLEDB/OleDb), Windows Forms, Visual Studio Code.

У кваліфікаційній роботі розроблено програмну систему для уніфікації та конвертації звітних DBF-файлів різних форм у єдиний шаблон. Об’єктом дослідження є процес обробки табличних даних у форматі DBF, а предметом — методи нормалізації їхньої структури, валідації та запису у нову таблицю. Метою роботи є створення модульної системи, що забезпечує:

- імпорт DBF-файлів довільної структури (на прикладі форм 1ds–5ds); - нормалізацію полів до єдиної схеми (idformsimp, orgcode, formcode, formtype, program, datesnput, fondcode, kindcode, formperiod, rownum, cell1…cell20); - експорт консолідованих даних у новий DBF-файл з заданою структурою.

- Обрано багаторівневу архітектуру з розділенням на:

- UI (Windows Forms) — вибір файлів, налаштування параметрів, індикація результатів;

- Application (ConversionManager) — координація послідовності обробки;

- Business Logic (DbfSchemaAnalyzer, ValidationEngine, DataNormalizer, Logger) — аналіз структури, валідація, нормалізація та логування; - Data Access (DbfReader, DbfWriter, OleDbConnectionFactory) — зчитування і запис DBF через VFPOLEDB .

У результаті система дозволяє автоматизовано обробити десятки DBF-файлів за один запуск, виявити й зафіксувати можливі помилки (відсутність

обов'язкових полів, некоректне кодування, дублікати тощо), а також сформувати готовий до імпорту у централізовані державні системи вихідний файл.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ 7

ВСТУП... 8 РОЗДІЛ 1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ .... 11 1.1 Історія та еволюція формату DBF .... 11 1.2 Порівняння з іншими форматами зберігання даних.. 13 1.3 Основні виклики обробки DBF-звітів у держсекторі.. 16 1.4 Реальні приклади та наслідки помилок у звітності .... 19 1.5 Огляд інструментів доступу та обробки DBF .... 22 1.6 Методи валідації та контролю даних .... 25 РОЗДІЛ 2 ПРОЕКТУВАННЯ СТРУКТУРИ СИСТЕМИ.. 29 2.1 Функціональні вимоги та сценарії використання... 29 2.2 Архітектурні принципи та багаторівнева структура .... 32 2.3 Опис основних модулів системи .... 34 2.4 Взаємодія компонентів: UML та потік обробки .... 38 2.5 Формування структури вихідного DBF-файлу .... 40 2.6 Механізми розширюваності та параметризації.. 42 РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ.. 46 3.1 Використані

технології та середовище розробки .... 46 3.2 Інтерфейс користувача... 48 3.3 Рівень доступу до даних (DAL).. 50 3.4 Бізнес-логіка та нормалізація даних (BLL) .... 52 3.5 Модуль валідації та обробки помилок.. 54

3.6 Логування та звітність .... 56 3.7 Модульні тести й перевірка якості коду.. 58 3.8 Оптимізація продуктивності та масштабованість .... 60 РОЗДІЛ 4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ.. 63 4.1 Підготовка до впровадження .... 63 4.2 Процес впровадження... 63 4.3 Інструкція користувача... 64 ВИСНОВКИ.. 66 СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.. 68 ДОДАТОК... 70

ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ

API (Application Programming Interface) — програмний інтерфейс прикладного програмування; набір засобів для взаємодії програмного забезпечення. *BLL (Business Logic Layer)* — рівень бізнес-логіки; відповідає за правила обробки та перевірки даних.

CSV (Comma-Separated Values) — формат файлу з розділенням значень комами;

використовується для представлення табличних даних у текстовому вигляді. *DAL (Data Access Layer)* — рівень доступу до даних; відповідає за зчитування та запис інформації з бази даних або файлів.

DBF (DataBase File) — файл бази даних; бінарний формат зберігання табличних даних, характерний для dBASE, FoxPro тощо.

JSON (JavaScript Object Notation) — текстовий формат представлення структурованих даних на основі синтаксису JavaScript.

OLE DB (Object Linking and Embedding Database) — інтерфейс доступу до різних джерел даних від Microsoft.

SQL (Structured Query Language) — мова структурованих запитів; використовується для роботи з реляційними базами даних.

UML (Unified Modeling Language) — уніфікована мова моделювання; застосовується для графічного опису архітектури програмних систем. *UI (User Interface)* — інтерфейс користувача; засоби взаємодії користувача з програмою.

VFPOLEDB (Visual FoxPro OLE DB Provider) — постачальник інтерфейсу доступу до DBF-файлів, що використовуються у Visual FoxPro.

XML (eXtensible Markup Language) — розширювана мова розмітки; використовується для зберігання і передачі структурованих даних. *.NET* — програмна платформа від Microsoft, яка забезпечує середовище для розробки та виконання додатків, зокрема на мові C#.

C# — об'єктно-орієнтована мова програмування від Microsoft, призначена для розробки програм під платформу .NET.

ВСТУП

В умовах стрімкої цифровізації та підвищення вимог до ефективності державного управління, забезпечення оперативного та точного обміну статистичною інформацією між установами набуває критичного значення. Проблема ускладнюється тим, що значна частина звітних даних у державному секторі продовжує зберігатися у застарілому форматі DBF, що створює перешкоди для їх централізованого збирання, зіставлення та комплексного аналізу. Різноманітність структур DBF-файлів зумовлює значні витрати часу на ручну

обробку та підвищує ймовірність помилок при конвертації даних.

Актуальність даної дипломної роботи полягає в розробці програмної системи уніфікації звітних даних для державних установ. Система передбачає автоматизацію процесу зчитування та перетворення DBF-файлів різних форматів у єдиний шаблон з подальшим збереженням результату в новій DBF-таблиці. Для реалізації поставлених завдань використовується мова програмування C# та технології доступу до DBF-файлів VFPOLEDB/OleDb. Розроблений інтуїтивно зрозумілий інтерфейс на основі Windows Forms має забезпечити високу продуктивність обробки даних та зручність у використанні кінцевим користувачем. Мета роботи

Метою даної дипломної роботи є розробка модульної програмної системи, що забезпечує:

- імпорт DBF-файлів різних форм звітності (на прикладі 1ds–5ds); -

перетворення даних до єдиної структури полів (idformsimp, orgcode, formcode, formtype, program, datesnput, fondcode, kindcode, formperiod, rownum, cell1…cell20);

- експорт трансформованих даних у новий DBF-файл із заданими

параметрами.

Для досягнення поставленої мети необхідно вирішити такі завдання: 1. Проаналізувати формат DBF, його особливості та можливості роботи з ним засобами VFPOLEDB/OleDb.

2. Спроектувати архітектуру програмної системи, включаючи розподіл на рівні представлення (UI), бізнес-логіки (BLL) та доступу до даних (DAL). 3. Розробити користувацький інтерфейс на основі Windows Forms (форми Form1–Form3) для забезпечення вибору вхідних файлів, налаштування параметрів перетворення та запуску процесу конвертації.

4. Реалізувати програмні методи, що забезпечують:

- імпорт даних з DBF-файлів у структуру DataTable (метод LoadInputTable);

- динамічне створення вихідної DBF-таблиці (метод CreateOutputTable); - обробку даних різних форм звітності та виконання параметризованих SQL-запитів (метод InsertData).

5. Провести тестування розробленої системи на реальних прикладах DBF файлів, оцінити коректність результатів перетворення та швидкодію системи.

Об’єкт та предмет дослідження

- Об'єкт дослідження: процес автоматизованої обробки та уніфікації табличних звітних файлів у форматі DBF, що використовуються в державних установах.

- Предмет дослідження: методи та алгоритми перетворення структурованих даних з DBF-форм довільної структури у єдиний шаблон, реалізовані мовою C# з використанням VFPOLEDB та Windows Forms.

Методи дослідження

У процесі дослідження застосовано такі методи:

- аналіз існуючих форматів DBF;

- проектування архітектури системи з використанням UML-діаграм; - розробка програмного забезпечення на основі парадигми подієво орієнтованого програмування з використанням мови C#;

- тестування результатів конвертації даних для оцінки їх коректності.

Структура роботи

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків.

- У першому розділі подано опис предметної області, включаючи аналіз формату DBF, проблем його використання в державному секторі та огляд існуючих інструментів для роботи з DBF-файлами.

- У другому розділі розглянуто етапи проектування системи, її архітектуру, функціональні вимоги та моделі використання.

- Третій розділ присвячено програмній реалізації модулів системи, вибору технологій та середовища розробки.

- У четвертому розділі представлено результати тестування розробленої системи та їх аналіз.

РОЗДІЛ 1
ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

У сучасних умовах державного управління обробка табличних звітів залишається однією з найболючіших точок у роботі бухгалтерів, аналітиків та IT спеціалістів. Незалежно від того, де саме — у центральних апаратах міністерств чи районних відділеннях соцзахисту — більшість даних все ще зберігається у застарілому, але надзвичайно розповсюдженому форматі DBF. У цьому розділі ми детально розкриємо, як історично утвердився цей формат, у чому полягають його особливості та обмеження, чому він досі не втрачає своєї актуальності, та якими інструментами користуються спеціалісти для обробки таких даних.

1.1 Історія та еволюція формату DBF

Формат DBF (DataBase File) — один із найстаріших бінарних форматів зберігання табличних даних, який з'явився ще у 1980-х роках у складі програмного забезпечення dBASE від компанії Ashton-Tate [3]. Його популярність стрімко зростала у зв’язку з потребою в ефективному зберіганні та обробці структурованих даних на персональних комп’ютерах, які мали обмежені ресурси.

На відміну від текстових форматів (як-от CSV), DBF одразу створювався як структурований бінарний контейнер, який дозволяв швидко й ефективно здійснювати доступ до конкретних полів і записів. Його структура містила заголовок із описом полів, блок записів фіксованої довжини та (опційно) memo

файли — окремі файли для зберігання довгих текстових полів [7].

1.1.1 Основні етапи розвитку

- dBASE II (1981) — перша популярна реалізація DBF. Формат мав жорстко фіксовану структуру, підтримував лише базові типи полів: символьні, числові, логічні. Створення та читання файлів відбувалося з DOS-командного середовища.

- dBASE III (1984) — додано підтримку memo-полів (.DBT) для зберігання великих обсягів тексту, введено поняття deleted-флагів для м'якого видалення записів.

- dBASE IV / FoxBASE (кінець 1980-х) — формат був стандартизований та поширений серед сторонніх розробників. З'явилася підтримка індексації через .NDX/.MDX.

- Visual FoxPro (1995–2007) — Microsoft випустила розширення формату у вигляді DBF Visual FoxPro 9, де були додані транзакції, підтримка SQL синтаксису, складні індекси (.CDX), зв'язки між таблицями, вбудовані механізми referential integrity.

Важливо, що всі ці модифікації залишалися зворотно сумісними: програми могли читати файли, створені попередніми версіями.

1.1.2 Популярність у державному секторі

Формат DBF став надзвичайно популярним у державному секторі пострадянських країн через такі фактори:

- Простота використання. DBF-файли можна відкривати у FoxPro, Excel, Access — без складних драйверів чи налаштувань.

- Автономність. Один DBF-файл — це завершена база даних. Його можна

передати електронною поштою чи на флешці, і його відразу буде видно. -

Поширення 1С та FoxPro у 1990–2000-х. Багато державних установ використовували системи обліку, які автоматично формували звітність у DBF.

1.1.3 Технічні особливості класичних DBF

- Фіксована довжина запису. Це забезпечувало надзвичайно швидкий прямий доступ до будь-якого рядка у таблиці.

- Memo-поля. У парі з основним .DBF-файлом створювався .FPT або .DBT-файл, де зберігалися довгі тексти — примітки, адреси тощо.

- Слабка підтримка метаданих. Формат не містив внутрішнього опису ключів, обмежень чи зв’язків — усе покладалося на програму, яка його читала.

1.1.4 Причини виживання формату донині

Попри те, що DBF не відповідає сучасним вимогам до інформаційної безпеки, масштабованості та цілісності, він продовжує використовуватися в державному секторі, особливо для формалізованих звітів, де структура таблиці є простою, а запити — передбачуваними.

До переваг, що забезпечили довголіття формату, належать:

- мінімальні вимоги до ресурсів (може працювати без сервера БД); -

простота імпорту/експорту у популярних офісних додатках; - десятиліття

звички бухгалтерів та відсутність коштів на модернізацію.

1.2 Порівняння з іншими форматами зберігання даних

Незважаючи на свій поважний вік, формат DBF і надалі застосовується у практиці звітності, особливо в органах державного управління. Проте в умовах

розвитку цифрових технологій постає закономірне питання: чому DBF досі використовується, і чим він відрізняється від сучасних форматів зберігання табличних даних? У цьому розділі порівняємо DBF з найбільш поширеними альтернативами: CSV, XML, JSON та реляційними СУБД.

1.2.1 DBF vs. CSV

Формат CSV (Comma-Separated Values) — це простий текстовий файл, у якому поля розділені комами або іншими символами. Його головна перевага — це доступність: майже будь-який текстовий редактор або табличний процесор (як-от Excel) може відкрити CSV.

Переваги DBF над CSV:

- Строга типізація. DBF зберігає типи полів (числовий, дата, логічний), тоді як CSV — лише текст.

- Фіксована довжина записів. Це дає змогу швидко читати великі обсяги даних без повного завантаження файлу в пам'ять.

- Немає проблем із роздільниками. У CSV, якщо текст містить кому, він повинен бути обгорнутий лапками, що створює проблеми при парсингу. Недоліки DBF:

- Менша універсальність — для відкриття потрібен спеціалізований софт або драйвер.

- Гірша сумісність з UNIX-середовищем та веб-платформами.

1.2.2 DBF vs. XML/JSON

Формати XML та JSON широко застосовуються в сучасних веб-системах, API, мобільних застосунках тощо. Вони дозволяють зберігати як прості, так і вкладені структури даних.

Переваги XML/JSON:

- Гнучкість: можна описувати довільні ієрархії та метадані. -

Стандартизованість: добре інтегруються з веб-технологіями та REST-інтерфейсами.

- Читаємість для людини.

- Чому DBF іноді виграє:

- Менший розмір файлу. DBF — бінарний, XML/JSON — текстові. -

Швидкість обробки. DBF не потребує парсерів і працює напряму через бібліотеки доступу до файлів.

- Простота впровадження у «офлайнових» системах. DBF можна обробляти без інтернету, без зовнішніх сервісів і без складної інфраструктури.

1.2.3 DBF vs. Реляційні СУБД (SQL Server, MySQL, PostgreSQL) Сучасні реляційні бази даних мають безліч можливостей: транзакції, реплікація, масштабованість, мова SQL тощо. Проте вони вимагають окремого програмного забезпечення, серверного середовища, адміністрування та ресурсів.

Коли DBF краще:

- У випадку автономної роботи без мережі (наприклад, локальна бухгалтерія).

- При мінімальних вимогах до інфраструктури (немає сервера БД). -

Коли потрібно миттєво передати файл (на флешці, в email). Коли краще SQL-СУБД:

- При обробці великих обсягів пов'язаних таблиць.

- Коли потрібна безпека, аудит, контроль доступу.

- Для вебсистем і багатокористувацького середовища.

Для наочного зіставлення, ключові характеристики розглянутих форматів даних та їх порівняння представлені в Таблиці 1.

Таблиця 1 – Порівняння форматів зберігання

Критерій	**DBF**	**CSV**	**XML/JSON**	**SQL-СУБД**
Тип даних	Типізовані поля	Текст	Гнучка структура	Типізовані поля
Формат	Бінарний	Текстовий	Текстовий	Бінарний + клієнт-сервер
Підтримка індексів	Так	Ні	Ні	Так
Розмір файлу	Невеликий	Малий	Великий	Сховище БД
Автономна робота	Так	Так	Так	Ні
Потреба у спеціальному ПЗ	Так (VFPOLEDB)	Ні	Так (парсер)	Так
Підтримка транзакцій	Частково (FoxPro 9)	Ні	Ні	Так
Адміністрування	Мінімальне	Мінімальне	Середнє	Високе

Формат DBF не є ідеальним і в багатьох аспектах поступається сучасним підходам. Проте в умовах, коли потрібно мати мінімальні вимоги до інфраструктури, обробляти прості таблиці локально і не залежати від сторонніх

сервісів, DBF залишається доцільним вибором. Саме тому в державному секторі він і досі широко застосовується, а розробка автоматизованих систем обробки таких файлів має практичну цінність.

1.3 Основні виклики обробки DBF-звітів у держсекторі

Незважаючи на практичну доступність формату DBF та його багаторічне використання в системах державного обліку, обробка таких файлів у реальних умовах супроводжується низкою технічних і організаційних труднощів. У сучасних державних установах переважає значна різноманітність підходів до формування звітності, що ускладнює інтеграцію даних, контроль за цілісністю та подальшу автоматизацію.

Нижче розглянемо основні проблеми, з якими стикаються ІТ-фахівці, аналітики та бухгалтери при обробці DBF-звітів.

1.3.1 Різнорідність шаблонів і структур

Однією з головних перешкод на шляху до автоматизованої обробки DBF файлів є відсутність єдиних структурних стандартів. Навіть у межах однієї форми звітності можуть існувати різні версії, у яких:

- однакові поля мають різні назви (наприклад: TotalAmount, SUM_TOTAL, Сума),

- поля розташовані в іншому порядку,

- використовуються локальні скорочення чи коди, що не мають пояснення.

Такий підхід унеможливлює застосування універсальних скриптів або SQL запитів без попереднього аналізу конкретного файлу. Результатом є необхідність ручного налаштування під кожен новий шаблон, що суттєво уповільнює обробку великих обсягів даних.

1.3.2 Нестабільність регламентів

Форми звітності, затверджені державними органами (Мінфіном, Держстатом тощо), періодично змінюються: додаються нові поля, змінюються назви колонок, редагуються правила заповнення.

Ці зміни не завжди супроводжуються відповідним оновленням програмного

забезпечення в установах. У результаті:

- частина організацій надсилає дані за застарілими шаблонами, -

інші вже перейшли на нові структури,

- деякі надсилають змішані набори з обома типами файлів. Це створює

труднощі при автоматизованому злитті звітів та потребує розробки адаптивних алгоритмів аналізу та конверсії.

1.3.3 Проблеми з кодуванням

DBF-файли можуть бути згенеровані в різних операційних системах (DOS, Windows, Linux) і з використанням різних кодувань: CP866, Windows-1251, UTF-8 тощо. Часто в заголовках DBF вказане кодування або відсутнє, або неправильне, що призводить до:

- спотворення кириличних символів при відкритті файлів,

- помилок під час парсингу рядків,

- некоректної обробки фільтрів та сортування.

Автоматичне визначення та нормалізація кодування потребує спеціальних механізмів, оскільки не всі бібліотеки (зокрема, у .NET) підтримують повноцінне розпізнавання charset DBF-файлів.

1.3.4 Відсутність метаданих та зв'язків

DBF — це самодостатній файл без схеми зв'язків. У ньому немає зовнішніх ключів, індексів (окрім простих .CDX/.IDX), або обмежень цілісності, характерних для реляційних баз даних. Це означає, що:

- жодним чином не вказано, які поля є обов'язковими;

- немає внутрішньої валідації типів значень;

- складно виявити дублікати або логічні помилки.

Таким чином, перевірка коректності даних перекладається повністю на програму або людину, яка їх обробляє.

1.3.5 Труднощі з інтеграцією у сучасні IT-системи

Сучасні платформи звітності (наприклад, державні портали, електронні сервіси або системи документообігу) зазвичай використовують SQL-сервери, REST API або XML/JSON як формати передачі. DBF-файли погано інтегруються в такі системи:

- потребують попередньої конвертації;

- мають обмежену підтримку драйверами для Linux або macOS; -

ускладнюють впровадження web-сервісів та автоматичного імпорту. У

зв'язку з цим виникає потреба створення проміжних конвертаційних модулів, які переводять дані з DBF у сучасні структури для подальшої обробки або завантаження.

1.3.6 Людський фактор

Оскільки значна частина звітності в державному секторі готується вручну або за допомогою напівавтоматизованих систем (наприклад, застарілих версій 1С чи FoxPro), у файлах часто зустрічаються:

- неправильно вказані дати (наприклад, 31.02.2023),

- текст у числових полях;

- дублі рядків;

- пусті обов’язкові поля.

Усі ці фактори потребують ретельної перевірки, формування логів та запровадження алгоритмів валідації ще до виконання об’єднання або експорту в централізовану базу.

Обробка DBF-файлів у держустановах — це не просто технічна задача, а комплексна проблема, яка включає питання сумісності, стандартизації, безпеки та організаційного контролю. Для її вирішення необхідне впровадження автоматизованих систем, які враховують усі особливості структури даних, мають адаптивну архітектуру та підтримують повну валідацію вхідної інформації.

1.4 Реальні приклади та наслідки помилок у звітності

Належна організація процесів формування, перевірки та передачі звітних даних є однією з найважливіших складових функціонування будь-якої державної установи. Від якості поданої звітності залежить не лише точність статистичних узагальнень, але й ефективність розподілу бюджетних коштів, планування реформ, соціальних виплат, кадрових рішень тощо.

На жаль, практика показує, що ручна або напівавтоматизована обробка даних у форматі DBF часто супроводжується типовими помилками. Їх поява зумовлена як технічними обмеженнями формату, так і людським фактором, а також організаційною складністю взаємодії між різними рівнями адміністративного апарату. Нижче розглянуто низку практичних ситуацій, які демонструють масштаб і потенційні наслідки помилок, що виникають у звітних процесах.

1.4.1 Типові ситуації помилок

Приклад 1: Проблема з кодуванням символів

Опис: у рамках чергової звітної кампанії бухгалтер райвідділу соцзахисту створив DBF-файл у середовищі DOS, де за замовчуванням використовується кодування CP866. Проте головне управління, куди надсилався файл, обробляло звіти на платформі Windows із використанням Excel, що очікував кодування Windows-1251. В результаті, під час відкриття даних символи кирилиці були викривлені — назви полів та текстові значення виглядали як нечитабельний набір знаків.

Коментар: Така ситуація типова, оскільки DBF не має надійного механізму фіксації використовуваного кодування в заголовку файлу. І навіть якщо байт

кодової сторінки формально присутній, багато програм ігнорують його або використовують власне тлумачення.

Наслідки: файл виявився непридатним для автоматизованої обробки. Фахівцям довелося вручну перекодовувати дані за допомогою сторонніх інструментів, що затримало обробку на понад 24 години.

Приклад 2: Використання застарілого шаблону

Опис: після впровадження оновленої версії звітної форми 2-ПВ, до якої було додано нове поле region_code, частина установ через брак інформації продовжила користуватися старим шаблоном. Це призвело до відсутності обов'язкових полів у сформованому DBF-файлі.

Коментар: Зміна шаблонів форм є звичним явищем, але через відсутність централізованої автоматичної системи оновлення або контролю за версіями такі зміни часто не доходять вчасно до виконавців на місцях.

Наслідки: автоматизована система збору звітності не змогла опрацювати неповні файли. Їх довелося виключати з загального потоку та повертати на доопрацювання. Це спричинило затримку звітності по всьому регіону. Приклад 3: Відмінності у форматах числових значень

Опис: під час формування фінансової звітності одна установа використовувала десяткову крапку (.), а інша — десяткову кому (,). У DBF ці символи не стандартизовано, тому залежно від середовища та локалізації системи ці значення трактувалися або як числові, або як текстові.

Коментар: Наявність таких розбіжностей свідчить про відсутність чітких регламентів щодо формату числових полів на рівні усієї системи звітності. Часто це питання залишено на розсуд розробника програмного забезпечення або виконавця на місцях.

Наслідки: частина показників не була врахована при агрегації, що призвело до помилкового заниження суми звіту. Лише після зовнішньої перевірки було виявлено відхилення від нормативних значень.

Приклад 4: Повторювані рядки у звітних даних

Опис: унаслідок технічного збою під час експорту з системи обліку 1С було

сформовано DBF-файл, що містив дублікати записів з однаковим rownum. Через відсутність у DBF механізмів контролю унікальності або первинних ключів, система не змогла автоматично виявити проблему.

Коментар: Такий збій часто виникає через неправильну логіку у користувацькому макросі або через ручне об'єднання файлів із різних джерел без попередньої перевірки.

Наслідки: у консолідованому звіті відображалась подвійна інформація, що спровокувало хибні аналітичні висновки та потребувало повторного формування підсумкових таблиць.

1.4.2 Узагальнені наслідки

Сукупність подібних ситуацій дозволяє зробити висновок, що навіть у випадку використання відносно «простого» формату, як-от DBF, ризики помилок залишаються високими. Часто вони мають не лише технічний, а й організаційний характер: недостатній контроль, брак інструкцій, відсутність системних оновлень, слабка комунікація між установами різного рівня.

Серед основних типових наслідків таких помилок можна виокремити: -

Затримки у звітності: через необхідність ручного доопрацювання, переформатування або повторного заповнення.

- Навантаження на IT-відділ: спеціалісти змушені витрачати час на

виправлення технічних недоліків замість виконання планових завдань. - Зростання

ризику адміністративних та фінансових санкцій: особливо у випадку подання несвоєчасної або некоректної інформації до вищих органів управління.

- Системні збої в приймаючому програмному забезпеченні, що очікує

строгої структури та типізації даних.

- Зниження довіри до даних: керівництво втрачає впевненість у

достовірності звітів, що ускладнює прийняття рішень.

1.5 Огляд інструментів доступу та обробки DBF

Формат DBF, хоч і є досить застарілим, все ще потребує підтримки та обробки в багатьох державних установах. У зв'язку з цим на ринку програмного забезпечення представлено широкий спектр інструментів і бібліотек, які дозволяють працювати з DBF-файлами на різних мовах програмування та платформах. Вибір конкретного засобу залежить від вимог до функціональності, продуктивності, підтримки кодувань, зручності розгортання та сумісності з існуючими IT-системами.

У цьому підрозділі розглядаються найбільш поширені інструменти доступу до DBF-файлів, які застосовуються в середовищі .NET, а також у деяких альтернативних технологіях.

1.5.1 VFPOLEDB (Visual FoxPro OLE DB Provider)

Один із найпотужніших і водночас найтрадиційніших способів роботи з DBF у середовищі Microsoft — це використання VFPOLEDB, офіційного постачальника OLE DB-доступу до файлів Visual FoxPro [1].

Переваги:

- Повна підтримка специфікації DBF (включаючи memo-поля, індекси,

транзакції).

- Можливість виконання SQL-запитів безпосередньо до DBF-файлів. -

Висока продуктивність при читанні та записі даних.

Обмеження:

- Не підтримується офіційно в нових версіях Windows (починаючи з

Windows 10 64-bit вимагає ручного встановлення).

- Працює лише в межах Windows-платформи.

Висновок: VFPOLEDB — оптимальний вибір для проектів на C# із високими вимогами до сумісності з Visual FoxPro, однак його використання потребує правильного розгортання на кінцевих робочих станціях.

1.5.2 DotNetDBF

DotNetDBF — це кросплатформна .NET-бібліотека з відкритим вихідним кодом, яка дозволяє читати та записувати DBF-файли без використання зовнішніх драйверів чи постачальників.

Переваги:

- Не потребує встановлення сторонніх компонентів.

- Працює на будь-якій платформі, що підтримує .NET або Mono. -

Зручна для вбудованих або переносних рішень.

Недоліки:

- Обмежена підтримка типів даних (наприклад, немає повноцінної

підтримки memo-полів).

- Менш стабільна робота з кириличними кодуваннями.

Висновок: підходить для простих задач, особливо коли немає можливості використовувати VFPOLEDB, або коли додаток має працювати під Linux/macOS.

1.5.3 XBase.NET, CodeBase та інші комерційні бібліотеки

Існує цілий ряд професійних платних рішень для обробки DBF-файлів, наприклад XBase.NET, CodeBase, Apollo DBF Library тощо. Вони, як правило, мають багатий функціонал і підтримують більшість специфікацій dBase/Clipper/Visual FoxPro.

Переваги:

- Повна підтримка всіх версій DBF.

- Оптимізація для великих обсягів даних.

- Розширені можливості: індексація, робота з кількома таблицями

одночасно, інтеграція з СУБД.

Недоліки:

- Потребують ліцензування.

- Не завжди мають українську або російську локалізацію.

Висновок: доречні для корпоративних систем із великим обсягом обробки та вимогами до надійності, але надлишкові для невеликих внутрішніх рішень.

1.5.4 Інструменти в інших мовах програмування

Хоча кваліфікаційна робота орієнтована на реалізацію в середовищі C#, варто згадати також про популярні засоби роботи з DBF у Python, R та інших мовах. Python:

- Бібліотеки dbfread, simpledbf, dbf дозволяють зчитувати й аналізувати дані без складного налаштування.

- Переваги — зручність у скриптах для аналітики або швидкої візуалізації.

- Пакети foreign, DBI використовуються для імпорту старих статистичних баз у DBF-форматі.

- Обмежена підтримка memo-полів, не рекомендовано для масової обробки.

Висновок: у науковому та дослідницькому середовищі використання DBF у Python або R має місце, проте у держсекторі такі підходи застосовуються рідко, оскільки не мають необхідної продуктивності та відповідності вимогам безпеки. Основні переваги, недоліки та особливості розглянутих інструментів для роботи з DBF-файлами узагальнено в Таблиці 2.

Таблиця 2 – Порівняння «інструментів»

Інструмент	**Платформа**	**Підтримка memo**	**Потрібен драйвер**	**Переваги**	**Недоліки**

VFPOLEDB	Windows	Так	Так	Висока сумісність, повний SQL	Вимагає встановлення, лише Windows
DotNetDBF	Кросплат.	Частково	Ні	Простота, легке розгортання	Обмежена підтримка типів
XBase.NET	Windows	Так	Вбудовано	Комерційна підтримка, стабільність	Платна ліцензія
dbf (Python)	Кросплат.	Частково	Ні	Скриптові рішення, гнучкість	Повільна обробка великих файлів
foreign (R)	Кросплат.	Ні	Ні	Інтеграція зі статистичними пакетами	Не підтримує великі масиви

На сучасному етапі існує кілька шляхів доступу до DBF-файлів, кожен з яких має свої переваги та обмеження. Для проектів у державному секторі, де важлива сумісність з наявними базами Visual FoxPro, найбільш доцільним є використання VFPOLEDB. Проте для простих або кросплатформних рішень з обмеженим функціоналом ефективними можуть бути бібліотеки на зразок DotNetDBF.

Ключовим критерієм вибору інструменту має бути баланс між функціональністю, надійністю, простотою розгортання та сумісністю з існуючою IT-інфраструктурою установи.

1.6 Методи валідації та контролю даних

У процесі обробки та консолідації звітної інформації, збереженої у форматі DBF, особливе значення має не лише правильне зчитування та структурування даних, але й перевірка їхньої достовірності, повноти та відповідності встановленим стандартам. В умовах, коли джерелами інформації є десятки або сотні різних підрозділів державної установи, валідація стає ключовим етапом, що

забезпечує якість підсумкових аналітичних звітів.

З огляду на обмеження самого формату DBF, який не підтримує схеми, обмеження цілісності або типів зв'язків, функції контролю даних мають реалізовуватися зовнішніми програмними засобами — у нашому випадку, в рамках розробленої системи обробки, написаної на C#.

1.6.1 Базова валідація структури

Найперше, що перевіряється при завантаженні DBF-файлу, — це його структура заголовка та наявність необхідних полів. У державних звітних формах часто використовуються стандартизовані назви полів, наприклад: - formcode — код форми;

- datesnput — дата подання;
- rownum — порядковий номер рядка;
- cell1 ... cell20 — числові значення по категоріях.

Якщо хоча б одне з ключових полів відсутнє, файл вважається непридатним для обробки і блокується на рівні завантаження.

Крім того, перевіряється відповідність імен полів очікуваним шаблонам. Якщо, наприклад, замість formcode у файлі наявне поле Form_Code або Форма, програма має мати можливість виконати автоматичне зіставлення або сповістити користувача про невідповідність.

1.6.2 Форматна валідація значень

Далі виконується перевірка типів і форматів даних, що дозволяє виявити потенційно помилкові або недопустимі записи:

Дати: поля на зразок datesnput повинні відповідати формату дд.мм.рррр. Наявність порожніх дат або некоректних значень (наприклад, «32.13.2023») фіксується у журналі.

Числові значення: поля cell1–cell20 повинні містити лише числові значення з

допустимою точністю. Якщо замість числа введено текст (наприклад, «n/a» або «-»), програма повинна автоматично або замінити його на 0, або попередити користувача.

Текстові поля: кодові значення (наприклад, formperiod, fondcode) мають відповідати певним дозволеним наборам (наприклад, тільки «1», «2», «4»). Для реалізації цих перевірок у C# зручно використовувати регулярні вирази (Regex), а також методи DateTime.TryParse() та Decimal.TryParse().

1.6.3 Перевірка унікальності та логічної узгодженості

Одним із ключових моментів є перевірка унікальності записів. У багатьох формах, зокрема у 1-ПВ, 2-ПВ, 5-ДС, кожен рядок повинен мати унікальний rownum. Повторення одного і того ж номера може вказувати на дублювання записів, яке спотворює підсумкові показники.

Також виконується логічна перевірка відповідності значень, наприклад: -

якщо fondcode задано як «0», то kindcode не може бути «65»; - якщо datesnput

перевищує поточну дату — це вважається помилкою; - якщо усі cellX значення

дорівнюють нулю, а rownum заповнено — це може бути свідченням відсутності фактичних даних.

У разі виявлення таких розбіжностей дані не видаляються, але позначаються у журналі перевірки як «підозрілі» для подальшого ручного аналізу.

1.6.4 Ведення журналів і звітність про помилки

Усі виявлені помилки та попередження мають бути зафіксовані у лог-файлі.

Такий файл (наприклад, conversion.log) містить наступну інформацію: - час і дата

початку перевірки;

- шлях до файлу, який перевіряється;

- перелік виявлених помилок із вказанням номера рядка, назви поля, типу помилки;

- загальну статистику: кількість успішно оброблених записів, кількість із помилками, кількість пропущених.

Цей журнал не лише служить засобом аудиту, а й дає змогу IT-фахівцям швидко локалізувати джерело проблеми у великому наборі файлів.

1.6.5 Інтерактивне сповіщення користувача

На додаток до логування важливо забезпечити оперативне інформування користувача. У рамках реалізованої системи це здійснюється за допомогою: - MessageBox-повідомлень про критичні помилки (наприклад, відсутній обов'язковий стовпець);

- індикаторів стану — кількість рядків з помилками відображається у вікні підсумків;

- (за потреби) — автоматичне формування email-звіту з логом, який може надсилатись у відділ підтримки.

Такі механізми дозволяють знизити рівень стресу для кінцевого користувача і забезпечують прозорість усіх етапів обробки.

У загальному підсумку можна стверджувати, що валідація є необхідною умовою надійної роботи зі звітними DBF-даними. Її ігнорування може призвести до суттєвих відхилень у результатах, які, у свою чергу, вплинуть на прийняття управлінських рішень на вищому рівні. Комплексна система перевірки, яка включає структурну, форматну, логічну та змістовну валідацію, повинна бути вбудована в кожну автоматизовану систему, що працює з даними в цьому форматі.

РОЗДІЛ 2

ПРОЕКТУВАННЯ СТРУКТУРИ СИСТЕМИ

2.1 Функціональні вимоги та сценарії використання

Ефективне проєктування будь-якої сучасної інформаційної системи передбачає не лише визначення її технічної реалізації, але й глибоке розуміння задач, які вона повинна вирішувати, а також обмежень, у межах яких система функціонуватиме. У випадку розробки програмної системи уніфікації звітних даних, яка має справу з великою кількістю різнотипних DBF-файлів, питання чіткого формування функціональних вимог набуває особливої актуальності.

В умовах реального використання в державному секторі система повинна відповідати низці вимог, які відображають очікування кінцевих користувачів (експертів, бухгалтерів, аналітиків) і технічні обмеження, пов'язані з платформами, форматами даних і рівнем цифрової грамотності персоналу.

Аналіз вимог до системи

На початковому етапі розробки було проведено функціональне моделювання, що дозволило виділити перелік ключових вимог до поведінки системи. Ці вимоги поділяються на основні групи: функціональні, нефункціональні та інтерфейсні.

Функціональні вимоги

Функціональні вимоги описують дії, які система повинна виконувати у

відповідь на вхідні дані та взаємодію з користувачем. Основними з них є: -

Завантаження DBF-файлів із довільною структурою.

Система має підтримувати завантаження як окремих файлів, так і групової пакетної обробки. При цьому формат і структура вхідних файлів можуть суттєво відрізнятися — як за кількістю полів, так і за назвами, типами та порядком їх розміщення.

- Автоматичне визначення критичних полів.

Незалежно від варіацій імен полів, система повинна розпізнати їх

відповідність стандартному шаблону (наприклад, formcode, datesnput, rownum) та

повідомити про відсутність обов'язкових елементів структури. - Нормалізація структури вхідних таблиць.

Концептуально система має здійснювати приведення даних до уніфікованої форми, що полягає у вирівнюванні назв, форматів, порядку полів, заповненні порожніх значень та приведенні чисел і дат до єдиного стандарту.

- Створення нової DBF-таблиці зі строго визначеною структурою. Після обробки система має сформувати вихідний файл у форматі DBF, який повністю відповідає затвердженій моделі. Така модель повинна бути сумісною з подальшим імпортом у централізовані державні системи обліку. - Ведення журналу подій.

Система повинна фіксувати усі кроки обробки: від часу запуску і списку оброблених файлів до переліку помилок, попереджень та пропущених записів. Цей журнал має бути зручним для подальшого аналізу, збереженим у текстовому форматі (наприклад, .log).

Нефункціональні вимоги

Нефункціональні вимоги визначають обмеження, які не стосуються безпосередньо функцій, але мають критичне значення для стабільності, ефективності й сумісності системи:

1. Продуктивність. Система повинна обробляти великі файли (до 100 тис. рядків) без помітного зниження швидкодії або зависань.

2. Сумісність. Підтримка роботи у середовищах Windows 7, 10, 11 без потреби встановлення додаткових СУБД або складних компонентів, окрім VFPOLEDB.

3. Зрозумілість. Інтерфейс користувача має бути інтуїтивним навіть для осіб без технічного досвіду — бухгалтерів, методистів, діловодів. 4. Безпека. Усі операції відбуваються локально, без потреби доступу до зовнішніх мереж, що гарантує захист персональних та фінансових даних.

5. Можливість розширення. Архітектура повинна дозволяти майбутнє додавання підтримки нових форматів (наприклад, XML або CSV), не змінюючи

основну логіку.

Типові сценарії використання. Опис типових сценаріїв дозволяє краще зрозуміти очікувану поведінку системи в реальних умовах експлуатації. Сценарій 1: Обробка окремого звіту вручну

- Користувач запускає програму з ярлика на робочому столі. - Через інтерфейс обирає DBF-файл, сформований у 1С або іншій системі.

- Програма автоматично визначає структуру та пропонує параметри конвертації.

- Після натискання кнопки «Конвертувати» виконується обробка та створюється новий файл.

- Користувач переглядає результат та лог-повідомлення.

Сценарій 2: Пакетна обробка десятків файлів

- Адміністратор звітності копіює всі DBF-файли за місяць у одну папку. - Система сканує директорію, перевіряє кожен файл на валідність і запускає автоматичну обробку.

- По завершенні формується один підсумковий уніфікований файл, а також зведений звіт про всі знайдені помилки.

Сценарій 3: Робота у віддаленому підрозділі без технічної підтримки - Користувач самостійно інсталює програму на свій комп’ютер. - Завдяки зрозумілому меню та мінімальним налаштуванням він може одразу почати роботу.

- У разі помилок (наприклад, відсутність поля rownum) система надає чітке повідомлення з поясненням і порадами.

2.2 Архітектурні принципи та багаторівнева структура

Проєктування архітектури програмної системи — це ключовий етап життєвого циклу програмного забезпечення, який значною мірою визначає її гнучкість, масштабованість, підтримуваність і стабільність у довгостроковій перспективі. У випадку розробки системи уніфікації звітних DBF-файлів для державних установ, архітектура повинна бути не лише технічно ефективною, але й адаптованою до умов реального використання, з урахуванням обмежень апаратного забезпечення, рівня кваліфікації користувачів та потреби у швидкому впровадженні змін.

З метою досягнення зазначених цілей у роботі було обрано багаторівневу архітектуру з чітким поділом відповідальностей між її складовими. Цей підхід базується на класичних концепціях Separation of Concerns (SoC) та SOLID принципів, що забезпечують ізольованість логіки, мінімізацію міжмодульних залежностей та простоту супроводу системи [9].

Базова модель архітектури

- Обрана модель реалізована у вигляді чотирирівневої структури: -

Presentation Layer (UI) — рівень взаємодії з користувачем; - Application Layer

(Control/Orchestration) — рівень координації бізнес логіки;

- Domain/Business Logic Layer (BLL) — рівень правил обробки даних; - Data

Access Layer (DAL) — рівень доступу до DBF-файлів. Кожен із зазначених рівнів виконує чітко визначену роль і не перетинається з іншими за обсягом відповідальності.

1. Presentation Layer (рівень представлення)

Цей рівень реалізовано за допомогою Windows Forms — перевіреної технології побудови графічних інтерфейсів, яка є вбудованою у .NET Framework та ідеально підходить для внутрішніх десктопних застосунків [10]. Основне призначення цього рівня:

- забезпечення зручного вибору вхідних DBF-файлів;

- введення користувачем параметрів звітності (код форми, дата, період тощо);

- запуск основних сценаріїв обробки (конвертація, перегляд логів); -

інформування користувача про результати та можливі помилки. Усі елементи управління згруповано у три основні форми (Form1, Form2, Form3), що дозволяє логічно розділити процес перегляду, налаштування та виконання операцій.

2. Application Layer (координаційний рівень)

Цей шар відповідає за послідовність дій, які виконуються під час обробки даних, але сам по собі не містить жодної бізнес-логіки. Його завдання — передати ініціативу відповідним компонентам бізнес-рівня у правильному порядку.
Основний клас:

ConversionManager — координує увесь процес обробки, послідовно викликаючи: DbfSchemaAnalyzer → ValidationEngine → DataNormalizer → DbfWriter

Наявність цього рівня дозволяє чітко відділити логіку роботи з формами від логіки обробки даних.

3. Domain Layer (бізнес-логіка)

Центральний рівень системи, в якому реалізовані усі правила, специфічні для обробки звітних DBF-файлів: перевірка структур, нормалізація імен, валідація записів тощо.

Основні компоненти:

- DbfSchemaAnalyzer — зчитує структуру вхідного DBF-файлу та будує внутрішню модель таблиці;

- DataNormalizer — трансформує дані до стандартизованої форми, з урахуванням назв, порядку та допустимих значень;

- ValidationEngine — перевіряє коректність значень, виявляє порожні або

невалідні поля;

- Logger — зберігає всі етапи обробки, попередження та помилки у log

файл.

Кожен з цих компонентів побудовано з дотриманням принципу єдиної відповідальності (Single Responsibility Principle), що дозволяє легко змінювати або тестувати логіку незалежно від інших частин системи.

4. Data Access Layer (доступ до даних)

Усі операції зчитування й запису DBF-файлів винесено в окремий рівень, що дозволяє ізолювати роботу з VFPOLEDB і забезпечити можливість майбутнього переходу на інші технології доступу до даних без зміни бізнес-логіки.
Компоненти:

- DbfReader — імпортує дані з DBF у внутрішню модель (DataTable); -

DbfWriter — створює нову таблицю та записує туди уніфіковані дані; -

OleDbConnectionFactory — централізовано формує рядки підключення до DBF-файлів.

Переваги обраної архітектури

- Гнучкість – легко додати нові функції, не змінюючи існуючі класи. -

Масштабованість – у майбутньому можна реалізувати веб-версію, повторно використовуючи бізнес-логіку.

- Тестованість – логіка обробки не прив’язана до інтерфейсу і може

перевірятися окремо.

- Легка підтримка – при оновленні форм звітності можна адаптувати лише

один модуль (наприклад, DataNormalizer), не змінюючи решти.

2.3 Опис основних модулів системи

Проєктування архітектури програмної системи передбачає не лише логічне структурування її рівнів, але й чітке формування функціонального складу — тобто конкретних модулів, які реалізують ту чи іншу частину бізнес-логіки або технічної взаємодії. У рамках розробки системи уніфікації звітних DBF-файлів ключовим завданням було створення набору модулів, кожен із яких має єдину відповідальність, добре визначену роль і чіткий набір функцій.

З метою досягнення високого рівня гнучкості, повторного використання та ізоляції змін було впроваджено п’ять логічних груп модулів: модулі керування

процесом, обробки структури, нормалізації, перевірки й доступу до даних. Далі подано опис кожної з груп з поясненням ролі окремих компонентів. 1. Модуль координації: ConversionManager

Цей компонент виконує роль центрального диспетчера обробки, що реалізує так звану «керовану конвеєрну модель» (pipeline). Його основне завдання — координувати виклик окремих служб і керувати порядком виконання основних етапів конвертації.

Основні функції:

- ініціація обробки вхідного файлу (або кількох);

- передача результатів між модулями: структура → валідація → нормалізація;

- контроль завершення процесу та формування зведеного результату; -

виклик логування проміжних і кінцевих статусів.

Завдяки ConversionManager система підтримує сценарій "один вхід — один вихід", а також сценарії пакетної обробки, що робить її гнучкою для різних режимів роботи користувача.

2. Модулі аналізу структури: DbfSchemaAnalyzer

Зчитування структури DBF-файлів є критично важливим кроком, оскільки такі файли часто мають нестандартні імена полів, різну послідовність колонок і нетипові типи даних. Саме цей модуль забезпечує побудову внутрішнього

представлення таблиці, яке далі використовується для валідації та трансформації.

Основні завдання:

- відкриття DBF-файлу та читання його метаінформації;

- побудова абстрактної моделі структури (список полів, типи, порядок); - автоматичне розпізнавання критичних полів (наприклад, formcode, rownum, cellX);

- обробка ситуацій, коли назви полів відрізняються від очікуваних (наприклад, Form_Code замість formcode).

Успішне виконання цього кроку є передумовою для подальшої обробки: якщо структура не піддається аналізу — обробку буде зупинено з відповідним повідомленням у лог.

3. Модуль нормалізації: DataNormalizer

Цей компонент виконує приведення отриманих вхідних даних до уніфікованого шаблону, який є обов'язковим для створення вихідного DBF-файлу. Процес нормалізації охоплює не лише назви полів, але й порядок колонок, типи даних і зміст окремих значень.

Типові дії модуля:

- перейменування полів згідно зі стандартною схемою (наприклад, Сума_Заг → cell1);

- перестановка колонок відповідно до шаблону;

- заповнення відсутніх колонок пустими значеннями (null або "0"); - адаптація типів даних (наприклад, текст → число, дата в рядку → DateTime);

- стандартизація кодувань текстових полів (з використанням EncodingHelper).

Модуль дозволяє використовувати словники відповідності назв, а також враховує специфіку різних форм звітності, які можуть мати унікальні особливості структури (наприклад, форми 2ds і 5ds).

4. Модуль валідації: ValidationEngine

Коректна робота з державними звітами передбачає не лише правильну структуру, але й вмістовну достовірність. Саме цей модуль виконує комплексну перевірку даних на відповідність встановленим правилам.

Основні типи перевірок:

- структурна: наявність обов’язкових полів;

- форматна: відповідність дат формату dd.MM.yyyy, правильність числових значень;

- логічна: взаємозалежність між полями (наприклад, якщо fondcode = 0, то kindcode не може бути 65);

- унікальність: перевірка повторів по rownum;

- змістовна: виявлення підозріло порожніх рядків, де всі cellX = 0. При виявленні помилок система або пропускає відповідні рядки, або помічає їх у журналі помилок для подальшого аналізу користувачем.

5. Модулі доступу до даних: DbfReader та DbfWriter

Ці модулі взаємодіють безпосередньо з файловою системою і забезпечують роботу з DBF-файлами через VFPOLEDB. Для зчитування використовується OleDbDataAdapter, а для запису — OleDbCommand із параметризованими запитами INSERT.

DbfReader:

- відкриває з’єднання до вхідного DBF-файлу;

- формує DataTable на основі SQL-запиту SELECT * FROM [table]; -

забезпечує контроль за кодуванням через параметр Collating Sequence. -

DbfWriter:

- створює нову таблицю зі строго визначеною структурою; - формує

динамічний SQL-запит на створення полів cell1–cell20; - виконує вставку кожного

рядка з перевіркою на коректність типів; - у разі помилки не перериває процес, а

додає запис до лог-файлу. Ці компоненти абстрагують усю специфіку форматів DBF від решти системи, що дозволяє спростити логіку у вищих рівнях.

6. Допоміжні компоненти

- EncodingHelper – для коректної обробки кирилиці та перетворення

кодових сторінок (CP866 → UTF-8 тощо).

- DateParser – для розпізнавання дат у різних форматах.

- Logger – фіксує всі дії системи, помилки, попередження, результати

перевірок.

- ErrorHandler – уніфікована обробка винятків із можливістю виводу

детального повідомлення користувачу.

2.4 Взаємодія компонентів: UML та потік обробки

Для повноцінного розуміння внутрішнього функціонування програмної системи доцільно не лише описати її архітектурні компоненти, але й проаналізувати логіку їхньої взаємодії у динаміці. Саме цей аспект розкривається через моделювання потоку даних та побудову UML-діаграм, які дозволяють візуалізувати ключові етапи процесу обробки DBF-файлів та відображають порядок виклику методів між модулями системи.

Загальний потік обробки даних у системі

Система реалізує послідовну обробку даних за принципом етапного конвеєра (pipeline), де кожен модуль виконує свою функцію і передає результат наступному. У типовому випадку взаємодія відбувається за наступним сценарієм:

1. Користувач у графічному інтерфейсі обирає DBF-файл для обробки; 2. ConversionManager ініціює робочий процес;

3. DbfReader зчитує структуру та вміст вхідного DBF;

4. DbfSchemaAnalyzer аналізує поля та формує внутрішню модель структури;

5. ValidationEngine перевіряє відповідність вмісту стандартним вимогам; 6. DataNormalizer перетворює структуру таблиці у потрібний формат; 7. DbfWriter створює новий DBF-файл на основі уніфікованої структури; 8. Logger фіксує усі події та результати у log-файл;

9. Користувач отримує повідомлення про успішність або помилки процесу.

UML-діаграми в контексті проєктування

Для формалізації цього процесу в роботі використано два типи UML-діаграм: Діаграма варіантів використання (Use Case Diagram)

Ця діаграма відображає зовнішню поведінку системи з погляду користувача. Основні варіанти використання:

- Завантажити DBF-файл;

- Перевірити структуру;

- Запустити конвертацію;

- Зберегти результат;

- Переглянути журнал.

Це дозволяє продемонструвати, що взаємодія користувача відбувається на рівні інтерфейсу, тоді як усі складні обчислювальні та трансформаційні процеси відбуваються у фоновому режимі.

Діаграма послідовності (Sequence Diagram)

Цей тип UML-діаграми описує порядок виклику методів та обмін повідомленнями між об’єктами системи під час одного конкретного сценарію — наприклад, конвертації одного DBF-файлу.

На діаграмі чітко простежується послідовність виклику об’єктів. Кожен з них не тільки виконує свою операцію, але й передає оброблений результат далі по ланцюгу. Наприклад, DbfSchemaAnalyzer передає модель структури ValidationEngine, а вже валідаційний модуль передає очищені й перевірені дані DataNormalizer.

Механізм обміну даними між модулями

Обмін інформацією між компонентами реалізовано через передачу об’єктів типу DataTable, List<FieldModel>, а також структурованих класів-конфігурацій (DTOs). Це забезпечує високу узгодженість даних, уникнення дублікатів і підвищення читабельності коду.

Кожен модуль має свій вхід і вихід у вигляді строго типізованих параметрів, що полегшує тестування, обробку помилок та рефакторинг у майбутньому. Підхід до обробки помилок і журналювання

На всіх етапах роботи система фіксує результати і помилки у log-файл. Для цього використовується спеціалізований модуль Logger, який отримує повідомлення від кожного з функціональних блоків:

- Початок/завершення обробки файлу;

- Кількість оброблених записів;

- Типи знайдених помилок;

- Час виконання кожного етапу.

Завдяки цьому користувач має можливість простежити весь ланцюг дій навіть у разі часткової або повної невдачі процесу.

2.5 Формування структури вихідного DBF-файлу

Один із ключових етапів функціонування програмної системи —

формування уніфікованого вихідного файлу у форматі DBF, який відповідає затвердженим шаблонам звітності, що використовуються в державних установах. Вихідний DBF файл повинен не лише містити всі обов'язкові поля, але й мати строго визначену структуру — як за порядком колонок, так і за їх типами.

Такий підхід забезпечує сумісність з централізованими інформаційними системами, які приймають звіти в автоматичному режимі та вимагають повної відповідності формату.

Принципи формування структури вихідного файлу
Вихідний файл будується на основі уніфікованої схеми, яка:

- однакова для всіх типів звітів;

- фіксована в кількості колонок та їхніх типах;

- не залежить від структури вхідного DBF-файлу;

- розрахована на імпорт у зовнішні автоматизовані системи. Таким чином,

будь-який вхідний файл (незалежно від назв чи порядку полів) нормалізується до одного стандартного вигляду. Детальний опис полів та їх призначення у структурі вихідного уніфікованого DBF-файлу наведено в Таблиці 3.

Таблиця 3 – Структура DBF – таблиці

Поле	Тип	Призначення
idformsimp	Numeric	Унікальний ідентифікатор звітного рядка
orgcode	Character	Код установи
formcode	Character	Ідентифікатор форми
formtype	Character	Тип звітної форми
program	Character	Програмне забезпечення (джерело створення)
datesnput	Date	Дата заповнення звіту

fondcode	Character	Код фонду (пенсійного, соціального тощо)
kindcode	Character	Вид діяльності або категорія
formperiod	Character	Період звітності (напр. «2024-03»)
rownum	Character	Унікальний номер рядка у формі
cell1…cell20	Numeric	Основні показники звіту (до 20 числових значень)

SQL-шаблон для створення таблиці

Для створення нової таблиці застосовується динамічно згенерований SQL запит. Приклад фрагменту такого запиту виглядає так:

```
CREATE TABLE [output_path] (
idformsimp N(10,0),
orgcode C(20),
formcode C(20),
formtype C(20),
program C(50),
datesnput D,
fondcode C(20),
kindcode C(20),
```



```
formperiod C(20),
rownum C(20),
cell1 N(16,2),
cell2 N(16,2),
 ...
cell20 N(16,2)
)
```

Цей SQL-оператор передається у DbfWriter як рядок для виконання через OLE DB-підключення (VFPOLEDB). Формат кожного поля суворо контролюється — особливо числових (N(16,2)), де вказано точність значення.

Особливості реалізації в коді

- Динамічне створення SQL-команди: Назви полів cellX формуються в циклі, залежно від обраної конфігурації форми.

- Типізація: Для кожного поля передбачено суворо визначений тип (Character, Date або Numeric), що відповідає правилам DBF.

- Обробка відсутніх полів: Якщо певні поля у вхідному файлі були відсутні, система автоматично додає їх у вихідний файл з порожніми значеннями або нулями.

- Універсальність: Незалежно від варіацій у вхідних даних, вихідна таблиця завжди має однакову структуру, що дозволяє легко підключати її до наступних етапів автоматизованої обробки (імпорту, аналізу, архівації). Узгодженість із зовнішніми вимогами

Сформований файл повністю відповідає формату, який застосовується в державних інформаційних системах (наприклад, ПФУ, Держстат, Мінсоцполітики тощо). Це дозволяє уникнути ручного редагування перед поданням, зменшити ризики відхилення звіту, а також прискорити цикл подачі/прийняття документації.

2.6 Механізми розширюваності та параметризації

У сучасному програмному забезпеченні особливої уваги заслуговують питання розширюваності — здатності системи адаптуватися до змін у

функціональних або технічних вимогах без повного перепроєктування — та параметризації — можливості налаштовувати поведінку системи без потреби змінювати програмний код. Обидва ці аспекти є критично важливими для розроблюваної системи уніфікації звітних DBF-даних, яка має функціонувати в

умовах постійної зміни форм звітності, регламентів, структур вхідних файлів тощо.

У цьому підрозділі розглянуто механізми, які забезпечують адаптивність та гнучкість програмної системи, а також потенціал її подальшого розвитку з мінімальними витратами на модифікацію.

Гнучкість за рахунок модульної структури

Одним із ключових принципів, реалізованих у системі, є чітке розмежування функціональних блоків за їхньою відповідальністю (див. підрозділ 2.2). Завдяки цьому можлива незалежна модернізація окремих модулів без ризику порушення загальної функціональності системи.

Приклади:

- для підтримки нового формату експорту (наприклад, CSV або Excel)

достатньо додати новий модуль на рівні DataAccessLayer, не змінюючи бізнес логіку;

- для адаптації до змін у назвах полів звітності необхідно оновити лише

шаблон нормалізації, не змінюючи код самого DataNormalizer.

Такий підхід значно спрощує супровід системи в умовах змін, особливо коли зміни стосуються лише зовнішніх форм даних, а не внутрішньої логіки обробки.
Конфігурація через зовнішні файли

Ще одним важливим інструментом адаптації є використання конфігураційних файлів, у яких зберігаються параметри роботи програми. Це дозволяє змінювати поведінку системи без потреби перекомпіляції або втручання в вихідний код.

Реалізовано такі типи параметризації:

1. Шаблони відповідності полів (FieldMapping.json):
Містить таблицю відповідностей між можливими назвами полів у вхідному файлі та їх уніфікованими назвами у вихідному (наприклад, "СумаЗаг" → cell1).

2. Основні налаштування (App.config / Settings.json):

- шлях до вихідної директорії;

- тип обраного експорту (DBF, CSV, XML);

- кількість числових полів (cellCount);

- допустимі діапазони значень;

- правила заповнення нульових рядків.

3. Мовні файли:

Для можливості локалізації інтерфейсу в майбутньому структура UI передбачає підключення мовних ресурсів.

Це робить систему динамічно конфігурованою та зручною для адміністраторів, які можуть змінювати логіку нормалізації й перевірок без втручання в код.

Розширюваність у логіці обробки

Програмна логіка системи реалізована з урахуванням можливості розширення класів через інтерфейси й спадкування, що дозволяє вводити нові варіанти обробки без порушення наявного функціоналу.

Приклади:

- інтерфейс INormalizer дозволяє створити нову реалізацію нормалізатора для спеціалізованих типів звітів (наприклад, форми медичних установ);

- розширення IValidator дозволяє створити нові набори валідаційних правил, які можна підключати у вигляді плагінів або через конфігураційний файл. Це особливо актуально у випадках, коли система використовується в декількох регіонах, де формати звітів можуть частково відрізнятися. Підтримка нових форматів та технологій

Незважаючи на початкову орієнтацію системи на формат DBF, структура дозволяє з часом розширити її функціональність для роботи з іншими форматами:

- CSV/Excel: через реалізацію альтернативного IDataReader / IDataWriter.

- JSON/XML: для подачі звітів через API або інтеграцію з веб платформами.

- Бази даних: підключення до SQL-серверів через Entity Framework або

Dapper для збереження проміжних результатів.

Таким чином, система має потенціал розвитку в бік сервіс-орієнтованої архітектури або хмарної платформи зі збереженням наявної бізнес-логіки.
Обробка непередбачуваних даних

В умовах використання програми у різних державних установах існує ризик появи нетипових файлів із порушенням стандарту. Система містить захищені механізми обробки таких випадків:

- пропуск записів, що не відповідають правилам;

- формування розгорнутого лог-файлу з описом проблем;

- підсвічування проблемних місць у повідомленні користувачу. Це дозволяє

запобігти аварійній зупинці роботи системи навіть за наявності «поганих» вхідних даних.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Використані технології та середовище розробки

Для розробки програмної системи, що виконує уніфікацію звітних даних державних установ на основі DBF-файлів, було обрано перевірене й стабільне середовище, що повністю відповідає сучасним вимогам до розробки прикладного ПЗ з інтерфейсом користувача, доступом до файлів баз даних і можливістю подальшого масштабування.

Основною мовою програмування проєкту є C#, яка входить до складу екосистеми .NET Framework. Саме ця технологія була обрана не випадково: вона забезпечує широкий спектр можливостей для роботи з файлами, формами,

системною взаємодією та базами даних, а також має багаторічну підтримку з боку корпорації Microsoft. Серед переваг C# — висока читаємість коду, підтримка об'єктно-орієнтованого програмування, наявність великої спільноти розробників і наявність багатьох інструментів для налагодження та оптимізації роботи програми.

Проєкт реалізовано у середовищі Microsoft Visual Studio 2019 Community Edition, яке на сьогодні є одним із найпоширеніших і найзручніших інструментів для розробки програм під Windows. Visual Studio забезпечує не лише зручний редактор коду, але й інтерактивний дизайнер форм (Windows Forms Designer), який дозволяє розробнику швидко та наочно створювати інтерфейси, розміщувати елементи управління (кнопки, поля, списки) та одразу бачити результат.

У ході реалізації системи основна увага приділялася тому, щоб програма могла запускатися на типових робочих місцях державних установ, без потреби встановлення складного або дорогого програмного забезпечення. Саме тому було ухвалене рішення на користь .NET Framework 4.7.2 — версії, яка має високу сумісність із різними поколіннями Windows, починаючи від Windows 7, і не вимагає встановлення додаткових серверних компонентів або middleware.

Для зчитування та запису файлів у форматі DBF (dBASE, Visual FoxPro) використано технологію VFPOLEDB (Visual FoxPro OLE DB Provider) — це

офіційний постачальник OLE DB-доступу, що дає змогу працювати з DBF-файлами як зі звичайними таблицями бази даних [1]. Такий підхід забезпечує високий рівень сумісності з тими форматами, які найчастіше застосовуються у державному секторі України, зокрема в галузі соціального захисту, статистики, пенсійного обліку тощо.

Формування SQL-запитів до DBF-таблиць (наприклад, SELECT * FROM [table] або CREATE TABLE ...) здійснюється безпосередньо через об'єкти OleDbConnection та OleDbCommand, які входять до стандартної бібліотеки ADO.NET. Таким чином, обробка файлів відбувається без проміжних перетворень у текстові формати, що дозволяє зберігати точність типів даних і запобігає спотворенням при експортуванні.

Структура коду підтримує логічне розділення на окремі частини або шари: -

рівень інтерфейсу користувача (UI) представлено трьома формами — Form1, Form2, Form3, які відповідають за вибір файлів, відображення логів і запуск процесів обробки;

- рівень бізнес-логіки (BLL), що реалізує перетворення, нормалізацію та

валідацію даних;

- рівень доступу до даних (DAL), що забезпечує підключення до DBF

файлів і виконання запитів.

Такий поділ дозволяє не лише краще організувати код, але й дає змогу легше адаптувати систему до нових вимог у майбутньому: наприклад, додати підтримку інших форматів звітності (XML, CSV), не змінюючи інтерфейс користувача чи ядро системи.

Слід зазначити, що проєкт не використовує сторонніх бібліотек через NuGet або інші менеджери пакетів, що відповідає вимогам до експлуатації програмного забезпечення у державному секторі, де зазвичай заборонено автоматичне завантаження зовнішніх компонентів із мережі. Усі функції реалізовані вручну, з опорою на стандартні можливості C# та .NET, що гарантує стабільність, контрольованість і простоту супроводу системи.

Загалом, обрані технології дозволили реалізувати програмну систему, яка не потребує інсталяції складних серверів, легко розгортається на звичайних

комп’ютерах без спеціального налаштування і має зрозумілий для користувача інтерфейс. Саме ці чинники забезпечують практичну доцільність використання даного програмного продукту в умовах реального державного документообігу.

3.2 Інтерфейс користувача

Одним із ключових елементів зручності будь-якої прикладної програми є її інтерфейс користувача. У межах розробленої системи для уніфікації DBF-звітності особлива увага приділялася створенню інтерфейсу, який би відповідав вимогам

кінцевих користувачів — працівників державних установ, які не завжди мають високий рівень технічної підготовки.

Інтерфейс реалізовано засобами Windows Forms, що є класичною технологією побудови графічних інтерфейсів у середовищі .NET Framework. Даний вибір обумовлений стабільністю роботи, простотою розробки та повною сумісністю з середовищем Windows, яке є домінуючим у більшості державних структур.

Програму побудовано за принципом поділу функціональності на кілька окремих вікон (форм), кожне з яких виконує чітко визначену роль у процесі роботи з даними. Такий підхід дозволяє уникнути перевантаження інтерфейсу зайвими елементами, спрощує логіку навігації та робить систему інтуїтивно зрозумілою навіть для новачків.

У розробленій системі передбачено три основні форми:

- Form1 — початкове вікно, в якому користувач обирає вхідний DBF файл, задає параметри звітності (якщо потрібно) та ініціює процес конвертації. Інтерфейс цієї форми містить кнопку відкриття файлу, текстове поле для виводу шляху до обраного DBF-файлу, та кнопку запуску обробки. У разі помилки при відкритті або обробці файлу користувач одразу бачить повідомлення про проблему.

- Form2 — додаткове вікно, яке відображає вміст завантаженого файлу у вигляді табличної сітки (DataGridView). Завдяки цьому користувач може попередньо переглянути структуру та зміст файлу, переконатися у правильності

обраних даних і побачити можливі порушення, такі як порожні поля чи некоректні значення.

- Form3 — вікно для перегляду результатів обробки та журналу помилок (log-файлу). Тут відображається статистика обробки (кількість рядків, кількість пропущених, виявлені помилки), а також виводиться короткий текстовий звіт. За потреби користувач може зберегти лог або передати його в IT-відділ для подальшого аналізу.

Інтерфейс системи побудовано з урахуванням принципів доступності та лаконічності. Усі елементи управління мають підписи українською мовою, розміщені логічно, відповідно до послідовності дій користувача: від завантаження файлу — до перегляду результату. Кнопки та поля мають фіксований розмір і не перекриваються при зміні роздільної здатності екрану.

Кожна дія користувача супроводжується візуальним зворотним зв’язком — у вигляді повідомлень, підказок або індикаторів стану. Наприклад, у разі відсутності обов’язкового поля rownum у вхідному файлі система виводить попередження ще до запуску конвертації. Подібна поведінка дозволяє уникнути помилок до початку трудомісткої обробки.

Варто зазначити, що інтерфейс не містить жодних прихованих або технічно складних функцій, які могли б призвести до непередбачуваних результатів. Усі кнопки відповідають одній конкретній дії, а конфігураційні параметри або автоматизовані — або ж задаються чітко визначеними списками. Це особливо важливо в умовах експлуатації системи в державних установах, де до стабільності та зрозумілості програмного забезпечення пред’являються високі вимоги.

Таким чином, реалізований інтерфейс користувача забезпечує простий і надійний механізм взаємодії з програмною системою, що дає змогу ефективно використовувати її навіть у віддалених підрозділах без постійної ІТ-підтримки. Завдяки структурованому підходу до побудови форм, дотриманню єдиних стилістичних принципів і передбаченню типових сценаріїв використання, інтерфейс можна вважати достатньо зручним для широкого кола користувачів.

3.3 Рівень доступу до даних (DAL)

Рівень доступу до даних (Data Access Layer, DAL) є критичним компонентом програмної системи, оскільки забезпечує пряме зчитування вхідних DBF-файлів та створення вихідних таблиць у відповідному форматі. Його головне завдання — ізолювати всі операції роботи з файловою базою від решти логіки програми, забезпечити стабільний обмін даними та підтримку уніфікованого каналу взаємодії з файловою системою через технологію OLE DB.

У межах реалізації було використано провайдер VFPOLEDB (Visual FoxPro

OLE DB Provider), що дозволяє здійснювати запити до DBF-файлів як до повноцінних таблиць бази даних [1]. На практиці це дало змогу уникнути ручного парсингу файлів, натомість користуватись знайомими конструкціями SQL, такими як SELECT, CREATE TABLE та INSERT INTO, що значно спростило реалізацію основних сценаріїв обробки.

Усі операції з файлами абстраговано в окремі методи, які реалізовано у вигляді допоміжного класу з відповідними функціями. Наприклад, метод LoadInputTable виконує підключення до каталогу з вхідним DBF-файлом, формує стандартний SQL-запит, зчитує вміст у структуру типу DataTable та передає її в обробку.

Загальний вигляд методу виглядає так:

```
public DataTable LoadInputTable(string filePath)
{
 string connString = $"Provider=VFPOLEDB.1;Data
Source={Path.GetDirectoryName(filePath)};Collating Sequence=General;";  using (var
conn = new OleDbConnection(connString))
 {
 conn.Open();
 string tableName = Path.GetFileNameWithoutExtension(filePath);  var adapter =
new OleDbDataAdapter($"SELECT * FROM {tableName}", conn);  var table = new
DataTable();
```



```
 adapter.Fill(table);
 return table;
 }
}
```

Такий підхід має одразу кілька переваг. По-перше, він забезпечує прозоре зчитування даних без втручання у структуру файлу, з автоматичним визначенням типів полів. По-друге, реалізація відбувається на основі вбудованих засобів .NET без використання сторонніх бібліотек, що відповідає вимогам до безпеки і стабільності у державних організаціях. По-третє, гнучкість підключення до каталогу дозволяє працювати з кількома файлами в межах одного запиту або

циклу.

Аналогічним чином реалізовано метод CreateOutputTable, який динамічно створює нову DBF-таблицю згідно з уніфікованою структурою. SQL-команда формується шляхом послідовного додавання назв і типів полів: *CREATE TABLE output.dbf (*

idformsimp N(10,0),

orgcode C(20),

formcode C(20),

...

cell20 N(16,2)

)

Виконання запиту здійснюється через об'єкт OleDbCommand, який відкриває з'єднання з цільовим каталогом і створює файл таблиці із заданими параметрами. Особливу увагу приділено коректному задання типів полів, оскільки формат DBF вимагає суворої відповідності між оголошенням типу (Character, Numeric, Date) і фактичними даними. Для числових полів використовується формат N(16,2), який забезпечує зберігання чисел із плаваючою комою до 14 знаків до коми та 2 — після неї.

Крім того, для кожного виклику методів DAL передбачено виключення (try catch) з логуванням помилок, що дозволяє фіксувати випадки, коли, наприклад, файл зайнятий іншою програмою, має пошкоджену структуру або містить

невизначені типи. Таке захисне програмування особливо важливе у ситуаціях, коли обробка відбувається автоматично або в пакетному режимі, без безпосередньої участі користувача.

Важливо зазначити, що всі операції читання і запису даних виконуються локально, без підключення до зовнішніх серверів, що є додатковим заходом безпеки в умовах обробки конфіденційних звітних даних.

У результаті, реалізований рівень доступу до даних є достатньо гнучким, розширюваним і безпечним. Він дозволяє працювати з файлами DBF у найбільш типовому форматі для державних установ і формує надійну основу для подальшої логіки валідації, нормалізації та об'єднання звітних даних у єдиний стандарт.

3.4 Бізнес-логіка та нормалізація даних (BLL)

Бізнес-логіка програмної системи виконує центральну роль у процесі уніфікації звітності. Саме на цьому рівні реалізовано перетворення «сирих» даних, зчитаних із вхідних DBF-файлів, у структуровану, стандартизовану форму, що відповідає вимогам центральних державних систем обліку. Цей рівень визначає правила обробки, валідації та формування вихідного набору даних, і забезпечує незалежність основної логіки від інтерфейсу користувача та джерела інформації.

На відміну від рівня доступу до даних, який виконує лише операції читання та запису, бізнес-логіка відповідає на запитання: що вважається правильними даними, як визначити відповідність структурі, які записи потрібно змінити, а які — відкинути, як привести поля до стандартної схеми, тощо.

Основу бізнес-логіки складає клас ConversionManager, який координує весь цикл обробки одного або кількох файлів:

1. Ініціює аналіз структури вхідної таблиці.
2. Передає дані на валідацію.
3. Застосовує правила нормалізації.
4. Викликає методи формування вихідного файлу.
5. Запускає логування результатів.

Окремі частини обробки реалізовані як самостійні модулі, кожен з яких виконує одну чітко визначену функцію. Наприклад:

- DbfSchemaAnalyzer — модуль, який читає структуру вхідної DBF таблиці, визначає наявні поля, їхні типи та послідовність. Він також виконує первинне зіставлення назв полів із шаблоном (наприклад, "Форма", "form_code", "codeform" → formcode). Це важливо, оскільки різні установи можуть формувати таблиці з різними найменуваннями колонок.

- DataNormalizer — компонент, який приводить таблицю до єдиної структури. У разі відсутності потрібного поля — створює його з порожніми значеннями; у разі неправильного типу — перетворює (наприклад, з string на decimal). Він також виконує стандартизацію форматів дат, очищення текстових

полів, упорядкування колонок згідно зі структурою вихідного DBF-файлу.

- InsertData — метод, який у циклі проходить по всіх рядках обробленої таблиці й формує для кожного з них SQL-запит типу INSERT INTO. Перед вставкою здійснюється контроль типів та значень. Якщо значення некоректне (наприклад, текст у числовому полі), воно замінюється на 0 або позначається в логах.

Приклад створення уніфікованого рядка в коді:

```
var insertCommand = new OleDbCommand("INSERT INTO [output.dbf] (formcode,
rownum, cell1, ...) VALUES (?, ?, ?, ...)", connection);
insertCommand.Parameters.AddWithValue("formcode",
normalizedRow["formcode"]); insertCommand.Parameters.AddWithValue("rownum",
normalizedRow["rownum"]); insertCommand.Parameters.AddWithValue("cell1",
Convert.ToDecimal(normalizedRow["cell1"]));
...
insertCommand.ExecuteNonQuery();
```

Особливістю реалізації є гнучкість логіки: нові поля можуть бути додані до схеми без необхідності переписувати весь код, завдяки використанню циклів та конфігураційних списків полів. Це дозволяє швидко адаптувати систему до змін у

регламентах звітності, що, як відомо, відбувається досить часто в державному секторі.

Ще однією важливою рисою є ізольованість логіки. Кожен модуль не залежить від конкретної форми чи зовнішнього вигляду даних, а працює з абстрактними структурами (наприклад, DataTable, Dictionary<string, object>). Такий підхід спрощує тестування, налагодження, а також можливість повторного використання коду в інших застосунках.

Загалом, реалізована бізнес-логіка забезпечує:

- коректну обробку файлів з довільною структурою;

- адаптацію під різні формати назв полів;

- збереження цілісності типів і значень;

- можливість масштабування під нові шаблони звітності.

Таким чином, шар бізнес-логіки виконує роль «розумного посередника» між необробленими файлами та стандартизованим виходом, і є ключовим елементом, який забезпечує відповідність усієї системи її головному призначенню — уніфікації звітних даних.

3.5 Модуль валідації та обробки помилок

Одним із ключових завдань під час обробки звітних файлів є виявлення та обробка помилок, які можуть виникати як у самих даних, так і в ході їхньої трансформації. Особливо це важливо в умовах роботи з DBF-файлами, які можуть формуватись вручну, експортуватися з різного програмного забезпечення або створюватись за застарілими шаблонами. Враховуючи це, в системі реалізовано окремий модуль валідації, який відповідає за перевірку даних перед вставкою у вихідну таблицю, а також механізм обробки помилок, що забезпечує стійку роботу навіть у разі часткових збоїв.

Валідація в системі відбувається на кількох рівнях:

1. Структурна валідація — перевірка наявності обов’язкових полів, таких як rownum, formcode, orgcode. У разі їхньої відсутності файл не допускається до подальшої обробки, а відповідна помилка записується в лог.

2. Форматна валідація — перевірка відповідності типу значення до очікуваного формату. Наприклад, якщо поле cell1 має бути числовим (N(16,2)), а у вхідному файлі в ньому міститься текстове значення («N/A» або пробіли), такий рядок або очищується, або відзначається як помилковий.

3. Логічна валідація — виявлення аномалій у даних, які можуть свідчити про помилки заповнення. Наприклад, рядок із однаковими нулями в усіх cellX полях або рядки без номера (rownum == 0) вважаються підозрілими й можуть бути пропущені або додатково позначені.

Усі ці перевірки реалізовано у класі ValidationEngine, який працює із

кожним записом вхідної таблиці до моменту його нормалізації та перенесення у вихідний файл. Перевірка реалізована у вигляді методів на кшталт:

```
public bool IsValidRow(DataRow row)
{
 if (row["rownum"] == DBNull.Value) return false;
 if (!decimal.TryParse(row["cell1"].ToString(), out _)) return false;
...
 return true;
}
```

Також особливу увагу приділено обробці виключень (exception handling), що забезпечує безпечне продовження роботи програми навіть у разі виникнення критичних помилок. Наприклад, якщо під час зчитування DBF-файлу виявляється заблокований ресурс або файл виявляється пошкодженим, програма не припиняє виконання, а повідомляє користувача через діалогове вікно і заносить інцидент до журналу.

Кожна спроба роботи з даними (читання, запис, перетворення) огорнута в блок try-catch, де фіксується тип помилки, її опис, а також контекст — наприклад, назва оброблюваного рядка або значення поля, що викликало помилку. Це дає змогу в подальшому легко виявити джерело проблеми навіть без доступу до самого файлу.

Усі помилки або попередження записуються до текстового лог-файлу (conversion.log), що формується автоматично при кожному запуску обробки. Формат записів у лог-файлі стандартизовано: кожен рядок містить мітку часу, тип повідомлення (INFO, WARNING, ERROR), опис і, за потреби, додаткові деталі (номер рядка, назва поля).

Приклад запису до лог-файлу:

[2025-03-20 14:12:47] WARNING: Missing field 'formcode' in input file. File skipped.
[2025-03-20 14:13:02] ERROR: Invalid decimal value in cell12, rownum = 73. Value: 'N/A'
[2025-03-20 14:13:04] INFO: File processed successfully. 128 rows inserted, 2 skipped.

Користувач може переглянути лог-файл у вікні результатів або зберегти його у

вигляді окремого документа для подальшого аналізу або передавання до технічної підтримки.

Завдяки реалізації багаторівневої валідації та системи обробки винятків програмна система демонструє стійкість до пошкоджених, неповних або нестандартних звітних файлів, що є звичним явищем у реальних умовах роботи з даними державних установ. Такий підхід дозволяє знизити ризик помилок при централізованій обробці звітності та мінімізувати участь IT-фахівців у ручному аналізі кожного випадку.

3.6 Логування та звітність

У процесі розробки програмної системи особливу увагу було приділено не лише функціональності основних оброблювальних механізмів, але й забезпеченню прозорості процесів, які відбуваються «за лаштунками» під час виконання операцій. З цією метою у програму інтегровано повноцінний механізм логування — автоматичного фіксування важливих подій, помилок, попереджень і підсумкових результатів обробки.

Логування виконує низку важливих функцій:

- дозволяє відстежити хід обробки навіть після завершення роботи

програми;

- дає змогу швидко виявити джерело помилок у разі неправильного

результату;

- створює історію дій користувача, що може бути використана як доказ

належного виконання процедури звітності;

- полегшує технічну підтримку, оскільки дає змогу передати лог-файл IT-

фахівцю без потреби повторно відтворювати ситуацію.

Усі повідомлення фіксуються у файлі conversion.log, який автоматично створюється в тій самій директорії, де зберігається оброблений звіт, або у

спеціальній підпапці logs. Формат лог-файлу обрано максимально простим і читабельним: кожен рядок містить дату і час події, її тип (INFO, WARNING, ERROR) та текст повідомлення, який коротко, але точно описує суть події.

Наприклад:

[2025-03-20 14:05:17] INFO: Starting file conversion: form_1ds_kyiv.dbf [2025-03-20 14:05:19] WARNING: Missing optional field 'cell5'. Filled with default value (0).

[2025-03-20 14:05:23] ERROR: Row #118 — cannot convert 'abc' to decimal in column 'cell12'

[2025-03-20 14:05:30] INFO: Conversion finished. Total rows: 132. Successful: 129. Skipped: 3.

Для реалізації логування у програмі створено окремий модуль Logger, який відповідає за запис повідомлень у файл, сортування за важливістю, а також — при потребі — дублювання ключових помилок у вікні повідомлень. Це дозволяє одночасно вести технічний облік подій і оперативно інформувати користувача про критичні проблеми.

Класифікація повідомлень виглядає наступним чином:

- INFO — загальні відомості про перебіг процесу (початок/завершення обробки, кількість записів, час виконання);

- WARNING — нетипові ситуації, які не заважають обробці, але потребують уваги (наприклад, відсутність неключового поля);

- ERROR — критичні помилки, через які окремі записи або файли не можуть бути оброблені (некоректне значення, неможливість запису, відсутність ключового поля).

На завершення кожної обробки формується підсумковий звіт, який також виводиться в інтерфейсі користувача. У ньому вказується:

- загальна кількість оброблених рядків;

- кількість успішно збережених записів;

- кількість пропущених рядків або тих, що спричинили помилки; -

наявність критичних або незначних попереджень;

- загальний час, витрачений на обробку файлу.

Окремо передбачена можливість збереження логів для зовнішнього аналізу, що дозволяє, наприклад, прикріпити їх до службової записки або звіту про збої, переданого до технічного підрозділу. Користувач також може вручну видалити або очистити старі логи, якщо вважає їх непотрібними, — усі лог-файли є звичайними текстовими .log файлами, які можна переглядати у будь-якому редакторі.

Таким чином, модуль логування та звітності відіграє роль інформаційного "чорного ящика", що дозволяє контролювати та аналізувати роботу програмної системи не лише в момент її використання, але й заднім числом. Це особливо важливо в умовах звітного контролю, де будь-яка помилка може призвести до значних організаційних наслідків, а прозоре ведення журналу подій дозволяє уникнути непорозумінь та забезпечити відповідальність і відтворюваність усіх етапів обробки.

3.7 Модульні тести й перевірка якості коду

Під час розробки програмного забезпечення для державного сектору важливо забезпечити не лише коректну роботу основного функціоналу, але й стійкість, надійність та передбачуваність поведінки системи при змінних вхідних даних або нестандартних ситуаціях. Саме тому одним із важливих етапів реалізації є тестування — процес перевірки логіки роботи окремих частин програми з метою виявлення помилок до моменту впровадження.

У межах розробки системи уніфікації звітності реалізовано набір модульних тестів (unit tests) для ключових компонентів бізнес-логіки, зокрема для класів, що відповідають за нормалізацію структури, перевірку правильності даних, формування SQL-запитів, а також базову перевірку коректності обробки вхідних таблиць.

Основна мета модульного тестування — ізольовано перевірити логіку кожного методу чи модуля незалежно від зовнішніх чинників, таких як графічний інтерфейс, файлове середовище чи інші підсистеми. Для цього створюються окремі тестові проєкти, які використовують штучно сформовані вхідні дані, що дозволяє легко відтворити та контролювати поведінку системи в межах конкретного сценарію.

Зокрема, були протестовані наступні аспекти:

- правильність перетворення типів (наприклад, з рядка у десяткове число);

- поведінка при відсутності обов'язкових полів;

- перевірка структури нормалізованої таблиці після обробки; -

імітація помилкових значень (null, "", N/A) у критичних полях; -

відповідність вихідної схеми стандарту шаблону DBF.

Приклад простого модульного тесту для методу перевірки значення в числовому полі:

```
[TestMethod]
public void Should_ReturnFalse_When_InvalidDecimal()
{
 var row = new Dictionary<string, object> { { "cell1", "abc" } };
var result = ValidationEngine.IsValidDecimal(row["cell1"]);
 Assert.IsFalse(result);
}
```

Тестування здійснювалося у вбудованому середовищі MSTest, що входить до складу Visual Studio, і дозволяє автоматично виконувати всі тести при кожному

оновленні проєкту. Це забезпечує так звану регресійну перевірку — контроль того, що зміни у коді не порушили вже реалізовану функціональність. Окрему увагу приділено перевірці граничних випадків, які часто стають джерелом

несподіваних помилок у реальних умовах:

- обробка порожніх таблиць;

- присутність однакових rownum;

- надлишкова кількість колонок, не передбачених стандартною схемою; -

поля з некоректними датами або значеннями типу DBNull. Крім модульних тестів, здійснювалась ручна перевірка інтерфейсу

користувача та сценаріїв «повної обробки» — від вибору вхідного файлу до створення вихідного результату. Це дозволило виявити не лише технічні, але й логічні помилки, пов’язані з послідовністю дій, зручністю інтерфейсу та поведінкою програми у нестандартних ситуаціях.

Хоча система не використовує складні CI/CD-платформи (через її автономний характер і відсутність публічного розгортання), сам підхід до тестування побудовано з урахуванням майбутнього розширення. Усі класи написано з дотриманням принципів розділення обов’язків, що спрощує тестування і дозволяє при потребі легко масштабувати набір тестів.

Загалом, реалізація модульного тестування значно підвищує якість і надійність коду, дозволяє виявити помилки на ранніх етапах та забезпечує упевненість у тому, що навіть при зміні окремих частин системи її загальна працездатність залишиться стабільною.

3.8 Оптимізація продуктивності та масштабованість

У розробці прикладного програмного забезпечення для обробки великого обсягу звітних даних особливе значення має продуктивність — тобто здатність системи обробляти великі обсяги інформації з мінімальними витратами часу та ресурсів. Це особливо актуально у випадках, коли програма використовується в централізованих структурах, де щомісяця необхідно обробляти десятки або сотні DBF-файлів, кожен з яких може містити тисячі записів.

Зважаючи на це, в ході реалізації системи особливу увагу було приділено оптимізації основних алгоритмів, а також забезпеченню масштабованості — тобто

здатності програмного продукту працювати з більшими обсягами даних без втрати стабільності чи суттєвого зниження продуктивності.

Під час профілювання було ідентифіковано ключові «вузькі місця» — насамперед, це циклічна обробка кожного рядка вхідної таблиці при конвертації, а також створення SQL-запитів INSERT для кожного окремого запису. Щоб уникнути зайвих витрат ресурсів, було впроваджено низку рішень:

Попереднє зчитування всієї таблиці у пам'ять у вигляді об'єкта DataTable, що дозволило працювати з даними без повторного звернення до файлової системи. Використання параметризованих запитів, які формуються один раз, після чого лише змінюються значення параметрів у кожному проході циклу. Це зменшило навантаження на інтерпретатор SQL і пришвидшило вставку даних. Локальна обробка помилкових записів без припинення основного процесу. Завдяки цьому програма не зупиняється при першій помилці, а продовжує обробку, пропускаючи лише проблемні рядки.

Під час тестування зразкових файлів було встановлено, що середній час обробки 10 000 записів становить приблизно 2–3 секунди на типовому офісному комп'ютері (Intel i5, 8 ГБ ОЗУ). Це свідчить про те, що система здатна працювати в умовах реального навантаження без необхідності у спеціальному технічному забезпеченні.

Крім продуктивності, реалізація враховувала також можливість масового оброблення кількох файлів. Хоча інтерфейс користувача призначений для роботи з одним файлом за раз, структура коду передбачає можливість виклику основного механізму у циклі, що дозволяє обробляти папку з кількома DBF-файлами в автоматичному режимі (наприклад, із застосуванням скрипту або консолі). Це створює передумови для майбутнього розширення функціоналу до повноцінного пакетного режиму.

Для підвищення масштабованості також:

- максимально уникалося збереження зайвих копій таблиць у пам'яті;

- забезпечено очищення тимчасових об'єктів після завершення кожного етапу обробки;

- створено гнучкий механізм роботи з файлами: будь-який розмір, будь яка кількість полів, за умови відповідності базовому формату.

Окремо варто відзначити, що система не використовує важких або ресурсомістких компонентів (наприклад, візуалізації графіки, складних мережевих підключень, сторонніх бібліотек), що робить її придатною для встановлення на малопотужні комп’ютери — зокрема, у віддалених філіях державних органів, де технічні ресурси обмежені.

Таким чином, застосування ряду практичних рішень щодо оптимізації та архітектурна гнучкість дозволили забезпечити високу продуктивність та адаптивність системи, що, у свою чергу, створює передумови для її успішного використання у широкому масштабі.

РОЗДІЛ 4
ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ

4.1 Підготовка до впровадження

Перед початком експлуатації розробленої системи уніфікації DBF-файлів необхідно здійснити комплекс підготовчих заходів, що забезпечать її коректне та ефективне функціонування. Перш за все, важливо провести ретельний аналіз існуючої IT-інфраструктури установи, включаючи оцінку апаратних ресурсів (наприклад, потужність серверів або робочих станцій, обсяг доступної пам'яті) та програмного забезпечення (операційні системи, СУБД), щоб переконатися у їх відповідності мінімальним вимогам системи.

Наступним кроком є налаштування необхідного програмного забезпечення, що може включати встановлення компонентів .NET Framework, якщо вони ще не встановлені, а також встановлення драйверів VFPOLEDB або інших бібліотек для забезпечення доступу до DBF-файлів. Важливим етапом є підготовка тестових даних, тобто набір DBF-файлів, які представляють різні форми звітності, що використовуються в установі. Це дозволить провести всебічне тестування системи та перевірити коректність її роботи з реальними даними.

І, нарешті, необхідно забезпечити належну підготовку кінцевих користувачів

системи. Це передбачає розробку детальних інструкцій користувача, проведення навчальних семінарів або тренінгів, щоб користувачі могли ефективно встановлювати, налаштовувати та використовувати систему у своїй повсякденній роботі.

4.2 Процес впровадження

Процес впровадження розробленої системи уніфікації DBF-файлів є критичним етапом, який визначає успіх її подальшої експлуатації. Існує кілька підходів до впровадження, і вибір конкретного залежить від таких факторів, як розмір установи, складність існуючих процесів обробки звітності та наявність ресурсів. Одним з можливих підходів є реалізація пілотного проекту. Це

передбачає розгортання системи в обмеженому масштабі, наприклад, в окремому підрозділі або для невеликої групи користувачів. Пілотний проект дозволяє протестувати систему в реальних умовах, виявити потенційні проблеми або недоліки та внести необхідні корективи перед повномасштабним впровадженням. Іншим підходом є поетапне розгортання, коли система впроваджується поступово в різних підрозділах або для різних типів звітності. Такий підхід забезпечує більш плавний перехід до нової системи, дозволяє надати належну підтримку та навчання користувачам на кожному етапі та мінімізує ризики. Нарешті, можливе одномоментне розгортання, коли система впроваджується одночасно у всій установі. Цей підхід є найбільш швидким, але вимагає ретельного планування, координації та підготовки, щоб забезпечити безперебійну роботу всіх процесів. Незалежно від обраного підходу, важливо забезпечити чітку комунікацію між розробниками, IT-відділом та кінцевими користувачами, а також надати достатньо ресурсів для навчання та підтримки.

4.3 Інструкція користувача

Розроблена система уніфікації DBF-файлів є інтуїтивно зрозумілим інструментом, що дозволяє користувачам легко конвертувати звітні дані у єдиний формат. Для забезпечення ефективної роботи з системою, нижче наведено детальну інструкцію з її використання.

Система не потребує складної процедури встановлення. Для початку роботи достатньо скопіювати виконуваний файл програми на комп'ютер користувача. Переконайтеся, що на комп'ютері встановлена необхідна версія .NET Framework (якщо це необхідно).

Перед початком конвертації користувачу необхідно налаштувати певні параметри:

- Вибір вхідних файлів: За допомогою інтерфейсу програми користувач обирає DBF-файли, які потрібно конвертувати. Система підтримує імпорт файлів різних форм звітності (1ds–5ds).

- Вибір вихідного файлу: Користувач вказує місцезнаходження та назву для збереження вихідного DBF-файлу, в який буде записано уніфіковані дані. - Налаштування відповідності полів: (Цей пункт може бути присутнім, якщо у вашій системі є можливість налаштування відповідності між полями вхідних та вихідних файлів). За потреби, користувач може налаштувати відповідність між полями вхідних DBF-файлів та полями єдиної структури (idformsimp, orgcode, formcode, formtype, program, datesnput, fondcode, kindcode, formperiod, rownum, cell1…cell20).

Після налаштування необхідних параметрів користувач запускає процес конвертації. Система автоматично зчитує дані з вхідних DBF-файлів, перетворює їх відповідно до заданої структури та зберігає результат у вихідному DBF-файлі.

У випадку виникнення помилок під час конвертації система виводить інформативні повідомлення, що дозволяють користувачу швидко ідентифікувати та усунути проблему. Можливі помилки включають:

- Некоректний формат вхідного файлу.

- Відсутність необхідних прав доступу до файлів.

- Помилки під час перетворення даних.

Ця інструкція надає загальний огляд використання системи. Залежно від конкретної реалізації вашої системи, можуть знадобитися додаткові пояснення або скріншоти для ілюстрації процесу.

ВИСНОВКИ

Кваліфікаційна робота присвячена розробці програмного забезпечення для автоматизованої обробки та уніфікації звітних даних у форматі DBF, що активно використовується в державних установах. Основною метою дослідження було створення універсальної системи, здатної забезпечити зчитування DBF-файлів із різною структурою, їх валідацію, нормалізацію та експорт у єдиний уніфікований шаблон для подальшого використання в централізованих інформаційних системах.

Проблематика, що лежить в основі дослідження, стосується великої кількості типових помилок і ускладнень, пов'язаних із ручною обробкою звітів: різнорідність форматів, відсутність єдиного стандарту, помилки в кодуванні та даних, складність перевірки й інтеграції. Розроблена система на базі мови програмування C# з використанням технології VFPOLEDB/OleDb дозволяє ефективно вирішити ці задачі, забезпечивши адаптивне перетворення звітних файлів у формат, що відповідає затвердженій структурі з полями типу idformsimp, orgcode, formcode, formtype, datesnput, fondcode, formperiod, rownum, cell1–cell20 тощо.

Система забезпечує автоматичну перевірку даних на коректність, включаючи валідацію типів, правильність форматів дат, відповідність кодувань, перевірку унікальності та логічної узгодженості. Механізм ведення журналів помилок і попереджень дозволяє оперативно виявляти та виправляти проблеми, що значно підвищує надійність підсумкової звітності. Інтерфейс, реалізований за допомогою Windows Forms, є інтуїтивно зрозумілим і не вимагає спеціальної технічної підготовки користувача, що особливо важливо для застосування у сфері держслужби.

У рамках дипломної роботи було досягнуто всі поставлені цілі: проведено аналіз формату DBF, визначено основні проблеми його обробки, спроєктовано багаторівневу архітектуру системи, розроблено програмні модулі для обробки,

перевірки та трансформації даних, проведено тестування на реальних прикладах

звітних файлів. Результати тестування підтвердили працездатність системи, її стабільність, коректність обробки та економію часу при масовій обробці даних. Разом з тим, у ході реалізації було виявлено ряд обмежень. Найсуттєвіші з них — це залежність від драйвера VFPOLEDB, обмеження платформи (переважна підтримка Windows), складність у розпізнаванні сильно відмінних структур без наявності шаблонів, а також відсутність повноцінної підтримки memo-полів. Для подальшого вдосконалення системи доцільно передбачити підтримку інших форматів даних (наприклад, CSV, XML), реалізацію онлайн-доступу через веб інтерфейс, інтеграцію з державними сервісами подання звітності, а також впровадження механізмів машинного навчання для автоматичного розпізнавання типових структур.

Таким чином, дана робота демонструє значний потенціал програмної автоматизації обробки звітних даних у державному секторі. Розроблена система дозволяє підвищити точність, швидкість та зручність обробки звітних DBF-файлів, що є важливим етапом у процесі цифровізації державного документообігу. Отримані результати можуть бути впроваджені як у локальних відділеннях держустанов, так і у масштабніших проектах із централізації звітності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Microsoft Corporation. Using the OLE DB Provider for Visual FoxPro. [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en us/sql/ado/guide/data/using-ole-db-provider-for-visual-foxpro (дата звернення: 25.05.2025)

2. DBFViewer.com. DBF Viewer & Editor – онлайн-інструменти та документація щодо роботи з DBF. [Електронний ресурс] – Режим доступу: https://dbfviewer.com/ (дата звернення: 25.05.2025)

3. Habr. История и особенности формата DBF. [Електронний ресурс] – Режим доступу: https://habr.com/ru/articles/456476/ (дата звернення: 25.05.2025) 4.

Вікіпедія – вільна енциклопедія. DBF (формат файлу). [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/DBF (дата звернення: 25.05.2025)

5. Stack Overflow. Практичні приклади роботи з DBF-файлами за тегом «dbf». [Електронний ресурс] – Режим доступу: https://stackoverflow.com/questions/tagged/dbf (дата звернення: 25.05.2025)

6. Amin, Hesham A. Read/write DBF files using C#. CodeProject. [Електронний ресурс] – Режим доступу: https://www.codeproject.com/Articles/21379/Read-write-DBF-files-using-C (дата звернення: 25.05.2025)

7. FileFormat.com. DBF File Format – Специфікація DBF-файлів. [Електронний ресурс] – Режим доступу: https://docs.fileformat.com/database/dbf/ (дата звернення: 25.05.2025)

8. Костров С. В. Архітектура програмного забезпечення: навч. посібник. – Київ: Видавництво Ліра-К, 2020. – 248 с.

9. Гамма Е., Хелм Р., Джонсон Р., Влісідес Дж. Шаблони проектування. – К.: Діалектика, 2002. – 366 с.

10. Microsoft Corporation. Платформа .NET. [Електронний ресурс] – Режим доступу: https://dotnet.microsoft.com/ (дата звернення: 25.05.2025) 11. NuGet. NuGet Gallery – Бібліотеки для C# і .NET. [Електронний ресурс] – Режим доступу: https://nuget.org/ (дата звернення: 25.05.2025)

ДОДАТОК

```
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 database_conversion
{
public partial class Form1 : Form
{
public Form1()
{
InitializeComponent();
}

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

private void вихідToolStripMenuItem_Click_1(object sender, EventArgs e)  {
this.Close();
}
```



```
private void відкритиБДToolStripMenuItem_Click(object sender, EventArgs e)  {
Form2 newForm = new Form2();
newForm.Show();
}

private void конвертуватиБДToolStripMenuItem_Click(object sender, EventArgs  e)
{
Form3 newForm = new Form3();
newForm.Show();
}
}
```

```
}

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

namespace database_conversion
{
 public partial class Form2 : Form
```



```
 {
 public Form2()
 {
 InitializeComponent();
 }

 private void button1_Click(object sender, EventArgs e)
 {
 openFileDialog1.Filter = "DBF файли (*.dbf)|*.dbf|Усі файли (*.*)|*.*";  if
(openFileDialog1.ShowDialog() == DialogResult.OK)
 {
 string dbfFilePath = openFileDialog1.FileName;
 string dbfDirectory = System.IO.Path.GetDirectoryName(dbfFilePath);  string
dbfFileName = System.IO.Path.GetFileName(dbfFilePath);
```

```
string connStr = $"Provider=VFPOLEDB.1;Data  Source={dbfDirectory};Collating
Sequence=machine;";

using (OleDbConnection conn = new OleDbConnection(connStr))  {
try
{
conn.Open();
string query = $"SELECT * FROM {dbfFileName}";  OleDbDataAdapter adapter
= new OleDbDataAdapter(query, conn);  DataTable dt = new DataTable();
adapter.Fill(dt);

dataGridView1.DataSource = dt;
}
```



```
catch (Exception ex)
{
MessageBox.Show("Помилка: " + ex.Message, "Помилка",  MessageBoxButtons.OK,
MessageBoxIcon.Error);
}
}
}
}
}
}

using System;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.IO;
using System.Linq;
```

```
using System.Windows.Forms;

namespace database_conversion
{
 public partial class Form3 : Form
 {
 public Form3()
 {
 InitializeComponent();
 textBox2.Text = DateTime.Now.ToString("dd.MM.yyyy");  }

 private void button1_Click(object sender, EventArgs e)
```



```
 {
 OpenFileDialog openDialog = new OpenFileDialog { Filter = "DBF файли
(*.dbf)|*.dbf" };
 if (openDialog.ShowDialog() != DialogResult.OK) return;
 var input = LoadInputTable(openDialog.FileName);

 SaveFileDialog saveDialog = new SaveFileDialog { Filter = "DBF файли
(*.dbf)|*.dbf", FileName = "NewDatabase.dbf" };
 if (saveDialog.ShowDialog() != DialogResult.OK) return;
 string outPath = saveDialog.FileName;
 string outName = Path.GetFileNameWithoutExtension(outPath);
string outDir = Path.GetDirectoryName(outPath);

 CreateOutputTable(outName, outDir);
 InsertData(input, textBox1.Text.Trim(), outName, outDir);

 MessageBox.Show("Дані успішно перенесено.", "Успіх",  MessageBoxButtons.OK,
MessageBoxIcon.Information);
```

```
}

private void CreateOutputTable(string tableName, string folder)  {
string connStr = $"Provider=VFPOLEDB.1;Data Source={folder};";  using
(var conn = new OleDbConnection(connStr))
{
conn.Open();
try { new OleDbCommand($"DROP TABLE {tableName}",
conn).ExecuteNonQuery(); } catch { }
string sql = $"CREATE TABLE {tableName} (" +
```



```
"idformsimp N(10,0), " +
"orgcode C(20), " +
"formcode C(20), " +
"formtype C(20), " +
"program C(50), " +
"datesnput D, " +
"fondcode C(20), " +
"kindcode C(20), " +
"formperiod C(20), " +
"rownum C(20), " +
string.Join(", ", Enumerable.Range(1, 20).Select(i => $"cell{i} N(16,2)")) +  ")";
new OleDbCommand(sql, conn).ExecuteNonQuery();
}
}

private DataTable LoadInputTable(string filePath)
{
string dir = Path.GetDirectoryName(filePath);
string name = Path.GetFileNameWithoutExtension(filePath);  string connStr =
```

```
$"Provider=VFPOLEDB.1;Data Source={dir};Collating  Sequence=machine;";
 var dt = new DataTable();
 using (var conn = new OleDbConnection(connStr))
 {
 conn.Open();
 new OleDbDataAdapter($"SELECT * FROM {name}", conn).Fill(dt);  }
 return dt;
 }
```



```
 private void InsertData(DataTable input, string formCode, string outName, string
outDir)
 {
 string destConn = $"Provider=VFPOLEDB.1;Data Source={outDir};";  using
(var conn = new OleDbConnection(destConn))
 {
 conn.Open();
 bool switchedToSecondPart = false;
 int emptyField1Count = 0;
 int form2dsPart = 1;
 int rowIndex = 0;

 foreach (DataRow row in input.Rows)
 {
 rowIndex++;
 if (formCode.Equals("2ds", StringComparison.OrdinalIgnoreCase) &&  rowIndex <= 5)
 continue;

 var vals = row.ItemArray;
 if (vals.Length < 2) continue;
```

```
string targetRowNum = "", targetFormCode = formCode;  int
startIndex = 2;

if (formCode.Equals("1ds", StringComparison.OrdinalIgnoreCase))  {
bool isField1Empty = vals[1] == DBNull.Value ||
string.IsNullOrWhiteSpace(vals[1].ToString());
```



```
emptyField1Count = isField1Empty ? emptyField1Count + 1 : 0;

if (emptyField1Count >= 2)
switchedToSecondPart = true;

targetRowNum = vals[1]?.ToString()?.Trim() ?? string.Empty;  if
(string.IsNullOrWhiteSpace(targetRowNum)) continue;

targetFormCode = switchedToSecondPart ? "1p" : "1a";  }
else if (formCode.Equals("2ds", StringComparison.OrdinalIgnoreCase))  {
string rawRownum = vals[1]?.ToString()?.Trim() ?? "";  emptyField1Count =
string.IsNullOrWhiteSpace(rawRownum) ?  emptyField1Count + 1 : 0;

if (emptyField1Count >= 2)
form2dsPart = Math.Min(form2dsPart + 1, 4);

if (!System.Text.RegularExpressions.Regex.IsMatch(rawRownum,  @"^\d{4}$"))
continue;

targetRowNum = rawRownum;
targetFormCode = $"2ds{form2dsPart}";
}
else if (formCode.Equals("3ds", StringComparison.OrdinalIgnoreCase) ||
formCode.Equals("4ds", StringComparison.OrdinalIgnoreCase))  {
```

```
targetRowNum = vals[1]?.ToString()?.Trim() ?? string.Empty;
```



```
if (string.IsNullOrWhiteSpace(targetRowNum)) continue;

targetFormCode = formCode;
}
else if (formCode.Equals("5ds", StringComparison.OrdinalIgnoreCase))  {
string field1 = vals[1]?.ToString()?.Trim() ?? "";
string field2 = vals.Length > 2 ? vals[2]?.ToString()?.Trim() ?? "" : "";

bool field1IsNumber = decimal.TryParse(field1, out _);  bool
field2HasBracketedNumber =  System.Text.RegularExpressions.Regex.IsMatch(field2,
@"^\(\d+\)$");  bool field1IsLetter =
System.Text.RegularExpressions.Regex.IsMatch(field1, @"[a-zA-Z]");  bool
field2IsLetter =  System.Text.RegularExpressions.Regex.IsMatch(field2, @"[a-zA-Z]");

if (field1IsNumber)
{
targetRowNum = field1;
}
else if (!field1IsNumber && field2HasBracketedNumber &&  !field1IsLetter)
{
targetRowNum =  System.Text.RegularExpressions.Regex.Match(field2,
@"\d+").Value;  startIndex = 3;
}
else
{
```



```
continue;
}

targetFormCode = formCode;
```

```
}
else
{
targetRowNum = vals[1]?.ToString()?.Trim() ?? string.Empty;  if
(string.IsNullOrWhiteSpace(targetRowNum)) continue;  }

var cols = new List<string> { "idformsimp", "orgcode", "formcode",  "formtype",
"program", "datesnput", "fondcode", "kindcode", "formperiod", "rownum"  };
cols.AddRange(Enumerable.Range(1, 20).Select(i => $"cell{i}"));  string sql =
$"INSERT INTO {outName} ({string.Join(",", cols)}) VALUES  ({string.Join(",",
cols.Select(_ => "?"))})";

using (var cmd = new OleDbCommand(sql, conn))
{
cmd.Parameters.AddWithValue("idformsimp", 0);
cmd.Parameters.AddWithValue("orgcode", "013705");
cmd.Parameters.AddWithValue("formcode", targetFormCode);
cmd.Parameters.AddWithValue("formtype", "15");
cmd.Parameters.AddWithValue("program", textBox3.Text);
DateTime.TryParse(textBox2.Text, out DateTime dtOut);
cmd.Parameters.AddWithValue("datesnput", dtOut);
cmd.Parameters.AddWithValue("fondcode",
comboBox1.SelectedItem?.ToString() ?? string.Empty);
```



```
cmd.Parameters.AddWithValue("kindcode",
comboBox2.SelectedItem?.ToString() ?? string.Empty);
cmd.Parameters.AddWithValue("formperiod",
comboBox3.SelectedItem?.ToString() ?? string.Empty);
cmd.Parameters.AddWithValue("rownum", targetRowNum);

for (int i = 1; i <= 20; i++)
```

```
{
int srcIndex = formCode.Equals("1ds",  StringComparison.OrdinalIgnoreCase) ? i :
startIndex + i - 1;
decimal cellValue = 0;
if (srcIndex < vals.Length && vals[srcIndex] != DBNull.Value)  {
var raw = vals[srcIndex].ToString();
if (decimal.TryParse(raw, out decimal parsed))  cellValue =
parsed;
}
cmd.Parameters.AddWithValue($"cell{i}", cellValue);  }

cmd.ExecuteNonQuery();
}
}
}
}
}
}

namespace database_conversion
{
```

КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
ДЕРЖАВНОГО НЕКОМЕРЦІЙНОГО ПІДПРИЄМСТВА
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

РЕЦЕНЗІЯ

на кваліфікаційну роботу

випускника спеціальності: 123 «Комп'ютерна інженерія»

відділення: комп'ютерної та програмної інженерії

циклова комісія: комп'ютерних систем та мереж

Владислав ПАВЛЮК
(ім'я, прізвище)

1. *Кваліфікаційна робота присвячена актуальній темі — автоматизації обробки звітних DBF-файлів, що широко використовуються в державному секторі. У роботі грамотно сформульовано мету й завдання дослідження, проведено глибокий аналіз формату DBF та практичних труднощів його використання.*
2. *Зміст роботи викладено логічно, чітко структуровано. Особливої уваги заслуговує технічне рішення – розробка програмного забезпечення мовою C# з використанням технології VFPOLEDB. Інтерфейс реалізовано на основі Windows Forms, що забезпечує зручність і доступність для кінцевого користувача.*
3. *До позитивних сторін слід віднести – адаптивну обробку даних різної структури, вбудовану валідацію, створення системи з врахуванням вимог держустанов, реалістичне економічне обґрунтування та практичне тестування.*
4. *Недоліки, пов'язані з обмеженнями платформи й драйвера VFPOLEDB, не знижують загальної якості роботи, а навпаки — демонструють вміння здобувача освіти критично оцінювати результати й визначати напрямки для подальшої модернізації проєкту.*
5. *Кваліфікаційна робота заслуговує оцінку «відмінно».*

Рецензент

викладач
(науковий ступінь, посада)

«____»__________ 2025 р. ____________ Тетяна НОВІК
(підпис) (ім'я, прізвище)

З рецензією ознайомлений ____________ Владислав ПАВЛЮК
(підпис) (ім'я, прізвище)