

Детектор Плагіату v. 2961 - Звіт оригінальності: 15.06.2026 9:13:06

Перевірений документ: Бак_пояснювальна_Клименко (1).docx Ліцензія: ВОЛОДИМИР МАТІЄВСЬКИЙ

? Тип пошуку: Пошук Перепису ? Знайдено мову: Uk

? Тип перевірки: Інтернет-перевірка

ТЕЕ і кодування: DocX n/a

Детальний аналіз документа документа:

? Співвідношення:

Плагіат 0% Оригінал 99.87%
Цитати 0.13%



? Графік розподілу:



? Джерела плагіату: 2

- 0% 1 1. <https://ukrreferat.com/chapters/komputerny-nauki/bazi-danih-referat.html>
0% 1 2. https://kkite.cnu.edu.ua/wp-content/uploads/sites/50/2021/02/KI_B_BK12-Системи-керув...

? Дані оброблених ресурсів: 199 - ОК / 11 - Помилка

? Важливі примітки:

Wikipedia:	Google Books:	Сервіси платних робіт:	Античіт:
[не знайдено]	[не знайдено]	[не знайдено]	Знайдена протидія!

? Звіт проти обману UACE:

- Статус: Аналізатор **Увімкнений** Нормалізатор **Увімкнений** схожість символів встановлено **100%**
- Виявлений відсоток забруднення UniCode: **11,3%** з обмеженням: 4%
- Відсоток невизнаних символів після нормалізації: **6,8%**
- Усі підозрілі символи будуть позначені фіолетовим кольором: **Abcd...**
- Знайдено невидимі символи: 0

Рекомендація з оцінки:

Аналізу цього звіту слід приділити особливу увагу! Підозрюється, що цей документ містить значну кількість символів, чужих мові документа. Це є прямим вказівкою на те, що автор документа використав спеціальне програмне забезпечення онлайн -веб -сервіс, щоб ефективно заплутати текст, намагаючись уникнути потенційного виявлення плагіату. Наполегливо рекомендується ескалювати цю справу! У разі сумнівів зверніться до служби підтримки детектора плагіату!

Статистика алфавіту та аналіз символів:

?

Активні посилання (URL-адреси, витягнуті з документа):

URL не знайдені

?

Виключено:

URL не знайдені

?

Включено:

URL не знайдені

Детальний аналіз документу:

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ДЗ

Знайдено посилань: 0,01%

id: 1

«ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА»

Навчально-науковий факультет математики (назва факультету, інституту) Кафедра інформаційних технологій та систем (назва кафедри) Пояснювальна записка до кваліфікаційної роботи за першим (бакалаврським) рівнем освіти на тему: Перенесення [HRM](#) на сучасні ОС та її модернізація Виконав: здобувач вищої освіти 4 курсу спеціальності [F2](#)

Знайдено посилань: 0%

id: 2

«Інженерія програмного забезпечення»

(шифр і назва напряму підготовки, спеціальності) _____ Віталій КЛИМЕНКО (прізвище та ініціали) Керівник _____ Микола СЕМЕНОВ (прізвище та ініціали) Рецензент _____ Владислав ІЩЕНКО (прізвище та ініціали) Полтава – 2026 ЗМІСТ ВСТУПЗ РОЗДІЛ 1. АНАЛІЗ ІНТЕРФЕЙСУ ЗАСТАРІЛОЇ [HRM](#)-СИСТЕМИ ТА СТРУКТУРИ БД6 1.1 Аналіз інтерфейсу застарілої [HRM](#)-системи6 1.2 Аналіз структури бд застарілої [HRM](#)-системи11 РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ26 2.1 Архітектурна модель програми26 2.2 Вибір інструментів для розробки, вибір СУБД та САБД34 РОЗДІЛ 3. РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ТА ЙОГО ТЕСТУВАННЯ40 3.1 Процес реалізації [Windows-Forms](#) застосунку40 3.2 Опис функціоналу застосунку48 3.3. Налаштування та тестування52 ЗАГАЛЬНІ ВИСНОВКИ55 СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ56 ДОДАТОК А.58 ДОДАТОК Б.59 ВСТУП Сучасний етап розвитку інформаційних технологій характеризується стрімким оновленням операційних систем, інструментів розробки та архітектурних підходів до створення програмного забезпечення. Багато підприємств та організацій продовжують використовувати застарілі системи управління людськими ресурсами ([HRM](#)), які були розроблені для старих операційних систем і не відповідають сучасним вимогам щодо безпеки, продуктивності, зручності використання та інтеграції з іншими корпоративними системами. Такі системи часто працюють на платформах, які більше не підтримуються виробниками, що створює суттєві ризики для стабільності роботи організації та захисту конфіденційних даних співробітників. Перенесення [HRM](#)-систем на сучасні операційні системи, зокрема [Windows 10](#), є актуальною задачею для багатьох українських підприємств. Це дозволяє не лише забезпечити сумісність із сучасним програмним та апаратним забезпеченням, але й впровадити сучасні методи модернізації інтерфейсу, оптимізувати роботу з базою даних, покращити безпеку та надійність системи. Особливу важливість має збереження всіх існуючих даних та функціональності під час міграції, а також забезпечення інтуїтивно зрозумілого інтерфейсу для користувачів, які можуть не мати глибоких технічних знань. На сьогодні існує низка підходів до міграції застарілих програмних систем, включаючи повну переписування коду, рефакторинг існуючої кодової бази, використання емуляторів або віртуалізацію старих середовищ. Однак більшість із цих підходів мають суттєві недоліки: високу вартість, тривалі терміни реалізації, втрату функціональності або зниження продуктивності. Особливо складним є перенесення систем, які використовують застарілі технології розробки, такі як старі версії мов програмування, застарілі фреймворки або специфічні архітектурні рішення. Мета роботи – перенесення застарілої [HRM](#)-системи на сучасні ОС та її модернізація. Для досягнення поставленої мети визначено завдання: провести аналіз інтерфейсу застарілої [HRM](#)-системи та визначити його основні недоліки та обмеження; дослідити структуру бази даних застарілої [HRM](#)-системи та оцінити її відповідність сучасним вимогам; спроектувати архітектурну модель нової [HRM](#)-системи з урахуванням вимог до сучасних операційних систем; здійснити вибір інструментів для розробки, СУБД та САБД, які забезпечать сумісність із [Windows 10](#) та високу продуктивність системи; реалізувати [Windows Forms](#)-застосунок відповідно до проєктованої архітектури та сформованих вимог; описати функціонал застосунку та забезпечити його зручність для кінцевих користувачів; провести налаштування та тестування розробленого застосунку для забезпечення його стабільності та надійності. Об'єктом дослідження є [HRM](#)-система. Предметом дослідження є технологія перенесення застарілих [HRM](#)-систем на сучасні ОС ([Windows 10](#)). Методи дослідження – теоретичний аналіз предметної області; порівняльний аналіз існуючих [HRM](#)-систем та підходів до їх міграції; системний аналіз інтерфейсу та структури бази даних застарілої системи; моделювання архітектури програмного забезпечення; об'єктно-орієнтоване

програмування (C#); методика розробки [Windows Forms](#)-застосунків; емпіричні методи тестування програмивного забезпечення; методи забезпечення якості програмного продукту. Бакалаврська робота складається з трьох розділів. У першому розділі здійснено аналіз інтерфейсу застарілої [HRM](#)-системи та структури її бази даних. Розглянуто основні недоліки існуючого інтерфейсу, оцінено його зручність для користувачів, виявлено проблеми сумісності з сучасними операційними системами. Досліджено структуру бази даних, проаналізовано зв'язки між таблицями, оцінено ефективність зберігання даних та визначено необхідність оптимізації. У другому розділі наведено опис архітектурної моделі нової [HRM](#)-системи, обґрунтовано вибір інструментів розробки, СУБД та САБД. Представлено загальну архітектуру програми, описано компоненти системи та їх взаємодію. Обґрунтовано вибір платформи [Windows Forms](#), мови програмування C# та системи керування базами даних з урахуванням вимог до сумісності з [Windows 10](#), продуктивності та зручності підтримки. Третій розділ присвячено процесу реалізації [Windows Forms](#)-застосунку, опису його функціоналу та результати тестування. Детально описано етапи розробки інтерфейсу, реалізацію ключових функцій системи, організацію роботи з базою даних. Наведено опис функціональних можливостей застосунку, результати налагодження та тестування, що підтверджують працездатність та стабільність розробленої системи.

РОЗДІЛ 1. АНАЛІЗ ІНТЕРФЕЙСУ ЗАСТАРІЛОЇ [HRM](#)-СИСТЕМИ ТА СТРУКТУРИ БД

1.1 Аналіз інтерфейсу застарілої [HRM](#)-системи.

Перед початком процесу модернізації [HRM](#)-системи необхідно було провести аналіз існуючого програмного забезпечення, яке тривалий час використовувалось у кадровому обліку організації. Особливістю цього етапу стало те, що доступ до вихідного коду старої системи був відсутній. У розпорядженні була лише скопійована версія програми, що працювала в середовищі операційної системи [Windows](#). Через це дослідження виконувалось методом реверсивного аналізу інтерфейсу користувача, поведінки елементів програми та логіки взаємодії між модулями. Для структуризації результатів аналізу були побудовані [UML](#)-діаграми, які дозволили відтворити загальну архітектуру програмного забезпечення та визначити ключові бізнес-процеси системи [1]. Проведений аналіз показав, що застаріла [HRM](#)-система була типовим настільним [WinForms](#)-застосунком, побудованим за монолітною архітектурою. Уся функціональність програми виконувалась в межах одного процесу, а інтерфейс користувача був тісно пов'язаний із логікою роботи та механізмами доступу до бази даних. Подібний підхід широко використовувався у корпоративних системах початку 2000-х років через простоту реалізації та високу швидкість розробки [2]. Головним стартовим вікном системи був модуль управління підрозділами. Відповідно до [UML](#)-діаграми, центральним елементом цього модуля виступав клас [DepartmentsView](#). Саме він об'єднував у собі функції перегляду структури організації, редагування даних та взаємодії користувача із кадровою інформацією. У межах цього класу були реалізовані методи [Save\(\)](#), [Reset\(\)](#) та [Delete\(\)](#), які забезпечували основні операції над записами. Рис. 1.1.

Діаграма класів модулю управління підрозділами

Структура інтерфейсу вікна була побудована за класичним принципом [WinForms](#)-застосунків. У лівій частині розташовувалось дерево підрозділів [TreeView](#), яке дозволяло переглядати організаційну структуру у вигляді ієрархії. Такий спосіб представлення інформації був зручним для великих установ із багаторівневою структурою факультетів, кафедр та відділів. Користувач міг розгортати та згортати вузли дерева, переміщаючись між різними підрозділами організації. Для пошуку підрозділів використовувались текстове поле [searchBox](#) та окрема кнопка [searchButton](#). Це свідчить про те, що механізм пошуку був реалізований у вигляді окремої операції, яка запускалась лише після натискання кнопки. У сучасних системах зазвичай використовується автоматична фільтрація в режимі реального часу, однак для старих [WinForms](#)-програм подібний підхід був типовим через обмеження продуктивності та особливості реалізації інтерфейсу. Основна інформація про підрозділ відображалась у центральній частині форми через набір вкладок [TabControl](#). У вкладках знаходились текстові поля для введення назви установи, скороченої назви, адреси, номера телефону та приміток. Крім того, використовувались комбіновані списки [ComboBox](#) для вибору керівника, факультету та типу підрозділу. Наявність комбінованих списків свідчить про використання довідникових таблиць та централізованого зберігання пов'язаних даних. Аналіз [UML](#)-діаграми дозволяє зробити висновок, що система використовувала реляційну модель даних. Основними сутностями були класи [Department](#), [Employee](#), [Faculty](#) та [DepartmentType](#). Кожна сутність містила унікальний ідентифікатор та набір атрибутів. Наприклад, клас [Department](#) мав поля [Name](#), [ShortName](#), [Address](#), [Phone](#) та [Notes](#). Клас [Employee](#) містив інформацію про працівника, включаючи ім'я та посаду. Подібний підхід відповідає принципам об'єктно-орієнтованого моделювання, коли кожен клас описує

окрему сутність предметної області. Зв'язки між класами демонструють структуру організації. Один факультет міг містити декілька підрозділів, а кожен підрозділ мав зв'язок із багатьма працівниками. Крім того, працівник міг виконувати роль керівника підрозділу. Це дозволяє зробити висновок, що база даних системи містила зовнішні ключі та таблиці зв'язків для підтримки цілісності інформації. Важливим компонентом системи був клас [DepartmentService](#), який виконував функції сервісного шару. Він містив методи [SaveDepartment\(\)](#), [DeleteDepartment\(\)](#), [LoadDepartments\(\)](#) та [SearchDepartments\(\)](#). Наявність окремого сервісного шару свідчить про спробу розділення логіки інтерфейсу та бізнес-логіки. Проте через монолітну структуру застосунку всі компоненти залишались взаємозалежними, що ускладнювало подальшу модернізацію та перенесення системи на інші операційні системи. Діаграма варіантів використання демонструє основні сценарії роботи користувача із системою. Користувач міг переглядати список підрозділів, виконувати пошук, редагувати записи, зберігати зміни та видаляти інформацію. Усі дії виконувались через графічний інтерфейс без використання зовнішніх інструментів або командного рядка. Це підтверджує орієнтацію системи на звичайних працівників кадрового відділу, які не мали спеціальної технічної підготовки. Рис 1.2. Діаграма варіантів використання модулю управління підрозділами

Наступним важливим елементом старої HRM-системи був модуль особистих карток працівників. Центральним класом цього модуля виступала форма [PersonnelForm](#), яка забезпечувала перегляд, редагування та збереження інформації про співробітників. Інтерфейс форми містив дерево підрозділів, таблицю працівників та набір вкладок із персональними даними. Використання компонента [DataGridView](#) для відображення списку співробітників свідчить про табличний принцип організації інформації. Після вибору підрозділу у дереві користувач отримував список працівників відповідного відділу. Подібний підхід дозволяв швидко переглядати кадрову інформацію, однак при великій кількості записів міг створювати проблеми з продуктивністю та швидкістю оновлення даних. Особиста картка працівника була поділена на декілька вкладок. Серед них були вкладки з основною інформацією, паспортними даними, інформацією про освіту та додатковими відомостями. Такий спосіб організації інтерфейсу дозволяв зменшити перевантаження форми та розділити інформацію за категоріями. Рис. 1.3. Діаграма класів модулю управління особистими картками

Основні персональні дані працівника включали прізвище, ім'я, по батькові, стать, дату народження та фотографію. Для відображення фотографії використовувався компонент [PictureBox](#). Аналіз структури класу [Employee](#) показує, що фотографія зберігалась не безпосередньо у базі даних, а у вигляді шляху до файлу через поле [PhotoPath](#). Це було типовим рішенням для старих настільних систем через обмеження швидкодії баз даних та складність роботи з великими двійковими об'єктами. У бізнес-шарі системи були присутні класи [Position](#), [PassportData](#) та [EducationRecord](#). Клас [PassportData](#) містив інформацію про серію та номер паспорта, а також орган видачі документа. Клас [EducationRecord](#) використовувався для зберігання даних про навчальний заклад, спеціальність та рік закінчення. Подібна структура дозволяла уникати дублювання інформації та відповідала принципам нормалізації баз даних [3].

Окрему увагу слід приділити сервісному шару системи. Клас [EmployeeService](#) відповідав за завантаження, пошук, збереження та видалення працівників. Саме через нього виконувалась взаємодія із базою даних [Database](#). Усі [SQL](#)-запити виконувались безпосередньо під час роботи користувача із формою, що свідчить про синхронний характер роботи програми. [Sequence Diagram](#) процесу збереження працівника демонструє типовий сценарій функціонування системи. Після натискання кнопки [Save](#) користувачем форма [PersonnelForm](#) передавала об'єкт [employee](#) до сервісного шару. Далі [EmployeeService](#) виконував [SQL](#)-запит [UPDATE Employee](#) у базі даних. Після успішного завершення операції користувачу відображалось повідомлення про успішне збереження. Рис 1.4. Діаграма послідовності модулю управління особистими картками

Подібний механізм мав ряд недоліків. Оскільки операція запису виконувалась синхронно, інтерфейс міг тимчасово блокуватись до завершення запиту. При роботі через повільне мережеве з'єднання або при великому обсязі даних це створювало затримки у роботі користувача. Крім того, у системі були відсутні сучасні механізми асинхронної взаємодії та кешування даних. Ще одним важливим модулем системи був модуль управління посадами у підрозділах. Його основою виступала форма [DepartmentsView](#), яка містила таблицю посад [positionGrid](#) та набір вкладок із характеристиками посади. У цьому модулі користувач міг створювати нові посади, редагувати їх параметри та призначати співробітників. Інтерфейс форми дозволяв задавати назву посади, кількість штатних одиниць, кількість вакантних місць та розмір заробітної плати. Для введення числових параметрів використовувався компонент

[NumericUpDown](#), який забезпечував перевірку коректності значень на рівні інтерфейсу. Система підтримувала додаткові довідники, зокрема графіки роботи, типи зайнятості, форми оплати праці та вимоги до освіти. Для цього використовувались класи [WorkSchedule](#), [EmploymentCondition](#), [PaymentType](#) та [EducationRequirement](#). Наявність великої кількості довідникових сутностей свідчить про складність кадрового обліку та необхідність підтримки різних категорій працівників. Рис. 1.5. Діаграма класів модулю управління посадами підрозділу Клас [Position](#) мав зв'язок із класом [Employee](#), що дозволяло визначати працівників, які займають конкретну посаду. Один підрозділ міг містити декілька посад, а одна посада могла бути пов'язана з кількома співробітниками. Це дозволяло реалізувати функціональність штатного розпису та контролю вакантних місць. [Sequence Diagram](#) для редагування посади демонструє аналогічний принцип взаємодії між інтерфейсом, сервісним шаром та базою даних. Після внесення змін користувач натискав кнопку [Save](#), після чого форма передавала об'єкт [position](#) до [PositionService](#). Далі сервіс виконував [SQL](#)-запит [UPDATE Position](#) у базі даних та повертав результат операції. Рис. 1.6. Діаграма послідовностей модулю управління посадами підрозділу

Проведений аналіз дозволяє зробити висновок, що стара [HRM](#)-система була побудована на основі архітектурних підходів, характерних для свого часу. Вона забезпечувала базову функціональність кадрового обліку, підтримувала роботу з організаційною структурою, особистими картками працівників та штатним розписом. Разом із тим система мала суттєві обмеження, серед яких жорстка залежність від [Windows](#), складність масштабування, застарілий інтерфейс та відсутність підтримки сучасних механізмів взаємодії. Через відсутність вихідного коду процес перенесення системи на сучасні операційні системи вимагав проведення детального реверсивного аналізу інтерфейсу та логіки роботи програми. Побудовані [UML](#)-діаграми дозволили відновити структуру системи, визначити взаємозв'язки між компонентами та сформувавши основу для подальшої модернізації програмного забезпечення.

1.2 Аналіз структури бд застарілої [HRM](#)-системи.

Одним із найважливіших етапів модернізації [HRM](#)-системи став аналіз структури бази даних, яка використовувалась у попередній версії програмного забезпечення. Саме база даних забезпечувала зберігання кадрової інформації, структури підрозділів, штатного розпису, особистих даних працівників та інформації про кадрові накази. Через тривалий період експлуатації структура БД поступово ускладнювалась, а окремі архітектурні рішення перестали відповідати сучасним вимогам до побудови інформаційних систем [4]. У ролі системи керування базами даних використовувалась [Firebird](#). На момент створення старої [HRM](#)-системи ця СКБД була популярним рішенням для корпоративних настільних застосунків завдяки підтримці [SQL](#), невеликим вимогам до апаратних ресурсів та можливості роботи у локальних мережах [5]. База даних функціонувала у межах класичної клієнт серверної архітектури, де значна частина бізнес-логіки виконувалась безпосередньо всередині СКБД через тригери, процедури та функції. У процесі дослідження було встановлено, що структура БД мала застарілу архітектуру та містила низку проблем, пов'язаних із нормалізацією даних. Незважаючи на це, однією з вимог керівництва було повне збереження структури бази даних без внесення критичних змін до таблиць або зв'язків між ними. Через це перенесення БД на нову версію [Firebird](#) виконувалось без перебудови архітектури, а основні зміни були реалізовані на стороні нового клієнтського застосунку. Основою системи були таблиці [OTDEL](#), [DOLJNOST](#), [SOTR](#), [SOTR_DOLJN](#) та [PRIKAZ](#). Вони формували ядро кадрового обліку та забезпечували взаємозв'язок між підрозділами, працівниками та посадами. Рис. 1.7. Діаграма основної структури бд

Таблиця [OTDEL](#) використовувалась для зберігання інформації про структурні підрозділи організації. Вона містила назву підрозділу, скорочену назву, адресу, номер телефону та логічні ознаки типу підрозділу. Наявність полів [IS_KAF](#) та [IS_FAC](#) свідчить про використання системи в освітній установі, де необхідно було розділяти факультети та кафедри. Особливістю таблиці [OTDEL](#) було використання поля [PARENT_ID](#), яке реалізовувало ієрархічну структуру підрозділів. Завдяки цьому один підрозділ міг містити вкладені структурні елементи. Подібна схема дозволяла будувати дерево організації, однак при складних [SQL](#)-запитах могла створювати додаткове навантаження на систему [6]. Також таблиця містила поле [SHEF_ID](#), яке використовувалось для зберігання керівника підрозділу. Фактично це був зв'язок із таблицею працівників [SOTR](#). Проте в старій системі частина зовнішніх ключів не контролювалась безпосередньо СКБД, а перевірка коректності виконувалась через програмний код клієнтського застосунку. Це було характерною особливістю багатьох старих корпоративних систем. Таблиця [DOLJNOST](#) використовувалась для зберігання інформації про посади у межах конкретних підрозділів. Вона містила назву посади, інформацію про оклади, кількість штатних одиниць, кількість вакантних місць та

тривалість відпустки. Також таблиця містила поле [IS_PED](#), яке визначало педагогічний характер посади. Під час аналізу було виявлено проблему дублювання однотипних даних. Наприклад, окремо зберігались значення [OKLAD_B](#) та [OKLAD_NB](#), що відповідали бюджетному та небюджетному фінансуванню. Аналогічна ситуація спостерігалась для кількості ставок і вакантних місць. Подібна структура суперечить принципам нормалізації, оскільки дублювання однотипних атрибутів ускладнює підтримку та збільшує ризик появи неузгоджених даних [7]. Центральною таблицею кадрової системи була таблиця [SOTR](#). Вона містила основну інформацію про працівників, включаючи прізвище, ім'я, по батькові, стать, дату народження, адресу проживання, номер телефону, електронну пошту, паспортні дані та ідентифікаційний код. Структура таблиці демонструє використання денормалізованого підходу до зберігання інформації. Частина персональних даних знаходилась безпосередньо в основній таблиці працівників, хоча відповідно до сучасних принципів проєктування БД їх доцільно було б винести в окремі сутності. Наприклад, паспортні дані зберігались через поля [PASP_SER](#) та [PASP_N](#), а контактна інформація розміщувалась разом із основними даними працівника. У таблиці [SOTR](#) також використовувались поля великої довжини типу [varchar](#). Подібний підхід був типовим для старих систем, де структура БД створювалась із запасом для уникнення помилок при введенні даних користувачами. Проте надмірна довжина полів негативно впливала на оптимізацію індексів та продуктивність запитів. Таблиця [SOTR_DOLIN](#) виконувала роль проміжної сутності між працівниками та посадами. Саме через неї реалізовувався механізм призначення працівника на певну посаду в конкретному підрозділі. У таблиці містились посилання на працівника, посаду та відділ, дата створення запису та ознака основного місця роботи. Крім того, у таблиці дублювались оклади та кількість ставок із таблиці [DOLINOST](#). Подібний підхід використовувався для збереження історичних значень, однак він створював ризик виникнення неузгодженості даних. У випадку зміни параметрів посади частина записів могла залишатись із застарілими значеннями. Таблиця [PRIKAZ](#) використовувалась для зберігання кадрових наказів. Вона містила назву документа, тип наказу та дату створення. Через зв'язок із таблицею [SOTR_DOLIN](#) забезпечувалась можливість відстеження кадрових призначень та змін штатного розпису. Окрему групу таблиць становив модуль персональної інформації працівника. До нього входили таблиці [EDUCATION](#), [FAMILY](#), [VOENKOM](#), [SOTR_PENS](#), [SOTR_INV](#) та [CHANGED_FAMILS](#). Всі ці таблиці були пов'язані із таблицею [SOTR](#) через зв'язок один до багатьох. Рис. 1.8. Діаграма структури частини бд, яка відповідає за персональну інформацію працівника. Таблиця [EDUCATION](#) використовувалась для зберігання інформації про освіту працівника. У ній містились дані про навчальний заклад, спеціальність, кваліфікацію та дату видачі диплома. Один працівник міг мати декілька записів про освіту, що дозволяло зберігати інформацію про декілька дипломів або підвищення кваліфікації. Таблиця [FAMILY](#) забезпечувала зберігання інформації про членів сім'ї працівника. У ній знаходились ПІБ родича, тип спорідненості та дата народження. Подібні дані використовувались для кадрового обліку та формування окремих видів державної звітності. Таблиця [VOENKOM](#) використовувалась для ведення військового обліку. Вона містила військове звання, категорію та військову спеціальність працівника. Наявність окремого військового модуля є характерною особливістю кадрових систем державних та освітніх установ. Таблиці [SOTR_PENS](#) та [SOTR_INV](#) забезпечували облік пенсійного статусу та інвалідності працівників. У них містились номери документів, групи інвалідності та дати початку й завершення відповідних статусів. Таблиця [CHANGED_FAMILS](#) використовувалась для зберігання історії зміни прізвищ працівника. Це дозволяло підтримувати коректність кадрових документів та забезпечувало відповідність архівної інформації. Аналіз структури персонального модуля показує, що окремі частини системи були побудовані відповідно до принципів реляційного проєктування. Дані про освіту, військовий облік та сімейний стан були винесені в окремі таблиці, що дозволяло уникати прямого дублювання інформації. Окремим компонентом БД був науковий модуль, який використовувався для обліку наукової діяльності працівників. Він включав таблиці [NAUCH_STEPEN](#), [UCH_ZVAN](#), [KVALIFICATION](#) та [SOTR_CHLEN_ACADEM](#). Рис. 1.9. Діаграма наукового модулю бд. Таблиця [NAUCH_STEPEN](#) містила інформацію про наукові ступені, тему дисертації та академічний статус працівника. Таблиця [UCH_ZVAN](#) використовувалась для обліку вчених звань та містила інформацію про кафедру і дату присвоєння звання. Таблиця [KVALIFICATION](#) забезпечувала облік курсів підвищення кваліфікації. У ній містились назви курсів та періоди проходження навчання. Це дозволяло автоматизувати контроль професійного розвитку співробітників. Таблиця [SOTR_CHLEN_ACADEM](#) використовувалась для зберігання інформації про членство працівників у наукових та академічних організаціях. Подібна функціональність була

характерною для [HRM](#)-систем навчальних закладів та наукових установ. У процесі модернізації значна частина тригерів, функцій та процедур [Firebird](#) була видалена. Основною причиною цього стало перенесення бізнес-логіки на сторону клієнтського застосунку. Такий підхід дозволив зменшити залежність системи від специфічних механізмів старої версії [Firebird](#) та спростити підтримку БД [8]. Перенесення бази даних на нову версію [Firebird](#) виконувалось із використанням спеціального [Python](#)-алгоритму та програмного забезпечення [IBExpert](#). [Python](#) застосовувався для автоматизації експорту, перевірки даних та імпорту інформації у нову систему. Основним завданням було збереження максимальної кількості даних без порушення зв'язків між таблицями. [IBExpert](#) використовувався для аналізу структури БД, перевірки сумісності та ручного виправлення окремих проблем перенесення. За допомогою цього інструмента виконувалась перевірка індексів, зовнішніх ключів та коректності структури таблиць [9]. Проведений аналіз показує, що база даних застарілої [HRM](#)-системи формувалась протягом тривалого часу та поступово розширювалась відповідно до нових вимог користувачів. У результаті система поєднувала як елементи коректного реляційного проєктування, так і значну кількість застарілих рішень, пов'язаних із дублюванням даних та недостатньою нормалізацією. Незважаючи на виявлені недоліки, структура БД була збережена майже без змін через вимоги сумісності та необхідність підтримки існуючих кадрових даних. Отримані результати аналізу стали основою для подальшої модернізації системи та перенесення її на сучасні програмні платформи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Архітектурна модель застосунку

Під час розробки модернізованої [HRM](#)-системи одним із ключових завдань було створення архітектури, яка забезпечувала б підтримку сучасних операційних систем, збереження сумісності із наявною базою даних та можливість подальшого розвитку програмного забезпечення. Основною ціллю перенесення системи була забезпечення стабільної роботи застосунку в середовищі [Windows](#) 10, оскільки попередня версія програмного забезпечення створювалась для значно старіших версій операційної системи [Windows](#) і мала проблеми із сумісністю на сучасних платформах. Для реалізації клієнтської частини було обрано технологію [WinForms](#) на платформі [.NET Framework](#). Використання [WinForms](#) було зумовлене необхідністю максимально зберегти логіку та принципи роботи старої [HRM](#)-системи. Крім того, застосування цієї технології дозволило уникнути повного перепроєктування інтерфейсу користувача та спростило процес адаптації персоналу до нової версії програмного забезпечення. Важливою перевагою обраної версії [.NET Framework](#) стала офіційна підтримка операційної системи [Windows](#) 10. Це забезпечило стабільну роботу інтерфейсу, підтримку сучасних системних бібліотек та коректне відображення елементів керування на нових версіях ОС. Також використання сучаснішої версії [.NET Framework](#) дозволило підвищити рівень безпеки застосунку та сумісність із новими драйверами, системними компонентами та оновленнями [Windows](#) [10]. Архітектурною основою застосунку було обрано шаблон [MVP \(Model View Presenter\)](#). Даний підхід дозволяє розділити логіку представлення, бізнес-логіку та роботу з даними, що значно спрощує підтримку та модернізацію програмного забезпечення. На відміну від монолітної архітектури старої [HRM](#)-системи, у якій інтерфейс безпосередньо взаємодівав із базою даних, [MVP](#) забезпечує чітке розділення відповідальностей між компонентами. Структура проєкту складається з трьох основних компонентів: [View](#), [Presenter](#) та [Model](#). Кожен із цих компонентів виконує окрему роль у роботі системи. Рис. 2.1. Структура проєкту

Компонент [View](#) відповідає за відображення інформації користувачу та обробку дій інтерфейсу. У проєкті цей рівень представлений інтерфейсом [IView](#) та класом [WinFormsView](#). Інтерфейс [IView](#) визначає набір методів і властивостей, необхідних для взаємодії із [Presenter](#). Такий підхід дозволяє зменшити залежність між інтерфейсом користувача та логікою застосунку. Клас [WinFormsView](#) реалізує інтерфейс [IView](#) та містить безпосередню реалізацію графічного інтерфейсу на основі [WinForms](#). Саме цей компонент відповідає за роботу форм, кнопок, таблиць, вкладок та інших елементів інтерфейсу користувача. Використання інтерфейсу [IView](#) дозволило відокремити логіку роботи від конкретної реалізації інтерфейсу. Завдяки цьому [Presenter](#) взаємодіє не з конкретною формою [WinForms](#), а з абстрактним представленням інтерфейсу. Подібний підхід значно спрощує тестування системи та забезпечує можливість подальшої зміни інтерфейсу без необхідності переписування бізнес-логіки. Компонент [Presenter](#) виконує роль посередника між інтерфейсом користувача та моделлю даних. Саме він обробляє події, які надходять із [View](#), виконує перевірку даних, викликає необхідні методи [Repository](#) та передає результати назад до інтерфейсу. На відміну від старої версії системи, де частина логіки виконувалась безпосередньо у формах [WinForms](#), у новій архітектурі вся основна логіка була перенесена до [Presenter](#). Це

дозволило зменшити зв'язність компонентів та зробити код більш структурованим. [Presenter](#) взаємодіє із [View](#) через інтерфейс [IView](#), а з рівнем даних через [Repository](#). Таким чином реалізується чітке розділення між інтерфейсом та роботою з базою даних. Подібний підхід відповідає сучасним принципам побудови корпоративних інформаційних систем. Рівень [Model](#) відповідає за роботу з даними та взаємодію із базою [Firebird](#). У межах цього компонента знаходяться класи [Repository](#) та [Entity](#). Клас [Entity](#) використовується для представлення сутностей бази даних у вигляді об'єктів програмного коду. Наприклад, працівник, підрозділ або посада представлені окремими класами із відповідними властивостями. Клас [Repository](#) забезпечує взаємодію між застосунком та базою даних [Firebird](#). Саме через нього виконуються [SQL](#)-запити, завантаження даних та збереження змін. Подібний підхід дозволяє ізолювати логіку роботи з БД від інших компонентів системи. Важливою особливістю нової архітектури стало перенесення значної частини функціональності зі сторони бази даних на сторону клієнтського застосунку. У старій [HRM](#)-системі велика кількість бізнес-логіки реалізовувалась через тригери та процедури [Firebird](#). У процесі модернізації значна частина цього функціоналу була перенесена до [Presenter](#) та [Repository](#). Подібне рішення дозволило спростити структуру бази даних, зменшити залежність від специфічних можливостей старих версій [Firebird](#) та підвищити контроль над логікою роботи застосунку. Крім того, це дало можливість легше відлагоджувати систему та вносити зміни без необхідності модифікації [SQL](#)-процедур. Ще однією перевагою використання [MVP](#) стало спрощення підтримки програмного забезпечення. Завдяки розділенню компонентів окремі частини системи можуть модифікуватись незалежно одна від одної. Наприклад, зміна структури інтерфейсу не потребує переписування механізмів роботи з базою даних. Крім того, використання [MVP](#) позитивно вплинуло на читабельність програмного коду. Усі компоненти системи мають чітко визначені обов'язки, що значно спрощує навігацію по проєкту та подальшу підтримку програмного забезпечення. Таким чином, обрана архітектурна модель дозволила створити сучасний [WinForms](#)-застосунок із підтримкою [Windows](#) 10 та збереженням сумісності із існуючою структурою бази даних [Firebird](#). Використання шаблону [MVP](#) забезпечило розділення відповідальностей між компонентами системи, підвищило підтримуваність програмного коду та створило основу для подальшого розвитку [HRM](#)-системи.

2.2 Вибір інструментів для розробки, вибір СУБД та САБД

Під час розробки модернізованої [HRM](#)-системи важливим етапом став вибір інструментів розробки, системи керування базами даних та допоміжного програмного забезпечення. Основними вимогами до обраних технологій були сумісність із наявною структурою бази даних, підтримка сучасних операційних систем [Windows](#), стабільність роботи та можливість подальшого супроводу системи. Оскільки однією з головних вимог проєкту було збереження існуючої структури бази даних, у якості СУБД було вирішено використовувати [Firebird](#) нової версії. Використання саме [Firebird](#) дозволило уникнути необхідності повного перепроєктування БД та забезпечило сумісність із накопиченими кадровими даними. Попередня версія [HRM](#)-системи працювала на застарілій версії [Firebird](#), яка мала обмеження щодо підтримки сучасних операційних систем та нових механізмів безпеки. Перехід на новішу версію [Firebird](#) дозволив підвищити стабільність роботи системи, покращити продуктивність [SQL](#)-запитів та забезпечити підтримку сучасних механізмів взаємодії із клієнтськими застосунками. Важливою перевагою [Firebird](#) є невеликі системні вимоги та простота розгортання. Це особливо важливо для корпоративних [HRM](#)-систем, які можуть використовуватись на комп'ютерах із різними технічними характеристиками. Крім того, [Firebird](#) підтримує класичну клієнт серверну архітектуру та забезпечує роботу багатокористувацького режиму. Ще однією причиною вибору [Firebird](#) стала висока сумісність із [WinForms](#)-застосунками та платформою [.NET Framework](#). Для взаємодії із БД використовувались стандартні механізми роботи з [SQL](#) та драйвери [Firebird](#) для [.NET](#), що дозволило реалізувати стабільний доступ до даних без необхідності створення додаткових проміжних сервісів. У ролі системи адміністрування бази даних використовувалось програмне забезпечення [FlameRobin](#). Даний інструмент був обраний через підтримку сучасних версій [Firebird](#), простоту використання та можливість швидкого адміністрування структури БД. [FlameRobin](#) застосовувався для створення та редагування таблиць, перевірки структури бази даних, аналізу [SQL](#)-запитів та контролю цілісності інформації. Крім того, цей інструмент використовувався для перевірки результатів перенесення даних зі старої версії [Firebird](#) на нову. Важливою перевагою [FlameRobin](#) є підтримка відкритої архітектури та невелике споживання ресурсів. На відміну від більш складних корпоративних систем адміністрування БД, [FlameRobin](#) забезпечує швидкий доступ до основного функціоналу та не потребує складного налаштування. Для розробки клієнтської частини застосунку використовувалось середовище [Visual Studio](#). Саме [Visual](#)

[Studio](#) забезпечувала повний цикл розробки [WinForms](#)-застосунку, включаючи створення графічного інтерфейсу, налагодження програмного коду та інтеграцію із базою даних. Вибір [Visual Studio](#) був зумовлений високим рівнем інтеграції із платформою [.NET Framework](#) та підтримкою [WinForms](#). Середовище дозволяло швидко створювати форми, таблиці, вкладки та інші елементи інтерфейсу користувача за допомогою візуального редактора. Крім того, [Visual Studio](#) забезпечувала зручний механізм відлагодження застосунку, що було особливо важливо під час перенесення функціональності зі старої [HRM](#)-системи. Завдяки інтегрованим інструментам діагностики вдалося значно спростити пошук помилок та перевірку сумісності різних модулів системи. У якості основної технології для побудови графічного інтерфейсу використовувався [WinForms](#). Даний фреймворк був обраний через необхідність максимально зберегти принципи роботи старої [HRM](#)-системи та забезпечити звичний інтерфейс для користувачів. Використання [WinForms](#) дозволило швидко реалізувати класичний настільний інтерфейс із підтримкою таблиць, вкладок, деревоподібної навігації та діалогових вікон. Також важливою перевагою [WinForms](#) є стабільна робота у середовищі [Windows 10](#) та повна сумісність із [.NET Framework](#). Одним із важливих функціональних компонентів нової [HRM](#)-системи стала автоматична система формування звітів. У попередній версії системи значна частина звітності створювалась вручну або через застарілі механізми генерації документів. У модернізованій системі було реалізовано автоматичне створення [PDF](#)-документів на основі [HTML](#)-шаблонів. Для реалізації цього функціоналу використовувалась бібліотека [PuppeteerSharp](#) та додаткові пакети для роботи з [HTML](#) і [PDF](#). [PuppeteerSharp](#) є [.NET](#)-обгорткою для [Chromium](#) та дозволяє програмно керувати браузером у фоновому режимі. [11] Основний принцип роботи системи звітів полягав у генерації [HTML](#)-документа із кадровими даними, після чого цей [HTML](#) автоматично конвертувався у [PDF](#)-файл. Подібний підхід дозволив створювати звіти зі складним форматуванням, таблицями та підтримкою стилів [CSS](#). Використання [HTML](#) як проміжного формату дало можливість значно спростити процес створення шаблонів звітів. На відміну від класичних механізмів генерації [PDF](#), де структура документа формується програмним кодом, [HTML](#)-шаблони можна легко змінювати та адаптувати без значної модифікації логіки застосунку. Ще однією перевагою [PuppeteerSharp](#) стала підтримка сучасного механізму рендерингу [Chromium](#). Завдяки цьому [PDF](#)-документи мали коректне форматування, підтримку шрифтів, таблиць та складних елементів верстки. Автоматична система звітності дозволила суттєво спростити формування кадрових документів та зменшити кількість ручної роботи користувачів. Крім того, використання [PDF](#) забезпечило зручність друку та сумісність із різними операційними системами. У процесі розробки також використовувались додаткові бібліотеки для роботи із даними, обробки [SQL](#)-запитів та взаємодії із [WinForms](#)-компонентами. Проте основна архітектура системи залишалась максимально простою для забезпечення стабільності роботи та спрощення подальшої підтримки програмного забезпечення. Таким чином, вибір інструментів розробки та програмних компонентів був орієнтований на забезпечення сумісності зі старою [HRM](#)-системою, підтримку [Windows 10](#) та можливість подальшого розвитку застосунку. Використання сучасної версії [Firebird](#), середовища [Visual Studio](#), технології [WinForms](#) та бібліотеки [PuppeteerSharp](#) дозволило створити стабільну систему кадрового обліку із підтримкою автоматичної генерації звітів.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ТА ЙОГО ТЕСТУВАННЯ

3.1 Процес реалізації [Windows Forms](#) застосунку

Після завершення етапів аналізу застарілої [HRM](#)-системи та проектування нової архітектури було розпочато безпосередню реалізацію [Windows Forms](#) застосунку. Основною задачею цього етапу стало створення сучасної системи кадрового обліку, яка б забезпечувала стабільну роботу у середовищі [Windows 10](#), підтримувала існуючу структуру бази даних [Firebird](#) та дозволяла поступову модернізацію функціоналу без необхідності повного переписування всієї системи. Розробка застосунку виконувалась із використанням платформи [.NET Framework](#) та технології [Windows Forms](#). Використання [WinForms](#) було обрано через необхідність максимально зберегти структуру старого інтерфейсу та забезпечити звичний принцип роботи для співробітників кадрового відділу. Крім того, [WinForms](#) має стабільну підтримку у [Windows 10](#) та добре інтегрується із [.NET Framework](#) [12]. На відміну від попередньої версії [HRM](#)-системи, у якій значна частина логіки була реалізована безпосередньо у формах або на стороні бази даних, новий застосунок будувався відповідно до архітектурного шаблону [MVP](#). Використання [Model View Presenter](#) дозволило розділити логіку інтерфейсу, роботу з даними та бізнес-логіку системи. Подібний підхід значно покращив підтримуваність коду та спростив процес подальшої модернізації програмного забезпечення [13]. У межах [MVP](#) компонент [View](#) відповідав за взаємодію із користувачем та відображення інформації. Для цього використовувались [Windows Forms](#) форми, таблиці

[DataGridView](#), вкладки [TabControl](#), текстові поля та комбіновані списки. [Presenter](#) забезпечував обробку подій інтерфейсу, виконання перевірок даних та взаємодію із репозиторіями. Рівень [Model](#) відповідав за структуру моделей та роботу із [Firebird](#). Одним із основних компонентів системи став [DepartmentPresentater](#), який забезпечував управління підрозділами, штатним розписом та кадровими переміщеннями. Після створення екземпляра [Presenter](#) виконувався виклик методу [LoadAllInfo](#), який забезпечував завантаження всіх необхідних даних для конкретного підрозділу. У процесі виконання цього методу завантажувались дані про підрозділ, список посад, інформація про працівників та кадрові накази. Це дозволяло одразу після відкриття форми відобразити актуальний стан кадрової структури організації. Рис. 3.1. Діаграма класів, основна структура MVP системи [Presenter](#) взаємодіяв із формою через інтерфейс [IDepartmentView](#). Подібний механізм дозволив уникнути жорсткої залежності між логікою застосунку та конкретною реалізацією [WinForms](#) форми. У результаті бізнес-логіка стала незалежною від графічного інтерфейсу, що значно спростило тестування окремих компонентів. У процесі ініціалізації [Presenter](#) виконував підписку на події форми. Зокрема, використовувались [SaveEvent](#), [DeleteEvent](#), [ResetEvent](#) та інші події користувача. Після натискання кнопок форми відповідні методи [Presenter](#) автоматично виконували необхідні дії, включаючи збереження інформації, видалення записів або оновлення даних. Рис. 3.2. Діаграма класів, взаємодія [DepartmentPresentater](#) з репозиторіями Для взаємодії із базою даних використовувався репозиторій, який реалізовував доступ до таблиць [Firebird](#). Через репозиторій виконувались [SQL](#)-запити, завантаження моделей та збереження змін. У коді активно застосовувались [LINQ](#)-запити для вибірки інформації із таблиць. Наприклад, при завантаженні підрозділу використовувався пошук запису за ідентифікатором та подальше перетворення результату у модель [Department](#). Використання [LINQ](#) дозволило зробити код більш зрозумілим та зменшити кількість прямого [SQL](#)-коду у застосунку. Крім того, це значно спростило підтримку системи та роботу із моделями даних [14]. Для відображення кадрової інформації активно використовувався компонент [DataGridView](#). Саме він став основою інтерфейсу роботи із працівниками, посадами та кадровими наказами. Усі таблиці створювались вручну, без використання автоматичної генерації колонок. Подібний підхід дозволив повністю контролювати структуру таблиць та форматування даних. Рис. 3.3. Приклад використання [DataGridView](#) У процесі реалізації використовувались [DataGridViewComboBoxColumn](#), які дозволяли реалізувати випадючі списки безпосередньо всередині таблиць. Наприклад, під час редагування посад користувач міг вибрати тип зайнятості або графік роботи через комбінований список. Для забезпечення автоматичного оновлення інтерфейсу використовувались [BindingSource](#) та [BindingList](#). Завдяки цьому після зміни колекції об'єктів таблиця автоматично оновлювалась без необхідності ручного перезавантаження компонентів. Подібний механізм суттєво спростив роботу із великими обсягами кадрових даних. Окрему увагу було приділено реалізації механізму створення працівників. У старій [HRM](#)-системі додавання працівника виконувалось через декілька окремих форм та складну систему кадрових наказів. У новій версії цей процес був значно спрощений. Метод [SaveNewSotr](#) відповідав за створення нового запису працівника. Перед збереженням інформації система перевіряла наявність дублюючих записів. Для цього виконувалось порівняння прізвища, імені та дати народження працівника. Якщо подібний запис уже існував у базі, система блокувала створення дублікату. Рис. 3.4. Діаграма послідовностей для методу [SaveNewSotr\(\)](#) Після проходження перевірки створювався об'єкт моделі [Sotr](#), у який записувались персональні дані працівника. До моделі включались ПІБ, адреса проживання, контактний номер телефону, електронна пошта та інша кадрова інформація. Для перевірки коректності введених даних використовувався клас [ModelDataValidation](#). Його основним завданням була централізована перевірка моделей перед збереженням у базу даних. Наприклад, перевірялась коректність заповнення обов'язкових полів, правильність формату дат та відсутність порожніх значень у критично важливих атрибутах. Подібний підхід дозволив уникнути дублювання коду перевірки у різних частинах застосунку та забезпечив єдиний механізм валідації даних. Рис. 3.5. Діаграма класів для [ModelDataValidation](#) Для створення нових посад використовувався метод [SaveNewDoljnost](#). У межах цього методу створювались об'єкти [Doljnost](#), які містили інформацію про назву посади, оклад, кількість ставок, кількість вакантних місць та тип фінансування. Після створення об'єктів вони додавались до репозиторію та зберігались через [SaveChanges](#). Подібний механізм дозволив реалізувати централізовану роботу із базою даних та забезпечив контроль над процесом збереження інформації. Важливою частиною реалізації стала підтримка кадрових переміщень працівників. Для цього використовувалась модель [SotrMove](#), яка забезпечувала зв'язок між

працівником, посадою, підрозділом та наказом. Після створення кадрового переміщення система автоматично формувала відповідні записи у таблицях [Firebird](#). Під час реалізації кадрових переміщень активно використовувались [LINQ](#)-запити та механізми зв'язування моделей. Це дозволило забезпечити коректне формування взаємозв'язків між таблицями та уникнути помилок при створенні нових кадрових записів. Для роботи із датами був створений окремий механізм [ConvertToDateTime](#). Його основним завданням була підтримка декількох форматів введення дати та автоматичне перетворення рядків у [DateTime](#). У випадку неправильного формату система генерувала помилку із поясненням причини. Подібний механізм був необхідний через використання різних форматів дат у старій [HRM](#)-системі та необхідність підтримки існуючих записів у [Firebird](#). Однією з важливих задач модернізації стало перенесення бізнес-логіки зі сторони бази даних на сторону клієнтського застосунку. У старій системі велика кількість перевірок та операцій реалізовувалась через тригери та процедури [Firebird](#). Це ускладнювало підтримку системи та робило її сильно залежною від конкретної версії СУБД. У новому застосунку значна частина цієї логіки була реалізована через [Presenter](#) та [Repository](#). Завдяки цьому вдалося спростити структуру БД та зменшити кількість [SQL](#)-процедур. Крім того, це значно полегшило відлагодження системи та внесення змін до логіки роботи застосунку. Окремим важливим компонентом системи став модуль роботи із кадровими наказами, реалізований через клас [ChoosePrikazPresentater](#). Даний [Presenter](#) відповідає за вибір, пошук та прив'язку наказів до різних кадрових операцій у системі. Саме через цей модуль забезпечується взаємодія між кадровими переміщеннями, відпустками, трудовим стажем, змінами посад та іншими кадровими процесами. У старій [HRM](#)-системі механізм роботи із наказами був реалізований через значну кількість окремих форм та процедур бази даних. У новій версії застосунку було створено універсальний [Presenter](#), який дозволяє використовувати єдиний механізм вибору наказів для різних модулів системи. Клас [ChoosePrikazPresentater](#) побудований відповідно до архітектури [MVP](#) та виконує роль посередника між формою [ChoosePrikazView](#) та репозиторієм [PrikazRepository](#). Через [Presenter](#) реалізовується логіка завантаження наказів, пошуку записів, обробки вибору користувача та оновлення даних у таблицях інтерфейсу. Конструктор класу приймає посилання на репозиторій, форму, поточний [Presenter](#) та таблицю [DataGridView](#), із якою необхідно виконувати взаємодію. Подібний підхід дозволяє використовувати один і той самий [Presenter](#) для різних кадрових модулів. Під час створення екземпляра класу виконується підписка на події [ChooseEvent](#) та [SearchEvent](#). Завдяки цьому система автоматично реагує на дії користувача та виконує відповідні методи [Presenter](#). Одразу після ініціалізації виконується метод [LoadData](#), який забезпечує початкове завантаження кадрових наказів із бази даних [Firebird](#). Якщо дата не передана, система завантажує всі накази у порядку спадання дати створення. Якщо ж користувач виконує пошук по конкретній даті, застосовується фільтрація записів. Особливістю реалізації є повторне створення репозиторію всередині методу [LoadData](#). Це дозволяє уникнути використання застарілого контексту [Entity Framework](#) та забезпечує актуальність даних після змін у базі. Для роботи із наказами використовується модель [Prikaz](#), яка містить інформацію про назву наказу, дату створення та тип документа. Додатково завантажується список [TypPrikaz](#), який використовується для заповнення [DataGridViewComboBoxColumn](#) у формі. Після отримання даних [Presenter](#) передає список наказів у [DataGridView](#) через властивість [DataSource](#). Завдяки використанню [DataBinding](#) оновлення таблиці виконується автоматично без ручного створення рядків. Окрему увагу у модулі приділено механізму пошуку наказів. Метод [HandleSearchData](#) реалізує логіку визначення типу пошукового запиту. Спочатку система отримує текст із поля пошуку та виконує перевірку, чи відповідає введене значення формату дати. Для цього використовується метод [IsDateFormatValid](#). Він підтримує два формати дати: короткий формат [dd.MM.yyyy](#) та повний формат із мілісекундами [dd.MM.yyyy HH:mm.SSS](#). Подібна реалізація була необхідна через використання різних форматів дат у старій кадровій системі. Якщо введене значення відповідає формату дати, система виконує пошук наказів за датою створення. У протилежному випадку запускається пошук по номеру або назві наказу. Для пошуку за номером використовується метод [LoadDataByNumber](#). У ньому застосовується [EF.Functions.Like](#), який дозволяє виконувати [SQL LIKE](#)-запит безпосередньо через [Entity Framework](#). Завдяки цьому користувач може знаходити накази навіть при частковому введенні номера документа. Після виконання пошуку результати автоматично відображаються у таблиці [DataGridView](#). Подібний підхід значно спрощує роботу користувача із великою кількістю кадрових документів. Важливою частиною реалізації став метод [ConvertToDateTime](#). Він відповідає за перетворення текстового представлення дати у тип [DateTime](#). У випадку неправильного формату система генерує виняток із

повідомленням про помилку. Подібний механізм був реалізований для забезпечення стабільної роботи системи та уникнення помилок при введенні даних користувачем. Оскільки стара HRM-система підтримувала різні формати дати, додаткова перевірка була критично важливою. Основною функцією [ChoosePrikazPresenter](#) є метод [HandleChooseRecord](#). Саме він забезпечує прив'язку вибраного наказу до конкретного кадрового процесу. Після вибору запису система визначає активний [DataGridView](#) та залежно від його назви виконує різні сценарії обробки даних. Для цього використовується оператор [switch](#), який містить окрему логіку для кожного кадрового модуля. Наприклад, у випадку роботи із таблицею [sotrPrikazData](#) система оновлює список кадрових призначень працівника. Вибраний наказ записується у модель [Sotr_Doljnost](#), після чого оновлюється дата створення кадрового запису. Після зміни даних викликається метод [LoadSotrs](#), який повторно завантажує список працівників та оновлює інтерфейс користувача. Це дозволяє миттєво відобразити зміни у таблицях [WinForms](#). Для таблиці [sotrCrtDoljData](#) реалізовано інший сценарій. У цьому випадку [Presenter](#) безпосередньо змінює значення комірок [DataGridView](#). До таблиці записується дата наказу, номер документа та його ідентифікатор. Подібний підхід використовувався для тимчасових таблиць створення нових кадрових записів, які ще не були збережені у базі даних. Особливо складною є логіка роботи із [formalDataGridView](#). У цьому випадку [Presenter](#) виконує оформлення звільнення або кадрового переміщення працівника. Після вибору наказу система оновлює модель [_sotrMove](#), записуючи інформацію про наказ звільнення та дату кадрової операції. Далі виконується зміна моделі [Sotr_Doljnost](#). Для реалізації механізму звільнення використовується спеціальний підхід. Поле [Dolj_Id](#) встановлюється у значення 0, а [OtdelId](#) змінюється на 99999999. Подібна схема фактично переводить працівника у стан без активної посади. Після цього запис видаляється із поточного списку кадрових призначень через [currSotrDoljState.Remove](#). Таким чином система забезпечує приховування звільненого працівника із активного штатного розпису. Окремо реалізована підтримка кадрового стажу через таблиці [seniorityDataGridView](#) та [newSeniorityDataGridView](#). У цих сценаріях вибраний наказ використовується для оновлення інформації про записи трудового стажу працівника. [Presenter](#) автоматично формує текстове представлення наказу у форматі "номер наказу від дата". Це значно спрощує відображення інформації у кадрових документах та звітах. Аналогічний механізм реалізовано для модуля відпусток. При роботі із [VacationData](#) та [NewVacationData](#) система записує ідентифікатор наказу та його назву у відповідні моделі відпусток. Важливою особливістю реалізації є підтримка великої кількості різних [DataGridView](#) через єдиний [Presenter](#). Це дозволило уникнути дублювання коду та створення окремих модулів для кожного типу кадрової операції. У процесі реалізації активно використовувались [BindingList](#) та колекції моделей. Завдяки цьому після зміни даних [DataGridView](#) автоматично оновлювався без необхідності ручного перезавантаження інтерфейсу. Окрему увагу було приділено механізму повторного використання [Presenter](#). Для цього реалізовано шаблон [Singleton](#) через метод [GetInstance](#). Під час першого виклику створюється новий екземпляр [ChoosePrikazPresenter](#). При повторному відкритті форми використовується вже існуючий [Presenter](#), однак виконується оновлення [DataGridView](#) та повторна підписка на події. Подібний підхід дозволив зменшити кількість створюваних об'єктів та уникнути дублювання логіки роботи із наказами. Проте використання [Singleton](#) у [WinForms](#) також створює певні ризики. Наприклад, при неправильному очищенні подій можливі витоки пам'яті або дублювання обробників подій. Для уникнення цієї проблеми у коді виконується відписка від [ChooseEvent](#) перед повторною підпискою. Рис. 3.6. Діаграма класів для [ChoosePrikazPresenter](#). У процесі реалізації [ChoosePrikazPresenter](#) значна частина логіки кадрових наказів була перенесена зі сторони бази даних у клієнтський застосунок. У старій системі подібні операції виконувались через SQL-процедури [Firebird](#). У новій версії застосунку більшість перевірок та механізмів оновлення даних реалізована безпосередньо у [Presenter](#). Подібний підхід значно спростив підтримку системи та дозволив реалізувати більш гнучкий механізм роботи із кадровими документами. Таким чином, [ChoosePrikazPresenter](#) став одним із ключових компонентів кадрової системи, який забезпечує централізовану роботу із наказами, підтримує взаємодію між різними кадровими модулями та реалізує складну логіку кадрових операцій у межах [WinForms](#) застосунку. Окрему увагу було приділено реалізації автоматичної генерації PDF-документів. У старій HRM-системі значна частина звітності створювалась вручну або через застарілі механізми експорту. У новій системі було створено автоматичний механізм формування документів на основі HTML-шаблонів. Рис. 3.7. Діаграма послідовностей для методу [SaveHtmlAsPdf\(\)](#). Для цього використовувалась бібліотека [PuppeteerSharp](#), яка дозволяє керувати [Chromium](#) у фоновому режимі. Спочатку система формувала HTML-

документ із кадровими даними, після чого [PuppeteerSharp](#) виконував рендеринг сторінки та автоматично створював [PDF](#)-файл [15]. Використання [HTML](#) як проміжного формату дозволило реалізувати гнучку систему шаблонів. Завдяки [CSS](#) стало можливим створення складних таблиць, форматування тексту та налаштування друкованого вигляду документа. Крім того, [Chromium](#) забезпечував коректне відображення шрифтів, таблиць та інших елементів документа незалежно від конкретного принтера або налаштувань системи. Це значно покращило якість кадрових звітів у порівнянні зі старою версією [HRM](#)-системи. У процесі реалізації також було створено механізми очищення тимчасових таблиць та повторного завантаження інформації після збереження змін. Це дозволяло миттєво оновлювати інтерфейс після створення нових записів та забезпечувало актуальність відображених даних. Важливою частиною роботи стало тестування застосунку. Основна увага приділялась перевірці стабільності роботи інтерфейсу, коректності взаємодії із [Firebird](#) та правильності роботи [CRUD](#)-операцій. У межах тестування перевірялись сценарії створення працівників, редагування посад, кадрових переміщень та формування [PDF](#)-звітів. Також виконувалась перевірка роботи системи при великій кількості записів у таблицях [DataGridView](#). Окремо тестувалась обробка помилок. Наприклад, перевірялась реакція системи на неправильний формат дат, спробу створення дублюючого працівника або видалення посади, яка вже пов'язана із кадровими записами. Під час тестування було підтверджено коректну роботу застосунку у [Windows 10](#). Усі основні функції [HRM](#)-системи працювали стабільно, а інтерфейс коректно відображався на сучасних версіях операційної системи. Таким чином, у процесі реалізації [Windows Forms](#) застосунку було створено сучасну кадрову систему із підтримкою [Firebird](#), архітектурою [MVP](#) та автоматичною генерацією [PDF](#)-звітів. Перенесення бізнес-логіки на сторону клієнтського застосунку дозволило спростити структуру бази даних та покращити підтримуваність системи. Використання сучасних бібліотек і механізмів роботи із даними забезпечило стабільну роботу [HRM](#)-системи та створило основу для її подальшого розвитку.

3.2 Опис функціоналу застосунку

Розроблений [Windows Forms](#) застосунок реалізує комплексну систему автоматизації кадрового обліку та забезпечує підтримку основних процесів роботи відділу кадрів. Під час модернізації [HRM](#)-системи основна увага приділялась не лише перенесенню функціоналу на сучасні операційні системи, але й підвищенню продуктивності, покращенню інтерфейсу користувача та спрощенню роботи із кадровою документацією. Одним із головних функціональних модулів системи є управління структурою підрозділів. Користувач може переглядати дерево організаційної структури, створювати нові підрозділи, редагувати інформацію про факультети, кафедри та інші структурні одиниці. Для кожного підрозділу підтримується зберігання контактної інформації, адреси, номера телефону, скороченої назви та інформації про керівника. Для навігації між підрозділами використовується [TreeView](#)-структура, що дозволяє швидко переміщуватись між елементами навіть при великій кількості записів. Завдяки цьому працівники кадрового відділу можуть оперативно знаходити необхідні підрозділи без необхідності відкривати додаткові форми або виконувати складний пошук. Система також підтримує повноцінне управління штатним розписом. Користувач має можливість створювати нові посади, редагувати оклади, кількість ставок, вакантні місця та інші кадрові параметри. Для кожної посади можуть задаватися графік роботи, тип зайнятості, вимоги до освіти та інші характеристики. Під час роботи із посадовими записами система автоматично відображає інформацію про працівників, які займають відповідні посади. Це дозволяє кадровому відділу контролювати поточний стан штатного розпису та швидко визначати наявність вакантних місць. Важливою частиною функціоналу є модуль персональних карток працівників. Для кожного співробітника система зберігає великий обсяг кадрової інформації, включаючи особисті дані, паспортні реквізити, контактну інформацію, військовий облік, освіту, наукові ступені та кадрову історію. Інтерфейс персональної картки реалізований через вкладки [TabControl](#), що дозволяє логічно розділити інформацію на окремі категорії. Працівник кадрового відділу може швидко переходити між вкладками стажу, відпусток, освіти, наукової діяльності та іншими модулями. Однією з ключових можливостей системи є підтримка кадрових переміщень. Застосунок дозволяє оформлювати прийняття працівників на роботу, переведення між підрозділами, зміну посад та звільнення співробітників. Усі кадрові операції автоматично фіксуються у кадровій історії працівника. Для оформлення кадрових дій використовується механізм кадрових наказів. Користувач може здійснювати пошук наказів за номером, назвою або датою створення. Після вибору документа наказ автоматично прив'язується до кадрового переміщення, стажу або відпустки. Окрему увагу під час модернізації було приділено швидкодії застосунку. У порівнянні зі старою версією [HRM](#)-системи новий

застосунок працює значно швидше. Це стало можливим завдяки використанню нової версії [Firebird](#), оптимізації [SQL](#)-запитів та перенесенню значної частини логіки зі сторони бази даних у клієнтський застосунок [20]. У старій [HRM](#)-системі велика кількість операцій виконувалась через тригери та [SQL](#)-процедури [Firebird](#). У новій версії значна частина перевірок, пошукових операцій та обробки даних реалізована безпосередньо у [Presenter](#)-класах [Windows Forms](#) застосунку. Подібний підхід дозволив зменшити навантаження на сервер бази даних та значно пришвидшити роботу інтерфейсу користувача. Особливо помітним це стало при відкритті великих таблиць працівників та кадрових наказів. Додаткове покращення продуктивності було досягнуте за рахунок використання сучасної версії [Firebird](#) із підтримкою оптимізованих механізмів кешування та багатокористувацької роботи [21]. У результаті модернізації швидкість виконання пошукових операцій та відкриття кадрових форм збільшилась у декілька разів у порівнянні зі старою системою. Крім того, суттєво зменшився час збереження кадрових змін у базі даних. Для відображення кадрової інформації активно використовується компонент [DataGridView](#). Усі основні таблиці підтримують редагування даних безпосередньо в інтерфейсі, автоматичне оновлення інформації та сортування записів. Система підтримує пошук працівників, наказів та посад за різними параметрами. Користувач може виконувати пошук за прізвищем, номером документа, датою наказу або іншими кадровими атрибутами. Під час реалізації пошукових механізмів використовувались оптимізовані [LINQ](#)-запити та механізми часткового пошуку через [EF.Functions.Like](#). Це дозволило забезпечити швидку роботу навіть при великій кількості записів у базі даних. Одним із важливих покращень системи стала реалізація [client-side](#) валідації. У старій [HRM](#)-системі перевірка даних переважно виконувалась після надсилання [SQL](#)-запиту до [Firebird](#), через що користувач отримував повідомлення про помилки із затримкою. У новому застосунку перевірка даних виконується безпосередньо у [Windows Forms](#) інтерфейсі. Наприклад, система автоматично перевіряє правильність формату дат, наявність обов'язкових полів та коректність введених числових значень. Якщо користувач вводить неправильні дані, система миттєво відображає повідомлення про помилку без необхідності взаємодії із базою даних. Це значно покращує зручність роботи та дозволяє швидше виправляти помилки введення. [Client-side](#) валідація також позитивно вплинула на стабільність роботи системи, оскільки значна кількість помилок тепер обробляється ще до моменту запису інформації у [Firebird](#). Важливою функціональною можливістю системи стала підтримка автоматичної генерації [PDF](#)-документів. Користувач може експортувати основні кадрові таблиці у [PDF](#)-файли для друку або архівного збереження. Зокрема, система підтримує конвертацію таблиць відпусток, стажу, кадрових наказів та інших кадрових модулів у [PDF](#)-документи. Для створення документів використовується механізм [HTML](#)-шаблонів та рендеринг через [Chromium](#). Під час генерації документа система автоматично формує [HTML](#)-сторінку із кадровими даними, після чого виконується її перетворення у [PDF](#)-файл. Завдяки цьому забезпечується коректне форматування таблиць, шрифтів та друкованих сторінок [22]. Рис. 3.8. [HTML](#)-шаблон, з якого формується [.pdf](#)-звіт У порівнянні зі старою [HRM](#)-системою механізм формування документів став значно більш гнучким. Працівники кадрового відділу можуть швидко створювати друковані документи без необхідності ручного копіювання даних у сторонні програми. Інтерфейс застосунку був розроблений із урахуванням специфіки роботи кадрового відділу. Основні елементи управління розташовані відповідно до логіки старої системи, що дозволило мінімізувати час адаптації користувачів після перенесення [HRM](#) на сучасні ОС. Водночас інтерфейс став більш інтуїтивно зрозумілим. Було покращено структуру вкладок, спрощено навігацію між кадровими модулями та реалізовано більш логічне групування інформації. Для забезпечення стабільної роботи система підтримує механізми обробки помилок. У випадку неправильного введення даних або виникнення помилки користувач отримує пояснення причини проблеми без аварійного завершення роботи програми. Значна увага також приділялась сумісності із застарілою структурою бази даних [Firebird](#). Попри модернізацію архітектури та інтерфейсу застосунок повністю підтримує існуючу структуру БД, що дозволило уникнути втрати кадрових даних під час перенесення системи. Таким чином, реалізований [Windows Forms](#) застосунок забезпечує повноцінну автоматизацію кадрового обліку та підтримує всі основні процеси роботи відділу кадрів. Завдяки модернізації архітектури, оптимізації клієнтської логіки та використанню сучасної версії [Firebird](#) вдалося значно підвищити швидкодію системи, покращити стабільність роботи та зробити інтерфейс більш зрозумілим для користувачів.

3.3. Налагодження та тестування

Після завершення основного етапу реалізації [Windows Forms](#) застосунку було виконано комплексне налагодження та тестування системи. Основною метою цього етапу стала перевірка стабільності роботи застосунку, коректності

взаємодії із базою даних [Firebird](#), перевірка кадрових операцій та забезпечення сумісності програми із сучасними версіями операційної системи [Windows](#). Особливістю тестування даної [HRM](#)-системи стало те, що новий застосунок повинен був повністю підтримувати структуру застарілої бази даних без зміни її архітектури. Через це під час налагодження значна увага приділялась перевірці сумісності нової клієнтської логіки зі старими таблицями [Firebird](#). Тестування виконувалось у середовищі [Windows](#) 10 із використанням сучасної версії [Firebird](#). Додатково перевірялась робота застосунку при багатокористувацькому режимі доступу до кадрової бази даних. Першим етапом налагодження стала перевірка запуску застосунку та коректності ініціалізації компонентів. Під час тестування перевірялась робота підключення до [Firebird](#), завантаження репозиторіїв та ініціалізація [Presenter](#)-класів. Особлива увага приділялась роботі механізму завантаження даних. Було протестовано відкриття великих таблиць працівників, кадрових наказів та штатного розпису. У процесі перевірки контролювалась швидкість виконання [SQL](#)-запитів та стабільність роботи [DataGridView](#). Результати тестування показали, що після оптимізації клієнтської логіки та переходу на нову версію [Firebird](#) швидкість відкриття форм значно покращилась у порівнянні зі старою [HRM](#)-системою. Особливо це стало помітно при роботі із кадровими таблицями, що містять велику кількість записів. Наступним етапом стало тестування [CRUD](#)-операцій. Було перевірено створення нових працівників, редагування кадрових записів, оновлення штатного розпису та видалення інформації із бази даних. Під час тестування створення працівників перевірялась коректність збереження особистих даних, кадрових наказів, інформації про освіту, стаж та військовий облік. Особлива увага приділялась перевірці правильності створення зв'язків між таблицями [Firebird](#). Також було протестовано механізм кадрових переміщень. У процесі тестування виконувалось створення нових призначень, переведення працівників між підрозділами та оформлення звільнення співробітників. Для перевірки механізму звільнення використовувались сценарії із оновленням записів у таблиці [SOTR_DOLIN](#). Система повинна була коректно приховувати звільнених працівників зі штатного розпису та зберігати кадрову історію співробітника. Окрему увагу під час тестування приділено модулю кадрових наказів. Було перевірено роботу пошуку наказів за номером, назвою та датою створення. Також тестувалась коректність прив'язки наказів до кадрових операцій. Рис. 3.9. Приклад тестування пошуку наказу за датою У процесі перевірки механізму пошуку виконувалось тестування різних форматів дати. Система повинна була підтримувати як короткий формат дати, так і повний формат із мілісекундами. Під час налагодження також перевірялась робота механізму [client-side](#) валідації. Було протестовано реакцію системи на неправильне введення дат, порожні обов'язкові поля та некоректні числові значення. [23] Наприклад, при введенні неправильного формату дати система повинна була миттєво відобразити повідомлення про помилку без надсилання [SQL](#)-запиту до [Firebird](#). Подібна поведінка значно покращила взаємодію користувача із застосунком та дозволила уникнути багатьох помилок введення. Також перевірялась робота механізму захисту від дублювання записів. Під час створення нового працівника система виконувала перевірку наявності співробітника із аналогічними персональними даними. У процесі тестування було підтверджено, що система коректно блокує створення дублікатів та відображає відповідне повідомлення користувачу. Важливим етапом стало тестування роботи [DataGridView](#). Оскільки саме цей компонент використовується для відображення більшості кадрових даних, перевірялась стабільність його роботи при великій кількості записів. Було протестовано сортування таблиць, редагування значень безпосередньо у клітинках, роботу [DataGridViewComboBoxColumn](#) та автоматичне оновлення даних через [BindingSource](#). Під час перевірки було підтверджено, що таблиці коректно оновлюються після змін у колекціях моделей без необхідності ручного перезавантаження інтерфейсу. [24] Окрему увагу приділено тестуванню механізму генерації [PDF](#)-документів. Перевірялась коректність формування [HTML](#)-шаблонів та робота рендерингу через [Chromium](#). У процесі тестування виконувалась генерація [PDF](#)-документів для таблиць стажу, відпусток та кадрових наказів. Контролювалась правильність форматування таблиць, відображення шрифтів та друкованого вигляду сторінок. Результати перевірки показали, що система коректно формує [PDF](#)-файли та підтримує друк кадрової документації без втрати форматування. Також було виконано тестування роботи застосунку при повторному відкритті форм та багаторазовому використанні [Presenter](#)-класів. Особлива увага приділялась перевірці механізмів підписки та відписки від подій [WinForms](#). У процесі налагодження було виявлено декілька потенційних проблем із дублюванням обробників подій при повторному відкритті форм. Для усунення цієї проблеми було реалізовано механізм попередньої відписки від подій перед повторною ініціалізацією

Presenter. Окремо виконувалось тестування роботи застосунку при довготривалому використанні. Перевірялась стабільність роботи системи при багаторазовому відкритті кадрових модулів та виконанні великої кількості операцій із базою даних. Під час тестування контролювалось використання оперативної пам'яті та відсутність критичних витоків ресурсів. У результаті перевірки було підтверджено стабільну роботу застосунку протягом тривалого часу без необхідності перезапуску програми. Важливим етапом стала перевірка сумісності застосунку із застарілою структурою [Firebird](#). Оскільки структура бази даних залишалась практично незмінною, система повинна була коректно працювати із таблицями, створеними для попередньої версії [HRM](#). У процесі тестування перевірялась правильність роботи старих кадрових записів після перенесення бази даних через [Python](#)-алгоритм та [IBExpert](#). Було підтверджено збереження кадрової інформації та коректність роботи нової версії застосунку із перенесеними даними. Для додаткової перевірки виконувалось тестування роботи пошукових механізмів при великій кількості записів у [Firebird](#). Зокрема, перевірялась швидкість пошуку працівників, наказів та кадрових переміщень. Результати тестування показали, що після модернізації застосунок працює значно швидше за стару [HRM](#)-систему. Оптимізація [SQL](#)-запитів, використання сучасної версії [Firebird](#) та перенесення бізнес-логіки у клієнтський застосунок дозволили суттєво покращити швидкодію системи. Таким чином, у процесі налагодження та тестування було перевірено всі основні компоненти [HRM](#)-системи, включаючи роботу із базою даних [Firebird](#), кадровими наказами, штатним розписом, генерацією [PDF](#)-документів та механізмами [client-side](#) валідації. Результати тестування підтвердили стабільність роботи застосунку, сумісність із [Windows](#) 10 та коректність роботи із перенесеною кадровою базою даних.

ЗАГАЛЬНІ ВИСНОВКИ У процесі виконання дипломної роботи було реалізовано перенесення застарілої [HRM](#)-системи на сучасні операційні системи із одночасною модернізацією її архітектури, клієнтської логіки та окремих функціональних модулів. У ході роботи було виконано аналіз інтерфейсу та структури бази даних старої кадрової системи. На основі [UML](#)-діаграм та аналізу поведінки існуючої програми було відтворено логіку роботи основних кадрових модулів без доступу до оригінального вихідного коду. Це дозволило забезпечити сумісність нової версії застосунку зі структурою попередньої [HRM](#)-системи та зберегти наявні кадрові дані. У результаті дослідження було встановлено, що стара система мала застарілу архітектуру, використовувала значну кількість [SQL](#)-тригерів та серверної логіки, а також мала обмежену сумісність із сучасними версіями [Windows](#). Для вирішення цієї проблеми було обрано архітектурний підхід [MVP](#), що дозволив розділити інтерфейс, бізнес-логіку та роботу із базою даних. [25] У процесі реалізації було створено новий [Windows Forms](#) застосунок на платформі [.NET Framework](#) із підтримкою [Windows](#) 10. Для роботи із базою даних використано сучасну версію [Firebird](#), що дозволило забезпечити стабільну роботу системи та покращити продуктивність кадрових операцій. Особлива увага приділялась перенесенню бази даних. Було реалізовано спеціальний [Python](#)-алгоритм та використано середовище [IBExpert](#) для міграції кадрової інформації із застарілої [Firebird](#) БД у нову версію системи без втрати даних та зміни структури таблиць. Під час модернізації значна частина серверної логіки була перенесена у клієнтський застосунок. Це дозволило реалізувати сучасну [client-side](#) валідацію, зменшити навантаження на базу даних та суттєво підвищити швидкодію системи. У результаті розробки було реалізовано повноцінний кадровий застосунок, який підтримує роботу із підрозділами, штатним розписом, персональними картками працівників, кадровими наказами, стажем, відпустками та кадровими переміщеннями. Також у системі реалізовано автоматичну генерацію [PDF](#)-документів на основі [HTML](#)-шаблонів та [Chromium](#)-рендерингу. Це дозволило спростити формування кадрової документації та автоматизувати процес створення звітів. У процесі тестування було перевірено стабільність роботи застосунку, коректність роботи із [Firebird](#), швидкодію кадрових модулів та сумісність із сучасними операційними системами [Windows](#). Результати тестування підтвердили правильність обраних архітектурних рішень та стабільну роботу системи при великій кількості кадрових записів. Таким чином, у ході дипломної роботи було успішно виконано перенесення застарілої [HRM](#)-системи на сучасні ОС та проведено її модернізацію. Реалізований застосунок забезпечує повноцінну автоматизацію кадрового обліку, підтримує сумісність із існуючою структурою бази даних та значно покращує швидкодію й зручність роботи користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ [Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 533 p.](#) [Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.](#) [Microsoft Corporation. Windows Forms Documentation. URL: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/](#) (дата звернення: 28.05.2026).

[Firebird Project. Firebird 4.0 Language Reference. URL: <https://firebirdsql.org/en/reference-manuals/>](#) (дата звернення: 28.05.2026). [Date C. J. An Introduction to Database Systems. 8th ed. Boston : Pearson, 2003. 1024 p. Silberschatz A., Korth H., Sudarshan S. Database System Concepts. 6th ed. New York : McGraw-Hill, 2010. 1376 p. IBCExpert Development Team. IBCExpert Documentation. URL: <https://www.ibexpert.net/ibe/>](#) (дата звернення: 28.05.2026). [Beaulieu A. Learning SQL. Sebastopol : O'Reilly Media, 2020. 338 p. Microsoft Corporation. .NET Framework Guide. URL: <https://learn.microsoft.com/en-us/dotnet/framework/>](#) (дата звернення: 28.05.2026). [Sauter M. The MVP Pattern for WinForms Applications. URL: <https://www.codeproject.com/Articles/10984/Model-View-Presenter-MVP-Design-Pattern>](#) (дата звернення: 28.05.2026). [Troelsen A., Japikse P. Pro C# 10 with .NET 6. New York : Apress, 2022. 1800 p. Richter J. CLR via C#. 4th ed. Redmond : Microsoft Press, 2012. 826 p. Oracle Corporation. SQL Language Reference. URL: <https://docs.oracle.com/en/database/>](#) (дата звернення: 28.05.2026). [Firebird Foundation. Firebird Project Documentation. URL: <https://firebirdsql.org/en/documentation/>](#) (дата звернення: 28.05.2026). [Microsoft Corporation. Entity Framework Core Documentation. URL: <https://learn.microsoft.com/en-us/ef/core/>](#) (дата звернення: 28.05.2026). [Freeman A. Pro ASP.NET Core MVC. 8th ed. New York : Apress, 2024. 1040 p. Firebird Project. Firebird 5.0 Release Notes. URL: <https://firebirdsql.org/en/firebird-5-0/>](#) (дата звернення: 28.05.2026). [IBPhoenix. Firebird Performance Guide. URL: <https://www.ibphoenix.com/articles/art-00000399>](#) (дата звернення: 28.05.2026). [Chromium Developers. Headless Chrome Documentation. URL: <https://developer.chrome.com/docs/chromium/headless>](#) (дата звернення: 28.05.2026). [Firebird Foundation. Firebird 5.0 New Features. URL: \[https://firebirdsql.org/file/documentation/release_notes/html/en/5_0/rlsnotes50.html\]\(https://firebirdsql.org/file/documentation/release_notes/html/en/5_0/rlsnotes50.html\)](#) (дата звернення: 28.05.2026). [SQLite Documentation Team. Database Performance Tuning Principles. URL: <https://www.sqlite.org/queryplanner.html>](#) (дата звернення: 28.05.2026). [Microsoft Corporation. Windows Forms DataGridView Control Documentation. URL: <https://learn.microsoft.com/en-us/dotnet/desktop/winforms/controls/datagridview-control-windows-form>](#) (дата звернення: 28.05.2026). [Myers G. J., Sandler C., Badgett T. The Art of Software Testing. Hoboken : John Wiley & Sons, 2011. 256 p. Microsoft Corporation. Debugging Windows Forms Applications. URL: <https://learn.microsoft.com/en-us/visualstudio/debugger/debugging-managed-code>](#) (дата звернення: 28.05.2026). [Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2015. 816 p.](#)

ДОДАТОК А. `internal class DepartmentPresentater { public DepartmentsRepository _repository { get; set; } public DepartmentView _view { get; set; } public int _depId { get; set; } private string _depName { get; set; } public BindingSource currSotrDoljState { get; set; } private BindingSource _jobs { get; set; } public BindingSource _empls { get; set; } public BindingSource _orders { get; set; } public SotrMove _sotrMove { get; set; } public Sotr_Doljnost _sotrDoljModel { get; set; } public DepartmentPresentater(DepartmentsRepository repository, DepartmentView view, int depId) { _repository = repository; _view = view; _depId = depId; LoadAllInfo(); _view.SaveEvent += HandleSaveEvent; _view.ResetEvent += HandleResetEvent; _view.DeleteEvent += HandleDelEvent; _view.ChoosePrikazEvent += HandleChoosePrikaz;`

ДОДАТОК Б. `internal class PersdataPresentater { private PersdataView _view; private PersdataRepository _repository; private int _sotrid; private string _depName; private List<int> _genSeniority; private BindingSource _families; private BindingSource _educations; private BindingSource _eduScies; private BindingSource _eduSciNames; private BindingSource _kvalifications; private BindingSource _docs; private BindingSource _akadems; private List<SotrMove> _allMoves; private List<Sotr_sotr> _sotr; private List<Pensioner> _pensioner; private List<Tck_tckRecords> _tckRecords; public BindingSource _seniorities; public BindingSource _vacations; public BindingSource _oldMoves; public BindingSource _currMoves; public BindingSource _prices;`

ДОДАТОК В. `internal class DepartmentsPresentater { private DepartmentsRepository _repository; private DepartmentsView _view; private string _depName; private int? _depId; private List<Otdel> _allDepartments; internal sealed class SotrListItem { public int Id { get; set; } public string FullName { get; set; } } public DepartmentsPresentater(DepartmentsRepository repository, DepartmentsView view) { _repository = repository; _view = view; LoadAllData(); _view.SelectionEvent += HandleSelection; _view.PrikazyEvent += HandlePrikazy; _view.PersdataEvent += HandlePersdata; _view.CreateDepEvent += HandleCreateDep;`

ДОДАТОК Г `private void LoadAllSotrs(string search) { _view._fioDataGrid4.Rows.Clear(); _view._fioDataGrid5.Rows.Clear(); List<Sotr_Doljnost> sotrsData = _repository.Sotr_Doljnosts.Where(d => d.OtdelId == 999999999).ToList(); List<Sotr> allSotrs = _repository.Sotrs.Where(d => (d.Famil + " " + d.Name + " " + d.Otch).Contains(search)).ToList(); List<Sotr delSotrs =`

```
sotrsData.Select(e = allSotrs.FirstOrDefault(s = s.Id == e.Sotr_Id)) .Where(s = s != null) .ToList();
if (delSotrs.Count != 0) { List<int> deletedSotrs = delSotrs.OrderBy(d = (d.Famil + " " + d.Name +
" " + d.Otch)).Select(d = d.Id).ToList(); List<SotrListItem> delSotrsClass =
LoadDataInBatches(deletedSotrs); ДОДАТОК Д. private void HandleSearchData(object sender,
EventArgs e) { int tabInd = _view._depTabCtrl.SelectedIndex; string searchTxt =
_view._searchTxt.Text; if (tabInd == 2) { SetKartochka(_depld.Value, searchTxt);
LoadAllSotrs(searchTxt); } } private void HandleReloadData(object sender, EventArgs e) { if
(_depld != null) { SetKartochka(_depld.Value,
" Знайдено посилань: 0% id: 3
""
); } LoadAllSotrs(
" Знайдено посилань: 0% id: 4
""
); } } ДОДАТОК Ж. namespace ViddilKadriy.Presentaters.Common { public class
ModelDataValidation { public void Validate(object model) { string errorMessage =
" Знайдено посилань: 0% id: 5
""
; var results = new List<ValidationResult> (); var context = new ValidationContext(model); bool
isValid = Validator.TryValidateObject(model, context, results, true); if (!isValid) { foreach (var
item in results) errorMessage +=
" _ " id: 6
" _ "
+ item.ErrorMessage +
" Знайдено посилань: 0% id: 7
" \"
; throw new Exception(errorMessage); } } } }
```

Відмова від відповідальності:

Цей звіт повинен бути правильно інтерпретований та проаналізований кваліфікованою особою, яка несе відповідальність за оцінку!

Будь-яка інформація, наведена у цьому звіті, не є остаточною та підлягає ручному огляду та аналізу. Дотримуйтесь вказівок: [Рекомендації з оцінки](#)

Детектор Плагіату - Право знати автора! ☐ SkyLine LLC