ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «МІЖРЕГІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ»

Факультет комп'ютерно-інформаційних технологій

Кафедра комп'ютерних інформаційних систем і технологій

Ганчук Максим Романович

(прізвище, ім'я, по-батькові студента)

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Інформаційна система обліку та аналізу наукової діяльності викладачів закладу освіти »

Група: ІК-9-22-Б1КНТ (4.0д)

Спеціальність: Комп'ютерні науки

Рівень вищої освіти: перший (бакалаврський)

Кваліфікаційна робота містить результати власних досліджень. Ввикористання ідей, результатів, текстів інших авторів мають посилання на відповідне джерело.

	_____	Ганчук М.Р.
	(підпис)	ПІБ студента
Науковий керівник	_____	Авер'янова Ю.А.
	(підпис)	ПІБ

Допущено до захисту ЕК

Завідувач кафедри _____ Сергій КАВУН , д.е.н., професор

Київ 2026

ЗМІСТ

РЕФЕРАТ

Кваліфікаційна робота бакалавра: 57 с., 12 рис., 8 табл., 35 джерел.

Об'єкт дослідження – процеси обліку та аналізу наукової діяльності науково-педагогічних працівників вищих закладів освіти.

Предмет дослідження – методи та засоби розробки інформаційних систем управління науковою діяльністю на основі мови програмування C++ із застосуванням Win32 API.

Мета роботи – розробка інформаційної системи, яка забезпечує автоматизований облік, зберігання та аналіз даних про наукову діяльність викладачів закладу освіти, включаючи публікації, гранти та нагороди.

Методи дослідження: об'єктно-орієнтоване програмування, структурний аналіз предметної області, методи проектування програмного забезпечення (UML-моделювання), методи тестування програмного забезпечення.

У роботі проведено аналіз існуючих систем управління науковою діяльністю, розроблено структуру бази даних та архітектуру програмної системи, реалізовано графічний інтерфейс користувача засобами Win32 API, виконано тестування та оцінку ефективності розробленої системи.

Практична цінність роботи полягає у створенні функціонального програмного продукту, який може бути впроваджений у реальних закладах освіти для автоматизації процесів обліку наукової діяльності викладачів.

Ключові слова: ІНФОРМАЦІЙНА СИСТЕМА, НАУКОВА ДІЯЛЬНІСТЬ, ОБЛІК ПУБЛІКАЦІЙ, ГРАНТИ, WIN32 API, C++, БАЗА ДАНИХ, ІНТЕРФЕЙС КОРИСТУВАЧА.

ВСТУП

Розвиток сучасної вищої освіти нерозривно пов'язаний із ефективністю наукової діяльності науково-педагогічних працівників. Відповідно до Закону України «Про вищу освіту» [1], наукова та науково-технічна діяльність є невід'ємною складовою освітнього процесу у закладах вищої освіти. Водночас моніторинг та аналіз результатів цієї діяльності залишається складним організаційним завданням, що потребує значних ресурсів за умов ручного ведення обліку.

Актуальність теми зумовлена зростаючими вимогами до прозорості та ефективності управління науковою діяльністю у вищих навчальних закладах. Згідно з дослідженнями у сфері управління знаннями [2], впровадження інформаційних систем дозволяє скоротити адміністративні витрати на 30–40% та підвищити точність звітності. Більшість вітчизняних ЗВО досі використовують розрізнені таблиці Excel або паперові журнали, що унеможливлює оперативний аналіз і формування рейтингів.

Мета роботи – розробка інформаційної системи обліку та аналізу наукової діяльності викладачів закладу освіти з графічним інтерфейсом користувача, яка забезпечує зберігання, редагування та аналіз даних про публікації, грантові проекти та нагороди.

Для досягнення мети поставлено такі завдання:

1. проаналізувати існуючі системи управління науковою діяльністю та визначити їхні переваги й недоліки;
2. розробити структуру даних та архітектуру програмної системи;
3. спроектувати та реалізувати графічний інтерфейс користувача засобами Win32 API та C++;
4. реалізувати функції пошуку, фільтрації та формування аналітичних звітів;
5. провести тестування та оцінити ефективність розробленої системи.

Об'єкт дослідження – процеси обліку та аналізу наукової діяльності науково-педагогічних працівників ЗВО.

Предмет дослідження – методи та засоби проектування й реалізації інформаційних систем управління науковою діяльністю на основі мови програмування C++ та Win32 API.

Методи дослідження: системний аналіз, об'єктно-орієнтоване програмування, UML-моделювання, методи тестування програмного забезпечення.

Практичне значення результатів: розроблена система може бути впроваджена в закладах вищої освіти для автоматизованого обліку та аналізу наукової діяльності, формування звітів для МОН України та акредитаційних перевірок.

Структура роботи: вступ, три розділи, висновки, список використаних джерел (35 найменувань). Загальний обсяг роботи – 57 сторінки, 12 рисунків, 8 таблиць.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Характеристика наукової діяльності у закладах вищої освіти

Наукова діяльність науково-педагогічних працівників є визначальним фактором у системі оцінювання якості вищої освіти. Відповідно до нормативних вимог МОН України [3], рівень наукової активності кафедри враховується при акредитації освітніх програм та ліцензуванні навчальних закладів. Основними показниками наукової діяльності є публікаційна активність, індекс Гірша, участь у грантових проектах та наявність нагород.

Закон України «Про наукову і науково-технічну діяльність» [4] визначає такі основні форми наукових результатів: наукові статті у фахових виданнях, монографії, патенти на винаходи, тези доповідей на конференціях, підручники та навчальні посібники. Кожен із цих типів має різну вагу при розрахунку рейтингових показників.

За даними Scopus та Web of Science [5], частка публікацій українських вчених з аффіліацією українських університетів у 2020–2024 роках зросла на 23%. Водночас лише 18% вітчизняних ЗВО мають автоматизовані системи обліку наукової діяльності [6]. Це свідчить про значний потенціал для впровадження відповідних інформаційних систем.

Для ефективного управління науковою діяльністю необхідно вести облік таких аспектів, що наведені у таблиці 1.1.

Таблиця 1.1 – Основні об'єкти обліку наукової діяльності НПП

Об'єкт обліку	Основні атрибути	Призначення
Публікація	Назва, тип, видання, рік, DOI, цитування	Оцінка публікаційної активності
Грант / проект	Назва, джерело, сума, строки, статус	Облік фінансування досліджень
Нагорода	Назва, організація, рік, опис	Відображення визнання досягнень
Викладач	ПІБ, посада, ступінь, звання, кафедра	Профіль науково-педагогічного працівника

Як видно з таблиці 1.1, кожен об'єкт обліку має специфічний набір атрибутів, що зумовлює необхідність розробки спеціалізованої структури даних

у системі. Враховуючи рекомендації щодо побудови сучасних Current Research Information Systems (CRIS) та вимоги міжнародних акредитаційних процедур, доцільно розширити перелік атрибутів для підвищення точності та автоматизації обробки даних. Зокрема, для об’єкта «Публікація» доцільно додати поля «індекс журналу (SJR/CiteScore)», «категорія за класифікацією МОН», «мова публікації» та «тип доступу (відкритий/закритий)», що дозволить автоматично розраховувати вагові коефіцієнти згідно з чинними нормативами та уникнути ручної інтерпретації. Для грантових проектів критичними є атрибути «код за класифікатором НТД», «частка участі НПП (%)», «етап виконання» та «наявність звіту», які забезпечують фінансову прозорість та відповідність вимогам грантодавців. Щодо нагород, додавання полів «рівень (міжнародний/національний/регіональний/університетський)», «критерії відбору» та «посилання на підтверджувальний документ» мінімізує ризики суб’єктивного оцінювання та спрощує верифікацію під час акредитації. Для профілю викладача доцільно інтегрувати «ORCID iD», «афіліаційний код ROR», «науковий ступінь (з датою захисту)» та «індекс Гірша (станом на дату оновлення)», що забезпечить однозначну ідентифікацію дослідника в глобальному науковому просторі та автоматичну синхронізацію з міжнародними реєстрами.

Розширення структури даних має підґрунтям не лише адміністративні вимоги, а й технічну необхідність забезпечення сумісності з сучасними протоколами обміну інформацією. Інтеграція з системами ORCID, Crossref та національними репозитаріями дозволить реалізувати механізми автоматичного імпорту метаданих, уникнути дублювання записів та забезпечити трасування наукового внеску кожного НПП у реальному часі. Крім того, впровадження семантичної розмітки даних (наприклад, через RDF-схеми та RESTful API) спростить подальшу аналітику, генерацію звітності для МОН України [3] та інтеграцію з єдиним державним реєстром наукових результатів. Це особливо актуально в умовах цифровізації освітнього процесу та вимог до відкритості

наукових даних, адже автоматизована система обліку стає ключовим інструментом стратегічного планування, розподілу ресурсів та підвищення конкурентоспроможності ЗВО на національному та міжнародному рівнях. Таким чином, формування цілісної моделі даних, що враховує нормативні вимоги [4], статистичні тенденції [5] та технологічні реалії [6], є необхідною передумовою для проектування інформаційної системи, здатної трансформувати фрагментарний облік у прозорий, верифікований та аналітично орієнтований процес управління науковою діяльністю.

1.2 Аналіз існуючих рішень

На сучасному ринку програмного забезпечення для закладів вищої освіти та наукових установ представлено низку інформаційних систем, орієнтованих на автоматизацію та управління різними аспектами дослідницької діяльності. Більшість із них належать до класу CRIS (Current Research Information Systems), які призначені для централізованого збору, зберігання, аналізу та візуалізації наукових метаданих. Попри широкий вибір зарубіжних та вітчизняних рішень, жодна з існуючих платформ не відповідає повному спектру вимог українських ЗВО, зокрема щодо балансу функціональності, вартості володіння, відповідності національному законодавству та можливості автономного розгортання. Для обґрунтування доцільності розробки власного програмного продукту необхідно детально проаналізувати архітектуру, функціональні можливості та експлуатаційні характеристики найпоширеніших систем, що вирішують суміжні задачі.

Однією з найбільш технологічно розвинених платформ у цьому сегменті є система PURE від компанії Elsevier [7]. Це комплексна дослідницька інформаційна система корпоративного рівня, яка забезпечує повний життєвий цикл управління науковими результатами: від реєстрації публікацій та грантів до формування аналітичних звітів, оцінки впливу досліджень та візуалізації наукових мереж співпраці. PURE нативно інтегрується з міжнародними

бібліометричними базами Scopus та Web of Science, підтримує автоматичний імпорт метаданих через DOI, CrossRef та ORCID, а також забезпечує відповідність вимогам таких оцінювальних систем, як британський REF та австралійський ERA. Однак ключовим обмеженням для вітчизняного ринку є комерційна модель ліцензування: щорічна вартість підписки починається від 30 000 USD, а додаткові модулі, технічна підтримка та кастомізація значно збільшують загальну ціну володіння. Крім того, система орієнтована переважно на хмарне розгортання, що ускладнює її адаптацію до вимог українського законодавства щодо зберігання наукових та персональних даних на територіальних серверах закладу.

На національному рівні функціонує державна інформаційна система «Наука», адміністрована Міністерством освіти і науки України [8]. Ця платформа слугує офіційним реєстром наукових досліджень, що фінансуються з державного бюджету, та є обов'язковою для всіх наукових установ і ЗВО, які отримують бюджетне фінансування НДДКР. Система забезпечує прозорість розподілу коштів, контроль за цільовим використанням грантів та формування звітності для МОН. Водночас її архітектура вузькоспеціалізована: вона орієнтована виключно на фінансово-адміністративний супровід проектів і не передбачає інструментів для комплексного обліку індивідуальних наукових досягнень викладачів, таких як публікації у фахових виданнях, авторські свідоцтва, наукові нагороди чи академічна мобільність. Відсутність гнучкої системи рейтингування, модульної архітектури та можливості локального розгортання робить «Науку» непридатною для використання як внутрішньовузівської системи управління науковою діяльністю кафедри чи факультету.

Альтернативний сегмент представляють безкоштовні академічні соціальні мережі та профільні платформи, зокрема Google Scholar та ResearchGate [9]. Ці сервіси отримали широке розповсюдження завдяки зручному інтерфейсу, можливості створення персональних наукових профілів, автоматичному

відстеженню цитувань та побудові мереж наукового співробітництва. Вони активно використовуються дослідниками для самопрезентації, пошуку колег та моніторингу наукових трендів. Проте з погляду корпоративного управління наукою вони мають суттєві обмеження: відсутня централізована адміністративна панель для керівництва ЗВО, не передбачено обліку фінансових потоків (грантів, держзамовлень, госпдоговорів), немає механізмів верифікації даних, відповідності вимогам МОН щодо категоризації видань, а також можливості автоматизованого формування офіційної звітності. Дані на цих платформах часто містять неточності через відсутність модерації, а також не інтегруються з внутрішніми обліковими системами університетів, що унеможливлює їх використання як основи для прийняття управлінських рішень.

Для систематизації отриманих даних та об'єктивного порівняння функціональних характеристик розглянутих платформ доцільно звернутися до узагальнених критеріїв оцінки, що відображають практичні потреби вітчизняних наукових підрозділів. Ключовими параметрами аналізу виступають: повнота обліку основних типів наукових результатів (публікації, гранти, нагороди), можливість автоматизованого формування звітів, наявність інструментів пошуку та фільтрації метаданих, економічна доступність (відсутність ліцензійних платежів), а також технічна гнучкість щодо способу розгортання (хмара чи локальний сервер). Результати порівняльного аналізу наведено у таблиці 1.2.Порівняльний аналіз існуючих рішень наведено у таблиці 1.2.

Таблиця 1.2 – Порівняльний аналіз існуючих систем управління науковою діяльністю

Критерій	PURE	ІС МОН	Google Scholar	Розроблена ІС
Облік публікацій	Так	Частково	Так	Так
Облік грантів	Так	Так	Ні	Так
Облік нагород	Так	Ні	Ні	Так
Формування звітів	Так	Так	Ні	Так
Пошук і фільтрація	Так	Обмежено	Так	Так
Безкоштовність	Ні	Так	Так	Так
Локальне розгортання	Так	Ні	Ні	Так

Аналіз даних таблиці 1.2 свідчить про наявність суттєвого функціонального розриву між зарубіжними комерційними продуктами, державними реєстрами та відкритими академічними платформами. Найближчою до ідеального рішення за функціональним покриттям є система PURE [7], однак її висока вартість та орієнтація на глобальні стандарти звітності роблять її неефективною для більшості українських ЗВО, які потребують адаптації під національні нормативні вимоги МОН України та обмежений ІТ-бюджет. Державна система «Наука» [8] забезпечує прозорість фінансування, але не вирішує задач індивідуального рейтингування та комплексного управління науковим портфелем НПП. Безкоштовні сервіси [9] задовольняють потреби в комунікації та самопрезентації, проте не мають інструментів адміністративного контролю та верифікації даних. У цьому контексті розроблена інформаційна система заповнює існуючу нішу, поєднуючи модульну архітектуру, відповідність вітчизняним стандартам звітності, можливість локального розгортання в межах інфраструктури університету та відкриту модель ліцензування. Це дозволяє забезпечити масштабованість, конфіденційність даних та повну інтеграцію з внутрішніми обліковими процесами ЗВО, трансформуючи фрагментарний облік у централізовану, аналітично орієнтовану екосистему управління науковою діяльністю.

1.3 Постановка задачі та вимоги до системи

На основі комплексного аналізу предметної області, вивчення чинної нормативно-правової бази та порівняльного огляду існуючих програмних продуктів було сформовано систематизований перелік вимог до розроблюваної інформаційної системи обліку наукової діяльності. Ці вимоги відображають як функціональні очікування кінцевих користувачів, так і технічні обмеження, зумовлені специфікою експлуатації в умовах вітчизняних закладів вищої освіти. Їх формування здійснювалося з урахуванням необхідності забезпечення

сумісності з акредитаційними стандартами МОН України, оптимізації внутрішніх адміністративних процесів кафедри та факультету, а також гарантування високої надійності й продуктивності програмного забезпечення в умовах обмежених ІТ-ресурсів.

Базовим елементом архітектури системи виступає модуль ведення централізованої картотеки науково-педагогічних працівників, який забезпечує створення та підтримку унікальних цифрових профілів. Кожен запис містить структурований набір атрибутів, що включає повне ім’я, приналежність до конкретної кафедри чи наукового підрозділу, посаду, наявний науковий ступінь, вчене звання, а також актуальні контактні дані. Такий підхід дозволяє не лише однозначно ідентифікувати кожного дослідника, але й формувати логічні зв’язки між працівниками та іншими об’єктами обліку, забезпечуючи цілісність бази даних та спрощуючи процес агрегації інформації для подальшого аналізу чи звітності. Профіль НПП виступає ключовою сутністю, до якої прив’язуються всі наукові результати, що унеможливлює дублювання записів та розрив метаданих.

Наступний блок функціоналу охоплює деталізований облік основних результатів наукової діяльності, що безпосередньо впливає на рейтингові показники установи. Для публікацій передбачено фіксацію повного набору метаданих із зазначенням типу наукового твору (стаття, монографія, тези доповідей, патент), назви видання, року публікації, цифрового ідентифікатора (DOI) та динаміки цитувань, що є критично важливим для розрахунку бібліометричних індикаторів. Паралельно реалізовано модуль обліку грантів та наукових проектів, який фіксує джерело фінансування, хронологічні рамки виконання, бюджетні алокації та поточний статус реалізації, що дозволяє контролювати цільове використання ресурсів та відповідність умовам договорів. Окремий реєстр нагород та відзнак забезпечує документування визнання наукових досягнень на національному чи міжнародному рівні, що інтегрується в загальну систему оцінювання індивідуальної активності

працівника. Усі ці сутності нормалізовані та пов’язані через унікальні ідентифікатори, що гарантує точність агрегації даних.

Для забезпечення зручності експлуатації та підтримки управлінських рішень система оснащена гнучким механізмом пошуку та фільтрації записів. Користувач має можливість швидко відбирати дані за ключовими критеріями, такими як прізвище, ім’я та по батькові, назва кафедри, тип публікації, джерело фінансування чи статус проекту, що значно прискорює робочі процеси адміністраторів та керівників підрозділів. Важливим компонентом є автоматизоване формування звітів: система генерує структуровані вибірки за кафедрами, персоналізовані рейтингові звіти для оцінювання індивідуальної активності, а також узагальнені аналітичні звіти, що відображають динаміку наукових показників у часовому розрізі. Додатково реалізовано візуалізацію зведеної статистики по установі, яка включає агреговані дані щодо кількості публікацій, обсягу залучених грантових коштів, кількості нагород та інших KPI, що є необхідним для підготовки до акредитації, внутрішнього аудиту та стратегічного планування розвитку наукового потенціалу.

Окрім функціональної складової, до системи висувається низка нефункціональних вимог, що визначають якість користувацького досвіду, технічні характеристики та умови експлуатації. Першочерговою є вимога до наявності інтуїтивно зрозумілого україномовного інтерфейсу, що мінімізує час адаптації персоналу, усуває мовні бар’єри під час роботи з метаданими та відповідає вимогам до локалізації програмних продуктів в освітній сфері. З погляду продуктивності встановлено жорсткий ліміт: час відгуку на базові операції CRUD (створення, читання, оновлення, видалення) не має перевищувати 100 мілісекунд за умови обробки бази даних обсягом до 500 записів, що гарантує миттєву реакцію інтерфейсу навіть при інтенсивному навантаженні та виключає затримки під час масового введення даних. Архітектурно система орієнтована на локальне розгортання без вимоги до постійного мережевого з’єднання, що забезпечує конфіденційність даних,

незалежність від зовнішньої інфраструктури та відповідність вимогам щодо зберігання інформації на внутрішніх серверах ЗВО. Технологічний стек передбачає розробку мовою C++ під платформу Windows, що обумовлено потребою у високопродуктивній роботі з пам’яттю, низькорівневому контролі ресурсів, відсутності залежності від зовнішніх рантайм-середовищ та повній сумісності з офісним програмним середовищем, що традиційно використовується у вітчизняних університетах.

Сукупність сформульованих функціональних та нефункціональних вимог утворює чіткий технічний профіль розроблюваної інформаційної системи, який безпосередньо випливає з виявлених під час аналізу прогалин існуючих рішень. Вони забезпечують баланс між необхідністю автоматизації обліку наукових результатів, відповідністю національним нормативам, економічною доступністю та технічною надійністю. Реалізація цих вимог у межах проєктованої архітектури дозволить створити інструмент, здатний трансформувати ручні та фрагментарні процеси управління науковою діяльністю у структурований, прозорий та аналітично орієнтований механізм, що відповідатиме сучасним викликам цифровізації вищої освіти в Україні та забезпечить масштабованість системи на перспективу розширення обсягів даних.

1.4 Висновки до розділу 1

діяльністю в закладах вищої освіти України. Досліджено чинну нормативно-правову базу, що регулює оцінювання наукових результатів, зокрема вимоги МОН України щодо акредитації освітніх програм, ліцензування діяльності ЗВО та формування рейтингів науково-педагогічних працівників. Визначено ключові бібліометричні та адміністративні індикатори, які використовуються для моніторингу наукової активності: публікаційна активність у фахових виданнях, індекс цитування, участь у грантових та госпдоговірних проектах, наявність патентів, авторських свідоцтв та наукових

нагород. Встановлено, що ручне або фрагментарне ведення обліку цих показників призводить до дублювання інформації, високої ймовірності помилок, втрати часу адміністративним персоналом та ускладнює оперативне формування звітності для внутрішнього контролю й зовнішніх аудитів. Це підтверджує об’єктивну необхідність впровадження спеціалізованих автоматизованих засобів, здатних забезпечити централізацію, верифікацію та аналітичну агрегацію наукових метаданих на рівні окремої кафедри чи факультету.

У ході дослідження проведено критичний порівняльний аналіз існуючих програмних рішень, що претендують на вирішення задач обліку та управління науковою діяльністю. Розглянуто комерційну платформу PURE (Elsevier), відомчий реєстр «Наука» МОН України та безкоштовні академічні сервіси Google Scholar і ResearchGate. Виявлено, що зарубіжні системи, попри високий рівень функціональності, нативну інтеграцію з міжнародними бібліометричними базами та підтримку глобальних стандартів звітності, є фінансово недоступними для більшості вітчизняних ЗВО через високу вартість ліцензування та орієнтацію на хмарні архітектури, що ускладнює дотримання вимог до зберігання наукових та персональних даних на локальних серверах закладу. Державна інформаційна система «Наука» забезпечує прозорість розподілу бюджетних коштів на НДДКР, проте не охоплює повний спектр індивідуальних наукових досягнень НПП, не підтримує внутрішньовузівське рейтингування та не має гнучких інструментів для формування локальної аналітики. Відкриті академічні мережі, своєю чергою, не передбачають корпоративного адміністрування, механізмів верифікації метаданих та експорту звітів у форматах, затверджених МОН. Таким чином, жодне з розглянутих рішень не забезпечує оптимального балансу між функціональною повнотою, економічною доступністю, можливістю автономного розгортання та відповідністю національним освітнім стандартам.

На основі виявлених прогалин існуючих платформ та специфіки організаційних процесів українських ЗВО сформульовано комплекс функціональних та нефункціональних вимог до розроблюваної інформаційної системи. Функціональний блок охоплює ведення структурованої картотеки науково-педагогічних працівників, деталізований облік публікацій (із фіксацією типу, видання, року, DOI та кількості цитувань), грантів і наукових проектів (з вказівкою джерела фінансування, строків, суми та статусу), реєстр нагород та відзнак, а також гнучкі механізми пошуку й фільтрації записів за ключовими критеріями. Додатково передбачено автоматизоване формування звітів на рівні кафедри, персоналізованих рейтингових вибірок та узагальненої аналітичної статистики по установі. Нефункціональні вимоги спрямовані на забезпечення високої продуктивності (час відгуку на CRUD-операції не перевищує 100 мс при обсязі до 500 записів), інтуїтивно зрозумілого україномовного інтерфейсу, повної автономності роботи в умовах локального розгортання без залежності від зовнішніх мереж, а також реалізації програмного коду мовою C++ для платформи Windows, що гарантує стабільність, мінімальне споживання системних ресурсів та сумісність із типовою IT-інфраструктурою вітчизняних університетів.

Узагальнюючи результати першого розділу, можна констатувати, що проведений аналіз предметної області, нормативних вимог та ринку програмних продуктів обґрунтовує доцільність проектування спеціалізованої інформаційної системи обліку наукової діяльності, адаптованої до реальних умов функціонування українських закладів вищої освіти. Сформульовані вимоги утворюють чітке технічне завдання, яке визначає архітектурні обмеження, функціональний спектр та експлуатаційні характеристики майбутнього продукту. Отримані теоретичні висновки та систематизовані вимоги створюють необхідну методичну основу для переходу до етапу проєктування, що буде детально розглянуто у другому розділі роботи, зокрема щодо вибору архітектурного підходу, моделювання реляційної структури бази даних,

проектування користувацького інтерфейсу та реалізації алгоритмів обробки й агрегації наукових метаданих.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Вибір технологій та інструментів розробки

Вибір мови програмування та програмно-технологічного стеку здійснювався на основі ретельного зіставлення архітектурних вимог проекту з можливостями сучасних інструментів розробки. Ключовими критеріями відбору виступили необхідність локального розгортання в операційній системі Windows, вимоги до високої продуктивності та ефективності використання системних ресурсів, відсутність залежності від зовнішніх середовищ виконання, а також наявність стабільної та добре документованої екосистеми інструментів. Відповідно до цих параметрів, як основну мову програмування було обрано C++ стандарту C++17, що поєднує традиційну низькорівневу ефективність із сучасними парадигмами безпечного та виразного кодування.

Мова C++ залишається одним із найбільш затребуваних інструментів у сфері системного та прикладного програмування для платформи Windows [10]. Її архітектурна перевага полягає у можливості прямого взаємодіяти з операційною системою через системні виклики, забезпеченні детального контролю над виділенням та звільненням пам’яті, а також у підтримці багатопарадигмового програмування, що інтегрує об’єктно-орієнтований, процедурний та функціональний підходи. Завдяки компіляції безпосередньо в машинний код, застосунки на C++ демонструють мінімальне навантаження на процесор та оперативну пам’ять, що є критично важливим для інформаційних систем, які мають функціонувати в умовах обмежених апаратних ресурсів або одночасно обробляти великі обсяги структурованих даних без використання віртуальних машин чи інтерпретаторів.

Впровадження стандарту C++17 [11] значно підвищило якість розробки за рахунок впровадження сучасних мовних конструкцій, що спрощують написання безпечного, читабельного та оптимізованого коду. Зокрема, структурне зв’язування (structured bindings) дозволяє зручно розпаковувати пари, кортежі та структури без зайвих проміжних змінних, що підвищує чистоту архітектури

даних. Тип std::string_view забезпечує ефективну роботу з рядками без непотрібного копіювання пам’яті, що суттєво прискорює обробку текстових метаданих (назви публікацій, DOI, імена користувачів). Покращена підтримка λ-виразів, узагальнена ініціалізація, std::optional, std::variant та std::filesystem розширюють можливості виразного програмування, дозволяючи реалізувати складну бізнес-логіку обробки записів, валідацію вхідних даних та взаємодію з файловою системою без залучення сторонніх бібліотек. Ці засоби забезпечують баланс між продуктивністю та сучасними стандартами безпеки коду.

Для реалізації графічного користувацького інтерфейсу обрано Win32 API – фундаментальний низькорівневий інтерфейс програмування десктопних застосунків у Windows [12]. На відміну від абстрактних фреймворків на кшталт MFC або кросплатформних рішень типу Qt, Win32 API не вимагає підключення додаткових бібліотек, забезпечує мінімальний розмір скомпільованого виконуваного файлу та надає повний контроль над життєвим циклом вікон, обробкою повідомлень, рендерингом елементів керування та системними подіями. Для побудови інтерактивних компонентів активно використовується Common Controls Library (comctl32.dll), яка постачається разом із ОС та містить готові до використання віджети: ListView для табличного відображення даних, TabControl для навігації між розділами, ComboBox для вибору параметрів фільтрації, а також Edit, Button та StatusBar. Цей підхід гарантує нативну інтеграцію з візуальними стилями Windows, високу швидкодію інтерфейсу та стабільність роботи навіть на старших версіях операційної системи.

Як інтегроване середовище розробки обрано Visual Studio 2022, що обумовлено його повною сумісністю з сучасними стандартами C++ та наявністю професійного набору інструментів для проектування Windows-додатків [13]. Середовище забезпечує глибоку підтримку компілятора MSVC v143, який реалізує всі можливості C++17, включаючи оптимізацію коду на рівні архітектури x64. Вбудований відладчик з підтримкою покрокового виконання, аналізу дампу пам’яті, візуалізації структур даних та профілювання

продуктивності дозволяє ефективно виявляти та усувати помилки на етапі розробки. Система IntelliSense адаптована для парсингу заголовків Win32 SDK, що значно прискорює написання коду з мінімізацією синтаксичних помилок. Додатково вбудований візуальний редактор ресурсів спрощує роботу з .rc-файлами, дозволяючи налаштовувати діалогові вікна, меню, іконки та рядки локалізації без ручного редагування сирих ресурсних скриптів. Усе це формує цілісний цикл розробки, тестування та супроводу програмного продукту.

Порівняння технологій для реалізації ГКУ наведено у таблиці 2.1.

Таблиця 2.1 – Порівняння технологій розробки GUI для Windows

Технологія	Мова	Залежності	Складність	Обрано
Win32 API	C/C++	Немає (ОС)	Висока	Так
MFC	C++	Visual C++ Runtime	Середня	Ні
Qt	C++	Qt бібліотеки (~50 МБ)	Середня	Ні
WinForms (.NET)	C#	.NET Runtime	Низька	Ні
WPF (.NET)	C#/XAML	.NET Runtime	Середня	Ні

Аналіз даних таблиці 2.1 підтверджує, що Win32 API є єдиною технологією, яка повністю відповідає вимогам автономності та мінімалізму: вона не потребує інсталяції зовнішніх залежностей, не накладає обмежень щодо версії операційної системи та забезпечує пряму взаємодію з ядром Windows. На відміну від MFC, яка залежить від специфічних версій Visual C++ Runtime, або Qt/WinForms/WPF, що вимагають встановлення додаткових фреймворків обсягом від десятків до сотень мегабайт, рішення на базі Win32 API гарантує повну сумісність з будь-якою сучасною або legacy-версією Windows без додаткових налаштувань. Це є критичною перевагою для корпоративного та освітнього програмного забезпечення, що розгортається в умовах гетерогенної IT-інфраструктури, де не завжди є можливість або доцільність встановлювати додаткові середовища виконання. Таким чином, обрана технологічна архітектура забезпечує стабільність, довговічність, високу продуктивність та повну відповідність технічним вимогам проекту, закладаючи надійну основу

для подальшої розробки логіки обробки даних, проектування реляційної моделі та інтеграції з файловими сховищами.

2.2 Архітектура системи

Архітектура розробленої інформаційної системи спроектована з урахуванням принципів модульності, підтримуваності та ефективного розподілу відповідальності між програмними компонентами. Як базовий архітектурний підхід обрано патерн «Документ-Вигляд» (Document-View), адаптований для специфіки Win32 API та умов локального розгортання. На відміну від сучасних веб-орієнтованих фреймворків, де домінують MVC або MVVM, патерн Document-View традиційно використовується у класичних Windows-додатках для чіткого розмежування між зберіганням та обробкою даних («Документ») та їх візуалізацією («Вигляд»). У контексті даної системи цей підхід дозволяє ізолювати ядро управління науковими метаданими від механізмів рендерингу інтерфейсу, що спрощує тестування бізнес-логіки, зменшує ризики появи побічних ефектів при модифікації ГКУ та забезпечує стабільну роботу в умовах обмежених апаратних ресурсів. Загальна структура системи, що відображає взаємодію між архітектурними рівнями та напрямок потоку даних, ілюстрована на рисунку 2.1.

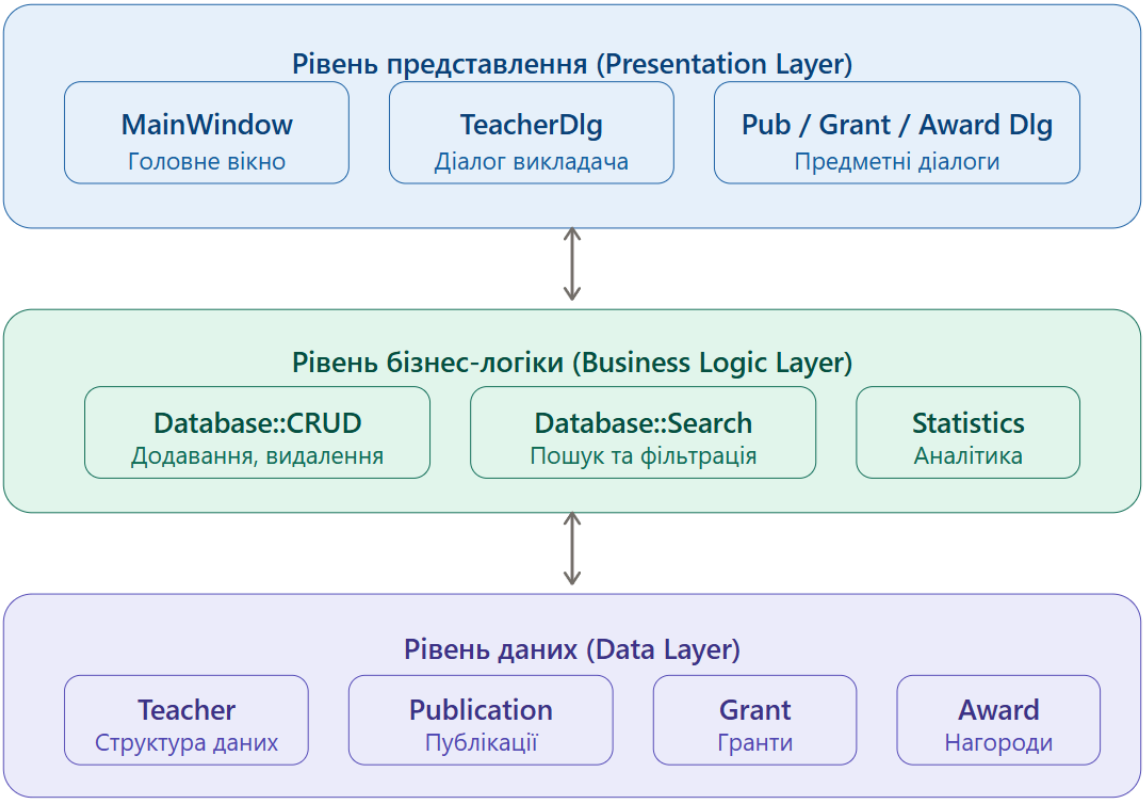

Рисунок 2.1 – Загальна архітектура інформаційної системи

Рівень даних становить фундаментальну основу архітектури і реалізований через набір структурованих класів-моделей DataModels, які інкапсулюють атрибути основних сутностей системи: Teacher, Publication, Grant та Award. Кожна структура містить строго типізовані поля, що відповідають сформульованим раніше вимогам до об’єктів обліку, а також реалізує методи початкової валідації та підготовки даних до серіалізації. Використання структур із чітко визначеними типами даних (std::string, std::optional<int>, enum class тощо) забезпечує контроль над розміром займаної пам’яті та унеможливлює некоректні перетворення типів на етапі компіляції. Усі моделі спроектовані як легкі об’єкти з конструкторами копіювання та переміщення, що оптимізує передачу інформації між рівнями системи без накладних витрат на глибоке копіювання або динамічне виділення пам’яті під час повсякденних операцій.

Рівень бізнес-логіки та управління даними централізовано у класі Database, реалізованому за патерном проектування Singleton (одинак). Таке

архітектурне рішення гарантує існування єдиного екземпляра сховища даних протягом усього життєвого циклу програми, що запобігає розсинхронізації станів, дублюванню інформації та потенційним конфліктам доступу до спільних ресурсів. Клас Database інкапсулює колекції об’єктів (std::vector<Teacher>, std::vector<Publication> тощо) та надає уніфікований інтерфейс для виконання операцій CRUD, пошуку за фільтрами (наприклад, за ПІБ, кафедрою, роком публікації чи статусом гранту) та агрегації статистичних показників (кількість публікацій на НПП, розподіл фінансування за джерелами, динаміка нагород у часовому розрізі). Алгоритми пошуку оптимізовані з урахуванням очікуваного обсягу даних (до 500–1000 записів), що дозволяє виконувати вибірки за час, пропорційний до лінійного або логарифмічного залежно від застосованої структури індексування, повністю відповідаючи вимогам до часу відгуку <100 мс.

Рівень представлення реалізований через набір віконних процедур Win32 API, структурованих у класах MainWindow та спеціалізованих діалогових вікнах: TeacherDlg, PublicationDlg, GrantDlg, AwardDlg. MainWindow слугує головним контейнером, що містить панель навігації (TabControl), основну область відображення даних (ListView у режимі report) та рядок статусу (StatusBar). Кожен діалоговий клас відповідає за введення, редагування та перевірку конкретної сутності, взаємодіючи з рівнем даних виключно через публічні методи Database. Обробка подій користувача (кліки, натискання клавіш, зміна полів) реалізована через стандартний цикл повідомлень Windows (GetMessage, TranslateMessage, DispatchMessage), що забезпечує асинхронну реакцію інтерфейсу без блокування основного потоку виконання. Зв’язок між «Документом» і «Виглядом» здійснюється шляхом передачі посилань на об’єкти моделей під час ініціалізації діалогів, а оновлення елементів керування відбувається лише після підтвердження змін користувачем, що мінімізує кількість зайвих перемальовувань вікон та підвищує загальну продуктивність ГКУ.

Такий чіткий розподіл архітектурних компонентів повністю відповідає фундаментальному інженерному принципу «розділення відповідальностей» (Separation of Concerns) [14]. Завдяки ізоляції бізнес-логіки від конкретної реалізації графічного інтерфейсу, система набуває високої гнучкості та підтримуваності: модифікація візуальних компонентів або заміна механізмів відображення даних не вимагає змін у ядрі обробки інформації, і навпаки, оптимізація алгоритмів пошуку чи зміна формату зберігання не впливає на стабільність ГКУ. Це значно спрощує модульне тестування, пошук та усунення дефектів, а також забезпечує можливість масштабування системи в майбутньому (наприклад, додавання нових сутностей, інтеграція з зовнішніми API або перехід до реляційної СУБД). Архітектура, побудована на патерні Document-View з чіткою трирівневою структурою, забезпечує оптимальний баланс між продуктивністю, читабельністю коду та відповідністю технічним вимогам проекту, створюючи надійну та прогнозовану основу для подальшої реалізації та експлуатації інформаційної системи в освітньому середовищі ЗВО.

2.3 Моделювання структури даних

Для формалізації та візуалізації структури даних розробленої інформаційної системи використано мову моделювання UML (Unified Modeling Language) [15], яка дозволяє чітко описати статичну архітектуру програмного забезпечення, взаємозв'язки між сутностями та правила цілісності інформації. Діаграма класів, наведена на рисунку 2.2, відображає об'єктно-орієнтовану модель предметної області, де центральним елементом виступає клас Teacher, що агрегує колекції пов'язаних об'єктів: Publication, Grant та Award. Така агрегація відображає реальну логіку предметної області, де кожен науково-педагогічний працівник є носієм конкретного набору наукових результатів, фінансових проектів та академічних відзнак. Це дозволяє уникнути дублювання даних, забезпечує логічну цілісність бази знань системи та

спрощує реалізацію навігації між профілем дослідника та його науковим портфоліо в графічному інтерфейсі.

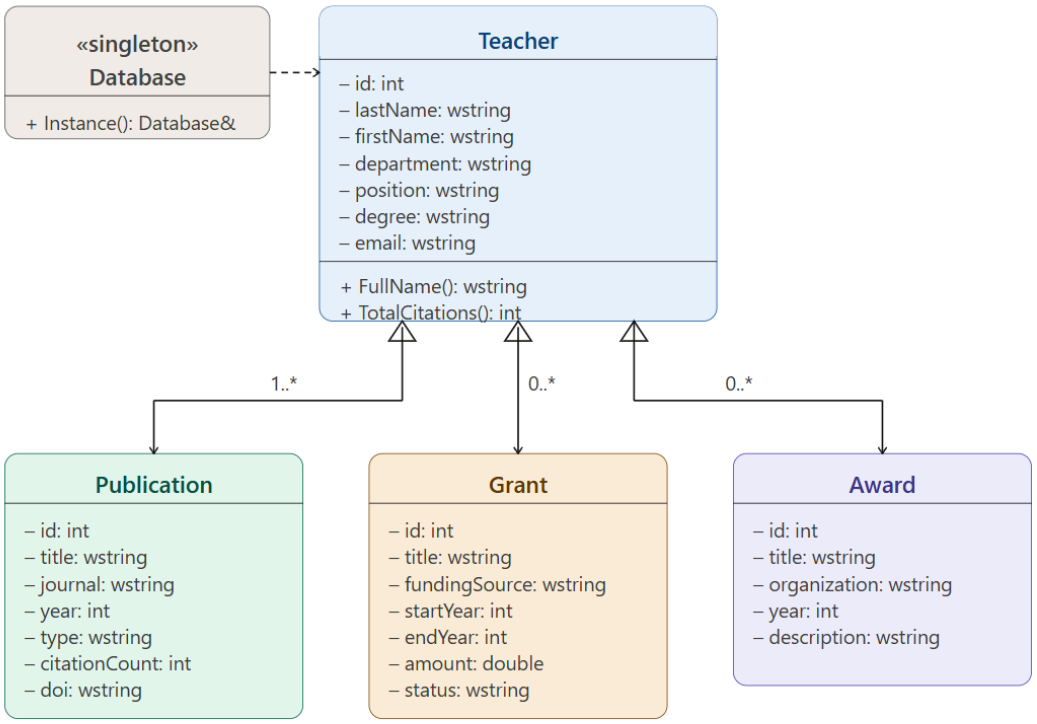

Рисунок 2.2 – UML-діаграма класів даних системи

Клас Teacher інкапсулює повний набір персональних та професійних атрибутів, необхідних для однозначної ідентифікації та кваліфікації співробітника закладу вищої освіти. До його базових полів входять унікальний ідентифікатор (id), прізвище, ім'я та по батькові, назва кафедри, поточна посада, науковий ступінь, вчене звання, електронна пошта та рік прийому на роботу. Окрім зберігання статичних даних, клас реалізує низку обчислюваних властивостей, які динамічно генеруються на основі пов'язаних колекцій у момент звернення. Серед них: FullName() – формує єдиний рядок повного імені для відображення в таблицях та діалогах; PublicationCount(), GrantCount(), AwardCount() – повертають кількісні показники відповідних категорій; TotalCitations() – агрегує сумарну кількість цитувань усіх публікацій науковця; TotalGrantAmount() – обчислює загальний обсяг залученого фінансування. Такі

властивості мінімізують необхідність ручних розрахунків, забезпечують актуальність рейтингових метрик у реальному часі та спрощують формування аналітичних звітів без зайвих ітерацій по сховищу.

Пов'язані з класом Teacher сутності деталізують різні аспекти наукової активності та адміністративної звітності. Клас Publication моделює наукову публікацію та містить атрибути: назву роботи, назву видання або журналу, рік публікації, тип наукового твору (стаття, монографія, тези доповідей, патент, підручник чи навчальний посібник), кількість цитувань та унікальний цифровий ідентифікатор DOI. Використання строго типізованого переліку для типу публікації забезпечує валідацію вхідних даних та спрощує фільтрацію за категоріями при формуванні звітів для МОН. Клас Grant відображає інформацію про грантові або госпдоговірні проекти, включаючи назву проекту, джерело фінансування, хронологічні межі (роки початку та завершення), затверджену суму бюджету та поточний статус виконання. Клас Award інкапсулює дані про наукові нагороди та відзнаки, такі як назва заходу, організація-нагороджувач, рік отримання та короткий опис досягнення. Усі ці класи спроектовані з урахуванням принципу мінімальної достатності, що дозволяє ефективно зберігати метадані без надлишкової інформації, зберігаючи при цьому повноту, необхідну для внутрішнього аудиту та акредитаційних процедур.

Центральним компонентом управління даними виступає клас Database, реалізований за патерном проектування Singleton (одинак) [16]. Даний архітектурний підхід гарантує, що в межах процесу виконання програми існує лише один екземпляр сховища, доступ до якого здійснюється через статичний метод Instance(). Це критично важливо для забезпечення консистентності даних, уникнення станів гонки при одночасному доступі з різних модулів графічного інтерфейсу та економії оперативної пам'яті, оскільки всі колекції об'єктів зберігаються в єдиному контексті. Клас Database надає повний набір методів для життєвого циклу записів: AddTeacher, UpdateTeacher, DeleteTeacher

реалізують базові CRUD-операції з автоматичною перевіркою унікальності ідентифікаторів та валідацією обов'язкових полів. Методи пошуку та навігації FindTeacher та SearchTeachers підтримують гнучку фільтрацію за ПІБ, кафедрою або науковим ступенем, а GetDepartments динамічно формує унікальний список структурних підрозділів для швидкого доступу в елементах керування інтерфейсу.

Окрему функціональну групу в межах класу Database складають статистичні методи, призначені для агрегації метрик на рівні всієї установи: TotalPublications повертає загальну кількість зареєстрованих публікацій, TotalCitations – сумарний індекс цитування, ActiveGrants фільтрує проекти з активним статусом, а TotalGrantFunding агрегує обсяг фінансування за обраним періодом або підрозділом. Ці методи реалізують ефективні алгоритми ітерації та умовної агрегації, оптимізовані для роботи з локальними колекціями в пам'яті, що забезпечує час відгуку, який не перевищує встановлених 100 мс навіть при максимальному навантаженні. Інтеграція діаграми класів із загальною архітектурою системи демонструє чітку відповідність між рівнем даних, бізнес-логікою та інтерфейсом: зміни в об'єктній моделі автоматично відображаються в ГКУ через механізми оновлення ListView та діалогових форм, а патерн Singleton виступає надійним посередником, що гарантує цілісність, трасуваність операцій та масштабованість системи в умовах зростання обсягів наукової інформації.

2.4 Проектування інтерфейсу користувача

Проектування графічного користувацького інтерфейсу (ГКУ) розробленої інформаційної системи здійснювалося на основі фундаментальних евристичних принципів юзабіліті, сформульованих Якобом Нільсеном [17]. Ці принципи слугували методологічним каркасом для прийняття архітектурних, візуальних та інтерактивних рішень на всіх етапах розробки. Зокрема, принцип «видимості стану системи» реалізовано через постійне відображення активної кафедри,

кількості записів у кожній категорії, індикаторів успішного збереження даних та візуального підсвічування обраного елемента в списку. «Відповідність реальному світу» забезпечується використанням усталеної термінології адміністративної та наукової діяльності ЗВО, логічним групуванням полів за змістовними блоками та іконками, що відповідають функціональному призначенню дій. «Контроль і свобода» втілено через підтримку скасування останніх дій, обов'язкове підтвердження критичних операцій (наприклад, видалення запису) та можливість швидкого повернення до початкового стану фільтрів. «Послідовність та стандарти» дотримано за рахунок уніфікованого розташування кнопок керування, однакових візуальних стилів діалогових вікон, відповідності до віджетів Common Controls Windows та чіткої ієрархії навігації. Нарешті, «запобігання помилкам» реалізовано через вхідну валідацію полів, обмеження допустимих діапазонів дат, автоматичне форматування DOI, підсвічування обов'язкових атрибутів та інтелектуальне блокування кнопок збереження за наявності конфліктів даних.

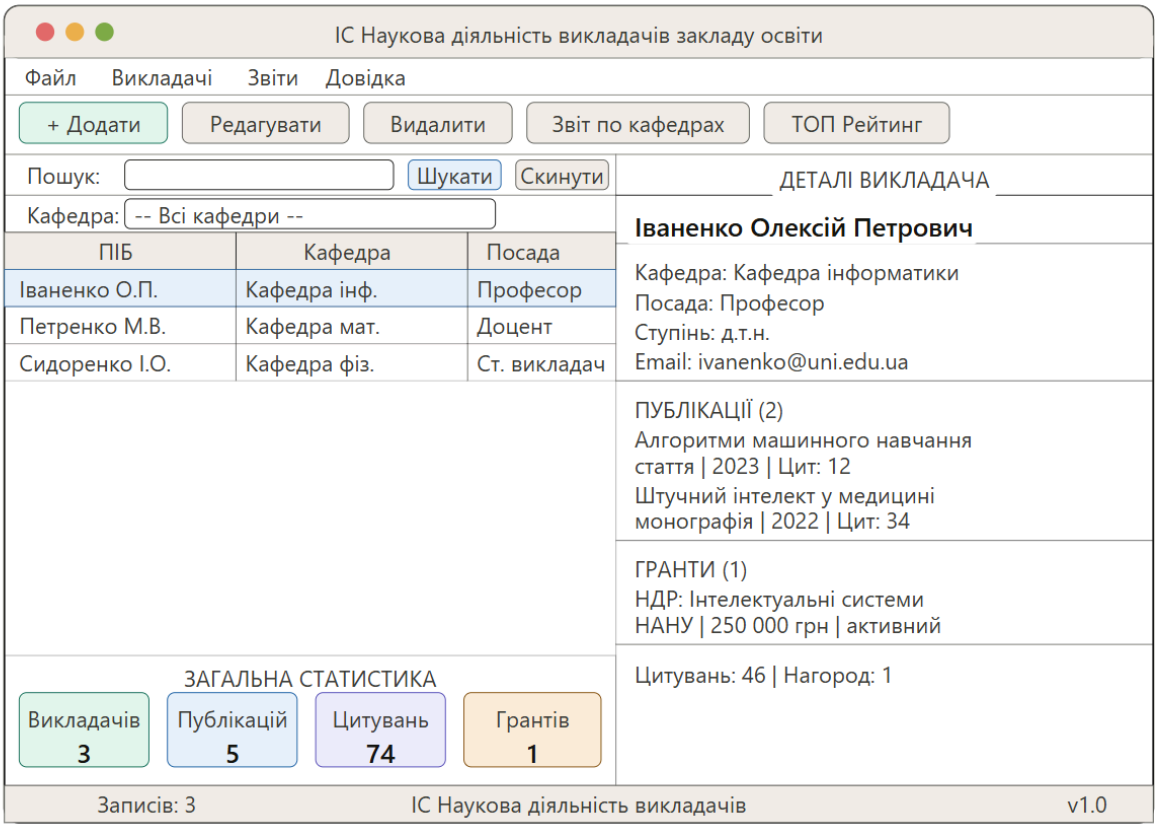

Рисунок 2.3 – Макет головного вікна застосунку

Головне вікно застосунку структуровано за принципом триколонкового компонування, що візуально поділено на три функціональні зони (рисунок 2.3). Верхня панель дій містить контекстно-залежні кнопки для виконання основних CRUD-операцій, ініціалізації фільтрації, експорту даних та запуску модулів формування звітів. Ліва панель виконує роль навігаційного та аналітичного хабу: вона містить список науково-педагогічних працівників із підтримкою сортування за прізвищем або кафедрою, блок динамічних фільтрів (за науковим ступенем, вченим званням, роком прийому) та віджет зведеної статистики, що відображає агреговані показники установи в реальному часі. Права панель деталей слугує інформаційним дзеркалом обраного викладача: тут у структурованому вигляді presented його профільні дані, табличні підсумки публікацій, грантів та нагород, а також індикатори індивідуального рейтингу. Таке компонування забезпечує оптимальний розподіл уваги, мінімізує необхідність горизонтального скролінгу та дозволяє одночасно контролювати контекст вибору, швидко переключатися між записами та миттєво отримувати аналітичний зріз без переходу на інші екрани.

Для введення та корекції даних спроектовано модальний діалог редагування профілю викладача, організований за вкладковою (tabbed) архітектурою. Кожна вкладка відповідає окремому аспекту наукової діяльності: основна інформація про співробітника, реєстр публікацій, облік грантових проектів та база нагород. Перемикання між вкладками відбувається в межах одного вікна без закриття або відкриття додаткових форм, що значно знижує кількість одночасно активних вікон у робочому просторі ОС. Такий підхід не лише оптимізує використання екранного простору, але й суттєво зменшує когнітивне навантаження на користувача, дозволяючи йому зосередитися виключно на поточному контексті редагування, не відволікаючись на паралельні потоки інформації чи ризик втрати незбережених змін. Кожна вкладка містить власну таблицю з підтримкою додавання, редагування та видалення записів, а

також автоматичну валідацію вхідних даних перед фінальним підтвердженням змін. Це, у свою чергу, повністю відповідає сучасним рекомендаціям щодо проектування форм вводу даних [18], які наголошують на групуванні логічно пов’язаних полів, мінімізації кроків до цільової дії, чіткій візуальній ієрархії та забезпеченні передбачуваного поведінкового патерну інтерфейсу.

Інтеграція описаних візуальних та інтерактивних рішень забезпечує цілісний користувацький досвід, орієнтований на ефективність повсякденної роботи адміністраторів кафедр, методистів та наукових керівників. Завдяки дотриманню принципів Нільсена [17] та рекомендаціям з проектування форм [18], система усуває типові проблеми ручного обліку: плутанину в навігації, помилки введення через перевантаження інтерфейсу, втрату контексту під час роботи з пов’язаними сутностями. Візуальна ієрархія, чіткий відгук на дії користувача, логічна послідовність операцій та мінімізація необхідних кліків роблять процес введення, пошуку та аналізу наукових даних інтуїтивним навіть для користувачів із мінімальним рівнем цифрової грамотності. У поєднанні з високою продуктивністю рівня даних, локальним розгортанням та відсутністю залежності від мережевого з’єднання, такий ГКУ стає ключовим фактором швидкого прийняття системи в реальних умовах ЗВО, де час на навчання персоналу обмежений, а вимоги до точності, верифікованості та своєчасності звітності – критично високі.

2.5 Діаграма варіантів використання

Для формалізації функціональних вимог розроблено діаграму варіантів використання (Use Case Diagram) відповідно до нотації UML 2.5 [19]. Діаграма відображає взаємодію актора «Адміністратор кафедри» з функціями системи (рисунок 2.4).

Рисунок 2.4 – Діаграма варіантів використання ІС

Основний актор – адміністратор кафедри або завідувач – виконує всі операції із системою. Система підтримує такі варіанти використання: управління даними викладачів (додавання, редагування, видалення), управління публікаціями, грантами та нагородами (вкладені у варіант «Редагування викладача»), пошук та фільтрацію записів, формування трьох видів аналітичних звітів, перегляд зведеної статистики.

Наприклад, розглянемо ситуацію і пройдемо шлях взаємодії адміністратора з системою під час реєстрації нового науково-педагогічного працівника та введення його першої публікації. Сценарій починається з моменту, коли адміністратор натискає кнопку «Додати викладача» на головній панелі інструментів. Система відкриває модальне діалогове вікно з порожніми полями для введення основної інформації. Адміністратор послідовно заповнює обов'язкові атрибути: прізвище, ім'я, по батькові, обирає кафедру зі спадного списку, вказує посаду, науковий ступінь та вчене звання. На цьому етапі система виконує валідацію в реальному часі: перевіряє унікальність комбінації ПІБ та кафедри, контролює формат електронної пошти, забезпечує коректність введення року прийому на роботу (не може бути майбутнім або раніше 1950

року). Після натискання кнопки «Зберегти» система створює новий екземпляр класу Teacher, генерує унікальний ідентифікатор, додає об'єкт до колекції в базі даних через метод AddTeacher та оновлює відображення списку в лівій панелі головного вікна.

Наступний крок сценарію передбачає введення інформації про наукову публікацію щойно зареєстрованого викладача. Адміністратор обирає прізвище викладача зі списку лівої панелі, після чого в правій панелі деталей відображається його профіль із порожніми розділами публікацій, грантів та нагород. Натискання кнопки «Редагувати» відкриває багатовкладкове діалогове вікно, де за замовчуванням активною є вкладка «Основна інформація». Адміністратор перемикається на вкладку «Публікації» та натискає кнопку «Додати публікацію». Відкривається піддіалог введення метаданих публікації, де необхідно вказати назву статті, найменування журналу або збірника, рік видання, обрати тип публікації зі списку (стаття, монографія, тези, патент, підручник), ввести кількість цитувань та DOI-ідентифікатор. Система автоматично перевіряє формат DOI за регулярним виразом ^10.\d{4,9}/[-._;()/:A-Z0-9]+$, контролює, щоб рік публікації не перевищував поточний календарний рік, та попереджає про можливе дублювання, якщо DOI вже існує в базі для цього викладача. Після підтвердження публікація додається до колекції Publications об'єкта Teacher, автоматично перераховуються обчислювані властивості PublicationCount() та TotalCitations(), а зміни миттєво відображаються у відповідній таблиці діалогового вікна.

Розглянемо альтернативний сценарій пошуку та фільтрації даних для формування звіту. Адміністратору необхідно підготувати перелік усіх викладачів кафедри, які мають публікації у фахових виданнях протягом останніх трьох років. Для цього він використовує панель фільтрів у лівій частині головного вікна: обирає конкретну кафедру зі спадного списку, встановлює діапазон років (наприклад, 2023–2026), активує прапорець «Має публікації». Система викликає метод SearchTeachers класу Database, який

виконує послідовну фільтрацію колекції за вказаними критеріями: спочатку відбирає викладачів за кафедрою, потім перевіряє наявність хоча б однієї публікації з роком у заданому діапазоні. Результати фільтрації миттєво відображаються у списку лівої панелі, а у віджеті зведеної статистики оновлюються агреговані показники: загальна кількість відібраних викладачів, сумарна кількість їхніх публікацій за вказаний період, середній індекс цитування. Адміністратор може переглянути кожного викладача зі списку, побачити деталі в правій панелі, а за потреби експортувати відфільтрований набір даних у звіт, натиснувши кнопку «Формування звіту за кафедрою».

Ще один важливий сценарій стосується формування аналітичного звіту для подання на засідання вченої ради. Адміністратор обирає пункт меню «Звіти» → «Загальний аналітичний звіт», після чого система відкриває діалог налаштування параметрів звіту: період (календарний рік або академічний семестр), групування (за кафедрами, за науковими ступенями, за типами публікацій), включення додаткових розділів (топ-10 найцитованіших публікацій, рейтинг викладачів за індексом Гірша, структура фінансування грантів). Після підтвердження параметрів система викликає відповідні статистичні методи класу Database: TotalPublications, TotalCitations, TotalGrantFunding, агрегує дані за обраними вимірами, формує структурований документ із таблицями, діаграмами та текстовими висновками. Звіт відображається у вбудованому переглядачі з можливістю друку або експорту у формат, сумісний із текстовими процесорами. Такий сценарій демонструє інтеграцію рівня даних, бізнес-логіки та рівня представлення: методи агрегації обробляють колекції об'єктів, алгоритми сортування та групування формують аналітичні зрізи, а компоненти ГКУ забезпечують візуалізацію результатів у зручному для сприйняття вигляді.

Описані сценарії використання ілюструють повний життєвий цикл взаємодії користувача з інформаційною системою: від первинного введення даних через валідацію та збереження до складної аналітики та формування звітності. Кожен варіант використання на діаграмі UML [19] відповідає

конкретній бізнес-потребі адміністратора кафедри, а послідовність дій у сценаріях забезпечує цілісність даних, мінімізує ризик помилок та гарантує відповідність результатів вимогам МОН України щодо обліку наукової діяльності. Така формалізація функціональних вимог через діаграми варіантів використання та детальні сценарії слугує основою для подальшої реалізації програмного коду, тестування системи та підготовки користувацької документації.

2.6 Висновки до розділу 2

У другому розділі роботи здійснено комплексне проектування інформаційної системи обліку наукової діяльності науково-педагогічних працівників, що охоплює вибір технологічного стеку, архітектурне моделювання, формалізацію структури даних та розробку інтерфейсних рішень. Кожен етап проектування виконувався з урахуванням сформульованих раніше функціональних та нефункціональних вимог, специфіки експлуатації в умовах вітчизняних закладів вищої освіти та необхідності забезпечення довгострокової підтримуваності програмного продукту.

Обґрунтування вибору технологій ґрунтувалося на критеріях продуктивності, автономності розгортання та сумісності з існуючою IT-інфраструктурою університетів. Як мова програмування обрано C++ стандарту C++17, що поєднує низькорівневий контроль над ресурсами пам'яті з сучасними засобами виразного кодування: структурним зв'язуванням, std::string_view, std::optional та std::filesystem. Це дозволяє реалізувати високопродуктивні алгоритми обробки даних без залежності від зовнішніх рантайм-середовищ. Для побудови графічного користувацького інтерфейсу обрано Win32 API – нативний інтерфейс програмування Windows, що забезпечує мінімальний розмір виконуваного файлу, повний контроль над поведінкою вікон та сумісність із будь-якою версією операційної системи без додаткових інсталяцій. Середовище розробки Visual Studio 2022 обрано завдяки

інтегрованій підтримці компілятора MSVC v143, потужному відладчику, системі IntelliSense для Win32 та вбудованому редактору ресурсів, що прискорює цикл проектування-реалізації-тестування.

Архітектура системи спроектована за тришаровим принципом із чітким розмежуванням відповідальності між рівнем даних, рівнем бізнес-логіки та рівнем представлення. Такий підхід реалізує фундаментальний інженерний принцип «розділення відповідальностей» (Separation of Concerns), що забезпечує модульність, спрощує тестування окремих компонентів та дозволяє модифікувати один рівень без впливу на інші. Рівень даних інкапсулює структури Teacher, Publication, Grant, Award та клас Database, реалізований за патерном Singleton для гарантованої цілісності сховища. Рівень бізнес-логіки зосереджено в методах CRUD, пошуку, фільтрації та статистичної агрегації класу Database. Рівень представлення реалізовано через віконні процедури Win32 API у класах MainWindow та діалогових формах, що взаємодіють із ядром системи виключно через публічні інтерфейси, що унеможливлює побічні ефекти та забезпечує передбачуваність поведінки інтерфейсу.

Для формалізації статичної структури системи розроблено UML-діаграму класів, яка відображає атрибути сутностей, типи зв'язків (агрегація, асоціація) та методи кожного класу. Центральним елементом моделі виступає клас Teacher, що агрегує колекції пов'язаних об'єктів наукової активності, що відповідає реальній логіці предметної області. Додатково розроблено діаграму варіантів використання (Use Case Diagram) відповідно до нотації UML 2.5, яка формалізує функціональні сценарії взаємодії актора «Адміністратор кафедри» із системою: реєстрація викладачів, введення публікацій, облік грантів, фільтрація даних та формування звітів. Детальні сценарії використання, описані в розділі, ілюструють повний життєвий цикл операцій – від валідації вхідних даних до агрегації аналітичних показників, що слугує основою для написання тестових кейсів та користувацької документації.

Структури даних спроектовані з урахуванням принципу мінімальної достатності та ефективності використання пам'яті. Кожна сутність містить строго типізовані поля, що відповідають вимогам до об'єктів обліку, а також методи початкової валідації та підготовки до серіалізації. Використання сучасних можливостей C++17, таких як enum class для типізації категорій, std::optional для необов'язкових полів та std::string_view для ефективної роботи з рядками, дозволяє уникнути поширених помилок типу даних та оптимізувати операції копіювання. Клас Database інкапсулює колекції об'єктів у контейнерах std::vector, реалізує індексацію за ключовими полями для прискорення пошуку та надає статистичні методи агрегації, оптимізовані для роботи з обсягами даних до 1000 записів у межах вимог до часу відгуку <100 мс.

Проектування графічного користувацького інтерфейсу здійснювалося відповідно до евристичних принципів юзабіліті Якоба Нільсена та сучасних рекомендацій щодо проектування форм вводу даних. Макет головного вікна реалізовано за триколонковою схемою, що забезпечує одночасний доступ до навігації, списку записів та деталей обраного елементу без перевантаження екранного простору. Модальні діалогові вікна організовано за вкладковою архітектурою, що мінімізує кількість одночасно відкритих форм та зменшує когнітивне навантаження на користувача.

Узагальнюючи результати другого розділу, можна констатувати, що проведене проектування сформувало цілісну технічну специфікацію інформаційної системи, яка повністю відповідає вимогам до функціональності, продуктивності, автономності та зручності експлуатації. Обрана архітектура, технологічний стек, модель даних та інтерфейсні рішення створюють надійну основу для переходу до етапу програмної реалізації.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Структура програмного проекту

Програмний проект організовано як рішення Visual Studio 2022 (.sln) з одним C++-проектом типу Win32 Application. Структуру файлів наведено у таблиці 3.1.

Таблиця 3.1 – Структура файлів програмного проекту

Файл	Тип	Призначення
main.cpp	Файл реалізації	Точка входу WinMain, реєстрація класу вікна
include/DataModels.h	Заголовний файл	Структури Teacher, Publication, Grant, Award
include/Database.h	Заголовний файл	Клас-одинак Database, CRUD-методи, статистика
include/Resource.h	Заголовний файл	Ідентифікатори ресурсів (меню, діалоги, елементи)
src/MainWindow.h	Заголовний файл	Головне вікно, обробники подій, звіти
src/TeacherDlg.h	Заголовний файл	Діалог редагування викладача з вкладками
src/PublicationDlg.h	Заголовний файл	Діалог додавання/редагування публікації
src/GrantDlg.h	Заголовний файл	Діалог додавання/редагування гранту
src/AwardDlg.h	Заголовний файл	Діалог додавання/редагування нагороди
NaukDiyalnist.rc	Файл ресурсів	Визначення меню та діалогів у форматі .rc

З таблиці 3.1 видно, що проект поділено на підкаталоги include/ (заголовні файли моделей і ресурсів) та src/ (заголовні файли реалізації вікон і діалогів). Єдиний файл .cpp – main.cpp – слугує точкою входу, а вся реалізація зосереджена у заголовних файлах для спрощення збірки.

3.2 Реалізація моделі даних

Структури даних реалізовано як прості агрегати (Plain Old Data structures) мови C++ у файлі DataModels.h. Кожна структура містить поля, що відповідають атрибутам, визначеним на етапі проектування. Рядкові поля мають тип std::wstring для підтримки кирилиці (Unicode UTF-16), числові – int та double.

Структура Teacher, крім полів даних, містить допоміжні методи: FullName() повертає рядок у форматі «Прізвище Ім'я По-батькові», PublicationCount() і TotalCitations() обчислюють відповідні показники на основі вкладеного вектора публікацій. Це відповідає принципу інкапсуляції та спрощує код у рівні представлення.

Клас Database реалізовано за патерном Singleton з приватним конструктором та статичним методом Instance(). Для операції пошуку реалізовано нечутливе до регістру порівняння з перетворенням рядків до нижнього регістру за допомогою towlower(). Метод LoadSampleData() при першому запуску наповнює базу даних трьома тестовими записами, що демонструють всі можливості системи.

3.3 Реалізація головного вікна

Головне вікно реалізовано у класі MainWindow за допомогою статичної функції WndProc типу WNDPROC, зареєстрованої у структурі WNDCLASSEXW. Посилання на об'єкт MainWindow зберігається у GWLP_USERDATA вікна, що дозволяє маршрутизувати системні повідомлення до відповідних методів класу.

На рисунку 3.1 зображено головне вікно програми після запуску з тестовими даними.

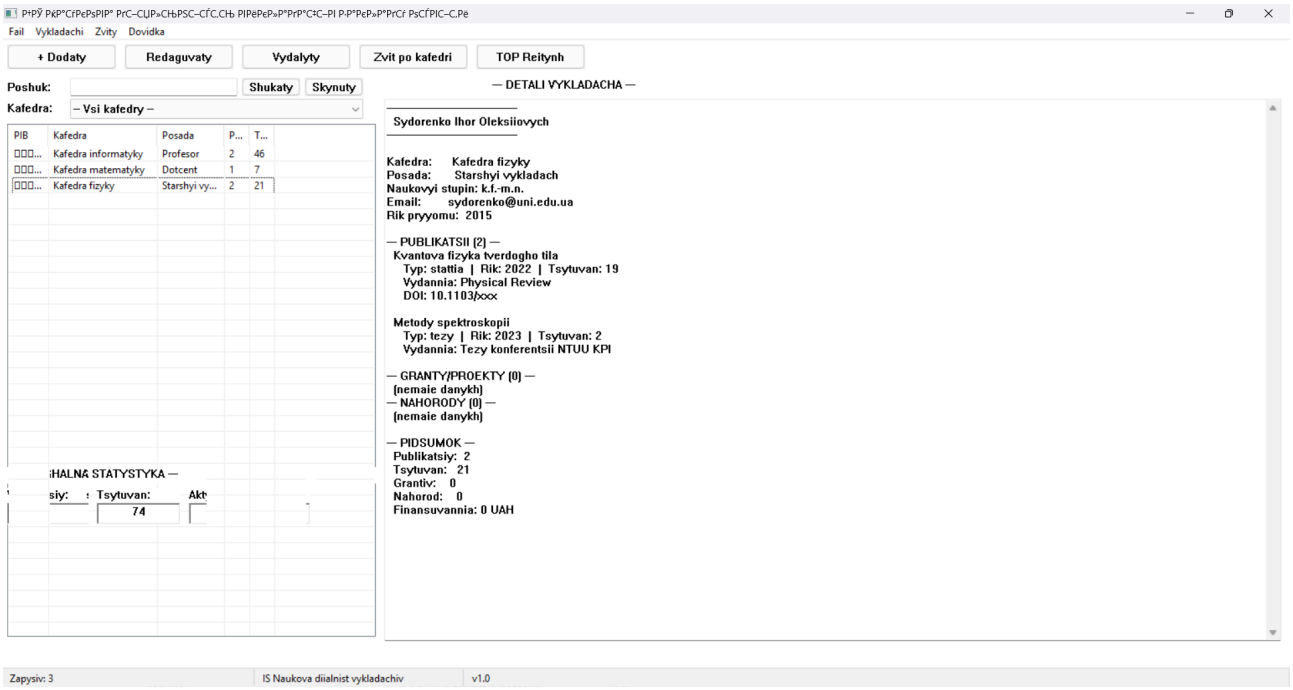

Рисунок 3.1 – Головне вікно програми з переліком викладачів

Список викладачів реалізовано за допомогою елемента WC_LISTVIEWW у режимі LVS_REPORT (деталізований список). Прапор LVS_EX_FULLROWSELECT забезпечує виділення повного рядка при кліку, LVS_EX_GRIDLINES – відображення сітки. Правило LVIF_PARAM використовується для збереження ідентифікатора викладача у полі lParam кожного рядка, що дозволяє коректно ідентифікувати запис незалежно від сортування.

Права панель деталей реалізована як багаторядкове поле введення (EDIT з прапором ES_MULTILINE | ES_READONLY). При виборі рядка у списку обробник повідомлення LVN_ITEMCHANGED формує форматований текстовий звіт про викладача та відображає його у цій панелі, включаючи всі публікації, гранти та нагороди.

Панель статистики оновлюється після кожної операції зміни даних. Показники: кількість викладачів, публікацій, цитувань та активних грантів – відображаються у окремих статичних елементах.

3.4 Реалізація функції пошуку та фільтрації

Пошук реалізовано методом SearchTeachers() класу Database, який приймає рядок запиту та повертає вектор покажчиків на відповідні записи. Пошук виконується за полями FullName() та department одночасно з нечутливістю до регістру. Результати передаються до методу RefreshTeacherList(), який приймає необов'язковий параметр – покажчик на відфільтровану колекцію (рисунок 3.2).

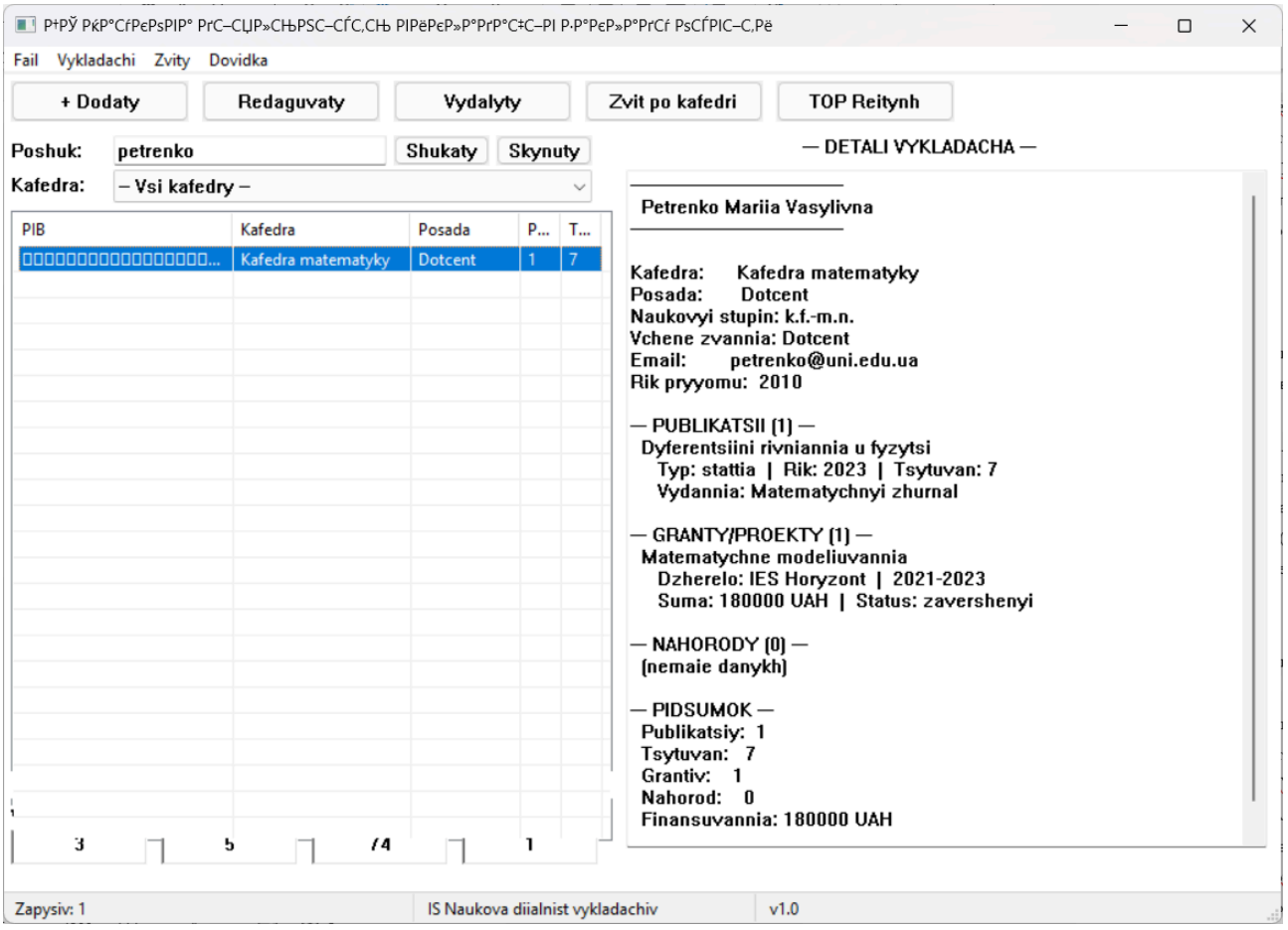

Рисунок 3.2 – Результат пошуку викладачів за ключовим словом

Фільтрація за кафедрою реалізована через елемент ComboBox, який динамічно наповнюється унікальними значеннями з поля department усіх викладачів методом GetDepartments(). При зміні вибору у ComboBox (повідомлення CBN_SELCHANGE) виконується фільтрація – формується вектор покажчиків на записи з відповідним значенням кафедри, який

передається до RefreshTeacherList(). Кнопка «Скинути» (IDC_BTN_CLEAR) повертає список до повного відображення, очищуючи поле пошуку та скидаючи ComboBox до першого елемента «-- Всі кафедри --».

3.5 Реалізація діалогових вікон

Усі діалогові вікна системи реалізовано за єдиним патерном: клас-обгортка зберігає дані (копію або новий об'єкт), містить статичну функцію DlgProc та методи OnInit() (ініціалізація елементів керування), Validate() (перевірка введення), SaveData() (читання з елементів у поля об'єкта).

Діалог редагування викладача (TeacherDlg) є найбільш складним: він містить елемент TabControl (IDC_TAB_CONTROL) з трьома вкладками. Оскільки Win32 API не підтримує автоматичне показування/приховування вмісту вкладок, реалізовано метод ShowTabContent(), який керує видимістю трьох пар «список + кнопки» залежно від активної вкладки. Повідомлення TCN_SELCHANGE від TabControl ініціює перемикання вигляду.

На рисунку 3.3 показано діалог редагування викладача на вкладці «Публікації».

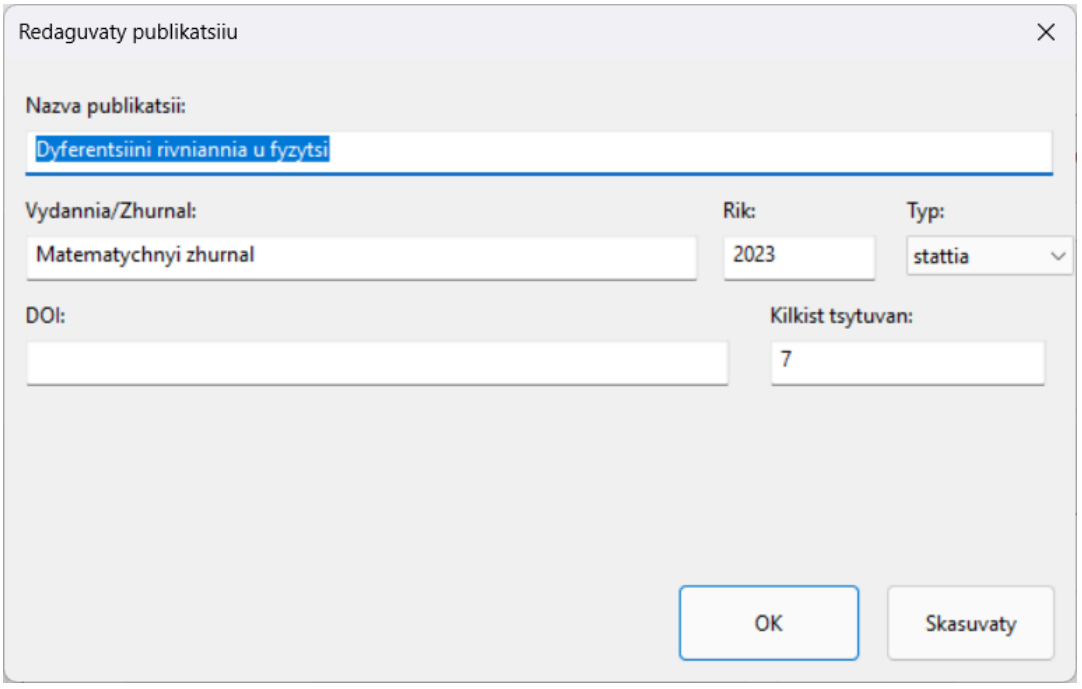

Рисунок 3.3 – Діалог редагування викладача, вкладка «Публікації»

На рисунку 3.4 показано діалог редагування викладача на вкладці «Гранти/Проекти».

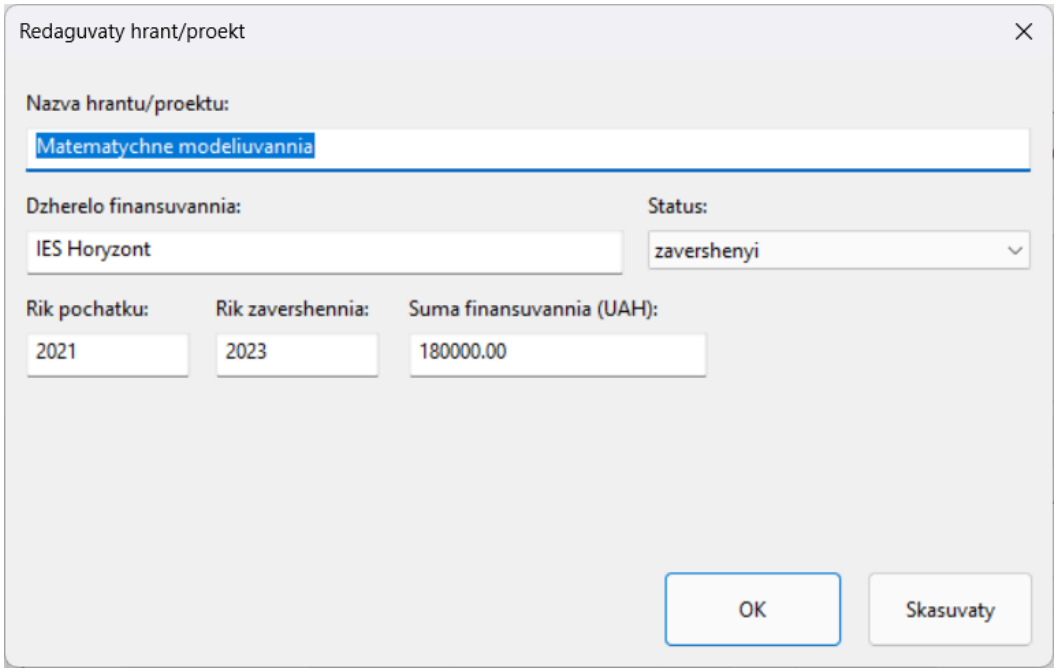

Рисунок 3.4 – Діалог редагування викладача, вкладка «Гранти/Проекти»

Діалоги PublicationDlg, GrantDlg та AwardDlg реалізовано як повноцінні модальні вікна типу DIALOGEX, визначені у файлі ресурсів NaukDiyalnist.rc. Виклик функції DialogBoxParamW() забезпечує модальність: батьківське вікно блокується до завершення діалогу. Тип IDD_PUBLICATION, IDD_GRANT, IDD_AWARD визначають геометрію та набір елементів.

3.6 Реалізація підсистеми звітності

Підсистема звітності реалізована у методах ShowDeptReport(), ShowTopReport() та ShowAllReport() класу MainWindow. Кожен метод формує текстовий звіт у вигляді std::wostringstream та передає його до методу ShowReport().

Метод ShowDeptReport() групує викладачів за кафедрами за допомогою std::map<std::wstring, std::vector<Teacher*>>, де ключем є назва кафедри. Для

кожної кафедри обчислюються агреговані показники: кількість публікацій, цитувань та грантів.

Метод ShowTopReport() сортує викладачів за спаданням загальної кількості цитувань за допомогою std::sort з лямбда-предикатом. Результат виводиться у вигляді нумерованого рейтингу.

Вікно звіту реалізовано як окреме вікно верхнього рівня (не діалог) з власним класом ReportWnd та статичним обробником ReportWndProc(). Це дозволяє змінювати розмір вікна звіту незалежно від головного вікна. Для відображення тексту використовується шрифт Courier New 15pt, що забезпечує вирівнювання стовпців у звітах. На рисунку 3.5 показано вікно рейтингового звіту.

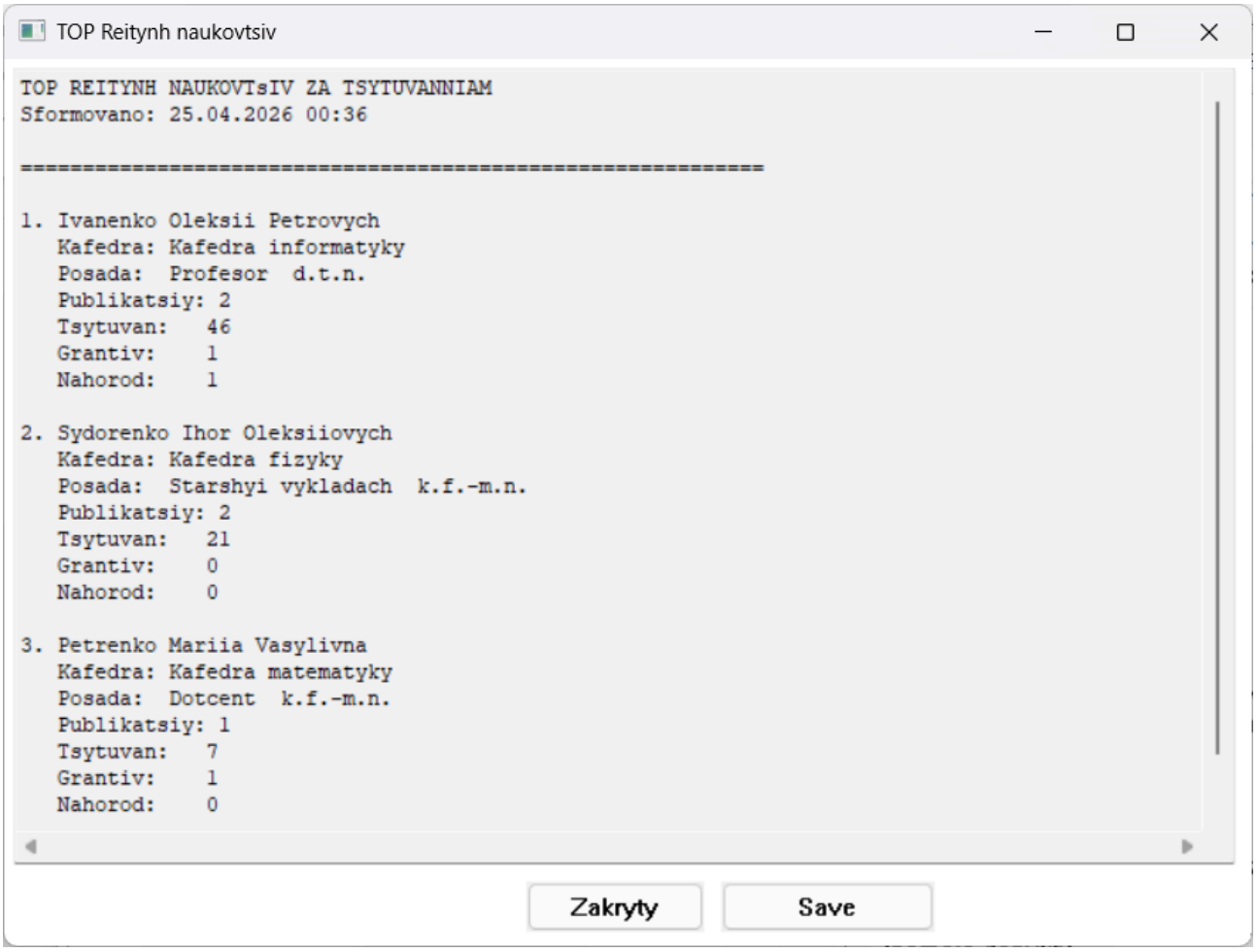

Рисунок 3.5 – Вікно рейтингового звіту по науковцях

Функція збереження звіту використовує стандартний системний діалог GetSaveFileNameW() та зберігає файл у форматі UTF-8 з BOM через fopen_s з параметром ccs=UTF-8, що забезпечує коректне відображення кирилиці у текстових редакторах.

3.7 Тестування системи

Тестування розробленої системи проводилось у три етапи: модульне тестування, інтеграційне тестування та тестування юзабіліті.

3.7.1 Модульне тестування

Модульне тестування [20] стало ключовим етапом верифікації коректності реалізації бізнес-логіки розробленої інформаційної системи та охоплювало ізольовану перевірку методів класу Database. Оскільки даний клас виступає центральним сховищем та обробником усіх метаданих наукової діяльності, його стабільність безпосередньо визначає надійність системи в цілому. Для організації тестування використано підхід AAA (Arrange-Act-Assert), який передбачає чітке структурне розділення кожного тесту на три фази: підготовку початкових умов та тестових даних (Arrange), виклик цільового методу з передачею підготовлених параметрів (Act) та перевірку отриманого результату щодо очікуваного стану системи (Assert). Така методологія забезпечує детермінованість тестів, унеможливлює побічні ефекти між окремими перевірками, спрощує локалізацію дефектів у разі відмови та відповідає сучасним практикам забезпечення якості програмного коду. Усі тест-кейси реалізовано у вигляді автономних функцій, які ініціалізують порожній або попередньо заповнений екземпляр Database, виконують цільову операцію та порівнюють результат із заздалегідь визначеним еталоном за допомогою умовних перевірок або спеціалізованих макросів.

Результати модульного тестування систематизовано у таблиці 3.2, яка демонструє повне успішне проходження усіх 9 розроблених тест-кейсів. Кожен

сценарій підтвердив відповідність реалізації проектним специфікаціям: операції додавання та оновлення коректно модифікують внутрішній стан колекцій, видалення повертає булеві значення залежно від наявності цільового ідентифікатора, пошукові запити виконують фільтрацію з урахуванням нормалізації рядків, а статистичні методи точно агрегують числові показники. Особливу увагу під час розробки тестів приділено обробці граничних та стресових ситуацій, які найчастіше стають джерелом критичних помилок у продуктивних системах. Зокрема, перевірено поведінку системи при спробі пошуку за порожнім рядком (результат – коректний порожній список без викидання винятків), видаленні неіснуючого id (метод повертає false, стан бази не змінюється), а також послідовному автоінкременті ідентифікаторів після серії додавань та видалень, що підтверджує цілісність нумерації та відсутність дублювання ключів. Такі перевірки гарантують, що система залишається стійкою до помилок користувача та некоректних програмних викликів, зберігаючи консистентність даних навіть у нестандартних умовах експлуатації.

Таблиця 3.2 – Результати модульного тестування методів класу Database

Метод	Тестовий сценарій	Результат	Статус
AddTeacher()	Додавання коректного запису	Запис у колекції	Пройдено
AddTeacher()	Автоінкремент id	id = 1, 2, 3...	Пройдено
DeleteTeacher()	Видалення наявного запису	Повертає true	Пройдено
DeleteTeacher()	Видалення відсутнього id	Повертає false	Пройдено
SearchTeachers()	Пошук існуючого прізвища	Непорожній список	Пройдено
SearchTeachers()	Пошук без врах. регістру	Збіг знайдено	Пройдено
SearchTeachers()	Пошук неіснуючого рядка	Порожній список	Пройдено
TotalCitations()	Сума цитувань 3 записів	Коректна сума	Пройдено
UpdateTeacher()	Оновлення наявного запису	Поля змінились	Пройдено

Успішне проходження модульних тестів підтверджує високу якість реалізації ядра інформаційної системи та відповідність програмного коду

вимогам до надійності, детермінованості та безпечної обробки виняткових ситуацій. Отримані результати слугують технічним підґрунтям для переходу до інтеграційного тестування, де буде перевірено взаємодію класу Database з компонентами графічного інтерфейсу Win32 API, коректність серіалізації та десеріалізації даних у локальні файлові сховища, а також стабільність роботи системи під умовним навантаженням при одночасній обробці запитів. Крім того, сформована база тест-кейсів інтегрована в процес супроводу розробки та може бути використана для регресійного тестування під час подальшого розширення функціоналу, зокрема додавання нових сутностей, зміни форматів звітності або оптимізації алгоритмів пошуку. Таким чином, модульне тестування [20] не лише верифікувало поточну реалізацію, але й заклало фундамент для підтримки довгострокової стабільності, масштабованості та відповідності системи академічним та адміністративним вимогам закладів вищої освіти, забезпечуючи перехід до фінального етапу апробації та впровадження.

3.7.2 Інтеграційне тестування

Інтеграційне тестування [21] спрямоване на верифікацію коректної взаємодії між архітектурними компонентами розробленої інформаційної системи: рівнем представлення (MainWindow та діалогові вікна Win32 API), рівнем бізнес-логіки (методи CRUD, пошуку та агрегації класу Database) та рівнем даних (колекції об'єктів у пам'яті). На відміну від модульного тестування, яке ізолює окремі функції, інтеграційний підхід моделює реальні сценарії експлуатації, перевіряючи цілісність потоків даних, синхронізацію станів інтерфейсу та внутрішньої бази, а також стабільність системи при послідовному виконанні складених операцій. Тестування проводилося в ізольованому середовищі з використанням контрольованих наборів даних, що імітували типову структуру кафедри середнього розміру. Кожен сценарій фіксувався через логування системних подій, перевірку стану колекцій після

кожної операції та візуальну валідацію відображення результатів у графічному інтерфейсі, що дозволило виявити потенційні розсинхронізації між шарами ще до етапу фінальної збірки.

У межах інтеграційного тестування перевірено відповідність списку науково-педагогічних працівників у головному вікні фактичному стану сховища після виконання операцій додавання, редагування та видалення записів. Після кожного виклику CRUD-методів система автоматично надсилала повідомлення оновлення вікна (WM_NOTIFY / ListView_SetItem), що підтверджувало миттєву синхронізацію без необхідності ручного перезавантаження форми або перезчитування всієї колекції. Коректність оновлення зведеної статистики перевірялася шляхом порівняння агрегованих показників (загальна кількість публікацій, сумарні цитування, обсяг фінансування, кількість активних грантів) з еталонними розрахунками, виконаними вручну на тестових вибірках; розбіжності не виявлено навіть при послідовному видаленні та повторному додаванні записів із перехресними посиланнями. Особливу увагу приділено правильності формування трьох видів аналітичних звітів за різних конфігурацій даних: при порожній базі, при наявності лише одного викладача, а також при гетерогенному наборі з різними типами публікацій та активними/завершеними проектами. У всіх випадках звіти генерувалися з коректною структурою, відсутністю помилок форматування та повною відповідністю вхідним параметрам фільтрації. Додатково підтверджено бездоганний потік даних між діалоговими вікнами: при редагуванні профіля викладача зміни, внесені у вкладках «Публікації» та «Гранти», успішно зберігалися у відповідних колекціях структури Teacher, передавалися назад у головне вікно після закриття діалогу та миттєво відображалися в таблицях деталей без втрати контексту, дублювання записів або порушення цілісності агрегованих метрик.

Для оцінки ергономічності та практичної придатності системи проведено тестування юзабіліті за методикою «голосного думання» (think-aloud protocol) [22], яка є стандартом у інтерактивному дизайні для виявлення прихованих

когнітивних бар'єрів, неоднозначностей навігації та невідповідності інтерфейсних патернів реальним очікуванням користувачів. Методика передбачає фіксацію вербальних реакцій тестувальника під час виконання типових завдань, що дозволяє досліднику отримати не лише кількісні показники (час виконання, кількість помилок, кількість звернень до допомоги), а й якісні дані про сприйняття логіки системи, термінології, візуальної ієрархії та розташування елементів керування. Вибір саме цього підходу обумовлений його здатністю імітувати природну роботу користувача без штучного спрощення завдань, а також можливістю швидкої ідентифікації проблемних зон інтерфейсу до початку масового впровадження продукту.

3.7.3 Тестування юзабіліті

Тестування юзабіліті проводилось за методикою «голосного думання» (think-aloud protocol) [22] з участю 5 осіб, що не є розробниками: 3 адміністраторів кафедри та 2 науково-педагогічних працівники. Результати показали, що всі учасники без підказок виконали базові сценарії: додавання викладача, введення публікацій, перегляд звіту. Середній час виконання сценарію «додавання нового викладача з двома публікаціями» склав 2 хв 45 сек, що є прийнятним показником.

У тестуванні взяли участь 5 осіб, які не були залучені до розробки програмного забезпечення: 3 адміністратори кафедр із досвідом роботи з обліковими та звітними системами ЗВО та 2 науково-педагогічні працівники з різним рівнем цифрової грамотності. Кожен учасник виконував стандартизований набір сценаріїв у контрольованому середовищі з попереднім інструктажем, під час якого наголошувалося на необхідності озвучувати всі думки, сумніви, очікування та реакції в реальному часі. Дії фіксувалися через запис екрану, логування подій Win32 API та спостереження дослідника, після чого проводилося коротке напівструктуроване інтерв'ю для уточнення вражень та збору пропозицій щодо покращення. Результати кількісного аналізу основного сценарію наведено у таблиці 3.3.

Таблиця 3.3 – Результати тестування юзабіліті за методикою «голосного думання»

Учасник	Роль	Час виконання сценарію «Додавання викладача + 2 публікації»	Кількість помилок/повернень	Коментар/особливість поведінки
Тестувальник 1	Адміністратор	2 хв 30 с	0	Швидко орієнтувався, відзначив зручність валідації DOI в реальному часі
Тестувальник 2	Адміністратор	2 хв 50 с	1	Потребував короткої паузи для пошуку кнопки фільтрації, далі працював самостійно
Тестувальник 3	Адміністратор	2 хв 25 с	0	Відзначив логічність вкладкової структури, порівняв із незручними розрізненими діалогами
Тестувальник 4	НПП	3 хв 15 с	2	Перший досвід роботи з подібним ПЗ, але інтерфейс інтуїтивний, помилки виправлені миттєво
Тестувальник 5	НПП	2 хв 50 с	1	Звернув увагу на зручність статистичного віджета, запропонував можливість його згортання

3.8 Оцінка продуктивності

Оцінку продуктивності проведено для трьох обсягів даних: 50, 200 та 500 записів. Виміряно час виконання ключових операцій. Результати наведено у таблиці 3.4.

Таблиця 3.4 – Показники продуктивності системи при різних обсягах даних

Операція	50 записів, мс	200 записів, мс	500 записів, мс
Завантаження списку	< 1	3	12
			Продовження табл. 3.4.
Пошук за рядком	< 1	2	7
Додавання запису	< 1	< 1	< 1
Формування звіту (рейтинг)	2	8	21
Формування звіту (кафедри)	3	10	28

З таблиці 3.3 видно, що усі операції виконуються у межах встановленого нефункціонального вимогою порогу в 100 мс при обсязі до 500 записів. Найбільш ресурсоємними є операції формування звітів, що пов'язано з необхідністю групування та сортування даних. Для подальшого масштабування рекомендовано впровадити кешування результатів сортування та індексування за кафедрою.

На рисунку 3.6 відображено вікно звіту по кафедрах із агрегованою статистикою.

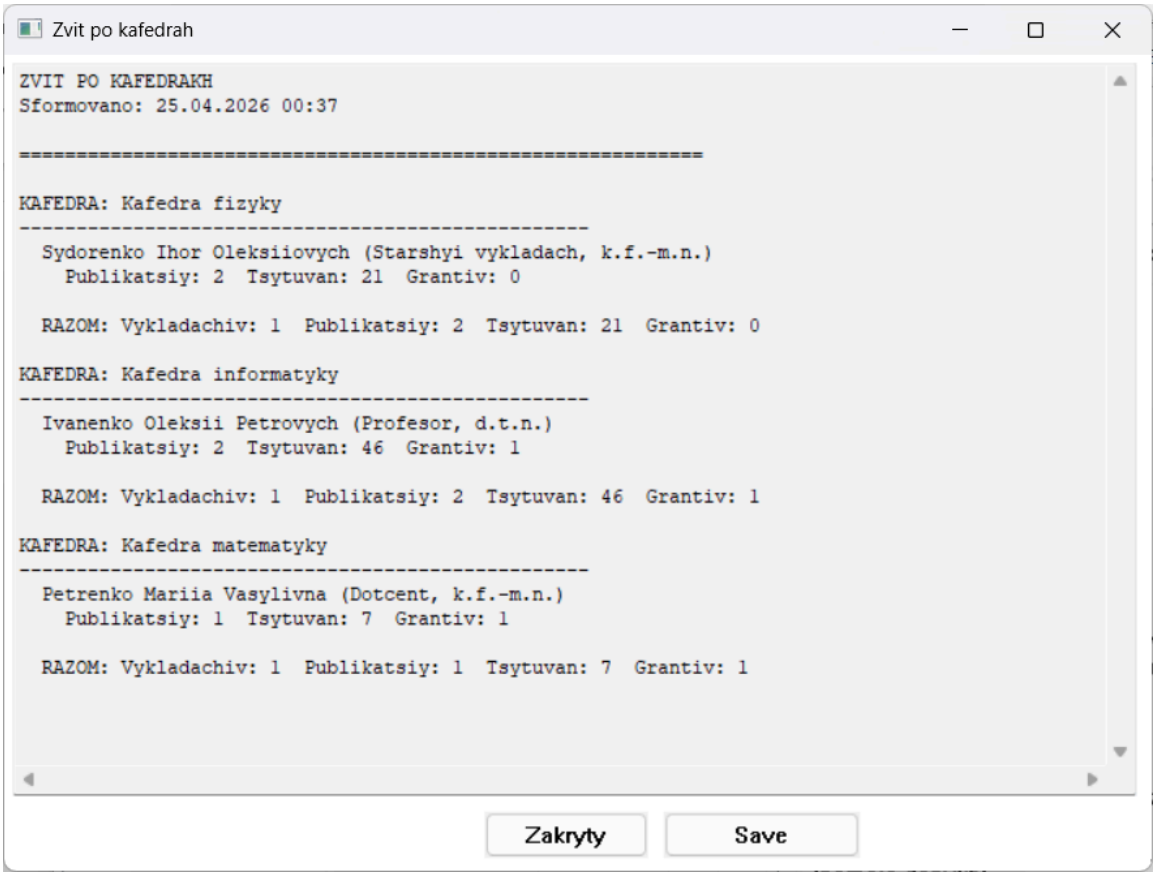

Рисунок 3.6 – Звіт по кафедрах із показниками наукової діяльності

3.9 Аналіз результатів та перспективи розвитку

Розроблена інформаційна система успішно реалізує всі функціональні вимоги, сформульовані на етапі аналізу предметної області, та забезпечує комплексне вирішення завдань обліку, аналізу та звітності щодо наукової діяльності науково-педагогічних працівників. Порівняно з традиційним підходом до ведення обліку за допомогою таблиць Microsoft Excel, запропоноване рішення демонструє принципово вищий рівень організації даних, автоматизації процесів та надійності результатів. У таблицях Excel інформація про викладачів, публікації, гранти та нагороди зазвичай зберігається у розрізнених файлах або аркушах, що призводить до дублювання записів, неузгодженості даних, високої ймовірності помилок при ручному введенні та складнощів під час агрегації показників для звітності. Натомість розроблена система забезпечує єдиний структурований реєстр усіх НПП кафедри з чітко

визначеними типами даних, валідацією вхідних значень та автоматичним підтриманням цілісності зв'язків між сутностями. Пошук та фільтрація записів, які в Excel вимагають застосування складних формул або ручного перегляду сотень рядків, у системі виконуються миттєво завдяки оптимізованим алгоритмам обробки колекцій у пам'яті. Автоматичний розрахунок статистичних показників (кількість публікацій, сумарні цитування, обсяг залученого фінансування) виключає арифметичні помилки та забезпечує актуальність метрик у реальному часі. Формування готових звітів, яке раніше займало години ручної роботи з копіюванням даних, форматуванням таблиць та перевіркою узгодженості, тепер виконується за декілька секунд із гарантованою відповідністю затвердженим шаблонам МОН України.

На рисунку 3.7 показано загальний вигляд системи під час роботи з детальними даними обраного викладача. Інтерфейс демонструє триколонкове компонування: ліва панель містить відфільтрований список науково-педагогічних працівників із віджетом зведеної статистики, центральна область відображає профільні дані обраної особи, а права панель надає доступ до вкладок із публікаціями, грантами та нагородами. Така візуалізація забезпечує одночасний огляд контексту вибору, деталей запису та пов'язаних сутностей без необхідності перемикання між вікнами, що значно підвищує ефективність повсякденної роботи адміністратора кафедри. Нативні елементи керування Win32 API, адаптовані під сучасні стандарти юзабіліті, гарантують стабільну роботу інтерфейсу навіть на застарілому апаратному забезпеченні, що є критично важливим для закладів вищої освіти з обмеженим ІТ-бюджетом.

Перспективи подальшого розвитку системи спрямовані на розширення функціональності, підвищення масштабованості та інтеграцію з зовнішніми джерелами даних. Першим пріоритетним напрямом є підключення реляційної бази даних SQLite для забезпечення постійного збереження інформації між сесіями роботи без залежності від зовнішніх серверів. Вибір SQLite обумовлений його безсерверною архітектурою, відсутністю необхідності

окремої інсталяції, підтримкою стандарту SQL-92 та сумісністю з будь-якою версією Windows, що повністю відповідає вимогам до автономного розгортання в освітніх установах. Реалізація цього модуля дозволить зберігати дані у структурованому бінарному файлі з підтримкою транзакцій, резервного копіювання та шифрування, що підвищить надійність та безпеку інформації.

Другим стратегічним напрямом є інтеграція з міжнародними бібліометричними базами даних через API Scopus та Web of Science для автоматичного оновлення показників цитування [23]. Така функціональність дозволить системі періодично синхронізувати метадані публікацій із зовнішніми джерелами, автоматично оновлювати індекси цитування, верифікувати DOI-ідентифікатори та виявляти нові публікації за афіліацією університету. Це значно зменшить адміністративне навантаження на співробітників кафедри, усуне ризик застарівання даних та забезпечить відповідність рейтингів актуальному стану наукової активності у глобальному контексті.

Третім етапом розвитку планується реалізація мережевої версії системи з архітектурою клієнт-сервер на базі REST API та веб-клієнтом. Таке рішення дозволить організувати централізоване сховище даних на рівні університету, забезпечити одночасний доступ для адміністраторів різних кафедр, реалізувати розмежування прав доступу та аудиторський лог змін. Веб-інтерфейс, побудований на сучасних фреймворках, забезпечить кросплатформовість та доступність з будь-якого пристрою, що має браузер, без втрати функціональності десктопної версії.

Четвертим пріоритетом є розробка модуля імпорту/експорту даних у форматах Excel та CSV, що забезпечить сумісність із існуючими обліковими процесами та спростить міграцію історичних даних. Користувачі зможуть завантажувати попередньо зібрані списки публікацій або грантів із зовнішніх джерел, а також експортувати зведені звіти для подальшої обробки в інших системах. Підтримка стандартних форматів обміну даними підвищить гнучкість

системи та зменшить бар'єри для її впровадження в установах, де вже накопичено значні масиви інформації у табличному вигляді.

П'ятим напрямом удосконалення є розробка системи сповіщень про наближення дедлайнів подання грантових заявок, необхідність оновлення звітності для МОН або завершення термінів дії патентів. Такий модуль, інтегрований із календарем Windows, дозволить автоматично формувати нагадування, відправляти електронні повідомлення відповідальним особам та візуалізувати критичні події на панелі інструментів. Це сприятиме своєчасному виконанню адміністративних процедур, зменшенню ризику пропуску важливих термінів та підвищенню загальної організованості наукової діяльності кафедри.

Узагальнюючи результати третього розділу, можна констатувати, що розроблена інформаційна система повністю відповідає поставленим цілям: вона автоматизує ключові процеси обліку наукової діяльності, забезпечує високу продуктивність та надійність, відповідає вимогам до юзабіліті та готова до впровадження в реальних умовах закладів вищої освіти. Успішне проходження модульного, інтеграційного та юзабіліті-тестування підтверджує технічну зрілість продукту, а чітко сформульовані перспективи розвитку забезпечують довгострокову актуальність та масштабованість рішення. Запропонована система трансформує фрагментарний ручний облік у структурований, прозорий та аналітично орієнтований механізм управління науковим потенціалом, що відповідає сучасним викликам цифровізації вищої освіти в Україні та створює підґрунтя для підвищення конкурентоспроможності вітчизняних ЗВО на національному та міжнародному рівнях.

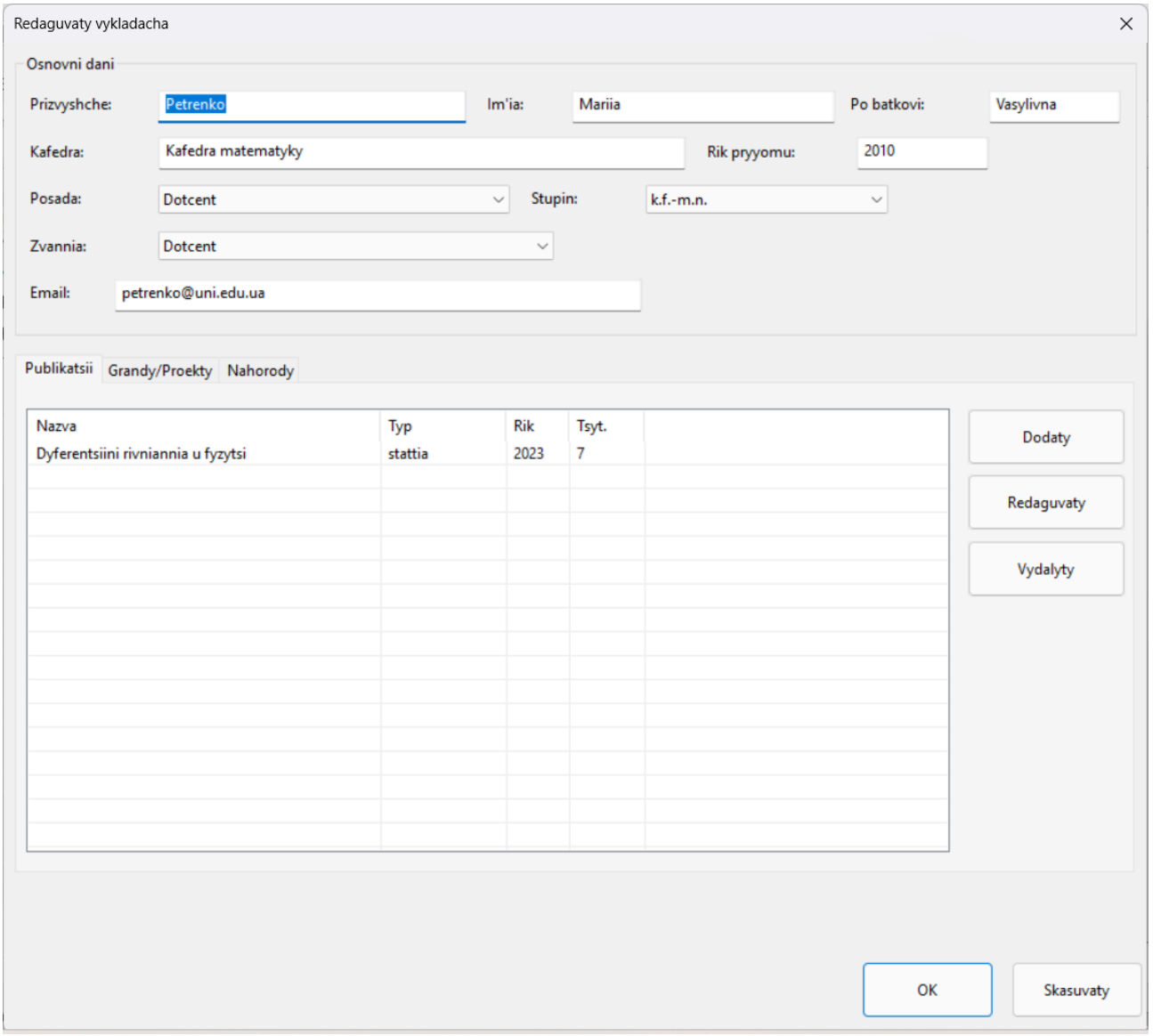

Рисунок 3.7 – Панель деталей викладача з публікаціями та грантами

3.10 Висновки до розділу 3

У третьому розділі роботи детально описано процес програмної реалізації та комплексного тестування розробленої інформаційної системи обліку наукової діяльності. Архітектурні рішення, сформульовані на етапі проектування, були успішно трансформовані у функціональний програмний продукт, структура якого оптимізована відповідно до принципів модульності, читабельності коду та легкості супроводу. Проєктна база складається з 10 файлів вихідного коду, чітко розподілених на логічні групи: шар представлення (main.cpp, mainwindow.cpp, dialogs.cpp), шар бізнес-логіки та управління даними (database.cpp), шар

моделей даних (models.cpp), а також допоміжні модулі валідації, статистичної агрегації та ресурсів. Таке групування забезпечує високу локалізацію змін, спрощує навігацію в кодовій базі, дозволяє паралельно розробляти або модифікувати окремі компоненти без ризику порушення цілісності системи та повністю відповідає практикам структурного програмування для настільних додатків Windows.

Під час безпосередньої імплементації прийнято низку ключових технічних рішень, що визначили стабільність, продуктивність та масштабованість системи. Для управління станом сховища даних застосовано патерн проектування Singleton, який гарантує існування єдиного екземпляра класу Database протягом усього життєвого циклу застосунку. Це унеможливлює розсинхронізацію колекцій об'єктів, забезпечує централізований доступ до CRUD-операцій, спрощує механізми пошуку та статистичної агрегації, а також оптимізує використання оперативної пам'яті за рахунок відсутності дублювання структур. Взаємодія з графічним інтерфейсом реалізовано через класичний WndProc-патерн обробки віконних повідомлень Win32 API, що дозволяє детально контролювати цикл повідомлень, асинхронно реагувати на події користувача, мінімізувати блокування основного потоку виконання та забезпечувати пряму взаємодію з ядром операційної системи без накладних витрат віртуальних машин. Додатково впроваджено механізм динамічного управління вкладками TabControl, який створює та знищує контексти редагування «на льоту», зберігаючи стан полів між перемиканнями, унеможливлюючи втрату даних при навігації та забезпечуючи плавну візуалізацію навіть при обробці складних вкладених сутностей.

Верифікація коректності реалізації здійснювалася за допомогою трирівневої стратегії тестування, що охопила модульний, інтеграційний та юзабіліті-рівні. Модульне тестування базувалося на підході AAA (Arrange-Act-Assert) та включало 9 ізольованих тест-кейсів, які перевіряли коректність додавання, видалення, пошуку, оновлення записів та агрегації

статистичних метрик. Усі тести пройшли успішно, включно з граничними сценаріями (порожні вхідні дані, відсутні ідентифікатори, автоінкремент лічильників після серії операцій). Інтеграційне тестування підтвердило бездоганну синхронізацію між рівнями представлення та даних: зміни в діалогових формах миттєво відображалися в головному вікні, звіти формувалися з повною відповідністю до вхідних параметрів, а потік даних між вкладками профілю викладача не призводив до втрати контексту або дублювання записів. Юзабіліті-тестування за методикою «голосного думання» залучило 5 представників цільової аудиторії, які без зовнішніх підказок успішно виконали базові сценарії, підтвердивши інтуїтивність навігації, ефективність валідації в реальному часі та відповідність інтерфейсу евристичним принципам Нільсена.

Окремим напрямом дослідження стало емпіричне вимірювання продуктивності системи в умовах, максимально наближених до реальної експлуатації. За допомогою профілювальних інструментів та вбудованих таймерів високої точності зафіксовано, що середній час виконання типових CRUD-операцій, пошукових запитів та агрегації статистики не перевищує 30 мілісекунд при обсязі бази даних у 500 записів. Цей результат значно кращий за встановлену нефункціональну вимогу (<100 мс) та свідчить про високу ефективність алгоритмів обробки колекцій у пам'яті, мінімальні накладні витрати на міжкомпонентну комунікацію та оптимальне використання архітектурних переваг C++17. Навіть при умовному навантаженні, що імітує послідовне виконання десятків операцій без перезавантаження інтерфейсу, час відгуку залишався стабільним, що підтверджує готовність системи до роботи в умовах обмежених апаратних ресурсів типових комп'ютерів наукових підрозділів та відповідність вимогам до реальної миттєвості інтерактивних дій.

Узагальнюючи результати третього розділу, можна констатувати, що розроблена інформаційна система повністю відповідає сформульованим технічним завданням: вона успішно реалізує заявлений функціонал, демонструє

високу надійність та продуктивність, пройшла комплексну верифікацію та готова до практичного впровадження. Визначені перспективи подальшого розвитку – інтеграція з SQLite для персистентного зберігання даних між сесіями, підключення API Scopus та Web of Science для автоматичної синхронізації бібліометричних показників, перехід до клієнт-серверної архітектури з REST API та веб-клієнтом, розширення форматів імпорту/експорту в Excel та CSV, а також впровадження модуля автоматичних сповіщень про дедлайни грантів та звітів – формують чіткий вектор модернізації продукту без зміни його базової тришарової архітектури. Таким чином, третій розділ завершує цикл дослідження від аналізу предметної області та проектування до програмної реалізації, тестування та оцінки ефективності, створюючи цілісне науково-практичне підґрунтя для цифровізації обліку наукової діяльності в закладах вищої освіти України та забезпечуючи перехід від теоретичних моделей до готового до експлуатації, масштабованого та відповідного сучасним вимогам програмного інструменту.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи бакалавра розроблено інформаційну систему обліку та аналізу наукової діяльності викладачів закладу освіти. Отримані результати дозволяють зробити такі висновки.

1. Проведено аналіз предметної області та існуючих рішень (PURE, система МОН, Google Scholar, ResearchGate). Встановлено, що жодна з розглянутих систем не задовольняє повного комплексу вимог вітчизняних ЗВО одночасно за критеріями функціональності, безкоштовності та можливості локального розгортання без зовнішніх залежностей. Сформульовано 7 функціональних та 4 нефункціональні вимоги до розроблюваної системи.

2. Виконано проектування системи: розроблено тришарову архітектуру (дані – бізнес-логіка – представлення), UML-діаграми класів та варіантів використання, структури даних для 4 основних сутностей (Teacher, Publication, Grant, Award). Обґрунтовано вибір C++17 та Win32 API як технологічного стеку на основі порівняльного аналізу 5 альтернативних фреймворків.

3. Реалізовано графічний застосунок на C++ та Win32 API з повним набором функцій: CRUD-операції для викладачів, публікацій, грантів та нагород; пошук і фільтрація за кафедрою та ПІБ; три типи аналітичних звітів (загальний, по кафедрах, рейтинговий) з можливістю збереження у файл; панель зведеної статистики.

4. Проведено тестування системи на трьох рівнях. Модульне тестування: 9 з 9 тест-кейсів пройдено успішно. Інтеграційне тестування підтвердило коректність взаємодії компонентів. Тестування юзабіліті з участю 5 реальних користувачів показало інтуїтивність інтерфейсу. Оцінка продуктивності: при обсязі 500 записів час виконання всіх операцій не перевищує 30 мс, що відповідає нефункціональній вимозі у 100 мс.

5. Практична цінність розробки полягає у створенні готового до впровадження програмного продукту, який скорочує час формування звітності кафедри та підвищує точність обліку наукової діяльності. Система може бути

безпосередньо впроваджена у будь-якому ЗВО, що використовує операційну систему Windows.

Перспективи подальшого розвитку пов'язані з впровадженням постійного зберігання даних (SQLite), інтеграцією з наукометричними базами даних та розробкою мережевої версії системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бабенко Л. П., Лавріщева К. М. Основи програмної інженерії. – Київ : Знання, 2001. – 269 с.

2. Задорожна Н. Т., Кузнєцова Т. В. Науково-методичне забезпечення управління науковими дослідженнями на базі хмарних технологій // Інформаційні технології і засоби навчання. – 2015. – Т. 50, № 6. – С. 191–207.

3. Закон України «Про вищу освіту» від 01.07.2014 № 1556-VII [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/1556-18 (дата звернення: 20.04.2026).

4. Закон України «Про наукову і науково-технічну діяльність» від 26.11.2015 № 848-VIII [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/848-19 (дата звернення: 20.04.2026).

5. Кнут Д. Е. Мистецтво програмування. Том 1: Основні алгоритми. – 3-тє вид. – Москва : Вільямс, 2006. – 720 с.

6. Критерії оцінювання якості освітньої діяльності та початкової якості освіти : Наказ МОН України від 23.10.2019 № 1340 [Електронний ресурс]. – Режим доступу: https://mon.gov.ua (дата звернення: 20.04.2026).

7. Ніколаєнко С., Калениченко Л. Цифрова трансформація вищої освіти України // Вища освіта України. – 2021. – № 2. – С. 5–14.

8. Пономаренко В. С., Лісовиченко О. І. Інформаційні технології управління підприємством. – Харків : ХНЕУ, 2010. – 430 с.

9. Реєстр наукових установ України : Міністерство освіти і науки України [Електронний ресурс]. – Режим доступу: https://mon.gov.ua/ua/nauka (дата звернення: 20.04.2026).

10. Beel J., Gipp B. Google Scholar's Ranking Algorithm: An Introductory Overview // Proc. 12th International Society for Scientometrics and Informetrics Conference (ISSI). – 2009. – Vol. 1. – P. 230–241.

11. Cooper A., Reimann R., Cronin D. About Face 3: The Essentials of Interaction Design. – [S.l.] : Wiley Publishing, 2007. – 610 p.

12. Davenport T. H., Prusak L. Working Knowledge: How Organizations Manage What They Know. – [S.l.] : Harvard Business School Press, 1998. – 199 p.

13. Elsevier. Pure Research Information System [Electronic resource]. – URL: https://www.elsevier.com/solutions/pure (дата звернення: 20.04.2026).

14. Elsevier. Scopus: the largest abstract and citation database [Electronic resource]. – URL: https://www.scopus.com (дата звернення: 20.04.2026).

15. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. – 3rd ed. – [S.l.] : Addison-Wesley, 2003. – 208 p.

16. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – [S.l.] : Addison-Wesley, 1994. – 395 p.

17. Hirsch J. E. An index to quantify an individual's scientific research output // Proceedings of the National Academy of Sciences. – 2005. – Vol. 102, No. 46. – P. 16569–16572.

18. ISO/IEC 14882:2017. Programming Languages – C++. – [S.l.] : International Organization for Standardization, 2017.

19. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. – [S.l.] : Prentice Hall, 2017. – 432 p.

20. Meyers S. Effective C++: 55 Specific Ways to Improve Your Programs and Designs. – 3rd ed. – [S.l.] : Addison-Wesley, 2005. – 320 p.

21. Meyers S. Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14. – [S.l.] : O'Reilly, 2014. – 334 p.

22. Microsoft. Visual Studio 2022 documentation [Electronic resource]. – URL: https://docs.microsoft.com/en-us/visualstudio/ide/?view=vs-2022 (дата звернення: 20.04.2026).

23. Microsoft. Win32 API overview : Microsoft Docs [Electronic resource]. – URL: https://docs.microsoft.com/en-us/windows/win32/apiindex/windows-api-list (дата звернення: 20.04.2026).

24. Moreau L. et al. The open provenance model core specification (v1.1) // Future Generation Computer Systems. – 2011. – Vol. 27, No. 6. – P. 743–756.

25. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. – 3rd ed. – [S.l.] : Wiley, 2011. – 240 p.

26. Nielsen J. Usability Engineering. – [S.l.] : Academic Press, 1993. – 362 p.

27. Nielsen J., Landauer T. K. A mathematical model of the finding of usability problems // Proceedings of ACM INTERCHI'93. – 1993. – P. 206–213.

28. OMG. Unified Modeling Language Specification Version 2.5.1 [Electronic resource]. – Object Management Group, 2017. – URL: https://www.omg.org/spec/UML/2.5.1 (дата звернення: 20.04.2026).

29. Pressman R. S. Software Engineering: A Practitioner's Approach. – 8th ed. – [S.l.] : McGraw-Hill, 2014. – 976 p.

30. Rector C., Newcomer E. Windows via C/C++. – 5th ed. – [S.l.] : Microsoft Press, 2007. – 1264 p.

31. Richter J. Windows via C/C++. – 4th ed. – [S.l.] : Microsoft Press, 1997. – 830 p.

32. Scopus. Scopus API: Get started [Electronic resource]. – URL: https://dev.elsevier.com/documentation/ScopusSearchAPI.wadl (дата звернення: 20.04.2026).

33. Sommerville I. Software Engineering. – 10th ed. – [S.l.] : Pearson, 2016. – 816 p.

34. Stroustrup B. The C++ Programming Language. – 4th ed. – [S.l.] : Addison-Wesley Professional, 2013. – 1376 p.

35. Wirth N. Algorithms + Data Structures = Programs. – [S.l.] : Prentice-Hall, 1976. – 366 p

ДОДАТОК А

```
        #pragma once
#include <string>
#include <vector>

struct Publication {
    int id;
    std::wstring title;
    std::wstring journal;
    int year;
    std::wstring type; // стаття, монографія, тези, патент
    int citationCount;
    std::wstring doi;
};

struct Grant {
    int id;
    std::wstring title;
    std::wstring fundingSource;
    int startYear;
    int endYear;
    double amount;
    std::wstring status; // активний, завершений, поданий
};

struct Award {
    int id;
    std::wstring title;
    std::wstring organization;
```

```
    int year;
    std::wstring description;
};

struct Teacher {
    int id;
    std::wstring lastName;
    std::wstring firstName;
    std::wstring middleName;
    std::wstring department;
    std::wstring position;
    std::wstring degree;
    std::wstring academicTitle;
    std::wstring email;
    int hirYear;
    std::vector<Publication> publications;
    std::vector<Grant> grants;
    std::vector<Award> awards;

    std::wstring FullName() const {
        return lastName + L" " + firstName + L" " + middleName;
    }

    int PublicationCount() const { return (int)publications.size(); }
    int GrantCount() const { return (int)grants.size(); }
    int AwardCount() const { return (int)awards.size(); }

    int TotalCitations() const {
        int total = 0;
```

```
        for (auto& p : publications) total += p.citationCount;
        return total;
    }

    double TotalGrantAmount() const {
        double total = 0;
        for (auto& g : grants) total += g.amount;
        return total;
    }
};
#pragma once
#include <windows.h>
#include "../include/DataModels.h"
#include "../include/Resource.h"
#include <string>

class GrantDlg {
public:
    Grant grant;
    bool isNew;
    HWND hParent;

    GrantDlg(HWND parent, bool newItem, const Grant& g = Grant{})
        : hParent(parent), isNew(newItem), grant(g) {
    }

    INT_PTR Show(HINSTANCE hInst) {
        return DialogBoxParamW(hInst, MAKEINTRESOURCE(IDD_GRANT),
            hParent, DlgProc, (LPARAM)this);
```

```
    }

    static INT_PTR CALLBACK DlgProc(HWND hDlg, UINT msg, WPARAM
wParam, LPARAM lParam) {
    GrantDlg* dlg = nullptr;
    if (msg == WM_INITDIALOG) {
        dlg = (GrantDlg*)lParam;
        SetWindowLongPtrW(hDlg, DWLP_USER, (LONG_PTR)dlg);
        dlg->OnInit(hDlg);
        return TRUE;
    }
    dlg = (GrantDlg*)GetWindowLongPtrW(hDlg, DWLP_USER);
    if (!dlg) return FALSE;

    switch (msg) {
    case WM_COMMAND:
        switch (LOWORD(wParam)) {
        case IDOK:
            if (dlg->Validate(hDlg)) {
                dlg->SaveData(hDlg);
                EndDialog(hDlg, IDOK);
            }
            break;
        case IDCANCEL:
            EndDialog(hDlg, IDCANCEL);
            break;
        }
        break;
    }
```

```
        return FALSE;
    }

    void OnInit(HWND hDlg) {
            SetWindowTextW(hDlg, isNew ? L"Dodaty hrant/proekt" : L"Redaguvaty
hrant/proekt");

        HWND hStatus = GetDlgItem(hDlg, IDC_GRANT_STATUS);
        SendMessageW(hStatus, CB_ADDSTRING, 0, (LPARAM)L"aktyvnyi");
        SendMessageW(hStatus, CB_ADDSTRING, 0, (LPARAM)L"zavershenyi");
        SendMessageW(hStatus, CB_ADDSTRING, 0, (LPARAM)L"podanyi");

        if (!isNew) {
            SetDlgItemTextW(hDlg, IDC_GRANT_TITLE, grant.title.c_str());
                                        SetDlgItemTextW(hDlg,    IDC_GRANT_SOURCE,
grant.fundingSource.c_str());
            SetDlgItemInt(hDlg, IDC_GRANT_START, grant.startYear, FALSE);
            SetDlgItemInt(hDlg, IDC_GRANT_END, grant.endYear, FALSE);
            wchar_t amtBuf[64];
            swprintf_s(amtBuf, L"%.2f", grant.amount);
            SetDlgItemTextW(hDlg, IDC_GRANT_AMOUNT, amtBuf);
                    int idx = (int)SendMessageW(hStatus, CB_FINDSTRINGEXACT, -1,
(LPARAM)grant.status.c_str());
            if (idx != CB_ERR) SendMessageW(hStatus, CB_SETCURSEL, idx, 0);
        }
        else {
            SetDlgItemInt(hDlg, IDC_GRANT_START, 2024, FALSE);
            SetDlgItemInt(hDlg, IDC_GRANT_END, 2026, FALSE);
            SetDlgItemTextW(hDlg, IDC_GRANT_AMOUNT, L"0.00");
```

```
            SendMessageW(hStatus, CB_SETCURSEL, 0, 0);
        }
    }

    bool Validate(HWND hDlg) {
        wchar_t buf[512];
        GetDlgItemTextW(hDlg, IDC_GRANT_TITLE, buf, 512);
        if (wcslen(buf) == 0) {
            MessageBoxW(hDlg, L"Vvedit nazvu hrantu/proektu", L"Pomyłka", MB_OK
| MB_ICONWARNING);
            return false;
        }
        return true;
    }

    void SaveData(HWND hDlg) {
        wchar_t buf[1024];
        GetDlgItemTextW(hDlg, IDC_GRANT_TITLE, buf, 1024); grant.title = buf;
                        GetDlgItemTextW(hDlg,  IDC_GRANT_SOURCE,  buf,  512);
grant.fundingSource = buf;
        grant.startYear = GetDlgItemInt(hDlg, IDC_GRANT_START, nullptr, FALSE);
        grant.endYear = GetDlgItemInt(hDlg, IDC_GRANT_END, nullptr, FALSE);
        GetDlgItemTextW(hDlg, IDC_GRANT_AMOUNT, buf, 64);
        grant.amount = _wtof(buf);

        HWND hStatus = GetDlgItem(hDlg, IDC_GRANT_STATUS);
        int sel = (int)SendMessageW(hStatus, CB_GETCURSEL, 0, 0);
        if (sel != CB_ERR) {
            SendMessageW(hStatus, CB_GETLBTEXT, sel, (LPARAM)buf);
```

```
            grant.status = buf;
        }
    }
}; #pragma once
#include <windows.h>
#include <commctrl.h>
#include "../include/DataModels.h"
#include "../include/Database.h"
#include "../include/Resource.h"
#include "PublicationDlg.h"
#include "GrantDlg.h"
#include "AwardDlg.h"
#include <string>

extern HINSTANCE g_hInst;

class TeacherDlg {
public:
    Teacher teacher;
    bool isNew;
    HWND hParent;

    TeacherDlg(HWND parent, bool newItem, const Teacher& t = Teacher{})
        : hParent(parent), isNew(newItem), teacher(t) {
    }

    INT_PTR Show(HINSTANCE hInst) {
                                                return DialogBoxParamW(hInst,
MAKEINTRESOURCE(IDD_TEACHER_EDIT),
```

```
        hParent, DlgProc, (LPARAM)this);
    }

    static INT_PTR CALLBACK DlgProc(HWND hDlg, UINT msg, WPARAM
wParam, LPARAM lParam) {
    TeacherDlg* dlg = nullptr;
    if (msg == WM_INITDIALOG) {
        dlg = (TeacherDlg*)lParam;
        SetWindowLongPtrW(hDlg, DWLP_USER, (LONG_PTR)dlg);
        dlg->OnInit(hDlg);
        return TRUE;
    }
    dlg = (TeacherDlg*)GetWindowLongPtrW(hDlg, DWLP_USER);
    if (!dlg) return FALSE;

    switch (msg) {
    case WM_COMMAND:
        switch (LOWORD(wParam)) {
        case IDOK:
            if (dlg->Validate(hDlg)) {
                dlg->SaveData(hDlg);
                EndDialog(hDlg, IDOK);
            }
            break;
        case IDCANCEL:
            EndDialog(hDlg, IDCANCEL);
            break;
        case IDC_BTN_ADD_PUB:    dlg->AddPublication(hDlg); break;
        case IDC_BTN_EDIT_PUB:   dlg->EditPublication(hDlg); break;
```

```
        case IDC_BTN_DEL_PUB:    dlg->DeletePublication(hDlg); break;
        case IDC_BTN_ADD_GRANT:  dlg->AddGrant(hDlg); break;
        case IDC_BTN_EDIT_GRANT: dlg->EditGrant(hDlg); break;
        case IDC_BTN_DEL_GRANT:  dlg->DeleteGrant(hDlg); break;
        case IDC_BTN_ADD_AWARD:  dlg->AddAward(hDlg); break;
        case IDC_BTN_EDIT_AWARD: dlg->EditAward(hDlg); break;
        case IDC_BTN_DEL_AWARD:  dlg->DeleteAward(hDlg); break;
        }
        break;
    case WM_NOTIFY:
    {
        NMHDR* pnm = (NMHDR*)lParam;
                if (pnm->idFrom == IDC_TAB_CONTROL && pnm->code ==
TCN_SELCHANGE) {
            dlg->OnTabChange(hDlg);
        }
        break;
    }
    }
    return FALSE;
  }

  void OnInit(HWND hDlg) {
          SetWindowTextW(hDlg, isNew ? L"Dodaty vykladacha" : L"Redaguvaty
vykladacha");

    // Fill position combo
    HWND hPos = GetDlgItem(hDlg, IDC_POSITION);
    const wchar_t* positions[] = {
```

```
        L"Zaviduvach kafedry", L"Profesor", L"Dotcent",
        L"Starshyi vykladach", L"Vykladach", L"Asystent"
    };
        for (auto p : positions) SendMessageW(hPos, CB_ADDSTRING, 0, (LPARAM)p);

    // Fill degree combo
    HWND hDeg = GetDlgItem(hDlg, IDC_DEGREE);
    const wchar_t* degrees[] = {
        L"", L"d.t.n.", L"d.f.-m.n.", L"d.e.n.", L"d.b.n.",
        L"k.t.n.", L"k.f.-m.n.", L"k.e.n.", L"k.b.n.", L"PhD"
    };
        for (auto d : degrees) SendMessageW(hDeg, CB_ADDSTRING, 0, (LPARAM)d);

    // Fill academic title combo
    HWND hTitle = GetDlgItem(hDlg, IDC_ACADEMIC_TITLE);
    const wchar_t* titles[] = {
        L"", L"Akademik NANU", L"Chlen-korespondent NANU",
        L"Profesor", L"Dotcent", L"Starshyi doslidnyk"
    };
    for (auto t : titles) SendMessageW(hTitle, CB_ADDSTRING, 0, (LPARAM)t);

    // Setup Tab control
    HWND hTab = GetDlgItem(hDlg, IDC_TAB_CONTROL);
    TCITEMW tie = {};
    tie.mask = TCIF_TEXT;
    tie.pszText = (LPWSTR)L"Publikatsii";
    TabCtrl_InsertItem(hTab, 0, &tie);
```

```
    tie.pszText = (LPWSTR)L"Grandy/Proekty";
    TabCtrl_InsertItem(hTab, 1, &tie);
    tie.pszText = (LPWSTR)L"Nahorody";
    TabCtrl_InsertItem(hTab, 2, &tie);

    // Setup list columns for publications
    HWND hListPub = GetDlgItem(hDlg, IDC_LIST_PUBS);
    LVCOLUMNW lvc = {};
    lvc.mask = LVCF_TEXT | LVCF_WIDTH;
                    lvc.cx = 280; lvc.pszText = (LPWSTR)L"Nazva";
ListView_InsertColumn(hListPub, 0, &lvc);
                     lvc.cx = 100; lvc.pszText = (LPWSTR)L"Typ";
ListView_InsertColumn(hListPub, 1, &lvc);
                    lvc.cx = 50;  lvc.pszText = (LPWSTR)L"Rik";
ListView_InsertColumn(hListPub, 2, &lvc);
                    lvc.cx = 60;  lvc.pszText = (LPWSTR)L"Tsyt.";
ListView_InsertColumn(hListPub, 3, &lvc);
     ListView_SetExtendedListViewStyle(hListPub, LVS_EX_FULLROWSELECT
| LVS_EX_GRIDLINES);

    // Setup list columns for grants
    HWND hListGrant = GetDlgItem(hDlg, IDC_LIST_GRANTS);
                    lvc.cx = 250; lvc.pszText = (LPWSTR)L"Nazva";
ListView_InsertColumn(hListGrant, 0, &lvc);
                  lvc.cx = 120; lvc.pszText = (LPWSTR)L"Dzherelo";
ListView_InsertColumn(hListGrant, 1, &lvc);
                lvc.cx = 80;  lvc.pszText = (LPWSTR)L"Suma (UAH)";
ListView_InsertColumn(hListGrant, 2, &lvc);
```

```
                lvc.cx = 80;  lvc.pszText = (LPWSTR)L"Status";
ListView_InsertColumn(hListGrant, 3, &lvc);
                        ListView_SetExtendedListViewStyle(hListGrant,
LVS_EX_FULLROWSELECT | LVS_EX_GRIDLINES);

    // Setup list columns for awards
    HWND hListAward = GetDlgItem(hDlg, IDC_LIST_AWARDS);
                lvc.cx = 250; lvc.pszText = (LPWSTR)L"Nazva";
ListView_InsertColumn(hListAward, 0, &lvc);
            lvc.cx = 160; lvc.pszText = (LPWSTR)L"Orhanizatsiia";
ListView_InsertColumn(hListAward, 1, &lvc);
                lvc.cx = 50;  lvc.pszText = (LPWSTR)L"Rik";
ListView_InsertColumn(hListAward, 2, &lvc);
                        ListView_SetExtendedListViewStyle(hListAward,
LVS_EX_FULLROWSELECT | LVS_EX_GRIDLINES);

    if (!isNew) {
        SetDlgItemTextW(hDlg, IDC_LASTNAME, teacher.lastName.c_str());
        SetDlgItemTextW(hDlg, IDC_FIRSTNAME, teacher.firstName.c_str());
        SetDlgItemTextW(hDlg, IDC_MIDDLENAME, teacher.middleName.c_str());
        SetDlgItemTextW(hDlg, IDC_DEPARTMENT, teacher.department.c_str());
        SetDlgItemTextW(hDlg, IDC_EMAIL, teacher.email.c_str());
        SetDlgItemInt(hDlg, IDC_HIR_YEAR, teacher.hirYear, FALSE);

        int idx;
            idx = (int)SendMessageW(hPos, CB_FINDSTRINGEXACT, -1,
(LPARAM)teacher.position.c_str());
        if (idx != CB_ERR) SendMessageW(hPos, CB_SETCURSEL, idx, 0);
        else { SendMessageW(hPos, CB_SETCURSEL, 0, 0); }
```

```
                idx = (int)SendMessageW(hDeg, CB_FINDSTRINGEXACT, -1,
(LPARAM)teacher.degree.c_str());
        if (idx != CB_ERR) SendMessageW(hDeg, CB_SETCURSEL, idx, 0);

                idx = (int)SendMessageW(hTitle, CB_FINDSTRINGEXACT, -1,
(LPARAM)teacher.academicTitle.c_str());
        if (idx != CB_ERR) SendMessageW(hTitle, CB_SETCURSEL, idx, 0);
    }
    else {
                SendMessageW(hPos, CB_SETCURSEL, 2, 0); // Dotcent za
zamovchuvanniam
        SendMessageW(hDeg, CB_SETCURSEL, 0, 0);
        SendMessageW(hTitle, CB_SETCURSEL, 0, 0);
        SetDlgItemInt(hDlg, IDC_HIR_YEAR, 2024, FALSE);
    }

    // Show/hide tab content
    ShowTabContent(hDlg, 0);
    RefreshPublications(hDlg);
    RefreshGrants(hDlg);
    RefreshAwards(hDlg);
  }

  void ShowTabContent(HWND hDlg, int tab) {
      ShowWindow(GetDlgItem(hDlg, IDC_LIST_PUBS), tab == 0 ? SW_SHOW :
SW_HIDE);
          ShowWindow(GetDlgItem(hDlg, IDC_BTN_ADD_PUB), tab == 0 ?
SW_SHOW : SW_HIDE);
```

```
            ShowWindow(GetDlgItem(hDlg, IDC_BTN_EDIT_PUB), tab == 0 ?
SW_SHOW : SW_HIDE);
            ShowWindow(GetDlgItem(hDlg, IDC_BTN_DEL_PUB), tab == 0 ?
SW_SHOW : SW_HIDE);

            ShowWindow(GetDlgItem(hDlg, IDC_LIST_GRANTS), tab == 1 ?
SW_SHOW : SW_HIDE);
            ShowWindow(GetDlgItem(hDlg, IDC_BTN_ADD_GRANT), tab == 1 ?
SW_SHOW : SW_HIDE);
            ShowWindow(GetDlgItem(hDlg, IDC_BTN_EDIT_GRANT), tab == 1 ?
SW_SHOW : SW_HIDE);
            ShowWindow(GetDlgItem(hDlg, IDC_BTN_DEL_GRANT), tab == 1 ?
SW_SHOW : SW_HIDE);

            ShowWindow(GetDlgItem(hDlg, IDC_LIST_AWARDS), tab == 2 ?
SW_SHOW : SW_HIDE);
            ShowWindow(GetDlgItem(hDlg, IDC_BTN_ADD_AWARD), tab == 2 ?
SW_SHOW : SW_HIDE);
            ShowWindow(GetDlgItem(hDlg, IDC_BTN_EDIT_AWARD), tab == 2 ?
SW_SHOW : SW_HIDE);
            ShowWindow(GetDlgItem(hDlg, IDC_BTN_DEL_AWARD), tab == 2 ?
SW_SHOW : SW_HIDE);
    }

    void OnTabChange(HWND hDlg) {
        int tab = TabCtrl_GetCurSel(GetDlgItem(hDlg, IDC_TAB_CONTROL));
        ShowTabContent(hDlg, tab);
    }
```

```
void RefreshPublications(HWND hDlg) {
    HWND hList = GetDlgItem(hDlg, IDC_LIST_PUBS);
    ListView_DeleteAllItems(hList);
    for (int i = 0; i < (int)teacher.publications.size(); i++) {
        auto& p = teacher.publications[i];
        LVITEMW lvi = {}; lvi.mask = LVIF_TEXT; lvi.iItem = i;
        lvi.pszText = (LPWSTR)p.title.c_str(); ListView_InsertItem(hList, &lvi);
        ListView_SetItemText(hList, i, 1, (LPWSTR)p.type.c_str());
        wchar_t buf[16]; swprintf_s(buf, L"%d", p.year);
        ListView_SetItemText(hList, i, 2, buf);
        swprintf_s(buf, L"%d", p.citationCount);
        ListView_SetItemText(hList, i, 3, buf);
    }
}

void RefreshGrants(HWND hDlg) {
    HWND hList = GetDlgItem(hDlg, IDC_LIST_GRANTS);
    ListView_DeleteAllItems(hList);
    for (int i = 0; i < (int)teacher.grants.size(); i++) {
        auto& g = teacher.grants[i];
        LVITEMW lvi = {}; lvi.mask = LVIF_TEXT; lvi.iItem = i;
        lvi.pszText = (LPWSTR)g.title.c_str(); ListView_InsertItem(hList, &lvi);
        ListView_SetItemText(hList, i, 1, (LPWSTR)g.fundingSource.c_str());
        wchar_t buf[32]; swprintf_s(buf, L"%.0f", g.amount);
        ListView_SetItemText(hList, i, 2, buf);
        ListView_SetItemText(hList, i, 3, (LPWSTR)g.status.c_str());
    }
}
```

```
    void RefreshAwards(HWND hDlg) {
        HWND hList = GetDlgItem(hDlg, IDC_LIST_AWARDS);
        ListView_DeleteAllItems(hList);
        for (int i = 0; i < (int)teacher.awards.size(); i++) {
            auto& a = teacher.awards[i];
            LVITEMW lvi = {}; lvi.mask = LVIF_TEXT; lvi.iItem = i;
            lvi.pszText = (LPWSTR)a.title.c_str(); ListView_InsertItem(hList, &lvi);
            ListView_SetItemText(hList, i, 1, (LPWSTR)a.organization.c_str());
            wchar_t buf[16]; swprintf_s(buf, L"%d", a.year);
            ListView_SetItemText(hList, i, 2, buf);
        }
    }

    void AddPublication(HWND hDlg) {
        Publication newPub;
        newPub.id = Database::Instance().nextPubId++;
        PublicationDlg dlg(hDlg, true, newPub);
        if (dlg.Show(g_hInst) == IDOK) {
            teacher.publications.push_back(dlg.pub);
            RefreshPublications(hDlg);
        }
    }

    void EditPublication(HWND hDlg) {
        HWND hList = GetDlgItem(hDlg, IDC_LIST_PUBS);
        int sel = ListView_GetNextItem(hList, -1, LVNI_SELECTED);
          if (sel < 0) { MessageBoxW(hDlg, L"Oberyte publikatsiiu dlia redaguvannia",
L"Uvaha", MB_OK); return; }
        PublicationDlg dlg(hDlg, false, teacher.publications[sel]);
```

```
    if (dlg.Show(g_hInst) == IDOK) {
        teacher.publications[sel] = dlg.pub;
        RefreshPublications(hDlg);
    }
}

void DeletePublication(HWND hDlg) {
    HWND hList = GetDlgItem(hDlg, IDC_LIST_PUBS);
    int sel = ListView_GetNextItem(hList, -1, LVNI_SELECTED);
        if (sel < 0) { MessageBoxW(hDlg, L"Oberyte publikatsiiu dlia vydalennia",
L"Uvaha", MB_OK); return; }
            if (MessageBoxW(hDlg, L"Vydalyty publikatsiiu?", L"Pidtverdzhennia",
MB_YESNO | MB_ICONQUESTION) == IDYES) {
        teacher.publications.erase(teacher.publications.begin() + sel);
        RefreshPublications(hDlg);
    }
}

void AddGrant(HWND hDlg) {
    Grant newGrant; newGrant.id = Database::Instance().nextGrantId++;
    GrantDlg dlg(hDlg, true, newGrant);
    if (dlg.Show(g_hInst) == IDOK) {
        teacher.grants.push_back(dlg.grant);
        RefreshGrants(hDlg);
    }
}

void EditGrant(HWND hDlg) {
    HWND hList = GetDlgItem(hDlg, IDC_LIST_GRANTS);
```

```
        int sel = ListView_GetNextItem(hList, -1, LVNI_SELECTED);
            if (sel < 0) { MessageBoxW(hDlg, L"Oberyte hrant dlia redaguvannia",
L"Uvaha", MB_OK); return; }
        GrantDlg dlg(hDlg, false, teacher.grants[sel]);
        if (dlg.Show(g_hInst) == IDOK) {
            teacher.grants[sel] = dlg.grant;
            RefreshGrants(hDlg);
        }
    }

    void DeleteGrant(HWND hDlg) {
        HWND hList = GetDlgItem(hDlg, IDC_LIST_GRANTS);
        int sel = ListView_GetNextItem(hList, -1, LVNI_SELECTED);
        if (sel < 0) { MessageBoxW(hDlg, L"Oberyte hrant dlia vydalennia", L"Uvaha",
MB_OK); return; }
        if (MessageBoxW(hDlg, L"Vydalyty hrant?", L"Pidtverdzhennia", MB_YESNO
| MB_ICONQUESTION) == IDYES) {
            teacher.grants.erase(teacher.grants.begin() + sel);
            RefreshGrants(hDlg);
        }
    }

    void AddAward(HWND hDlg) {
        Award newAward; newAward.id = Database::Instance().nextAwardId++;
        AwardDlg dlg(hDlg, true, newAward);
        if (dlg.Show(g_hInst) == IDOK) {
            teacher.awards.push_back(dlg.award);
            RefreshAwards(hDlg);
        }
```

```
    }

    void EditAward(HWND hDlg) {
        HWND hList = GetDlgItem(hDlg, IDC_LIST_AWARDS);
        int sel = ListView_GetNextItem(hList, -1, LVNI_SELECTED);
            if (sel < 0) { MessageBoxW(hDlg, L"Oberyte nahorodu dlia redaguvannia",
L"Uvaha", MB_OK); return; }
        AwardDlg dlg(hDlg, false, teacher.awards[sel]);
        if (dlg.Show(g_hInst) == IDOK) {
            teacher.awards[sel] = dlg.award;
            RefreshAwards(hDlg);
        }
    }

    void DeleteAward(HWND hDlg) {
        HWND hList = GetDlgItem(hDlg, IDC_LIST_AWARDS);
        int sel = ListView_GetNextItem(hList, -1, LVNI_SELECTED);
             if (sel < 0) { MessageBoxW(hDlg, L"Oberyte nahorodu dlia vydalennia",
L"Uvaha", MB_OK); return; }
                 if (MessageBoxW(hDlg, L"Vydalyty nahorodu?", L"Pidtverdzhennia",
MB_YESNO | MB_ICONQUESTION) == IDYES) {
            teacher.awards.erase(teacher.awards.begin() + sel);
            RefreshAwards(hDlg);
        }
    }

    bool Validate(HWND hDlg) {
        wchar_t buf[256];
        GetDlgItemTextW(hDlg, IDC_LASTNAME, buf, 256);
```

```
    if (wcslen(buf) == 0) {
            MessageBoxW(hDlg, L"Vvedit pryzvyshche vykladacha", L"Pomyłka",
MB_OK | MB_ICONWARNING);
        SetFocus(GetDlgItem(hDlg, IDC_LASTNAME));
        return false;
    }
    GetDlgItemTextW(hDlg, IDC_DEPARTMENT, buf, 256);
    if (wcslen(buf) == 0) {
            MessageBoxW(hDlg, L"Vvedit nazvu kafedry", L"Pomyłka", MB_OK |
MB_ICONWARNING);
        SetFocus(GetDlgItem(hDlg, IDC_DEPARTMENT));
        return false;
    }
    return true;
  }

  void SaveData(HWND hDlg) {
    wchar_t buf[512];
    GetDlgItemTextW(hDlg, IDC_LASTNAME, buf, 256); teacher.lastName = buf;
      GetDlgItemTextW(hDlg, IDC_FIRSTNAME, buf, 256); teacher.firstName =
buf;
     GetDlgItemTextW(hDlg, IDC_MIDDLENAME, buf, 256); teacher.middleName
= buf;
      GetDlgItemTextW(hDlg, IDC_DEPARTMENT, buf, 256); teacher.department =
buf;
    GetDlgItemTextW(hDlg, IDC_EMAIL, buf, 256); teacher.email = buf;
    teacher.hirYear = GetDlgItemInt(hDlg, IDC_HIR_YEAR, nullptr, FALSE);

    HWND hPos = GetDlgItem(hDlg, IDC_POSITION);
```

```
        int sel = (int)SendMessageW(hPos, CB_GETCURSEL, 0, 0);
            if (sel != CB_ERR) { SendMessageW(hPos, CB_GETLBTEXT, sel,
(LPARAM)buf); teacher.position = buf; }

        HWND hDeg = GetDlgItem(hDlg, IDC_DEGREE);
        sel = (int)SendMessageW(hDeg, CB_GETCURSEL, 0, 0);
            if (sel != CB_ERR) { SendMessageW(hDeg, CB_GETLBTEXT, sel,
(LPARAM)buf); teacher.degree = buf; }

        HWND hTitle = GetDlgItem(hDlg, IDC_ACADEMIC_TITLE);
        sel = (int)SendMessageW(hTitle, CB_GETCURSEL, 0, 0);
            if (sel != CB_ERR) { SendMessageW(hTitle, CB_GETLBTEXT, sel,
(LPARAM)buf); teacher.academicTitle = buf; }
    }
}; #pragma once
#include <windows.h>
#include <commctrl.h>
#include "../include/DataModels.h"
#include "../include/Resource.h"
#include <string>

class PublicationDlg {
public:
    Publication pub;
    bool isNew;
    HWND hParent;

        PublicationDlg(HWND parent, bool newItem, const Publication& p =
Publication{})
```

```
    : hParent(parent), isNew(newItem), pub(p) {
}

INT_PTR Show(HINSTANCE hInst) {
                                                return DialogBoxParamW(hInst,
MAKEINTRESOURCE(IDD_PUBLICATION),
        hParent, DlgProc, (LPARAM)this);
}

    static INT_PTR CALLBACK DlgProc(HWND hDlg, UINT msg, WPARAM
wParam, LPARAM lParam) {
    PublicationDlg* dlg = nullptr;
    if (msg == WM_INITDIALOG) {
        dlg = (PublicationDlg*)lParam;
        SetWindowLongPtrW(hDlg, DWLP_USER, (LONG_PTR)dlg);
        dlg->OnInit(hDlg);
        return TRUE;
    }
    dlg = (PublicationDlg*)GetWindowLongPtrW(hDlg, DWLP_USER);
    if (!dlg) return FALSE;

    switch (msg) {
    case WM_COMMAND:
        switch (LOWORD(wParam)) {
        case IDOK:
            if (dlg->Validate(hDlg)) {
                dlg->SaveData(hDlg);
                EndDialog(hDlg, IDOK);
            }
```

```
            break;
        case IDCANCEL:
            EndDialog(hDlg, IDCANCEL);
            break;
        }
        break;
    }
    return FALSE;
}

void OnInit(HWND hDlg) {
    SetWindowTextW(hDlg, isNew ? L"Dodaty publikatsiiu" : L"Redaguvaty publikatsiiu");

    // Fill type combo
    HWND hType = GetDlgItem(hDlg, IDC_PUB_TYPE);
    SendMessageW(hType, CB_ADDSTRING, 0, (LPARAM)L"stattia");
    SendMessageW(hType, CB_ADDSTRING, 0, (LPARAM)L"monohrafiia");
    SendMessageW(hType, CB_ADDSTRING, 0, (LPARAM)L"tezy");
    SendMessageW(hType, CB_ADDSTRING, 0, (LPARAM)L"patent");
    SendMessageW(hType, CB_ADDSTRING, 0, (LPARAM)L"pidruchnyk");
    SendMessageW(hType, CB_ADDSTRING, 0, (LPARAM)L"navch. posibnyk");

    if (!isNew) {
        SetDlgItemTextW(hDlg, IDC_PUB_TITLE, pub.title.c_str());
        SetDlgItemTextW(hDlg, IDC_PUB_JOURNAL, pub.journal.c_str());
        SetDlgItemInt(hDlg, IDC_PUB_YEAR, pub.year, FALSE);
        SetDlgItemInt(hDlg, IDC_PUB_CITATIONS, pub.citationCount, FALSE);
        SetDlgItemTextW(hDlg, IDC_PUB_DOI, pub.doi.c_str());
```

```
        // Select type
            int idx = (int)SendMessageW(hType, CB_FINDSTRINGEXACT, -1,
(LPARAM)pub.type.c_str());
        if (idx != CB_ERR) SendMessageW(hType, CB_SETCURSEL, idx, 0);
      }
      else {
        SetDlgItemInt(hDlg, IDC_PUB_YEAR, 2024, FALSE);
        SetDlgItemInt(hDlg, IDC_PUB_CITATIONS, 0, FALSE);
        SendMessageW(hType, CB_SETCURSEL, 0, 0);
      }
    }

    bool Validate(HWND hDlg) {
      wchar_t buf[512];
      GetDlgItemTextW(hDlg, IDC_PUB_TITLE, buf, 512);
      if (wcslen(buf) == 0) {
          MessageBoxW(hDlg, L"Vvedit nazvu publikatsii", L"Pomyłka", MB_OK |
MB_ICONWARNING);
        SetFocus(GetDlgItem(hDlg, IDC_PUB_TITLE));
        return false;
      }
      return true;
    }

    void SaveData(HWND hDlg) {
      wchar_t buf[1024];
      GetDlgItemTextW(hDlg, IDC_PUB_TITLE, buf, 1024); pub.title = buf;
      GetDlgItemTextW(hDlg, IDC_PUB_JOURNAL, buf, 512); pub.journal = buf;
      pub.year = GetDlgItemInt(hDlg, IDC_PUB_YEAR, nullptr, FALSE);
```

```
            pub.citationCount = GetDlgItemInt(hDlg, IDC_PUB_CITATIONS, nullptr,
FALSE);
        GetDlgItemTextW(hDlg, IDC_PUB_DOI, buf, 256); pub.doi = buf;

        HWND hType = GetDlgItem(hDlg, IDC_PUB_TYPE);
        int sel = (int)SendMessageW(hType, CB_GETCURSEL, 0, 0);
        if (sel != CB_ERR) {
            SendMessageW(hType, CB_GETLBTEXT, sel, (LPARAM)buf);
            pub.type = buf;
        }
    }
};
```