РІВНЕНСЬКИЙ ДЕРЖАВНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ

Факультет математики та інформатики
(повна назва факультету)

Кафедра інформатики та прикладної математики
(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри

_____ доц. Батишкіна Ю. В.
(підпис) (прізвище, ініціали)

«___»_____20___ р.

Дипломний проект (робота)

ступеня «Магістр»

з напряму підготовки (спеціальності) 014.09 – Середня освіта (Інформатика)

на тему: Створення автоматизованої інформаційної системи «Кафедра»

Виконав: студент II курсу, групи М-І-21

Табачук Роман Васильович _____
(прізвище, ім'я, по-батькові студента) (підпис)

Керівник: к.ф.-м.н., доц. Шахрайчук М. І. _____
(посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент: доц. Сяський В. А. _____
(посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент: доц. Батишкіна Ю. В. _____
(посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичених матеріалів з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Рівне – 2020 року

ЗМІСТ

СПИСОК УМОВНИХ ТЕРМІНІВ ТА ПОЗНАЧЕНЬ

БД – набір логічно пов’язаних даних та їх опис, які використовують сумісно для задоволення інформаційних потреб організацій.

SQL (англ. Structured Query Language) – мова структурованих запитів, яка застосовується для модифікації та управління даними. даними.

XAML – декларативна мова розмітки, призначена для спрощення створення інтерфейсу користувача програм .NET Framework.

XML – стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет.

JSON – текстовий формат обміну даними між комп’ютерами.

WPF (англ. Windows Presentation Foundation) – графічна (презентаційна) система .NET Framework, що має пряме відношення до XAML.

MVVM (Model-View-ViewModel) – шаблон проектування для відокремлення розробки графічного інтерфейсу від розробки бізнес логіки.

LINQ (англ. Language Integrated Query) – компонент .NET Framework, який додає нативні можливості виконання запитів даних до мов, що входять у .NET.

Excel Interop – технологія .NET Framework для взаємодії із табличним процесором Microsoft Excel.

Попап (англ. pop up – вискакувати) – спливаюче вікно.

ВСТУП

З кожним роком інформаційні технології займають все більше місця в нашому житті. Не залишається жодної сфери діяльності людини, де б не використовувались програмні засоби, створені для автоматизації різноманітних процесів, зокрема документообігу. При значному збільшенні обсягів інформації діловодство на паперових носіях скоротилось. Цього досягнуто за рахунок розвитку сучасних технологій роботи з документальною інформацією: застосування автоматизованого введення документів в комп'ютер; впровадження текстового і графічного видів обробки документів, що дає змогу просто й оперативно вносити в них зміни; застосування систем електронного документообігу та сучасних засобів оргтехніки; швидкого доступу до довідкової інформації через міжнародні бази даних та відповідні комп'ютерні мережі[1].

Серед державних структур, що активно використовують комп'ютерні програми для автоматизації документообігу є заклади вищої освіти. Робота з документами відіграє одну з найголовніших ролей в діяльності будь-якого ЗВО, тому кожен співробітник повинен вміти правильно використовувати ті чи інші програмні технології для створення та обробки документів. На даний момент найпоширенішими такими технологіями є Microsoft Word та Microsoft Excel. Це досить потужні інструментарії, які дають можливість швидко та легко маніпулювати різноманітними видами документів, однак вони містять досить загальний функціонал, що не дає можливості максимально автоматизувати спеціалізовану діяльність, пов'язану з документообігом. Для того, щоб додатково мінімізувати специфічні процеси потрібно створювати власні програмні засоби, що оптимізують ряд конкретних завдань працівників конкретної сфери діяльності.

Ця дипломна робота спрямована на те, щоб автоматизувати роботу співробітників кафедри, найбільш важливої структурної одиниці закладу вищої освіти. Однією з основних вимог до кафедри є організація навчального процесу, що у свою чергу, призводить до значних витрат часу на опрацювання

однотипних даних та виконання рутинних процесів з організації викладання предметів і курсів для груп студентів. Кафедра виконує значну кількість функцій однією з яких є розподіл навчального навантаження між викладацьким складом. Цей процес включає в себе формування, використання та затвердження навчальних планів спеціальностей, освітньо-кваліфікаційних характеристик та освітньо-професійних програм. На сьогоднішній день розподіл навчального навантаження працівників кафедри найчастіше виконується у ручному режимі та за допомогою електронних таблиць. Такий підхід не дає змогу якісно і повноцінно здійснювати контроль над розподілом годин по навантаженню кожного викладача та вимагає багато часу. З цієї причини виникає потреба у розробці автоматизованої інформаційної системи «Кафедра». Дана система дозволить звести до мінімуму число помилок при розподілі навчального навантаження між працівниками кафедри закладу вищої освіти та в цілому полегшить ряд трудомістких процесів, пов'язаних із розподілом навантаження.

Проблема даної дипломної роботи полягає у вивченні та розв'язанні наступних питань:

- Дослідження предметної області та уточнення вимог до програмного засобу.
- Проектування структури бази даних.
- Реалізація засобами об'єктно-орієнтованих технологій програмного засобу.

Тема. «Створення автоматизованої інформаційної системи «Кафедра»».

Система розрахована на те, щоб автоматизувати діяльність працівників кафедр ЗВО та мінімізувати число однотипних помилок, що виникають під час роботи з документами. Основною задачею роботи є автоматизоване генерування таблиці з навчальним навантаженням та швидкий доступ до інформації про викладачів і предметів, які вони викладають.

Актуальність. Функціонал програмного засобу спрямований на забезпечення швидкого доступу до бази даних, зручного маніпулювання нею, а

також автоматизоване створення документів. Ряд ефективних рішень, закладених в проект системи надає користувачеві можливість:

- Перегляду і редагування даних викладачів та списку предметів, які вони викладають.
- Додавання і видалення викладачів та предметів.
- Впорядкування та вибірки даних предметів за різноманітними критеріями.
- Імпорт предметів із файлів формату xls, xlsx, які містять в собі робочі навчальні плани.
- Автоматизованого створення документів на основі наданої користувачем інформації.

Об'єкт. Автоматизація навчального процесу у закладі вищої освіти.

Предмет. Програмний засіб дозволяє переглядати та редагувати інформацію, що зберігається в базі даних. Інтерфейс системи передбачає можливість швидкого пошуку потрібних даних, їх сортування та вибірку. А також автоматизованого створення документів на основі отриманої від користувача інформації.

Цілі роботи:

- Підібрати інструментарій, який дає можливість автоматизувати процеси аналізу предметної області для з'ясування функціональних вимог та проектування програмного виробу.
- Розробити матеріали, для використання у навчальному процесі.
- Використовуючи середовище Visual Studio .NET створити базу даних та розробити програмний засіб для маніпулювання нею.

Завдання. Для досягнення поставлених цілей потрібно:

- Розробити вимоги до програмного виробу.
- Спроектувати та розробити базу даних.
- Навчитися оперувати базою даних через програмний застосунок.
- Навчитися проектувати і розробляти інтерфейс користувача з використанням об'єктно-орієнтованих технологій.

- Реалізувати засобами об’єктно-орієнтованих технологій, зокрема WPF, Excel Interop, програмний засіб.
- Навчитися оперувати елементами інтерфейсу через код програмної частини.

Гіпотеза. Створення бази даних та практична реалізація додатка для маніпулювання нею значно скоротить час користувачів на пошук та дослідження потрібної інформації та зведе до мінімуму кількість помилок. Автоматизована генерація документів значно знизить шанс потрапляння до них хибних даних, так як процес генерації потребує мінімального втручання користувача.

Апробація результатів магістерської роботи. Основні результати роботи доповідалися та обговорювалися на XIII науково-практичній конференції «Інформаційні технології в професійній діяльності», що проводилась 18 листопада 2020 року.

Обсяг роботи. Дипломна робота складається зі вступу, трьох розділів, висновку та списку використаних джерел. Повний обсяг роботи складає 43 сторінки машинного тексту, в тому числі 17 рисунків.

РОЗДІЛ 1. Постановка завдання

1.1. Область застосування

Автоматизована інформаційна система «Кафедра» була створена для використання працівниками кафедр Рівненського державного гуманітарного університету. Однак, за умови, що система покаже ефективний результат, можливе впровадження системи і в інших ЗВО в подальшому.

За допомогою програмного застосунку та набору JSON файлів, які формують собою базу даних, співробітники кафедр матимуть змогу зручно та швидко переглядати інформацію про предмети та викладачів, редагувати їх, а також здійснювати контроль над розподілом навчального навантаження і створювати електронні документи, пов’язані з розподілом.

Ця інформаційна система повинна звести до мінімуму число однотипних помилок, які допускаються співробітниками кафедри під час виконання їх основних задач. Поставлена мета має бути досягнута за рахунок автоматизації процесів створення електронних документів, а також спеціальних правил валідації даних, які захистять систему від надходження хибної інформації.

Варто зазначити, що не останню роль відіграє інтерфейс користувача програми. Адже навіть добре написаний застосунок буде абсолютно неефективним, якщо його інтерфейс є незручним і незрозумілим для користувача. Тому в нашій інформаційній системі велику увагу приділено графічній верстці макетів. Завдяки добре продуманому розташуванню елементів користувач може швидко ознайомитись з основним функціоналом системи та не витрачати зайвий час на пошук потрібних інструментів.

Завдяки максимальній оптимізації програмного застосунку та використання сучасних технологій для обробки документів Excel, роботи з файловою системою, серіалізації та десеріалізації даних, користувачі матимуть змогу здійснювати основні операції максимально швидко та без найменших затримок.

1.2. Дослідження існуючого програмного забезпечення

Серед вже існуючих програмних додатків, призначених для використання на кафедрах ЗВО, варто зазначити пакет програм «Деканат» для закладів вищої освіти III-IV рівнів акредитації розроблений приватним підприємством «Політек-СОФТ». Цей пакет дозволяє створити та підтримувати базу даних, в якій реєструється та формується така інформація:

- структура навчального процесу закладу вищої освіти (факультети, кафедри, спеціальності, спеціалізації, навчальні плани, академічні та збірні групи, підгрупи, лекційні потоки);
- дані щодо навантаження кафедр (з генерацією звіту за Ф. У-4.01);
- результати обрахунку штатів кафедр;
- дані щодо всіх викладачів закладу вищої освіти та їх планового навантаження, розклад їх роботи;
- дані щодо всіх студентів закладу вищої освіти та подій, що відбуваються з ними під час навчання (оцінки, відвідування занять, рух студентів);
- аудиторний фонд закладу вищої освіти, його використання, розклад занять[2].

З вище описаних пунктів можна зробити висновок, що пакет є багатофункціональним та допомагає вирішити основні проблеми працівників кафедри. Однак попри всі його переваги він є платним. Розробники не надають можливості обмеженого тестування функціоналу продукту і це є значним недоліком, адже після ознайомлення з програмою може виявитись, що вона не задовольняє потреб користувача і кошти витрачені дарма.

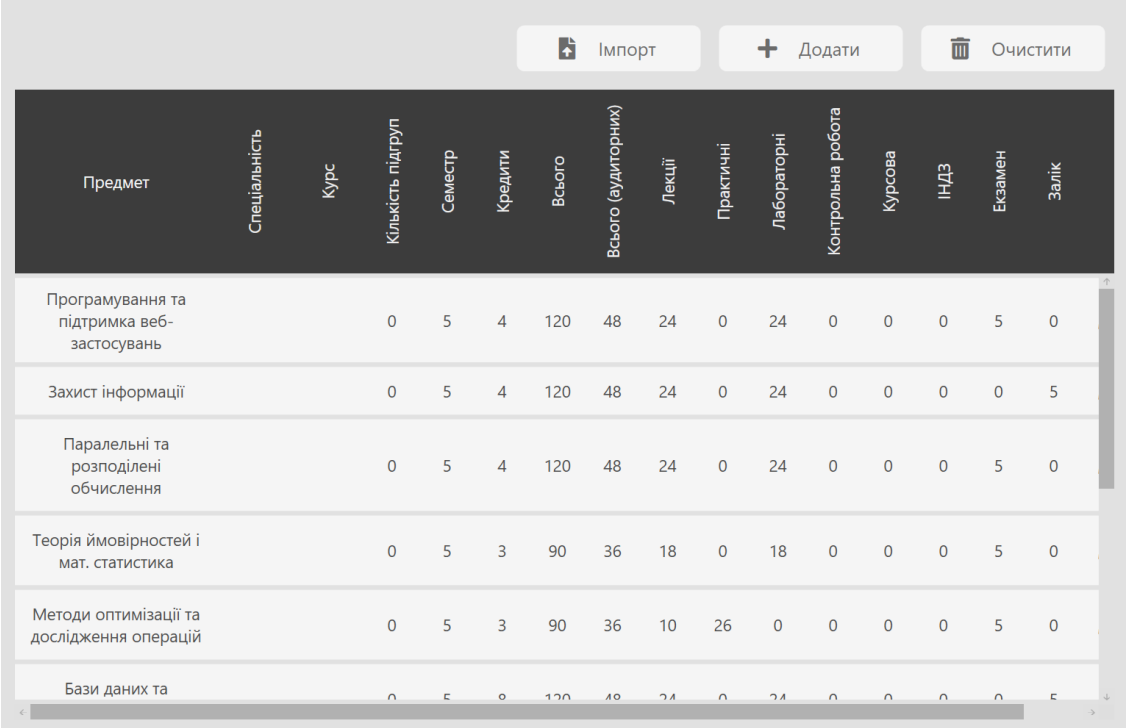
Отже, раціонально створити програмний застосунок, який допоміг би працівникам кафедри у вирішенні їх задач і при цьому був абсолютно безкоштовний та знаходився в загальному доступі.

1.3. Характеристика розроблених модулів

В автоматизованій інформаційній системі «Кафедра» функціонують наступні модулі:

- **Модуль предметів.**

Модуль предметів представляє собою таблицю, яка містить дані про всі предмети в базі даних (рис. 1.3.1). Користувач має змогу додавати предмети вручну або ж імпортувати їх з XLS файлів. Окрім цього наявна можливість редагування та видалення даних. За потреби можна повністю очистити список предметів за допомогою кнопки «Очистити».



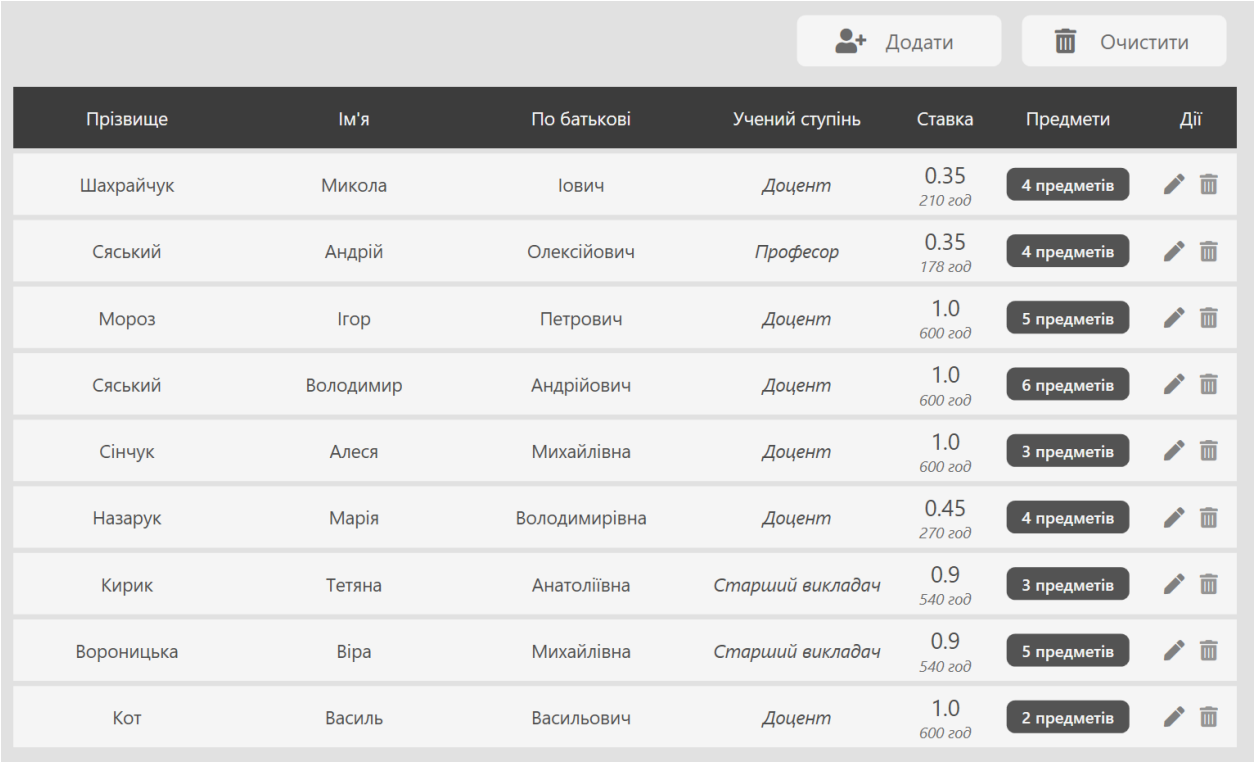
Предмет	Спеціальність	Курс	Кількість підгруп	Семестр	Кредити	Всього	Всього (аудиторних)	Лекції	Практичні	Лабораторні	Контрольна робота	Курсова	ІНДЗ	Екзамен	Залік
Програмування та підтримка веб-застосувань			0	5	4	120	48	24	0	24	0	0	0	5	0
Захист інформації			0	5	4	120	48	24	0	24	0	0	0	0	5
Паралельні та розподілені обчислення			0	5	4	120	48	24	0	24	0	0	0	5	0
Теорія ймовірностей і мат. статистика			0	5	3	90	36	18	0	18	0	0	0	5	0
Методи оптимізації та дослідження операцій			0	5	3	90	36	10	26	0	0	0	0	5	0
Бази даних та			0	5	8	120	48	24	0	24	0	0	0	0	5

Рис.1.3.1. Модуль предметів.

З метою оптимізації в таблицю завантажується лише обмежене число предметів. Інші дані завантажуються по мірі прокручування таблиці вниз. При цьому користувачу не потрібно очікувати появу нової інформації на екрані, процес відбувається миттєво.

- **Модуль викладачів.**

Модуль викладачів є схожим на модуль предметів і також представляє собою таблицю, однак дещо іншої структури (рис. 1.3.2). Окрім текстових елементів таблиця містить кнопки, що зазначають число предметів, на яких спеціалізується викладач. При натисненні на кнопку відкривається нове вікно, в якому користувач має змогу змінити список предметів.



Прізвище	Ім'я	По батькові	Учений ступінь	Ставка	Предмети	Дії
Шахрайчук	Микола	Іович	*Доцент*	0.35 *210 год*	4 предметів	
Сяський	Андрій	Олексійович	*Професор*	0.35 *178 год*	4 предметів	
Мороз	Ігор	Петрович	*Доцент*	1.0 *600 год*	5 предметів	
Сяський	Володимир	Андрійович	*Доцент*	1.0 *600 год*	6 предметів	
Сінчук	Алеся	Михайлівна	*Доцент*	1.0 *600 год*	3 предметів	
Назарук	Марія	Володимирівна	*Доцент*	0.45 *270 год*	4 предметів	
Кирик	Тетяна	Анатоліївна	*Старший викладач*	0.9 *540 год*	3 предметів	
Вороницька	Віра	Михайлівна	*Старший викладач*	0.9 *540 год*	5 предметів	
Кот	Василь	Васильович	*Доцент*	1.0 *600 год*	2 предметів	

Рис.1.3.2. Модуль викладачів.

- **Модуль учених ступенів.**

Модуль учених ступенів містить в собі дані про максимальне число годин (навантаження) для окремого вченого ступеня (рис. 1.3.3). Наявна функція редагування максимального числа годин. Додатково користувач може переглянути який учений ступінь має той чи інший викладач.

Викладач	Старший викладач	Доцент	Професор
Максимальна кількість годин: 600	*Максимальна кількість годин:* 600	*Максимальна кількість годин:* 600	*Максимальна кількість годин:* 510
	Кирик Т. А.	Шахрайчук М. І.	Сяський А. О.
	Вороницька В. М.	Мороз І. П.	
		Сяський В. А.	
		Сінчук А. М.	
		Назарук М. В.	
		Кот В. В.	

Рис. 1.3.3. Модуль учених ступенів.

- **Модуль норм часу.**

Модуль норм часу містить в собі норми часу для планування й обліку навчальної роботи (рис. 1.3.4). Як і в модулі учених ступенів, користувач має змогу змінювати кількість годин для того чи іншого виду навчальної роботи.

Вид роботи	Норма в годинах	Вид розподілу
Консультація	2	на групу
Залік	2	на групу
Екзамен	3	на групу
Контрольна робота	0,1	на студента
Курсова робота	3	на студента
Бакалаврська робота	20	на студента
Магістерська робота	28	на студента

Рис. 1.3.4. Модуль норм часу.

- **Модуль розподілу навантаження.**

Модуль розподілу навантаження представляє собою список викладачів та 2 таблиці: із нерозподіленим навантаженням та навантаженням вибраного викладача (рис. 1.3.5). Користувач має змогу формувати навантаження автоматично або присвоювати навантаження викладачеві в ручному режимі.

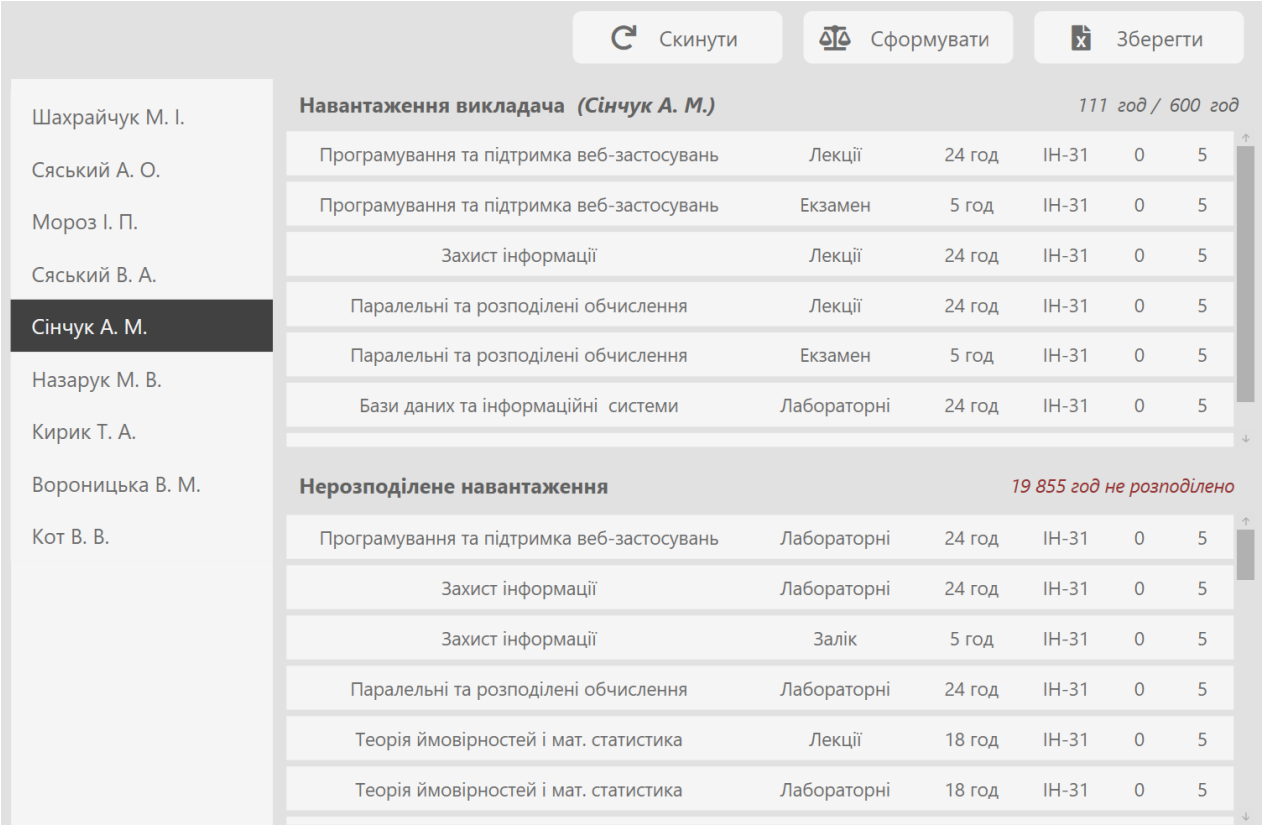


Рис. 1.3.5. Модуль розподілу навантаження.

РОЗДІЛ 2. Програмна реалізація

2.1. Вибір технологій реалізації

У розробці будь-якого програмного застосунку важливим є вибір технологій реалізації. Потрібно детально розглянути кожен із можливих варіантів та вибрати саме той, який найкраще підійде для вирішення поставлених завдань. У результаті неправильного вибору технологій розробники можуть зіткнутись із серйозними проблемами під час подальших етапів розробки. Технологія може виявитись «сирою», містити критичні баги, конфліктувати з іншими технологіями або ж просто не підходити для реалізації певних речей. Тому до цього етапу потрібно відноситись особливо уважно і обирати вже перевірені технології.

Для реалізації автоматизованої інформаційної системи «Кафедра» була обрана мова програмування C#, так як вона є досить гнучкою і містить багато готових рішень. На сьогоднішній момент мова програмування C# є однією з найпотужніших мов, яка швидко розвивається і найбільш затребувана мова в IT-галузі. На даний момент на ній пишуться найрізноманітніші програми: від невеликих десктопних програм до великих веб-порталів і веб-сервісів, які обслуговують щодня мільйони користувачів.

C# вже не молода мова. І C# і вся платформа .NET вже пройшли великий шлях. Перша версія мови вийшла разом з релізом Microsoft Visual Studio .NET в лютому 2002 року. Поточною версією мови є версія C# 9.0, яка вийшла 10 листопада 2020 року разом з релізом .NET 5. C# є мовою з Сі-подібним синтаксисом і близька в цьому відношенні до C++ і Java.

C# є об’єктно-орієнтованою і в цьому плані багато запозичила у Java і C++. Наприклад, C# підтримує поліморфізм, спадкування, перевантаження операторів, статичну типізацію. Об’єктно-орієнтований підхід дозволяє вирішити завдання з побудови великих, але в той же час гнучких, масштабованих і розширюваних додатків. C# продовжує активно розвиватися, і з кожною новою

версією з'являється все більше цікавих функціональностей, як, наприклад, лямбда-вирази, динамічне зв'язування, асинхронні методи і т. д.

Коли кажуть C#, нерідко мають на увазі технології платформи .NET (Windows Forms, WPF, ASP.NET, Xamarin). І, навпаки, коли кажуть .NET, нерідко мають на увазі C#. Однак, хоча ці поняття пов'язані, ототожнювати їх неправильно. Мова C# була створена спеціально для роботи з фреймворком .NET, проте саме поняття .NET дещо ширше.

Якось Білл Гейтс сказав, що платформа .NET – це найкраще, що створила компанія Microsoft. Можливо, він мав рацію. Фреймворк .NET представляє потужну платформу для створення додатків. Можна виділити наступні її основні риси:

- **Підтримка декількох мов**

Основою платформи є загальномовне середовище виконання Common Language Runtime (CLR), завдяки чому .NET підтримує декілька мов: поряд з C# це також VB.NET, C++, F#, а також різні діалекти інших мов, прив'язані до .NET, наприклад, Delphi .NET. При компіляції код на будь-якій з цих мов компілюється в збірку спільною мовою CIL (Common Intermediate Language) – свого роду асемблер платформи .NET. Тому за певних умов ми можемо зробити окремі модулі однієї програми на окремих мовах.

- **Кросплатформеність**

.NET є платформою, яку можна переносити (з деякими обмеженнями). Наприклад, остання версія платформи на даний момент – .NET 5 підтримується на більшості сучасних ОС Windows, MacOS, Linux. Використовуючи різні технології на платформі .NET, можна розробляти програми на мові C# для різних платформ – Windows, MacOS, Linux, Android, iOS, Tizen.

- **Потужна бібліотека класів**

.NET представляє єдину для всіх підтримуваних мов бібліотеку класів. І який би додаток ми не збиралися писати на C# – текстовий редактор, чат або складний веб-сайт – так чи інакше ми задіємо бібліотеку класів .NET.

- **Різноманітність технологій**

Загальномовне середовище виконання CLR і базова бібліотека класів є основою для цілого стека технологій, які розробники можуть задіяти при побудові тих чи інших додатків. Наприклад, для роботи з базами даних в цьому стеку технологій призначена технологія ADO.NET і Entity Framework Core. Для побудови графічних додатків з багатим насиченим інтерфейсом – технологія WPF і UWP, для створення простіших графічних додатків – Windows Forms. Для розробки мобільних додатків – Xamarin. Для створення веб-сайтів і веб-додатків – ASP.NET і т. д.

До цього варто додати фреймворк Blazor, який працює поверх .NET і дозволяє створювати веб-додатки як на стороні сервера, так і на стороні клієнта. А в майбутньому буде підтримувати створення мобільних додатків і, можливо, десктоп-додатків.

- **Продуктивність.**

Згідно ряду тестів веб-додатки на .NET 5 в ряді категорій сильно випереджають веб-додатки, побудовані за допомогою інших технологій. Додатки на .NET 5 в принципі відрізняються високою продуктивністю.

Також ще треба відзначити таку особливість мови C# і фреймворка .NET, як автоматичне прибирання сміття. А це означає, що нам в більшості випадків не доведеться, на відміну від C++, піклуватися про звільнення пам’яті. Вищезазначене загальномовне середовище CLR саме викличе збирач сміття і очистить пам’ять[3].

В якості підсистеми для створення графічного десктоп-додатка була обрана технологія WPF (Windows Presentation Foundation) (рис. 2.1.1). На відміну від технології Windows Forms, WPF є сучаснішою та має набагато більше корисних особливостей. Наприклад, графічний інтерфейс у WPF можна описувати за допомогою мови розмітки XAML, що дає змогу відділити його від логіки додатка. Якщо при створенні традиційних додатків на основі Windows Forms за рендеринг елементів управління і графіки відповідали такі частини ОС Windows, як User32 і GDI+, то додатки WPF засновані на DirectX. У цьому

полягає ключова особливість рендерингу графіки у WPF: використовуючи WPF, значна частина роботи з відображення графіки, як найпростіших кнопок, так і складних 3D-моделей, лягає на графічний процесор на відеокарті, що також дозволяє скористатися апаратним прискоренням графіки.

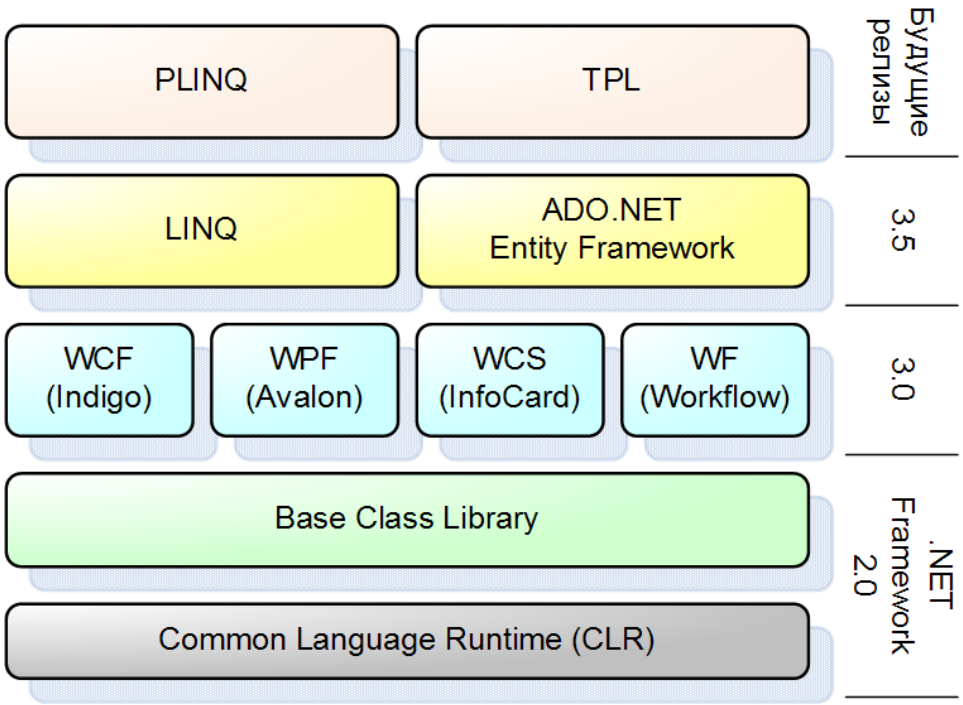


Рис.2.1.1. Технологія WPF в складі платформи .NET [4].

Однією з важливих особливостей є використання мови декларативної розмітки інтерфейсу XAML, заснованої на XML: можна створювати насичений графічний інтерфейс, використовуючи або декларативне оголошення інтерфейсу, або код на керованих мовах C # і VB.NET, або поєднувати і те, й інше.

Серед основних переваг WPF можна виділити:

- Використання традиційних мов .NET-платформи – C# і VB.NET для створення логіки додатка.
- Можливість декларативного визначення графічного інтерфейсу за допомогою спеціальної мови розмітки XAML, заснованій на XML. Ця мова представляє альтернативу програмному створенню графіки та елементів управління, а також можливість комбінувати XAML і C#/VB.NET.

- Незалежність від розмірів екрану: бо у WPF всі елементи вимірюються в незалежних від пристрою одиницях, додатки на WPF легко масштабуються під різні екрани з різним розміром.
- Нові можливості, яких складно було досягти у Windows Forms, наприклад, створення тривимірних моделей, прив'язка даних, використання таких елементів, як стилі, шаблони, теми та ін.
- Хороша взаємодія з Windows Forms, завдяки чому, наприклад, в додатках WPF можна використовувати традиційні елементи управління з Windows Forms.
- Багаті можливості зі створення різних додатків: це і мультимедіа, і двомірна і тривимірна графіка, і багатий набір вбудованих елементів управління, а також можливість самим створювати нові елементи, створення анімацій, прив'язка даних, стилі, шаблони, теми і багато іншого.
- Апаратне прискорення графіки – незалежно від того, чи ми працюємо з 2D або 3D, графікою або текстом, всі компоненти програми транслюються в об'єкти, зрозумілі Direct3D, і потім візуалізуються за допомогою процесора на відеокарті, що підвищує продуктивність, робить графіку більш плавною.
- Створення додатків під множину ОС сімейства Windows – від Windows XP до Windows 10.

У той же час WPF має певні обмеження. Незважаючи на підтримку тривимірної візуалізації, для створення додатків з великим числом тривимірних зображень, перш за все ігор, краще використовувати інші засоби – DirectX або спеціальні фреймворки, такі як Monogame або Unity.

Також варто враховувати, що в порівнянні з додатками на Windows Forms обсяг програм на WPF і споживання ними пам'яті в процесі роботи в середньому трохи вище. Але це з лишком компенсується більш широкими графічними можливостями і підвищеною продуктивністю під час відображення графіки[5].

Під час проектування архітектури додатка був використаний патерн MVVM (Model-View-ViewModel). Цей патерн дозволяє відокремити логіку додатка від візуальної частини (подання). Він є архітектурним, тобто він задає загальну архітектуру програми.

Патерн MVVM був представлений Джоном Госманом (John Gossman) в 2005 році як модифікація шаблону Presentation Model і був спочатку спрямований на розробку додатків в WPF. І хоча зараз цей патерн вийшов за межі WPF і застосовується в різних технологіях, в тому числі при розробці під Android, iOS, проте WPF є досить показовою технологією, яка розкриває можливості цього патерну.

MVVM складається з трьох компонентів: моделі (Model), моделі представлення (ViewModel) і представлення (View) (рис. 2.1.2):

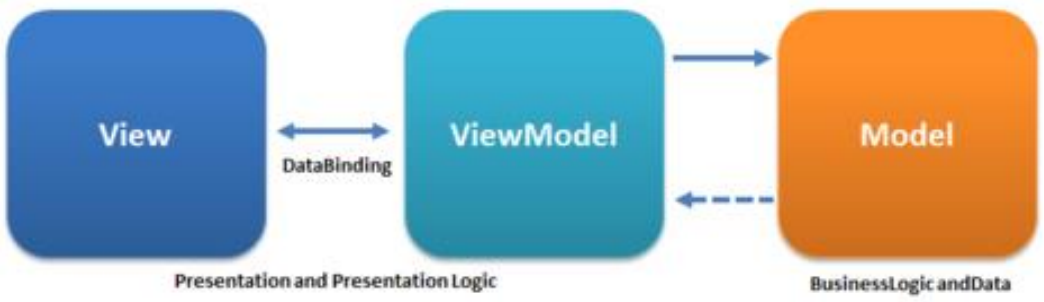


Рис.2.1.2. Структура патерна MVVM [6].

- **Model**

Модель описує використовувані в додатку дані. Моделі можуть містити логіку, безпосередньо пов’язану з цими даними, наприклад, логіку валідації властивостей моделі. У той же час модель не повинна містити ніякої логіки, пов’язаної з відображенням даних і взаємодією з візуальними елементами управління.

Нерідко модель реалізує інтерфейси INotifyPropertyChanged або INotifyCollectionChanged, які дозволяють повідомляти систему про зміни

властивостей моделі. Завдяки цьому полегшується прив'язка до подання, хоча знову ж пряма взаємодія між моделлю і представленням відсутня.

- **View**

View або представлення визначає візуальний інтерфейс, через який користувач взаємодіє з додатком. Стосовно WPF, представлення – це код у XAML, який визначає інтерфейс у вигляді кнопок, текстових полів та інших візуальних елементів.

Хоча вікно (клас Window) у WPF може містити як інтерфейс в XAML, так і прив'язаний до нього код C#, проте в ідеалі код C# не повинен містити якоїсь логіки, крім хіба що конструктора, який викликає метод InitializeComponent і виконує початкову ініціалізацію вікна. Вся ж основна логіка додатка виноситься в компонент ViewModel.

Однак іноді у файлі пов'язаного коду все ж може знаходитися певна логіка, яку важко реалізувати в рамках патерна MVVM у ViewModel. Представлення не виконує жодних дій за рідкісним винятком, а виконує дії в основному за допомогою команд.

- **ViewModel**

ViewModel або модель представлення пов'язує модель і представлення через механізм прив'язки даних. Якщо в моделі змінюються значення властивостей, при реалізації моделлю інтерфейсу INotifyPropertyChanged автоматично йде зміна відображуваних даних в представленні, хоча безпосередньо модель і представлення не пов'язані. ViewModel також містить логіку для отримання даних з моделі, які потім передаються в представлення. Також ViewModel визначає логіку для оновлення даних в моделі.

Через те що елементи представлення, тобто візуальні компоненти типу кнопок, не використовують події, то представлення взаємодіє з ViewModel за допомогою команд. Наприклад, користувач хоче зберегти введені в текстове поле дані. Він натискає на кнопку і тим самим відправляє команду до ViewModel. А ViewModel вже отримує передані дані і відповідно до них оновлює модель.

Підсумком застосування патерну MVVM є функціональний розподіл програми на три компонента, які простіше розробляти і тестувати, а також в подальшому модифікувати і підтримувати[7].

Для зберігання даних було вирішено використовувати набір файлів формату JSON. Для цього був написаний спеціальний функціонал для швидкого зчитування та запису даних. JSON – це текстовий формат обміну даними між комп'ютерами (рис. 2.1.3). JSON базується на тексті, може бути прочитаним людиною. Формат дає змогу описувати об'єкти та інші структури даних. Цей формат використовується переважно для передачі структурованої інформації через мережу (завдяки процесу, який називають серіалізацією).

```
1   [
2     {
3       "Id": "6f521d51-2de4-44c3-8799-53301ffa1509",
4       "LastName": "Шахрайчук",
5       "FirstName": "Микола",
6       "MiddleName": "Іович",
7       "AcademicStatus": 2,
8       "Rate": 0.35,
9       "SubjectsSpecializesIn": [
10        "Візуальне програмування",
11        "Додаткові розділи системного програмування",
12        "Обєктно-зорієнтований аналіз та проектування",
13        "Програмне забезпечення обчислювальних систем"
14      ]
15    },
16    {
```

Рис.2.1.3. Приклад даних в форматі JSON.

Розробив і популяризував формат Дуглас Крокфорд. JSON знайшов своє головне призначення в написанні веб-програм, а саме при використанні технології AJAX. JSON, що використовується в AJAX, виступає як заміна XML (використовується в AJAX) під час асинхронної передачі структурованої інформації між клієнтом та сервером. При цьому перевагою JSON перед XML є

те, що він дозволяє складні структури в атрибутах, займає менше місця і прямо інтерпретується за допомогою JavaScript в об’єкти[8]. Цей спосіб взаємодії з даними є гарною альтернативою повноцінній базі даних, наприклад SQL, так як не потребує встановлення додаткових інструментів для керування БД. Отже користувач може самостійно встановити програмний застосунок «Кафедра», не звертаючись за допомогою до програміста. Перенесення даних з одного комп’ютера на інший передбачає собою просту заміну однієї папки. Так само просто можна маніпулювати резервними копіями даних.

Як інструмент взаємодії із табличним процесором Excel було вирішено використати бібліотеку Excel Interop від Microsoft. Ця бібліотека дозволяє виконувати швидке зчитування та запис даних Excel. Окрім цього в Excel Interop доступне застосування формул та зміна стильового оформлення таблиці.

2.2. Розробка графічного інтерфейсу

Графічний інтерфейс відіграє значну роль в будь-якій сучасній програмі, бо навіть найпотужніший програмний застосунок з погано проробленим інтерфейсом є абсолютно безкорисним. Тому одним з основних завдань на етапі розробки програми є проектування зручного дизайну, який без жодних труднощів міг би опанувати навіть недосвідчений користувач.

Інтерфейс має коректно відображатись при будь-якому розмірі монітора. Він повинен бути гнучким та підлаштовуватись під масштаби вікна. Текст має бути читабельним, а елементи не повинні налазити один на одного. Важливу роль грає стильове оформлення. Користувач краще оцінить програмний продукт, якщо він буде мати приємний дизайн.

Графічний інтерфейс інформаційної системи «Кафедра» був повністю написаний на мові розмітки XAML. Як допоміжна бібліотека використовувався Material Design In XAML Toolkit. Ця бібліотека містить в собі ряд вже готових рішень для розробки графічного інтерфейсу. Це допомагає значно оживити

додаток та надати йому певного стильового оформлення без допомоги дизайнера.

Раціональним є використання в програмі графічних іконок, так як вони не лише роблять інтерфейс привабливішим, а й дозволяють користувачу інтуїтивно зрозуміти призначення того чи іншого елемента дизайну. В системі «Кафедра» був використаний пакет Font Awesome 5 (рис. 2.2.1). Цей пакет містить 1609 безкоштовних іконок. Всі іконки мають векторний формат, що дозволяє розтягувати їх до будь-якого можливого розміру без втрати якості.

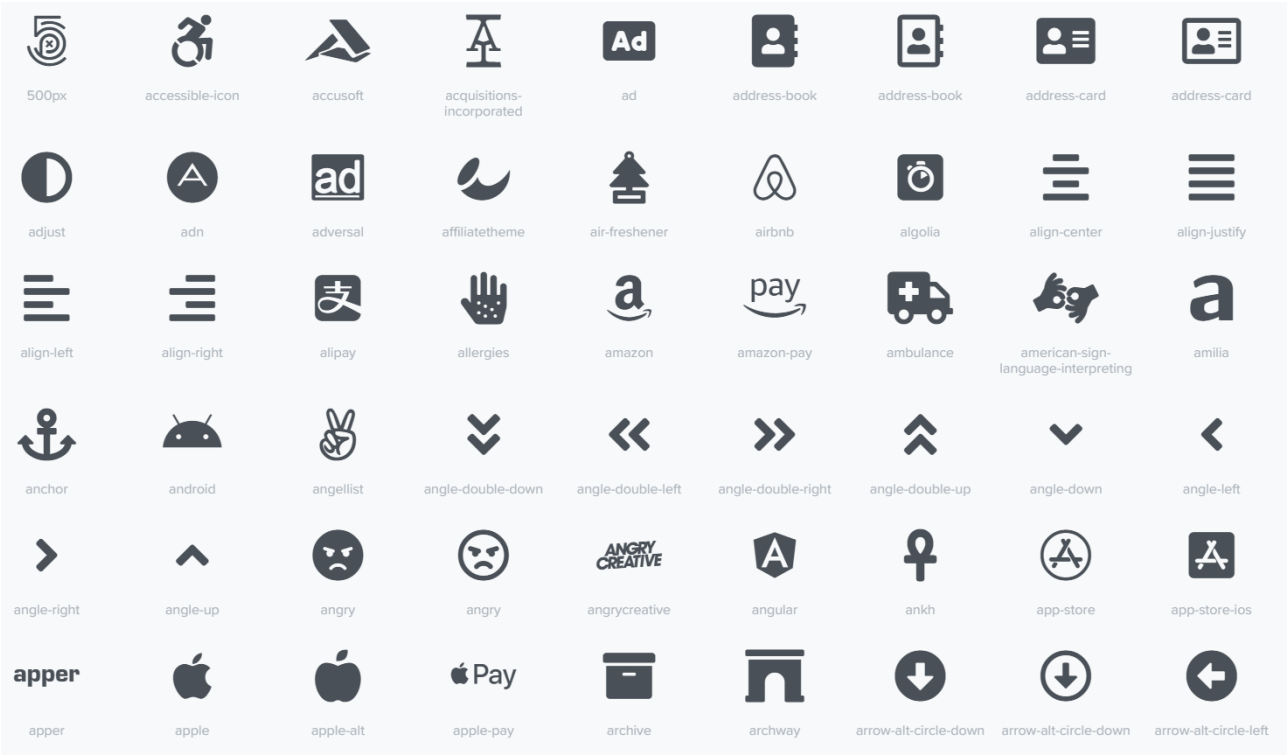


Рис.2.2.1. Іконки Font Awesome 5 [9].

Ініціалізація інформаційної системи «Кафедра» може зайняти певний час. Тривалість залежить від обсягу даних, які мають бути зчитані та завантажені в систему. Під час цього процесу було вирішено відображати прелоадер – невелике напівпрозоре вікно в центрі екрану з назвою програми та анімованим прогрес-баром (рис. 2.2.2). Так користувач розуміє, що виконуються певні процеси підготовки до роботи і невдовзі програма буде запущена. При потребі користувач має змогу пересунути вікно в інше місце на екрані або згорнути його.

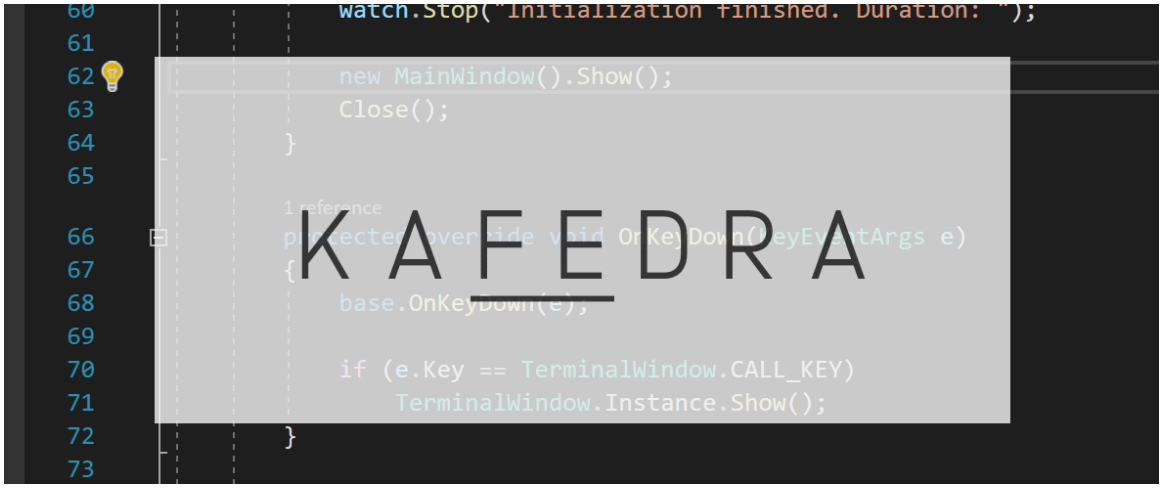


Рис.2.2.2. Прелоадер.

Реалізація прелоадера на мові XAML:

```
<Window
    xmlns:local="clr-namespace:KafedraApp.Windows"
    x:Class="KafedraApp.Windows.PreloaderWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
    mc:Ignorable="d"
    WindowStyle="None"
    Height="300" Width="700"
    Background="Transparent"
    AllowsTransparency="True"
    ResizeMode="NoResize"
    Icon="{DynamicResource Icon}"
    MouseDown="MovePreloader">
    <Grid>
        <Rectangle
            Fill="{DynamicResource BackgroundColor}"
            Opacity="0.8"/>
        <StackPanel VerticalAlignment="Center" Margin="115,0">
            <Label
                x:Name="TitleLbl"
                Content="K A F E D R A"
                FontSize="80"
                Foreground="{DynamicResource ForegroundColor}"
                Opacity="0.9"
                Padding="0"
                HorizontalAlignment="Center"
```

```
                HorizontalContentAlignment="Center"
                FontFamily="/KafedraApp;component/Resources/
                Fonts/#Anodina Light"/>
            <ProgressBar
                Foreground="{DynamicResource ForegroundColor}"
                HorizontalAlignment="Center"
                Width="{Binding ElementName=TitleLbl, Path=ActualWidth}"
                Background="Transparent"
                Opacity="0.9"
                Height="5"
                BorderThickness="0"
                IsIndeterminate="True">
            </ProgressBar>
        </StackPanel>
    </Grid>
</Window>
```

Після закінчення ініціалізації відкривається головне вікно програми, зліва якого розташовується меню, а справа контент обраної закладки (рис. 2.2.3). Для поліпшення читабельності коду та запобігання його роздуванню код основних елементів головного вікна був винесений в окремі файли проекту.

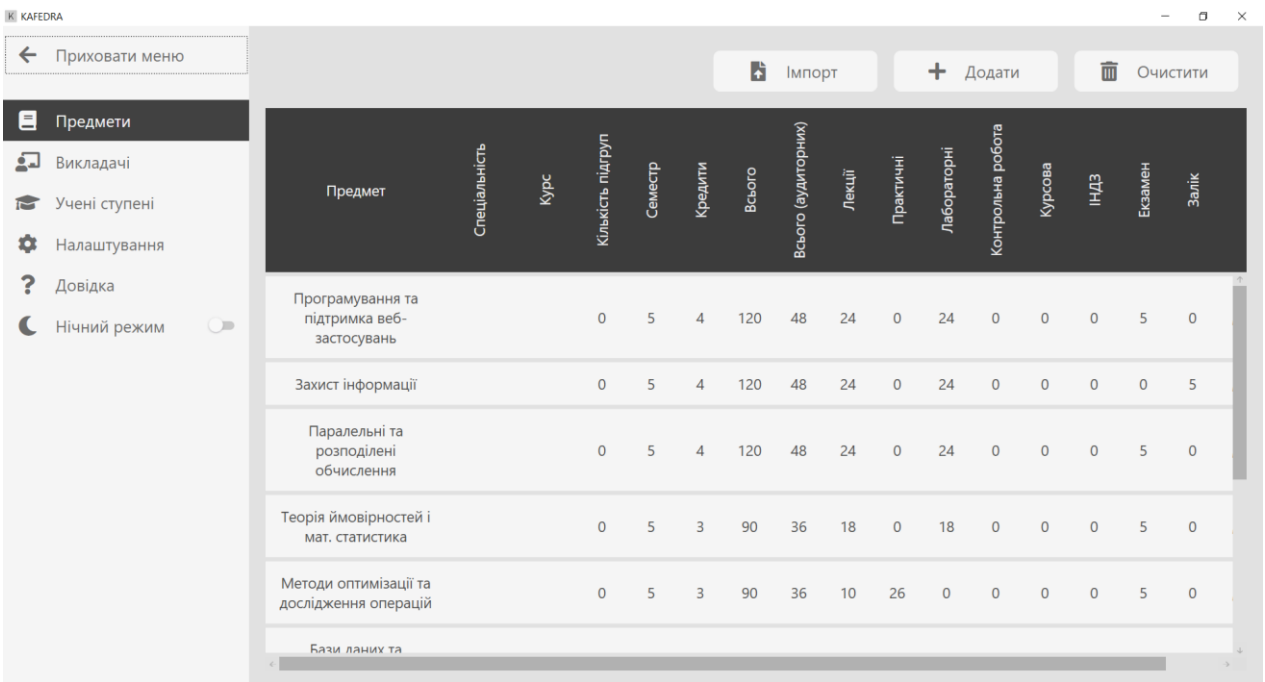


Рис. 2.2.3. Головне вікно додатка.

Контент вікна змінюється відповідно до обраної користувачем закладки. Ця особливість реалізована за допомогою елемента ContentControl, який містить

в собі ряд тригерів даних, що відображають той чи інший контент в залежності від значення властивості CurrentSection перелічувального типу. Коли користувач натискає на бажану закладку, значення CurrentSection змінюється і ContentControl автоматично відображає відповідний елемент. Отже для додавання нової закладки достатньо лише описати новий тригер. Це дозволяє уникнути клонування коду та скоротити тривалість розробки додатка.

Реалізація головного вікна на мові XAML:

```
<Window
    x:Class="KafedraApp.Windows.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
    xmlns:local="clr-namespace:KafedraApp.Windows"
    xmlns:views="clr-namespace:KafedraApp.Views"
    xmlns:viewModels="clr-namespace:KafedraApp.ViewModels"
    mc:Ignorable="d"
    WindowState="Maximized"
    Icon="{DynamicResource Icon}"
    Background="{DynamicResource DarkBackgroundColor}"
    Width="1080" Height="720"
    MinWidth="800" MinHeight="500"
    Title="KAFEDRA">

    <Window.Resources>
        <DataTemplate x:Key="SubjectsSection" DataType="{x:Type
views:SubjectsView}">
            <views:SubjectsView/>
        </DataTemplate>
        <DataTemplate x:Key="TeachersSection" DataType="{x:Type
views:TeachersView}">
            <views:TeachersView/>
        </DataTemplate>
        <DataTemplate x:Key="AcademicStatusesSection" DataType="{x:Type
views:AcademicStatusesView}">
            <views:AcademicStatusesView/>
        </DataTemplate>
        <DataTemplate x:Key="SettingsSection" DataType="{x:Type
views:SettingsView}">
            <views:SettingsView/>
        </DataTemplate>
        <DataTemplate x:Key="HelpSection" DataType="{x:Type
views:HelpView}">
            <views:HelpView/>
```

```
        </DataTemplate>
    </Window.Resources>
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <views:SideBarView Background="{DynamicResource
BackgroundColor}"/>
        <ContentControl
            Grid.Column="1"
            Content="{Binding }"
            Margin="20,15">
            <ContentControl.Style>
                <Style TargetType="{x:Type ContentControl}">
                    <Setter Property="ContentTemplate"
Value="{StaticResource SubjectsSection}" />
                    <Style.Triggers>
                        <DataTrigger Binding="{Binding CurrentSection}"
Value="{x:Static viewModels:MainViewModel+Sections.Teachers}">
                            <Setter Property="ContentTemplate"
Value="{StaticResource TeachersSection}" />
                        </DataTrigger>
                        <DataTrigger Binding="{Binding CurrentSection}"
Value="{x:Static viewModels:MainViewModel+Sections.AcademicStatuses}">
                            <Setter Property="ContentTemplate"
Value="{StaticResource AcademicStatusesSection}" />
                        </DataTrigger>
                        <DataTrigger Binding="{Binding CurrentSection}"
Value="{x:Static viewModels:MainViewModel+Sections.Settings}">
                            <Setter Property="ContentTemplate"
Value="{StaticResource SettingsSection}" />
                        </DataTrigger>
                        <DataTrigger Binding="{Binding CurrentSection}"
Value="{x:Static viewModels:MainViewModel+Sections.Help}">
                            <Setter Property="ContentTemplate"
Value="{StaticResource HelpSection}" />
                        </DataTrigger>
                    </Style.Triggers>
                </Style>
            </ContentControl.Style>
        </ContentControl>
    </Grid>
</Window>
```

За потреби користувач може приховати меню, натиснувши на відповідну кнопку зверху. Після натиснення спрацьовує анімація звуження меню до ширини іконок. Також в додатку реалізована зміна кольорової схеми. Ця функція

реалізована за допомогою динамічних ресурсів, які представляють собою кольори з унікальними ключами. Кожен файл з окремою кольоровою схемою містить набір кольорів з однаковими ключами, але різними значеннями. Отже розробнику достатньо лише змінити назву обраної теми, а кольорова схема зміниться автоматично. Це також дозволяє швидко додавати нові кольорові схеми або видаляти вже існуючі.

Ще однією особливістю інформаційної системи «Кафедра» є реалізація власного сервісу діалогових вікон. Замість використання класу System.Windows.MessageBox було вирішено відображати власноруч створені попапи всередині головного вікна (рис. 2.2.4). Дане рішення допомогло б зберегти оригінальний стиль додатка та реалізувати додаткові функції, які недоступні з використанням System.Windows.MessageBox.

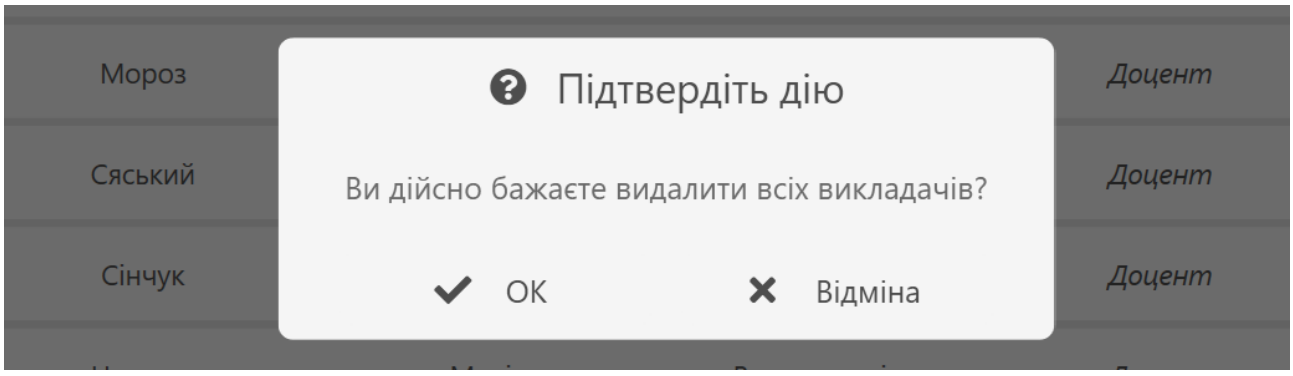


Рис. 2.2.4. Попап з запитом про підтвердження дії.

2.3. Збереження даних та доступ до них

Як було зазначено раніше, дані було вирішено зберігати в текстових файлах у форматі JSON. Саме для цього був розроблений власний міні-фреймворк, що дозволяє швидко та зручно записувати і зчитувати дані з JSON файлів. Цей фреймворк працює сумісно з бібліотекою Newtonsoft.Json. Під час запуску програми розпочинається зчитування всіх даних із файлів відповідної папки. Далі виконується їх серіалізація, тобто приведення тексту до програмних типів. Після чого дані повністю готові до маніпуляцій всередині програми.

Як колекція для зберігання списку об’єктів в додатку використовується ObservableCollection. Головна особливість цієї колекції полягає в можливості через виклик події CollectionChanged сповіщати систему про зміну даних всередині. Так можна реалізувати автоматичний запис даних при кожній модифікації колекції та уникнути випадкових втрат при несподіваній поломці.

Однак даний спосіб не завжди є доцільним. Наприклад, при імпорті предметів з таблиці Excel не раціонально записувати дані при додаванні в колекцію нового об’єкта, так як це значно уповільнить процес імпорту. Доцільнішим буде викликати запис даних після остаточного завантаження всіх предметів з таблиці. Фреймворк також дає можливість виконувати запис та зчитування даних асинхронно, що дозволяє не перекривати головний потік, а виконувати маніпуляції в бекграунді.

Всі типи об’єктів, з якими працює фреймворк спадкується від абстрактного базового класу BaseModel. Цей тип реалізує інтерфейс INotifyPropertyChanged, що допомагає сповістити систему про зміни всередині об’єкта.

Реалізація BaseModel на мові C#:

```
public abstract class BaseModel : INotifyPropertyChanged
{
	[JsonProperty(Order = -int.MaxValue)]
	public string Id { get; set; }

	public BaseModel()
	{
		Id = Guid.NewGuid().ToString();
	}

	public event PropertyChangedEventHandler PropertyChanged;

	protected virtual void OnPropertyChanged([CallerMemberName] string
		propertyName = null)
	{
		 PropertyChanged?.Invoke(this, new
		 PropertyChangedEventArgs(propertyName));
	}

	protected virtual bool SetProperty<T>(ref T storage, T value,
		[CallerMemberName] string propertyName = "")
	{
```

```
            if (EqualityComparer<T>.Default.Equals(storage, value))
                return false;

            storage = value;
            OnPropertyChanged(propertyName);
            return true;
        }
}
```

Також BaseModel містить в собі властивість Id типу string. У всіх об’єктів значення Id є унікальним та присвоюється при першому створенні об’єкта. Завдяки цій властивості можна виконувати швидкий пошук потрібного об’єкта і реалізувати зв’язки між об’єктами. Унікальність значень досягається за рахунок використання GUID (англ. Globally Unique Identifier) – статистично унікальний 128-бітний ідентифікатор. Загальне число унікальних ключів настільки велике, що ймовірність того, що у світі будуть незалежно згенеровані два однакових ключі, вкрай мала [10].

Всі моделі системи містять кастомний атрибут CollectionNameAttribute. Цей атрибут зберігає в собі ім’я файла, в якому будуть зберігатись дані тої чи іншої моделі. Для моделі Teacher це є Teachers, а Subject – Subjects. Відповідно дані будуть зберігатись у файлах Teachers.json та Subjects.json. Додатково був написаний власний JSON конвертер, який ініціалізує список зовнішніх об’єктів за наявності списку їх ідентифікаторів.

Реалізація цього JSON конвертера на мові C#:

```
public class BaseModelJsonConverter : JsonConverter
{
    public override bool CanConvert(Type objectType)
    {
        return objectType.BaseType == typeof(BaseModel)
            || (objectType.GetInterfaces()
            .Any(x => x.Name == "IEnumerable")
            && objectType.GenericTypeArguments[0].BaseType ==
            typeof(BaseModel));
    }

    public override object ReadJson(
        JsonReader reader,
        Type objectType,
```

```
        object existingValue,
        JsonSerializer serializer)
    {
        if (reader.TokenType == JsonToken.StartObject)
        {
            var item = JObject.Load(reader);
            PopulateIds(item, objectType);
            return item.ToObject(objectType);
        }
        else
        {
            var array = JArray.Load(reader);

            foreach (var jtoken in array)
            {
                PopulateIds(jtoken,
                objectType.GenericTypeArguments[0]);
            }

            return array.ToObject(objectType);
        }
    }

    public override void WriteJson(
        JsonWriter writer,
        object value,
        JsonSerializer serializer)
    {
        var jtoken = JToken.FromObject(value);

        if (jtoken is JObject jobj)
        {
            WriteIdsToJObject(jobj, value).WriteTo(writer);
        }
        else if (jtoken is JArray jarr)
        {
            writer.WriteStartArray();

            for (int i = 0; i < jarr.Count; ++i)
            {
                var item = (value as
                IEnumerable<BaseModel>).ElementAt(i);
                WriteIdsToJObject(jarr[i] as JObject,
                item).WriteTo(writer);
            }

            writer.WriteEndArray();
        }
    }

    private JObject WriteIdsToJObject(JObject jobj, object obj)
```

```
{
      var props = obj?.GetType()?.GetProperties()?
      .Where(x =>
      x?.GetValue(obj) is IEnumerable<BaseModel>
      && x?.GetCustomAttributes(false)?
      .Any(y => y?.GetType()?.Name == nameof(JsonIgnoreAttribute))
      == true);

      if (props?.Any() == true)
      {
            foreach (var prop in props)
            {
                  var ids = (prop.GetValue(obj) as
                  IEnumerable<BaseModel>)?
                  .Select(x => x?.Id)?.Where(x =>
                  !string.IsNullOrEmpty(x));

                  if (ids?.Any() == true)
                        jobj.Add(new JProperty(prop.Name, new
                        JArray(ids)));
            }
      }

      return jobj;
}

private void PopulateIds(JToken jtoken, Type objectType)
{
      var obj = jtoken.ToObject(objectType);

      var props = obj?.GetType()?.GetProperties()?
      .Where(x => x?.PropertyType?.GenericTypeArguments?.Any() ==
      true && x?.PropertyType?.GenericTypeArguments[0]?.BaseType ==
      typeof(BaseModel) && !Attribute.IsDefined(x,
      typeof(JsonIgnoreAttribute)));

      if (props?.Any() == true)
      {
            foreach (var prop in props)
            {
                  var ids = jtoken[prop.Name].ToArray();
                  prop.SetValue(obj,
                  Activator.CreateInstance(prop.PropertyType));

                  foreach (var id in ids)
                  {
                        var list = prop.GetValue(obj);
                        var baseModel =
                        Activator.CreateInstance(prop.PropertyType.Gen
                        ericTypeArguments[0]);
```

```
                        var idProp =
                        baseModel.GetType().GetProperty("Id");
                        idProp.SetValue(baseModel,
                        id.Value<string>());
                        prop.PropertyType.GetMethod("Add").Invoke(list
                        , new object[] { baseModel });
                    }
                }
            }

            jtoken = JToken.FromObject(obj);
        }
}
```

2.4. Реалізація основних функцій

Серед основних функцій інформаційної системи «Кафедра» треба виділити:

- **Відображення даних**

Ця функція реалізована за допомогою ObservableCollection, яка містить список об’єктів для відображення та елемента ItemsControl, що відображає об’єкти в заданому програмістом вигляді. Зв’язок ItemsControl та списку об’єктів виконується за допомогою прив’язки властивості ItemsSource до колекції за допомогою елемента Binding. Завдяки особливостям ObservableCollection елемент ItemsControl буде автоматично оновлювати відображення при будь-яких змінах всередині колекції.

Відображення об’єкта задається у властивості ItemTemplate. В системі «Кафедра» це елемент Grid із масивом елементів Label, які зв’язані властивістю Content із окремими властивостями кожного об’єкта в колекції. Всі елементи Grid розміщені в елементі StackPanel з вертикальною орієнтацією. Отриманий результат візуально сприймається користувачем як таблиця.

Задання ItemsControl в коді XAML на прикладі SubjectsView:

```
<ItemsControl ItemsSource="{Binding SubjectsToShow}">
	<ItemsControl.ItemsPanel>
		<ItemsPanelTemplate>
			<StackPanel Orientation="Vertical"/>
		</ItemsPanelTemplate>
	</ItemsControl.ItemsPanel>
	<ItemsControl.ItemTemplate>
		<DataTemplate>
			<Grid
				Margin="0,5,0,0"
				Background="{DynamicResource BackgroundColor}">

					...

			</Grid>

		</DataTemplate>

	</ItemsControl.ItemTemplate>

</ItemsControl>
```

- **Редагування та додавання даних**

Для редагування та додавання даних були створені окремі попапи із полями для вводу інформації. Якщо в попап при виклику передається об’єкт, він показує повідомлення про редагування, дані об’єкта автоматично відображаються в полях для вводу. Якщо при виклику попап нічого не отримує, поля для вводу залишаються порожніми, а зверху відображається повідомлення про додавання нового об’єкта.

Метод додавання викладача:

```
private async void AddTeacher()
{
	if (IsBusy)
		return;

	IsBusy = true;

	var teacher = await _dialogService.ShowTeacherForm();

	if (teacher != null)
		Teachers?.Add(teacher);
```

```
        IsBusy = false;
}
```

Після того як користувач натискає на кнопку підтвердження дії, програма перевіряє введені дані на валідність. Кожна модель містить в собі публічний метод IsValid, який у свою чергу містить набір правил, яким має відповідати модель. Якщо хоча б одне правило не виконується, метод повертає значення false як результат і присвоює текст помилки out-параметру error. Після чого текст помилки показується користувачу. Якщо ж об’єкт пройшов перевірку на валідність, метод повертає значення true і об’єкт додається в колекцію.

Метод IsValid для моделі Teacher:

```
public bool IsValid(out string error)
{
        if (string.IsNullOrWhiteSpace(LastName))
        {
                error = "Вкажіть прізвище";
                return false;
        }

        if (string.IsNullOrWhiteSpace(FirstName))
        {
                error = "Вкажіть ім'я";
                return false;
        }

        if (string.IsNullOrWhiteSpace(MiddleName))
        {
                error = "Вкажіть по батькові";
                return false;
        }

        if (Rate < 0 || Rate > 2)
        {
                error = "Вказана ставка не є коректною";
                return false;
        }

        error = null;
        return true;
}
```

- **Імпорт предметів із таблиць Excel**

Також в інформаційній системі «Кафедра» наявна функція імпорту предметів із таблиць Excel. Як було зазначено раніше, ця функція була реалізована за допомогою технології Excel Interop. Коли користувач натискає кнопку «Імпортувати», запускається метод ImportSubjects, який спочатку викликає діалогове вікно для вибору файлів. Після того як файли були вибрані, на екрані з'являється попап із прогресом імпортування, запускається метод зчитування предметів. Дані щодо прогресу попап отримує за допомогою підписки на подію LoadProgressChanged. Після закінчення імпортування викликається подія LoadCompleted і попап закривається. Предмети додаються в колекцію за допомогою циклу for.

Метод для завантаження предметів з таблиці Excel:

```
public static List<Subject> GetSubjects(string[] filePaths)
{
	var subjects = new List<Subject>();
	var args = new ExcelLoadProgressChangedEventArgs
	{
		TotalSheets = filePaths.Length
	};

	foreach (var filePath in filePaths)
	{
		args.CurrentSheetName = filePath.Split('\\').Last();
		args.CurrentSheetLoadingProgress = 0;
		LoadProgressChanged?.Invoke(args);

		if (_app == null)
			_app = new Application();

		var workBook = _app.Workbooks.Open(filePath, 0, true, 5, "",
		"", false, XlPlatform.xlWindows, "", true, false, 0, true,
		false, false);
		var workSheet = (Worksheet)workBook.Sheets[1];

		_titles = GetTitles(workSheet);
		var rowsCount = GetWorkSheetRowsCount(workSheet);

		string leftTopCell = $"{ DataStartColumn }{ DataStartRow }";

		string rightBottomCell = $"{ (char)(DataStartColumn +
		_titles.Count - 1) }{ DataStartRow + rowsCount - 1 }";
```

```
            _data = GetData(workSheet, leftTopCell, rightBottomCell);

            for (int i = DataStartRow; i < rowsCount - 1; ++i)
            {
                  var subject = GetItem<Subject>(i - DataStartRow);

                  if (subject != null)
                        subjects.Add(subject);

                  args.CurrentSheetLoadingProgress = (i - DataStartRow +
                  1) * 100 / (rowsCount - DataStartRow - 1);
                  LoadProgressChanged?.Invoke(args);
            }

            ++args.LoadedSheets;
            LoadProgressChanged?.Invoke(args);

            Marshal.ReleaseComObject(workSheet);
            workBook.Close(0);
            Marshal.ReleaseComObject(workBook);
      }

      _app.Quit();
      _app = null;
      LoadCompleted?.Invoke(null, null);
      return subjects;
}
```

Розділ 3. ІНСТРУКЦІЯ З ВИКОРИСТАННЯ

3.1. Основні операції

Після того як програма запустилась, на екрані відображається головне вікно. У лівій частині вікна знаходиться меню. За допомогою першої кнопки зверху користувач має змогу згорнути або розгорнути меню. Після виходу з програми це налаштування збережеться. Нижче цієї кнопки розташовані закладки. Обрана закладка виділяється чорним кольором. Контент в правій частині вікна змінюється в залежності від обраної закладки. Внизу меню знаходиться кнопка «Нічний режим», яка дозволяє змінювати кольорову схему додатка (рис. 3.1.1). Після виходу з програми кольорова схема також буде збережена.

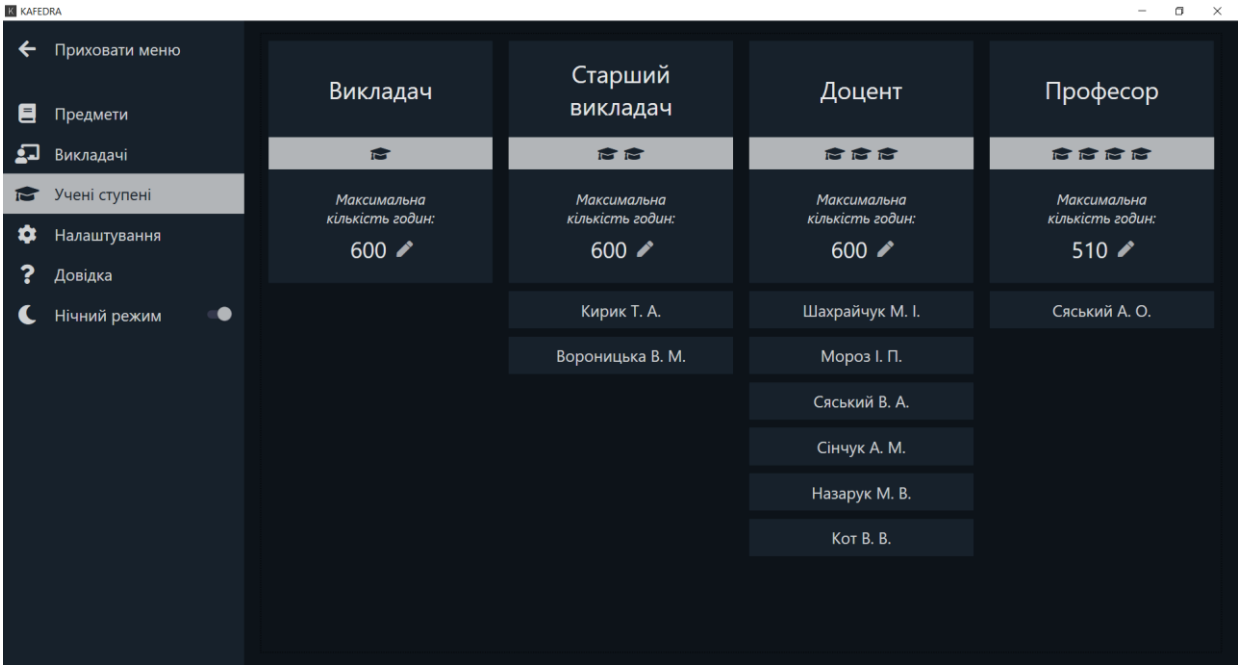


Рис. 3.1.1. Інтерфейс додатка в нічному режимі.

Для керування предметами потрібно перейти в закладку «Предмети» та обрати потрібну операцію. Для додавання нового предмета потрібно натиснути кнопку «Додати» у верхній лівій частині екрана. Після цього на екрані з’явиться

попап із полями вводу (рис. 3.1.2). Далі потрібно ввести інформацію про предмет у відповідні поля і натиснути кнопку «Додати». Якщо введена інформація є валідною, попап закриється і новий предмет буде додано до списка.

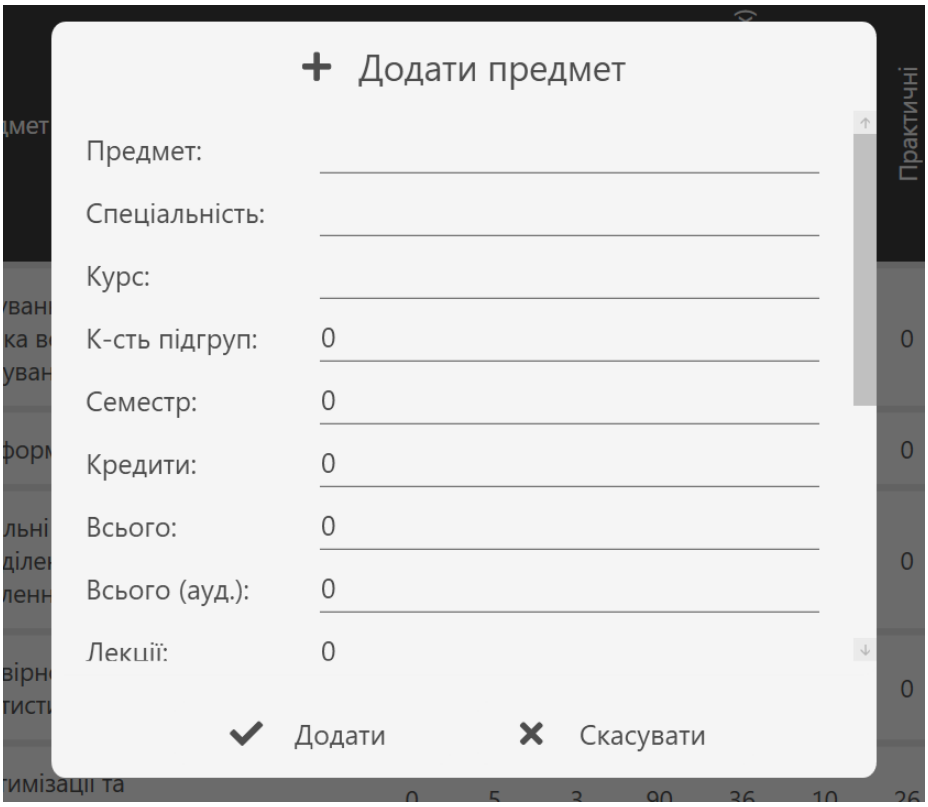


Рис. 3.1.2. Попап для додавання предмета.

Для редагування предмета потрібно натиснути іконку олівця справа від його даних. Після чого відкриється аналогічний попап, тільки з уже заповненими даними. Щоб видалити предмет, потрібно натиснути на іконку урни поруч із іконкою олівця і підтвердити дію. Користувач може видалити одразу всі предмети за допомогою кнопки «Очистити» в правій верхній частині екрана. Після видалення предметів їх неможливо буде відновити, тому краще зробити резервну копію файла.

Перелічені вище операції доступні також для викладачів. Окрім даних операцій наявне редагування списка предметів, які може вести викладач. Щоб відредагувати список, потрібно натиснути на чорну кнопку з числом предметів в рядку з відповідним викладачем. Після цього на екрані з’явиться попап, в правій

частині якого знаходяться предмети, на яких спеціалізується викладач, а в лівій – всі інші предмети (рис. 3.1.3). Предмети розташовані в алфавітному порядку. Для перенесення предмета з одного списку в інший потрібно натиснути на відповідний предмет. Після завершення роботи – натиснути на кнопку «Зберегти».

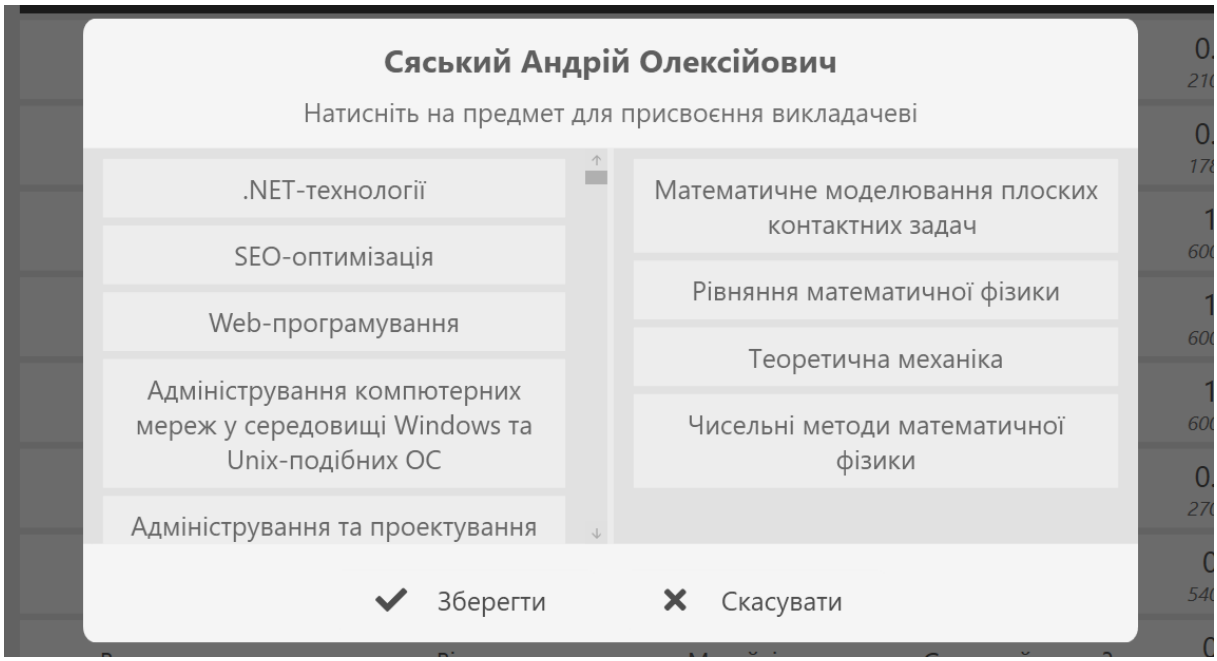


Рис. 3.1.3. Попап зі списками предметів.

Для того, щоб змінити максимальне число годин для певного вченого звання, потрібно перейти на закладку «Учені ступені» та натиснути на іконку олівця всередині панелі відповідного ступеня. Після цього змінити число годин на потрібне значення і зберегти зміни, натиснувши на іконку галочки. Введене значення повинне бути в діапазоні від 1 до 2000.

3.2. Імпортування предметів

Якщо користувач має таблиці з предметами, він може скористатись функцією імпортування предметів. Для цього потрібно перейти на закладку «Предмети» та натиснути кнопку «Імпорт». Після виконання цих дій на екрані з’явиться стандартне вікно для вибору файлів. Користувач може обрати файли

лише формату XLS, XLSX або XLSM. Після підтвердження вибору в програмі відкриється попап із прогресом імпортування (рис. 3.2.1).

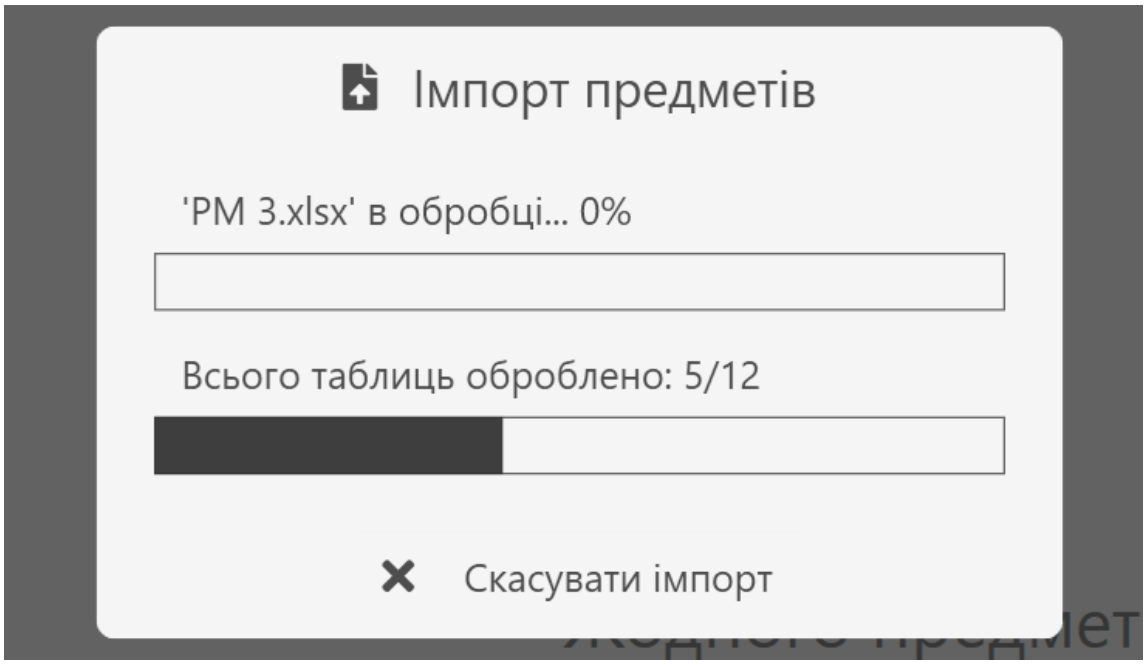


Рис. 3.2.1. Попап із прогресом імпортування.

Зверху відображається прогрес імпортування поточного файла в процентному співвідношенні. Знизу – загальний прогрес для всіх обраних файлів. Користувач може скасувати імпортування у будь-який момент натиснувши на кнопку внизу попапа. Після закінчення імпортування попап закриється, а в таблиці відобразяться нові предмети.

3.3. Розподіл навантаження

Однією із найголовніших функцій системи є розподіл навантаження. За допомогою даної функції користувач має можливість розподіляти навчальне навантаження між викладачами. Зверху вікна відображається навантаження вибраного викладача, а знизу нерозподілене навантаження (рис. 3.3.1). Для того, щоб програма виконала автоматичний розподіл, користувачеві необхідно всього лише натиснути кнопку «Сформувати» зверху вікна. Після чого навчальне навантаження буде розподілене між всіма викладачами згідно списку дисциплін,

на яких вони спеціалізуються, та їхньої ставки. В разі, якщо користувачу потрібно внести додаткові зміни в розподіл, він може зробити це вручну. Для цього потрібно обрати потрібного викладача в списку зліва та натиснути на години, які потрібно присвоїти викладачеві або навпаки – вилучити. У вікні користувач також має змогу бачити кількість годин, присвоєних вибраному викладачеві, та загальну кількість нерозподілених годин.

Нерозподілене навантаження *19 855 год не розподілено*

Нейронні мережі	Залік	7 год	ІН-41	0	7
Розробка програмного забезпечення під мобільні платформи	Лекції	20 год	ІН-41	0	8
Розробка програмного забезпечення під мобільні платформи	Лабораторні	30 год	ІН-41	0	8
Розробка програмного забезпечення під мобільні платформи	Залік	8 год	ІН-41	0	8
Розподілені інформаційно-аналітичні системи	Лекції	28 год	ІН-41	0	8

Рис. 3.3.1. Нерозподілене навантаження.

ВИСНОВКИ

В результаті написання дипломної роботи «Створення автоматизованої інформаційної системи «Кафедра»» були виконані наступні завдання:

- **Дослідження існуючих програмних засобів**

Було досліджено ряд уже існуючих додатків з аналогічним призначенням. В ході дослідження були визначені їхні основні переваги та недоліки. Завдяки цьому, під час розробки системи вдалось уникнути невдалих рішень та перейняти ефективні особливості кожного з додатків.

- **Реалізація простої та гнучкої бази даних**

Була реалізована максимально проста база даних, що являє собою сукупність текстових файлів формату JSON. Таким чином, для встановлення системи «Кафедра» не потрібно додатково встановлювати та налаштовувати спеціальні інструменти для взаємодії з БД. Для неї також був написаний власний фреймворк на мові програмування C#, який дозволяє використовувати зв'язки «один-до-одного» і «один-до-багатьох» в реалізованій базі даних. Окрім цього, фреймворк виконує швидкий запис та зчитування даних з БД.

- **Забезпечення швидкої взаємодії із табличним процесором Excel**

Завдяки детальному вивченню багатьох інтернет-ресурсів, був написаний оптимальний алгоритм зчитування даних з файлів Excel за допомогою бібліотеки Excel Interop. Проведена ретельна оптимізація процесів парсингу даних, яка дозволила обробляти значні об'єми інформації за лічені секунди. Результат є досить ефективним, враховуючи те, що обробка цього самого об'єму даних виконується близько хвилини в аналогічних програмних засобах.

- **Розробка сучасного та зручного графічного інтерфейсу**

Було досліджено провідні тенденції UX-дизайну, внаслідок чого вдалось розробити візуально приємний та практичний користувацький інтерфейс додатка. Для кожного графічного елемента був прописаний власний стиль, який не лише надавав би йому гарного вигляду, а й дозволив користувачу інтуїтивно зрозуміти його призначення в системі. Значний період часу був витрачений на

створення анімацій, призначених зробити програму більш живою та сучасною. Реалізовано так званий «Нічний режим», що змінює кольорову схему інтерфейсу зі світлої на темну.

Враховуючи безкоштовний загальний доступ до програмного застосунку, його багатофункціональність та зручний інтерфейс, можна з впевненістю сказати, що інформаційна система «Кафедра» значно переважає над вже існуючими продуктами з аналогічним призначенням. У випадку успішної апробації додатка співробітниками Рівненського державного гуманітарного університету планується впровадження системи в інші ВНЗ України.

Підводячи підсумки, можна впевнено стверджувати, що цілі даної дипломної роботи є досягнутими. Автоматизована інформаційна система «Кафедра» є повністю готовою до використання цільовими користувачами. В подальшому система буде вдосконалюватись та модифікуватись відповідно до потреб користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Організація сучасного діловодства [Електронний ресурс] – Режим доступу: http://www.dut.edu.ua/uploads/l_1294_51446278.pdf.
2. Пакет програм «Деканат» для вищих навчальних закладів III-IV рівнів акредитації [Електронний ресурс] – Режим доступу: http://www.politek-soft.kiev.ua/index.php?do=products&product=deanery34.
3. C# і .NET | Введение [Електронний ресурс] – Режим доступу: https://metanit.com/sharp/tutorial/1.1.php.
4. WPF – Википедия [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Windows_Presentation_Foundation.
5. WPF | Введение [Електронний ресурс] – Режим доступу: https://metanit.com/sharp/wpf/1.php.
6. Model-view-viewmodel – Wikipedia [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel.
7. WPF | Паттерн MVVM [Електронний ресурс] – Режим доступу: https://metanit.com/sharp/wpf/22.1.php.
8. JSON – Вікіпедія [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/JSON.
9. Icons | Font Awesome [Електронний ресурс] – Режим доступу: https://fontawesome.com/icons?d=gallery.
10. GUID – Вікіпедія [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/GUID.