**Форма № Н-6.01у**

Кременчуцький національний університет імені Михайла Остроградського

(повне найменування вищого навчального закладу)

Кафедра автоматизації та інформаційних систем

(повна назва кафедри, циклової комісії)

**КУРСОВИЙ ПРОЄКТ**

**(РОБОТА)**

з дисципліни «Сучасні мови об’єктно-орієнтованого програмування»

(назва дисципліни)

на тему Розробка WPF додатку «Розрахунок матриці долі”

Студента \_2\_ курсу КН–22–2 групи

Ступінь вищої освіти «Бакалавр»

(бакалавр, магістр)

Спеціальність 122 – «Комп’ютерні науки»

Освітньо-професійна програма

«Комп’ютерні науки»

Бутник Дмитро Віталійович

(прізвище та ініціали)

Керівник старший викладач кафедри АІС

\_\_\_\_\_\_\_ Бельська В. Ю.\_\_\_\_\_\_\_\_\_\_\_

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_. Оцінка: ЄКTС \_\_\_\_

Члени комісії  . В. Ю. Бельська

(підпис) (ініціали та прізвище)

І. Г. Оксанич

(підпис) (ініціали та прізвище)

В. В. Найда

(підпис) (ініціали та прізвище)

м. Кременчук 2023 рік

**Форма № Н-9.01у**

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

Кафедра автоматизації та інформаційних систем

Дисципліна «Сучасні мови об’єктно-орієнтованого програмувння»

Освітній ступінь \_\_ «Бакалавр»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Спеціальність 122 – «Комп’ютерні науки»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Освітня програма «Комп’ютерні науки»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Курс 2 група \_\_КН-22-2 семестр \_\_3\_\_

**ЗАВДАННЯ**

**НА КУРСОВИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТУ**

\_\_\_\_\_\_\_Бутнику Дмитру Віталійовичу\_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по-батькові)

1. Тема роботи: Розробка WPF додатку “Розрахунок матриці долі”
2. Термін здачі студентом роботи: 1 грудня 2023 р
3. Вихідні дані до роботи: ColorThemes.json (зберігає кольорові теми для програми), Settings.json (містить значення налаштувань проекту),
4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):

постановка завдання, аналіз технічного завдання, розробка алгоритму роботи програми, опис моделі даних, структура програмного забезпечення, функціональна схема та інтерфейс програми.

1. Перелік графічного матеріалу:
2. Дата видачі завдання: 1 жовтня 2023 р.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| №  пор. | Назва етапів курсового проекту | Терміни виконання етапів проекту | Вказівки та  зауваження викладача (з зазначенням дати консультації) | Оцінювання етапів проекту | | |
| за націо- нальною шкалою | за шкалою ЄКТС | кількість балів |
| 1 | Етап 1  Аналіз предметної області | 01.10.23–  25.10.23 |  |  | | |
| 2 | Етап 2  Створення моделі даних | 16.10.23–  28.10.23 |  |  | | |
| 3 | Етап 3  Розробка віконного інтерфейсу та створення основного програмного коду додатку | 29.10.23–  24.11.23 |  |  | | |
| 4 | Етап 4  Тестування програмного коду | 16.10.23  29.11.23 |  |  | | |
| 5 | Етап 5  Оформлення пояснювальної записки | 25.11.23–  31.11 .23 |  |  | | |
| 6 | Етап 9  Захист | 01.12.23 |  |  | | |
|  | Разом | 8 тижнів |  |  | | |

Студент Бутник Дмитро Віталійович

(підпис)

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_ В. Ю. Бельська \_\_\_\_

(підпис) (ініціали та прізвище)

«1» жовтня 2023 р.

РЕФЕРАТ

Курсова робота містить 53 сторінки, 2 розділи, 15 рисунків, 1 таблиця, 5 використаних джерел.

Об’єкт розробки – WPF додаток «Розрахунок матриці долі”.

Мета: створення багатовіконного WPF додатку, що реалізує додаток «Розрахунок матриці долі».

Під час виконання завдання, поставленого на курсову роботу, було виконано аналіз алгоритму програми “Розрахунок матриці долі”. Визначено основні функціональні та не функціональні вимоги до додатку, що створюється. Побудована модель даних, що дозволить вирішити наступні задачі: розробка алгоритму запису даних користувача, розробка алгоритму для розрахунку матриці, опис та додаткові відомості, функція збереження у PDF форматі, створення віконного інтерфейсу задля графічного відображення матриці, тощо.

В якості мови програмування для створення WPF додатку використовувалася мова C# та середовище програмування Visual Studio 22.

Результатом виконання всіх етапів є застосунок, який розраховує матрицю через введення даних користувача, можливість вибору розрахунку, вивід опису та збереження матриці у PDF форматі.

MVVM, WPF, DATA BINDING, ICOMMMAND, ООП

**Зміст**

[Вступ 2](#_Toc152801546)

[1 Опис предметної області 4](#_Toc152801547)

[1.1 Три загальні акроніми, які використовуються в .NET, їх призначення 4](#_Toc152801548)

[1.2 Аналіз технічного завдання на роботу 5](#_Toc152801549)

[1.2.1 Функціональні вимоги 5](#_Toc152801550)

[1.2.2 Нефункціональні вимоги 5](#_Toc152801551)

[1.3 Опис алгоритму основних задач/підзадач у роботі 5](#_Toc152801552)

[Висновки до розділу 7](#_Toc152801553)

[2 Опис розробки програмного забезпечення 8](#_Toc152801554)

[2.1 Структура програмного забезпечення 8](#_Toc152801555)

[2.2 Опис роботи програми 10](#_Toc152801556)

[2.3 Функціональна схема програми 11](#_Toc152801557)

[2.4 Опис розробки модулів VievModel та Model проеку 13](#_Toc152801558)

[Висновки до розділу 19](#_Toc152801559)

[Висновки 20](#_Toc152801560)

[Список літератури 21](#_Toc152801561)

[Додаток А класи Model 22](#_Toc152801562)

[Додаток Б класи VievModel 29](#_Toc152801563)

[Додаток В Тестування додатку 49](#_Toc152801564)

ВСТУП

На сьогоднішній день, найбільш популярною платформою для створення графічного інтерфейсу в середовищі C# є Windows Presentation Foundation (WPF). WPF широко використовується для розробки різноманітних програм, охоплюючи модель програми, ресурси, елементи управління, графіку, макети, прив'язки даних, документи та аспекти безпеки.

Технологія WPF (Windows Presentation Foundation) є частиною екосистеми платформи .NET і являє собою підсистему для побудови графічних інтерфейсів.

Ця платформа має вражаючий набір компонентів, які дозволяють розробникам створювати зручні та ефективні інтерфейси. Однією з ключових особливостей WPF є потужна система прив'язок (Binding), що стала необхідним інструментом для серйозного програмування. Використання прив'язок дозволяє зручно маніпулювати та відображати дані, пов'язуючи їх з елементами інтерфейсу.

При використанні WPF можна легко створити інтуїтивно зрозумілий інтерфейс для програми. Механізми прив'язок дозволяють ефективно реалізувати функції редагування та відображення заявок та інших даних, надаючи користувачам зручний та інтуїтивний інтерфейс для взаємодії з програмою.

Якщо при створені традиційних додатків на основі WinForms за візуалізацію елементів керування відповідали такі частини ОС Windows, як User32 та GDI+, то додатки WPF засновані на **DirectX**.В цьому і є головна відмінність рендерингу графіки у WPF: велика частина роботи по візуалізації графіки, як і звичайних кнопок, так і складних 3D-моделей, лежить на графічному процесорі на відеокарті, що також дає можливість користуватись апаратним прискоренням графіки.

Ще однією важливою особливістю WPF є використання мови декларативної розмітки XAML, заснованого на XML: ви можете створити гарний графічний інтерфейс, використовуючи або декларативне оголошення інтерфейсу, або код на керуючих мовах, як C#.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Три загальні акроніми, які використовуються в .NET, їх призначення

В індустрії програмування сучасних технологій вкрай важливо освоювати основні концепції та стандарти, які стоять за роботою використовуваних технологій. Однією із важливих аспектів вивчення є розуміння загальних акронімів, що активно використовуються в області .NET-технологій.

IL (Intermediate Language): IL або мова проміжного рівня є формою представлення програмного коду на рівні, що є проміжним між вихідним кодом програми та виконавчим кодом. Вона є платформонезалежною та використовується компіляторами мов програмування .NET для створення виконавчого коду.

CIL (Common Intermediate Language): CIL - це стандартізована версія мови проміжного рівня, призначена для використання різними мовами програмування. Вона надає спільний набір інструкцій та правил для побудови виконавчого коду, що забезпечує взаємодію різних мов на платформі .NET.

CLI (Common Language Infrastructure): CLI є стандартом, який визначає загальні правила для розробки програмного забезпечення на платформі .NET. Він охоплює систему типів, обробку винятків, збірку мусору та інші аспекти, які забезпечують взаємодію мов програмування з виконавчим середовищем.

Розуміння цих акронімів необхідне для розробників, оскільки вони надають ключові знання щодо внутрішніх процесів та можливостей платформи .NET. Засвоєння концепцій IL, CIL та CLI дозволяє ефективно використовувати можливості .NET для створення високоефективних та переносних додатків.

1.2 Аналіз технічного завдання на роботу

Мета курсового проєкту – Розробити WPF додаток «Розрахунок матриці долі».

Під час розробки моделей даних та графічного інтерфейсу були висунуті наступні вимоги:

**1.2.1 Функціональні вимоги**

1. Можливість занесення даних користувача : ім’я та дата народження.

2. Можливість вибору матриці : індивідуальна або сумісна, після чого з’являються додаткові поля для занесення даних.

3. Вивід інформації – отримання розрахованої матриці долі та її опис.

4. Можливість перегляду додатковго опису при подвійному натисканні на аркани.

5. Збереження отриманої інформації – за бажанням користувача, можна зберегти інформацію у PDF форматі.

**1.2.2 Нефункціональні вимоги**

1. Зовнішній інтерфейс має бути реалізованим за допомогою WPF.

2. Використання паттерну MVVM.

3. Усі поля введення повинні бути захищені від пустого або некоректного введення.

4. Додаток має бути багатовіконним.

1.3 Опис алгоритму основних задач/підзадач у роботі

Розробка алгоритмів функціонування програми в контексті розрахунку матриці долі у середовищі C# WPF передбачає створення моделі роботи та функціональності програми як єдиної системи. На даному етапі необхідно передбачити обмеження, які можуть у майбутньому вплинути на коректне розширення функціоналу програми.

Програма розрахунку матриці долі розробляється на мові програмування C# з використанням технології WPF. Основна мета програми - обчислення та відображення матриці долі користувача, а також забезпечення можливості збереження цієї матриці у форматі PDF.

Основні кроки роботи програми:

1. Вибір та введення даних користувача:

- На першому вікні програми користувач обирає тип матриці (індивідуальна чи сумісна).

- Вводить особисті дані, такі як ім'я та дата народження.

2. Розрахунок матриці долі:

- На основі введених даних та обраного типу матриці програма виконує необхідні обчислення для отримання матриці долі.

- Результат розрахунків відображається на другому вікні.

3. Опис та функція збереження матриці:

- На другому вікні виводиться розрахована матриця разом із відповідним описом.

- Кнопка "Зберегти у PDF" дозволяє користувачеві зберегти отриману матрицю та її опис у форматі PDF.

4. Обробка помилок та обмежень:

- Валідація введених даних:

Перед розрахунками перевіряти правильність формату дати народження, який має відповідати стандарту (дд.мм.рррр) та некорректних символів.

- Відображення повідомлень про помилку:

Якщо введена дата не відповідає очікуваному формату, вивести повідомлення користувачеві з поясненням про необхідність введення коректних даних.

5. Розширення функціоналу:

- Алгоритми функціонування програми розроблені таким чином, щоб легко дозволяти розширення функціональності у майбутньому.

Висновки до розділу

У першому розділі розглянуті базові принципи ООП, створена розширена постановка задачі, проаналізована предметна область, сформовано вимоги до функціональних можливостей програми та розглянуто технічне завдання. Складенj план розробки програми та функції, що повинні бути реалізовані у програмі.

2 ОПИС РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Структура програмного забезпечення

Структура програмного забезпечення є поєднанням усіх застосованих модулів програми, що утворюють працюючу систему. Кожний модуль повинен реалізовувати функції програми, що вирішують основні завдання проєкту.

Програма розроблена в інтерфейсі Windows Presentation Foundation (WPF), що призначений для побудови графічних інтерфейсів.

На рис. 2.1 зображено оглядач рішень програмного застосунку. В табл.2.1 представлено опис модулів(класів) проекту курсової роботи

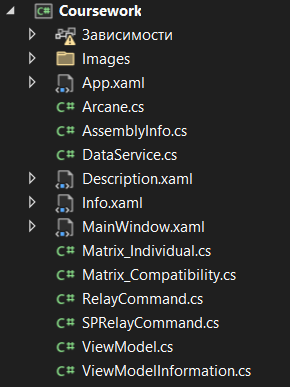
****

Рисунок 2.1 – Оглядач рішень

Таблиця 2.1 – Призначення модулів програми

|  |  |
| --- | --- |
| **Назва модуля** | **Призначення модуля** |
| Arcane.cs | Клас, що використовується як модель даних для арканів |
| DataService.cs | Клас, що використовується для передачі даних між вікнами |
| MainWindow.xaml | Вікно, яке зображує поля для внесення даних користувача та вибору типу матриці |
| MainWindow.xaml.cs | Клас вікна, що містить методи для коректної роботи вікна MainWindow.xaml |
| Matrix\_Individual.cs | Клас, який містить медоди для обчислення індивідуальної матриці долі |
| Matrix\_Compatibility.cs | Клас, який містить медоди для обчислення матриці сумісності |
| RelayCommands.cs | Клас, який використовується для реалізації патерна команди (Command pattern) в програмній розробці |
| SPRelayCommands.cs | Клас, який використовується для реалізації патерна команди (Command pattern) з передачею типу даних в програмній розробці |
| ViewModel.cs | Клас, що приймає введені користувачем данні |
| ViewModelInformation.cs | Клас, що отримує оброблені дані та відповідає за логіку виведення інформації |
| Info.xaml | Сторінка, яка виводить матрицю та її опис |
| Info.xaml.cs | Клас сторінки, що містить методи для коректної роботи вікна Info.xaml |
| Description.xaml | Вікно з описом обраного аркана |
| Description.xaml.cs | Клас сторінки, що містить методи для коректної роботи вікна Info.xaml |

Під час створення програми було створено 3 вікна до них було створено 2 моделі вигляду, як показано на рис. 2.2. Всі класи моделей зв’язані класом DataService і реалізують клас INotifyPropertyChanged.



Рисунок 2.2 – Діаграма класів вікон

Файли Model знаходяться у додатку А, ViewModel у додатку Б, тестування програми у додатку В.

2.2 Опис роботи програми

Програма на C# WPF розрахунку матриці долі розробляється для створення моделі роботи та функціонування як єдиної системи. На початковому вікні користувач обирає тип матриці (особисту або сумісну) та вводить особисті дані, такі як ім'я та дата народження.

Після введення даних користувачем і натискання на відповідну кнопку, програма переходить на друге вікно, де здійснюється розрахунок матриці долі. Результат виводиться на екрані разом з описом та іншою необхідною інформацією.

Особливою функціональністю є можливість збереження розрахованої матриці у форматі PDF. Користувач може викликати цю функцію, натискавши на відповідну кнопку. Програма генерує PDF-файл, включаючи в себе розраховану матрицю, опис та іншу інформацію.

При розробці алгоритмів функціонування, необхідно передбачити можливі обмеження, що можуть виникнути при розширенні функціональності програми. Це включає в себе обробку різних сценаріїв введення даних, забезпечення коректності розрахунків, а також надійність функції збереження у PDF.

**2.3 Функціональна схема програми**

Детальний опис й пояснення процесів окремих функцій розробляємої системи утворюють функціональну схему програми(рис. 2.3).

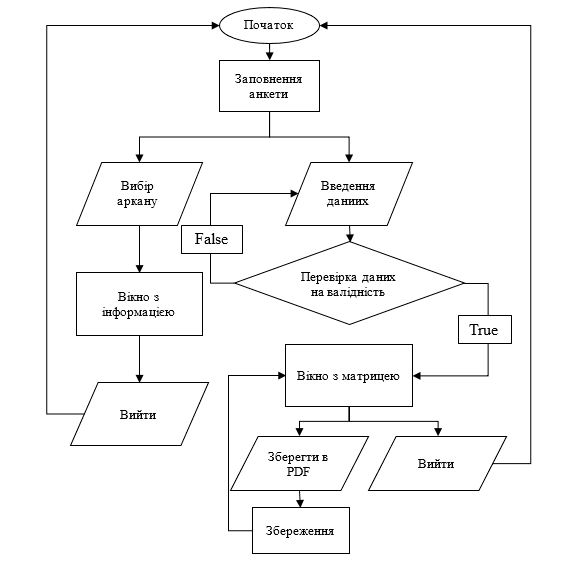


Рисунок 2.3 – Функціональна схема програмного коду

Як видно з рис.2.4 після запуску за стосунку першим відображається вікно з меню вибору типу матриці та вводом даних користувача. З цього вікна можна потрапити на вікно з описом обраного аркану або після вводу даних розрахувати матрицю та потрапити на сторінку з зображенням матриці та її описом.

З сторінки з матрицею, користувач може повернутися на початкове вікно і ввести нові данні або зберегти матрицю та її опис у PDF форматі.

2.4 Опис розробки модулів VievModel та Model проеку

Arcane.cs - один з основних класів моделей даних в рамках цього проекту. Цей клас містить в собі 5 полів та конструктор з параметрами та перевантажений метод ToString(), як зображено на рис. 2.4



Рисунок 2.4 – Структура класу Arcane

Поле A\_Path\_Image використовується для зберігання шляху до картинки у резурсах проекту.

Дані для полів цього класу зчитуються з файлу Arcanes.json у ресурсах проекту.

Натупним важливим класом є Matrix\_Individual.cs. Цей клас має 3 поля, та медоди для обчислення елементів індивідуальної матриці долі, як зображено на рис. 2.5. Також у цьому класі реалізований інтерфейс INotifyPropertyChanged.

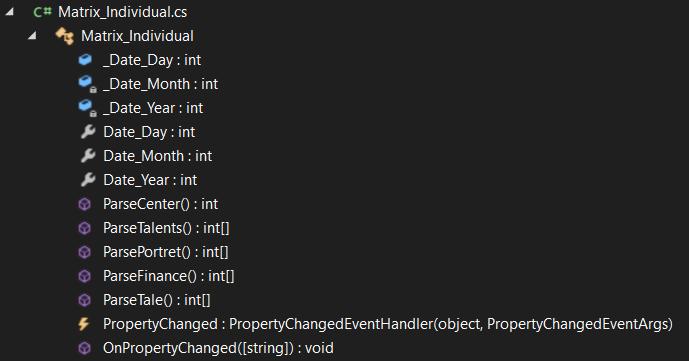


Рисунок 2.5 – Структура класу Matrix\_Individual

Методи ParseCenter(), ParseTalents(), ParsePortret(), ParseFinance() та ParseTale() використовуються для розрахунку значень арканів у матриці для відповідних ланок.

Реалізований метод INotifyPropertyChanged для полів, які вводить користувач на головній сторінці програми.

Також клас Matrix\_Compatibility.cs має 2 поля та методи для розрахунку елементів матриці сумісності, як зображено на рис. 2.6



Рисунок 2.6 – Структура класу Matrix\_Compatibility

Параметри класу Matrix\_Compatibility.cs приймають значень двох об’єктів класу Matrix\_Individual.cs. Які вже мають портібні дані для розрахунку, які ввів користувач.

Класи DataService.cs, RelayCommand.cs, SPRelayCommand.cs, ViewModel.cs та ViewModel.cs утворюють основні складові шаблону проекту для використання паттерну MVVM (Model-View-ViewModel) в додатках, розроблених на платформі .NET. Клас DataService.cs відповідає за передачу даних між класами ViewModel.cs та ViewModelInformation.cs, а також храніння цих даних для подальшого використання. RelayCommand.cs та SPRelayCommand.cs дають можливість створення команд для взаємодії між елементами користувацького інтерфейсу. SPRelayCommand.cs використовується для створення команд у яких існують функції, які приймають значення будь-якого типу даних. ViewModel.cs та ViewModelInformation.cs надають фунціонал для подій, які використовуються в ViewModel.

Розглянемо клас ViewModel.cs (рис. 2.7) , який прив’язується засобом DataContext до початкового вікна та до сторінки з детальним описом конкретно обраного аркану. Відповідає за прийняття значень анкети початкового вікна. Наслідує інтерфейс INotifyPropertyChanged. Його основна функція полягає в управлінні введенням дати/т народження користувача/ів, вибору типу матриці.



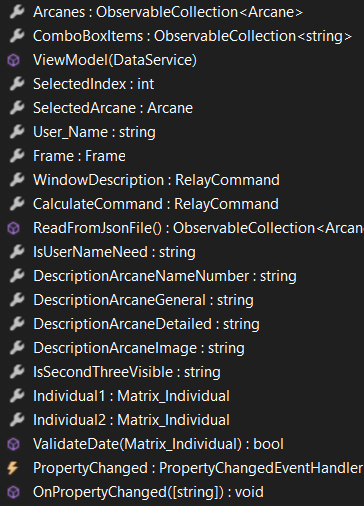


Рисунок 2.7 – Cтруктура класу ViewModel

Цей клас має метод ValidateDate(Matrix\_Individual data), який приймає один аргумент – об’єкт класу Matrix\_Individual.cs. Він перевіряє значення введеної користувачем дати на валідність. Якщо дата народження ,введена у вірному форматі, метод повертає значення true, в противному випадку повертає – false.

Метод IsUserNameNeed перевіряє чи потрібно користувачеві вводити своє ім’я, в залежності від обраного ним типу матриці.

Команда CalculateCommand створена за допомогою класу RelayCommand.cs. При натисанні кнопки на користувацькому інтерфейсі перевіряє введені дані на валідність за допомогою методу ValidateDate, якщо дані валідні виводить сторінку Description.xaml і передає дані у DataService. Якщо ведені дані невалідні – виводить MessageBox з відповідною помилкою.

Метод SelectedIndex отримує значення (індекс) матриці, яку користувач хоче створити. В залежності від виду, на користувацькому інтерфейсі виводить додаткові поля для вводу другої дати народження за допомогою методу методу IsSecondThreeVisible та згортає поле для вводу вводу ім’я методом IsUserNameNeed.

Метод SelectedArcane повертає об’єкт класу Arcane.cs, якщо користувач обирає елемент з панелі ListBox користувацьго інтерфейсу. Повернене значення використовується для команди WindowDescription, яка виводить вікно з описом відповідно обраного аркану.

Метод ReadFromJsonFile() використовується для зчитування списку об’єктів класу Arcane.cs з файлу з базою даних арканів Arcanes.json. Зчитані за допомогою цього методу об’єкти вносяться у список ObservableCollection та розташовуються на панелі ListBox користувацького інтерфейсу. Для роботи методу було встановлено NuGet пакет Newtonsoft.Json.

Наступний клас ViewModelInformation.cs (рис. 2.8) отримує валідовані дані з класу DataService.cs. Наслідує інтерфейс INotifyPropertyChanged. Прив’язується до сторінки Info.xaml засобом DataContext. Головна мета цього класу – це обробка отриманих даних від користувача під моделі даних Matrix\_Individual та Matrix\_Compatibility. Має відповідні поля пов’язані з медодами класів моделі даних: (center1, finance1, portret1, talents1, tale1) – для індивідуальної матриці долі та додаткові поля (center2, finance2, portret2, talents2, tale2) – для ситуації, якщо користувач хоче розрахувати матрицю сумісності.

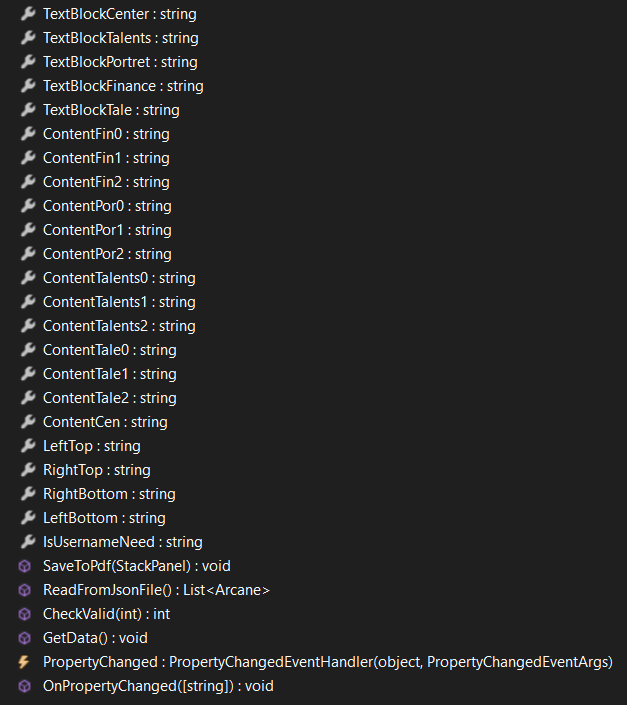


Рисунок 2.8 – структура класу ViewModelInformation

Метод GetData() використовується для використання методів класів моделі даних для розрахунку елементів матриці з тих даних, що надаються з класу DataService.cs. В залежності від попереднього вибору користувача розраховується один набір даних або два.

Метод CheckValid(int it) перевіряє на валідність формату елементи для матриці сумісності. Якщо елемент не валідний, метод відформатовує його.

Метод IsUsernameNeed перевіряє, в залежності від вибору користувача, чи потрібно виводити його ім’я в матриці. Також, якщо користувач залишив поле з ім’ям пустим, метод не виводить поле в користувацький інтерфейс також.

Метод ReadFromJsonFile() використовується для зчитування списку об’єктів класу Arcane.cs з файлу з базою даних арканів Arcanes.json. Зчитані за допомогою цього методу об’єкти вносяться у список ObservableCollection та в залежності від того, яке значення має елемент матриці виводить інформацію про відповідний об’єкт в сторінку Іnfo.xaml.

Команда BackToMainWindow при натисканні кнопки на користувацькому інтерфейсі виконує перехід до початкового вікна MainWindow.xaml.

Метод SaveToPdf(StackPanel panel) використовується для збереження панелі, яка вказана як аргумент методу, у файл PDF формату. Для роботи методу втановлено NuGet пакет iTextSharp.LGPLv2.Core та допоміжний для нього пакет BouncyCastle.Cryptography. Метод виконує рендеринг відповідної панелі за точками та вказаними параметрами для правильного відображення контенту у PDF файлі.

Висновки до розділу

У другому розділі розглянуті: ієрархія класів, ініціалізація даних, функціональна схема програми та опис основних функціональних вікон програми, їх призначення та властивості.

Крім того, розроблено інтерфейс програми і детально описано керівництво користувача системи.

ВИСНОВКИ

Курсовий проєкт спрямований на створення WPF додатку «Розрахунок матриці долі». Розглянуті базові принципи ООП, проаналізована предметна область та розглянуто технічне завдання, реалізовано можливість вводу даних користувача з перевіркою на їх валідність, можливість вибору матриці для розрахунку, перегляд матриці та її розширенний опис з можливістю збереження у PDF форматі.

Для виконання даного проєкту була використана мова програмування C#, яка добре підходить для розробки програм з використанням принципів ООП.

Важливим етапом розробки стали планування й реалізація користувальницького інтерфейсу. Основною метою було задоволення ряду потреб: комфортність роботи з інтерфейсом, швидкодія при обміні даних, ефективність управління, реалізація необхідних функцій. Використання приємних для більшості людей кольорів, декількох вікон, а також захист від способів некоректного введення.

Таким чином, у курсового проєкту продемонстровані найважливіші методи роботи з програмним середовищем WPF, його інструментами та функціями.

Як результат виконання був створений багатовіконний WPF додаток “Матриця долі”

СПИСОК ЛІТЕРАТУРИ

1. Документація Microsoft. URL: https://docs.microsoft.com/
2. Паттерн MVVM C#. URL: https://metanit.com/sharp/wpf/22.1.php
3. Інтернет ресурс C# Corner URL: https://www.c-sharpcorner.com/
4. Офіційна документація Microsoft для WPF: URL: https://learn.microsoft.com/ en-us/dotnet/desktop/wpf/?view=netdesktop-8.0
5. Інтернет ресурс GitHub. URL: https://github.com/miptleha/dotnet\_dev

Додаток А  
класи Model

**Клас Matrix\_Individual**

public class Matrix\_Individual : INotifyPropertyChanged

{

public int \_Date\_Day;

private int \_Date\_Month;

private int \_Date\_Year;

public int Date\_Day

{

get { return \_Date\_Day; }

set

{

\_Date\_Day = value;

OnPropertyChanged("Date\_Day");

}

}

public int Date\_Month

{

get

{

return \_Date\_Month;

}

set

{

\_Date\_Month = value;

OnPropertyChanged("Date\_Month");

}

}

public int Date\_Year

{

get { return \_Date\_Year; }

set

{

\_Date\_Year = value;

OnPropertyChanged("Date\_Year");

}

}

public int ParseCenter()

{

int center = Date\_Month;

if (Date\_Day > 22)

{

center += Date\_Day / 10 + Date\_Day % 10;

}

else

{

center += Date\_Day;

}

string parsedYear = (Date\_Year / 1000 + (Date\_Year / 100) % 10 + (Date\_Year / 10) % 10 + Date\_Year % 10).ToString();

int sum = 0;

foreach (char digit in parsedYear)

{

sum += int.Parse(digit.ToString());

}

center += sum;

center \*= 2;

if (center > 22) center = center / 10 + center % 10;

return center;

}

public int[] ParseTalents()

{

int[] talents = new int[3];

talents[0] = Date\_Month;

talents[1] = talents[0] + ParseCenter();

if (talents[1] > 22) talents[1] = talents[1] / 10 + talents[1] % 10;

talents[2] = talents[1] + talents[0];

if (talents[2] > 22) talents[2] = talents[2] / 10 + talents[2] % 10;

return talents;

}

public int[] ParsePortret()

{

string Day = \_Date\_Day.ToString();

int[] portret = new int[3];

if (int.Parse(Day) > 22)

{

portret[0] = int.Parse(Day[0].ToString()) + int.Parse(Day[1].ToString());

}

else

{

portret[0] += int.Parse(Day);

}

portret[1] = portret[0] + ParseCenter();

if (portret[1] > 22) portret[1] = portret[1] / 10 + portret[1] % 10;

portret[2] = portret[1] + portret[0];

if (portret[2] > 22) portret[2] = portret[2] / 10 + portret[2] % 10;

return portret;

}

public int[] ParseFinance()

{

string year = \_Date\_Year.ToString();

int[] finance = new int[3];

int sumYear = year.Sum(digit => int.Parse(digit.ToString()));

finance[0] = sumYear;

if (finance[0] > 22) finance[0] = finance[0] / 10 + finance[0] % 10;

finance[1] = finance[0] + ParseCenter();

if (finance[1] > 22) finance[1] = finance[1] / 10 + finance[1] % 10;

finance[2] = finance[1] + finance[0];

if (finance[2] > 22) finance[2] = finance[2] / 10 + finance[2] % 10;

return finance;

}

public int[] ParseTale()

{

int[] tale = new int[3];

tale[0] = ParseTalents()[0] +

ParsePortret()[0] + ParseFinance()[0];

if (tale[0] > 22) tale[0] = tale[0] / 10 + tale[0] % 10;

tale[1] = tale[0] + ParseCenter();

if (tale[1] > 22) tale[1] = tale[1] / 10 + tale[1] % 10;

tale[2] = tale[1] + tale[0];

if (tale[2] > 22) tale[2] = tale[2] / 10 + tale[2] % 10;

return tale;

}

public event PropertyChangedEventHandler PropertyChanged;

public void OnPropertyChanged([CallerMemberName] string prop = "")

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(prop));

}

}

**Клас Matrix\_Compatibility**

public class Matrix\_Сompatibility

{

public Matrix\_Individual first\_person { get; set; }

public Matrix\_Individual second\_person { get; set; }

public Matrix\_Сompatibility(Matrix\_Individual? first\_person, Matrix\_Individual? second\_person)

{

this.first\_person = first\_person;

this.second\_person = second\_person;

}

public int ParseCenter()

{

int center = first\_person.ParseCenter() +

second\_person.ParseCenter();

if (center > 22)

{

string temp = center.ToString();

center = int.Parse(temp[0].ToString()) +

int.Parse(temp[1].ToString());

}

return center;

}

public int[] ParseTalents()

{

int[] talents = new int[3];

talents[0] = first\_person.ParseTalents()[0] + second\_person.ParseTalents()[0];

talents[0] = CheckValid(talents[0]);

talents[1] = talents[0] + ParseCenter();

talents[1] = CheckValid(talents[1]);

talents[2] = talents[1] + talents[0];

talents[2] = CheckValid(talents[2]);

return talents;

}

public int[] ParsePortret()

{

int[] portret = new int[3];

portret[0] = first\_person.ParsePortret()[0] + second\_person.ParsePortret()[0];

portret[0] = CheckValid(portret[0]);

portret[1] = portret[0] + ParseCenter();

portret[1] = CheckValid(portret[1]);

portret[2] = portret[1] + portret[0];

portret[2] = CheckValid(portret[2]);

return portret;

}

public int[] ParseFinance()

{

int[] finance = new int[3];

finance[0] = first\_person.ParseFinance()[0] + second\_person.ParseFinance()[0];

finance[0] = CheckValid(finance[0]);

finance[1] = finance[0] + ParseCenter();

finance[1] = CheckValid(finance[1]);

finance[2] = finance[1] + finance[0];

finance[2] = CheckValid(finance[2]);

return finance;

}

public int[] ParseTale()

{

int[] tale = new int[3];

tale[0] = first\_person.ParseTale()[0] + second\_person.ParseTale()[0];

tale[0] = CheckValid(tale[0]);

tale[1] = tale[0] + ParseCenter();

tale[1] = CheckValid(tale[1]);

tale[2] = tale[0] + tale[1];

tale[2] = CheckValid(tale[2]);

return tale;

}

public int CheckValid(int it)

{

if (it > 22) it = it / 10 + it % 10;

return it;

}

}

**Клас RelayCommand**

public class RelayCommand : ICommand

{

private readonly Action<object> execute;

private readonly Predicate<object> canExecute;

private Action saveToPdf;

public RelayCommand(Action<object> execute, Predicate<object> canExecute = null)

{

this.execute = execute ?? throw new ArgumentNullException(nameof(execute));

this.canExecute = canExecute;

}

public RelayCommand(Action saveToPdf)

{

this.saveToPdf = saveToPdf;

}

public bool CanExecute(object parameter)

{

return canExecute == null || canExecute(parameter);

}

public void Execute(object parameter)

{

execute(parameter);

}

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

}

**Клас SPRelayCommand**

public class SPRelayCommand<T> : ICommand

{

private readonly Action<T> \_execute;

private readonly Predicate<T> \_canExecute;

public SPRelayCommand(Action<T> execute, Predicate<T> canExecute = null)

{

\_execute = execute ?? throw new ArgumentNullException(nameof(execute));

\_canExecute = canExecute;

}

public bool CanExecute(object parameter)

{

return \_canExecute == null || \_canExecute((T)parameter);

}

public void Execute(object parameter)

{

\_execute((T)parameter);

}

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

}

**Клас DataService**

public class DataService : INotifyPropertyChanged

{

public Matrix\_Individual matrix1 { get; set; }

public Matrix\_Individual matrix2 { get; set; }

public int \_index { get; set; }

public string \_user\_name { get; set; }

public class Matrix\_Indivdual

{

public int MyProperty { get; set; }

}

public int Index

{

get { return \_index; }

set

{

if (\_index != value)

{

\_index = value;

OnPropertyChanged(nameof(Index));

}

}

}

public string User\_Name

{

get { return \_user\_name; }

set

{

if (\_user\_name != value)

{

\_user\_name = value;

OnPropertyChanged(nameof(User\_Name));

}

}

}

public Info InfoFrame { get; set; }

public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged(string propertyName)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

}

**Клас Arcane**

public class Arcane

{

public int A\_Num { get; set; }

public string? A\_Name { get; set; }

public string? A\_General\_Description { get; set; }

public string? A\_Detailed\_Description { get; set; }

public string? A\_Path\_Image { get; set; }

public Arcane(int number, string name, string general, string detailed, string path)

{

A\_Num = number;

A\_Name = name;

A\_General\_Description = general;

A\_Detailed\_Description = detailed;

A\_Path\_Image = path;

}

public Arcane() { }

public override string ToString()

{

return $"{A\_Num}:{A\_Name}:\n{A\_General\_Description}";

}

}

Додаток Б  
класи VievModel

**Клас ViewModel**

public class ViewModel : INotifyPropertyChanged

{

private DataService \_dataService;

private int \_selectedIndex;

private Arcane \_selectedArcane;

private string \_user\_name;

private Frame \_frame;

public RelayCommand \_windowDescription;

private RelayCommand \_calculateCommand;

public ObservableCollection<Arcane> Arcanes { get; set; }

public ObservableCollection<string> ComboBoxItems { get; set; }

public ViewModel(DataService dataService)

{

Arcanes = ReadFromJsonFile();

ComboBoxItems = new ObservableCollection<string> {

"Індивідуальна матриця долі",

"Матриця сумісності"

};

\_dataService = dataService;

dataService.matrix1 = new Matrix\_Individual();

dataService.matrix2 = new Matrix\_Individual();

}

// Приймає значення індексу комбобокса

public int SelectedIndex

{

get => \_selectedIndex;

set

{

\_selectedIndex = value;

OnPropertyChanged("SelectedIndex");

OnPropertyChanged("IsSecondThreeVisible");

OnPropertyChanged("IsUserNameNeed");

}

}

public Arcane SelectedArcane

{

get { return \_selectedArcane; }

set

{

\_selectedArcane = value;

OnPropertyChanged("SelectedArcane");

}

}

// Приймає значення ім'я користувача

public string User\_Name

{

get => \_user\_name;

set

{

\_user\_name = value;

OnPropertyChanged("User\_Name");

}

}

public Frame Frame

{

get => \_frame;

set

{

\_frame = value;

OnPropertyChanged("Frame");

}

}

public RelayCommand WindowDescription

{

get

{

return \_windowDescription ?? (\_windowDescription = new RelayCommand(obj =>

{

if (SelectedArcane != null)

{

Description description = new Description(this);

description.Show();

}

else

{

MessageBox.Show("Оберіть правильний аркан!");

}

}

));

}

}

// Команда для кнопки, яка виконує перехід на нове вікно

public RelayCommand CalculateCommand

{

set { \_calculateCommand = value; }

get

{

return \_calculateCommand ?? (\_calculateCommand = new RelayCommand(obj =>

{

if (SelectedIndex == 0 && ValidateDate(Individual1))

{

\_dataService.User\_Name = \_user\_name;

var vievmodel2 = new ViewModelInformation(\_dataService);

\_dataService.InfoFrame = new Info(\_dataService);

Frame.Content = \_dataService.InfoFrame;

}

else if (SelectedIndex == 1 && ValidateDate(Individual2) && ValidateDate(Individual1))

{

\_dataService.User\_Name = \_user\_name;

var vievmodel2 = new ViewModelInformation(\_dataService);

\_dataService.InfoFrame = new Info(\_dataService);

Frame.Content = \_dataService.InfoFrame;

}

else

{

MessageBox.Show("ENTER RIGHT DATA");

}

}

));

}

}

// Зчитування арканів з .json файла

public static ObservableCollection<Arcane> ReadFromJsonFile()

{

string jsonFromFile = File.ReadAllText("Arcanes.json");

ObservableCollection<Arcane> objects = JsonConvert.DeserializeObject<ObservableCollection<Arcane>>(jsonFromFile);

return objects;

}

public string IsUserNameNeed

{

get

{

if (SelectedIndex == 1)

{

return "Collapsed";

}

else

return "Visible";

}

}

// Метод для виводу полів для матриці сумісності

public string DescriptionArcaneNameNumber

{

get

{

return SelectedArcane.A\_Num + " : " + SelectedArcane.A\_Name;

}

}

public string DescriptionArcaneGeneral

{

get

{

return SelectedArcane.A\_General\_Description;

}

}

public string DescriptionArcaneDetailed

{

get

{

return SelectedArcane.A\_Detailed\_Description;

}

}

public string DescriptionArcaneImage

{

get

{

return SelectedArcane.A\_Path\_Image;

}

}

public string IsSecondThreeVisible

{

get

{

if (SelectedIndex == 1)

{

\_dataService.Index = 1;

return "Visible";

}

else

\_dataService.Index = 0;

return "Collapsed";

}

}

public Matrix\_Individual Individual1

{

get { return \_dataService.matrix1; }

set

{

\_dataService.matrix1 = value;

OnPropertyChanged("Matrix\_Individual");

}

}

public Matrix\_Individual Individual2

{

get { return \_dataService.matrix2; }

set

{

\_dataService.matrix2 = value;

OnPropertyChanged("Matrix\_Individual");

}

}

// Функція, яка перевіряє валідність, введених у об'єкт класу, даних

public bool ValidateDate(Matrix\_Individual data)

{

if (data.Date\_Month < 1 || data.Date\_Month > 12)

{

return false;

}

int daysInMonth = DateTime.DaysInMonth(data.Date\_Year, data.Date\_Month);

if (data.Date\_Day < 1 || data.Date\_Day > daysInMonth)

{

return false;

}

if (data.Date\_Year > 9999 || data.Date\_Year < 1000)

{

return false;

}

return true;

}

public event PropertyChangedEventHandler PropertyChanged;

public void OnPropertyChanged([CallerMemberName] string prop = "")

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(prop));

}

}

**Клас ViewModelInformation**

public class ViewModelInformation : INotifyPropertyChanged

{

private DataService \_dataService;

private Matrix\_Individual \_matrix1;

private Matrix\_Individual \_matrix2;

private Matrix\_Сompatibility \_matrix3;

private List<Arcane> \_arcanes;

private int \_index;

public int[] finance1;

public int[] portret1;

public int[] talents1;

public int[] tale1;

public int center1;

public int[] finance2;

public int[] portret2;

public int[] talents2;

public int[] tale2;

public int center2;

public ViewModelInformation(DataService dataService)

{ \_dataService = dataService;

\_index = \_dataService.\_index; \_arcanes = ReadFromJsonFile(); \_matrix1 = \_dataService.matrix1;

\_matrix2 = \_dataService.matrix2;

\_matrix3 = new Matrix\_Сompatibility(\_dataService.matrix1, \_dataService.matrix2);

\_user\_name = \_dataService.User\_Name;

GetData();

}

// Отримання данних з використанням інтерфейсу INotifyPropertyChanged

private string \_user\_name;

public string User\_Name

{

get { return "Ім'я користувача: " + \_user\_name; }

set

{

\_user\_name = value;

OnPropertyChanged("User\_Name");

}

}

private StackPanel \_matrix;

public StackPanel Matrix

{

get => \_matrix;

set

{

\_matrix = value;

OnPropertyChanged(nameof(Matrix));

}

}

private Button \_PDF\_button;

public Button PDF\_button

{

get => \_PDF\_button;

set

{

\_PDF\_button = value;

OnPropertyChanged(nameof(PDF\_button));

}

}

private Button \_Back\_ToMain;

public Button Back\_ToMain

{

get => \_Back\_ToMain;

set

{

\_Back\_ToMain = value;

OnPropertyChanged(nameof(Back\_ToMain));

}

}

private SPRelayCommand<StackPanel> \_saveToPdfCommand;

public SPRelayCommand<StackPanel> Save\_To\_PDF

{

get

{

return \_saveToPdfCommand ?? (\_saveToPdfCommand = new SPRelayCommand<StackPanel>(obj =>

{

PDF\_button.Visibility = Visibility.Collapsed;

Back\_ToMain.Visibility = Visibility.Collapsed;

SaveToPdf(Matrix);

PDF\_button.Visibility = Visibility.Visible;

Back\_ToMain.Visibility = Visibility.Visible;

}));

}

}

private RelayCommand \_backToMainWidow;

public RelayCommand BackToMainWindow

{

get

{

return \_backToMainWidow ?? (\_backToMainWidow = new RelayCommand(obj =>

{

\_dataService.InfoFrame.Content = null;

}));

}

}

// Прив'язка даних

// Текстблоки

public string TextBlockCenter

{

get

{

bool exists = \_arcanes.Any(x => x.A\_Num == int.Parse(ContentCen));

if (exists)

return "\n" + \_arcanes[int.Parse(ContentCen) - 1].ToString();

else return "Error";

}

}

public string TextBlockTalents

{

get

{

bool exists1 = \_arcanes.Any(x => x.A\_Num == int.Parse(ContentTalents0));

bool exists2 = \_arcanes.Any(x => x.A\_Num == int.Parse(ContentTalents1));

if (exists1 && exists2)

return "\n" + \_arcanes[int.Parse(ContentTalents0) - 1].ToString() +

"\n" + \_arcanes[int.Parse(ContentTalents1) - 1].ToString();

else return "Error";

}

}

public string TextBlockPortret

{

get

{

bool exists1 = \_arcanes.Any(x => x.A\_Num == int.Parse(ContentPor0));

bool exists2 = \_arcanes.Any(x => x.A\_Num == int.Parse(ContentPor1));

if (exists1 && exists2)

return "\n" + \_arcanes[int.Parse(ContentPor0) - 1].ToString() +

"\n" + \_arcanes[int.Parse(ContentPor1) - 1].ToString();

else return "Error";

}

}

public string TextBlockFinance

{

get

{

bool exists1 = \_arcanes.Any(x => x.A\_Num == int.Parse(ContentFin0));

bool exists2 = \_arcanes.Any(x => x.A\_Num == int.Parse(ContentFin1));

if (exists1 && exists2)

return "\n" + \_arcanes[int.Parse(ContentFin0) - 1].ToString() +

"\n" + \_arcanes[int.Parse(ContentFin1) - 1].ToString();

else return "Error";

}

}

public string TextBlockTale

{

get

{

bool exists1 = \_arcanes.Any(x => x.A\_Num == int.Parse(ContentTale0));

bool exists2 = \_arcanes.Any(x => x.A\_Num == int.Parse(ContentTale1));

if (exists1 && exists2)

return "\n" + \_arcanes[int.Parse(ContentTale0) - 1].ToString() +

"\n" + \_arcanes[int.Parse(ContentTale1) - 1].ToString();

else return "Error";

}

}

// Лейбли

public string ContentFin0

{

get

{

if (\_index == 0) return finance1[0].ToString();

else return finance2[0].ToString(); }

set

{

string s;

if (\_index == 0) s = finance1[0].ToString();

else s = finance2[0].ToString();

s = value;

OnPropertyChanged(s);

}

}

public string ContentFin1

{

get

{

if (\_index == 0) return finance1[1].ToString();

else return finance2[1].ToString(); }

set

{

string s;

if (\_index == 0) s = finance1[1].ToString();

else s = finance2[1].ToString();

s = value;

OnPropertyChanged(s);

}

}

public string ContentFin2

{

get

{

if (\_index == 0) return finance1[2].ToString();

else return finance2[2].ToString(); }

set

{

string s;

if (\_index == 0) s = finance1[2].ToString();

else s = finance2[2].ToString();

s = value;

OnPropertyChanged(s);

}

}

public string ContentPor0

{

get

{

if (\_index == 0) return portret1[0].ToString();

else return portret2[0].ToString(); }

set

{

string s;

if (\_index == 0) s = portret1[0].ToString();

else s = portret2[0].ToString();

s = value;

OnPropertyChanged(s);

}

}

public string ContentPor1

{

get

{

if (\_index == 0) return portret1[1].ToString();

else return portret2[1].ToString(); }

set

{

string s;

if (\_index == 0) s = portret1[1].ToString();

else s = portret2[1].ToString();

s = value;

OnPropertyChanged(s);

}

}

public string ContentPor2

{

get

{

if (\_index == 0) return portret1[2].ToString();

else return portret2[2].ToString(); }

set

{

string s;

if (\_index == 0) s = portret1[2].ToString();

else s = portret2[2].ToString();

s = value;

OnPropertyChanged(s);

}

}

public string ContentTalents0

{

get

{

if (\_index == 0) return talents1[0].ToString();

else return talents2[0].ToString(); }

set

{

string s;

if (\_index == 0) s = talents1[0].ToString();

else s = talents2[0].ToString();

s = value;

OnPropertyChanged(s);

}

}

public string ContentTalents1

{

get

{

if (\_index == 0) return talents1[1].ToString();

else return talents2[1].ToString(); }

set

{

string s;

if (\_index == 0) s = talents1[1].ToString();

else s = talents2[1].ToString();

s = value;

OnPropertyChanged(s);

}

}

public string ContentTalents2

{

get

{

if (\_index == 0) return talents1[2].ToString();

else return talents2[2].ToString(); }

set

{

string s;

if (\_index == 0) s = talents1[2].ToString();

else s = talents2[2].ToString();

s = value;

OnPropertyChanged(s);

}

}

public string ContentTale0

{

get

{

if (\_index == 0) return tale1[0].ToString();

else return tale2[0].ToString(); }

set

{

string s;

if (\_index == 0) s = tale1[0].ToString();

else s = tale2[0].ToString();

s = value;

OnPropertyChanged(s);

}

}

public string ContentTale1

{

get

{

if (\_index == 0) return tale1[1].ToString();

else return tale2[1].ToString(); }

set

{

string s;

if (\_index == 0) s = tale1[1].ToString();

else s = tale2[1].ToString();

s = value;

OnPropertyChanged(s);

}

}

public string ContentTale2

{

get

{

if (\_index == 0) return tale1[2].ToString();

else return tale2[2].ToString(); }

set

{

string s;

if (\_index == 0) s = tale1[2].ToString();

else s = tale2[2].ToString();

s = value;

OnPropertyChanged(s);

}

}

public string ContentCen

{

get

{

if (\_index == 0) return center1.ToString();

else return center2.ToString();

}

set

{

string s;

if (\_index == 0) s = center1.ToString();

else s = center2.ToString();

s = value;

OnPropertyChanged(s);

}

} public string LeftTop

{

get

{ if (\_index == 0) return CheckValid((talents1[0] + portret1[0])).ToString();

else return CheckValid((talents2[0] + portret2[0])).ToString();

}

set

{

string s;

if (\_index == 0) s = CheckValid((talents1[0] + portret1[0])).ToString();

else s = CheckValid((talents2[0] + portret2[0])).ToString();

s = value;

OnPropertyChanged(s);

}

}

public string RightTop

{

get

{ if (\_index == 0) return CheckValid((talents1[0] + finance1[0])).ToString();

else return CheckValid((talents2[0] + finance2[0])).ToString();

}

set

{

string s;

if (\_index == 0) s = CheckValid((talents1[0] + finance1[0])).ToString();

else s = CheckValid((talents2[0] + finance1[0])).ToString();

s = value;

OnPropertyChanged(s);

}

}

public string RightBottom

{

get

{ if (\_index == 0) return CheckValid((tale1[0] + finance1[0])).ToString();

else return CheckValid((tale2[0] + finance2[0])).ToString();

}

set

{

string s;

if (\_index == 0) s = CheckValid((tale1[0] + finance1[0])).ToString();

else s = CheckValid((tale2[0] + finance2[0])).ToString();

s = value;

OnPropertyChanged(s);

}

}

public string LeftBottom

{

get

{ if (\_index == 0) return CheckValid((tale1[0] + portret1[0])).ToString();

else return CheckValid((tale2[0] + portret2[0])).ToString();

}

set

{

string s;

if (\_index == 0) s = CheckValid((tale1[0] + portret1[0])).ToString();

else s = CheckValid((tale2[0] + portret2[0])).ToString();

s = value;

OnPropertyChanged(s);

}

}

// Видимість поля для користувача

public string IsUsernameNeed

{

get

{

if (\_index == 0 && \_user\_name != null)

{

return "Visible";

}

else

return "Collapsed";

}

}

// Функціональні методи

public static void SaveToPdf(StackPanel panel)

{

try

{

double startX = panel.ActualWidth / 2;

double startY = panel.ActualHeight / 2;

double width = panel.ActualWidth - startX;

double height = panel.ActualHeight - startY;

Rect rect = new Rect(startX, startY, width, height); RenderTargetBitmap renderBitmap = new RenderTargetBitmap(

(int)(rect.Width \* 4.01), (int)(rect.Height \* 4.01),

384d, 384d, PixelFormats.Pbgra32); DrawingVisual drawingVisual = new DrawingVisual();

using (DrawingContext drawingContext = drawingVisual.RenderOpen())

{

VisualBrush visualBrush = new VisualBrush(panel);

drawingContext.DrawRectangle(visualBrush, null, new Rect(new Point(), rect.Size));

} renderBitmap.Render(drawingVisual);

PngBitmapEncoder encoder = new PngBitmapEncoder();

encoder.Frames.Add(BitmapFrame.Create(renderBitmap)); using (MemoryStream ms = new MemoryStream())

{

encoder.Save(ms);

ms.Position = 0; Document document = new Document(new iTextSharp.text.Rectangle(0, 0, (float)rect.Width, (float)rect.Height), 0, 0, 0, 0);

PdfWriter writer = PdfWriter.GetInstance(document, new FileStream("Matrix.pdf", FileMode.Create));

document.Open(); iTextSharp.text.Image pdfImage = iTextSharp.text.Image.GetInstance(ms);

pdfImage.ScaleAbsolute((float)rect.Width, (float)rect.Height); document.Add(pdfImage);

document.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

}

} public static List<Arcane> ReadFromJsonFile()

{

string jsonFromFile = File.ReadAllText("Arcanes.json");

List<Arcane> objects = JsonConvert.DeserializeObject<List<Arcane>>(jsonFromFile);

return objects;

}

public static int CheckValid(int it)

{

if (it > 22) it = it / 10 + it % 10;

return it;

} public void GetData()

{

if (\_index == 0)

{

finance1 = \_matrix1.ParseFinance();

portret1 = \_matrix1.ParsePortret();

talents1 = \_matrix1.ParseTalents();

tale1 = \_matrix1.ParseTale();

center1 = \_matrix1.ParseCenter();

}

else

{

finance2 = \_matrix3.ParseFinance();

portret2 = \_matrix3.ParsePortret();

talents2 = \_matrix3.ParseTalents();

tale2 = \_matrix3.ParseTale();

center2 = \_matrix3.ParseCenter();

}

} public event PropertyChangedEventHandler PropertyChanged;

public void OnPropertyChanged([CallerMemberName] string prop = "")

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(prop));

}

}

Додаток В  
Тестування додатку

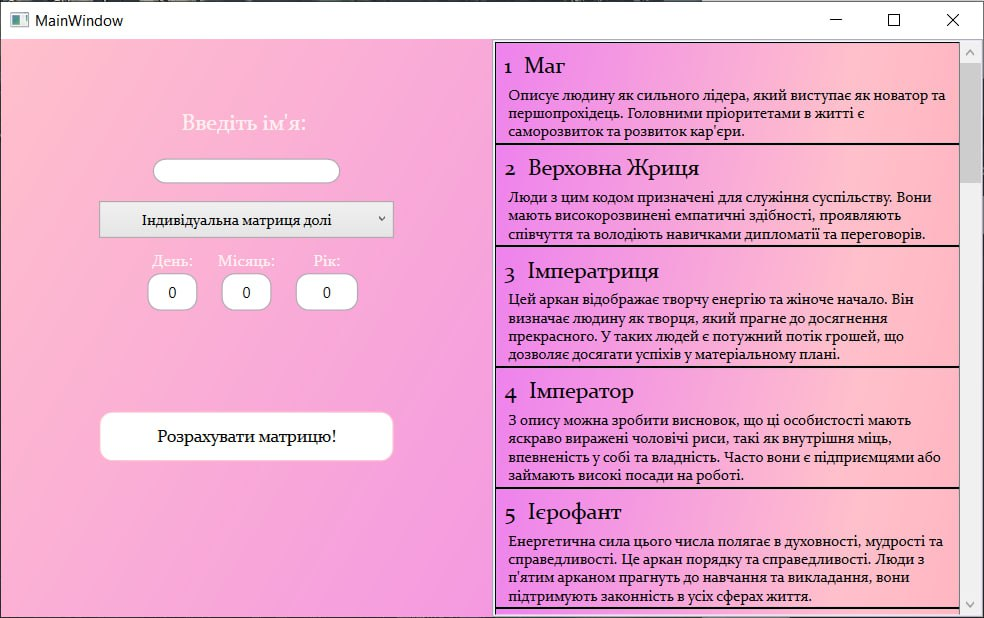


Рисунок В.1 – початкове вікно

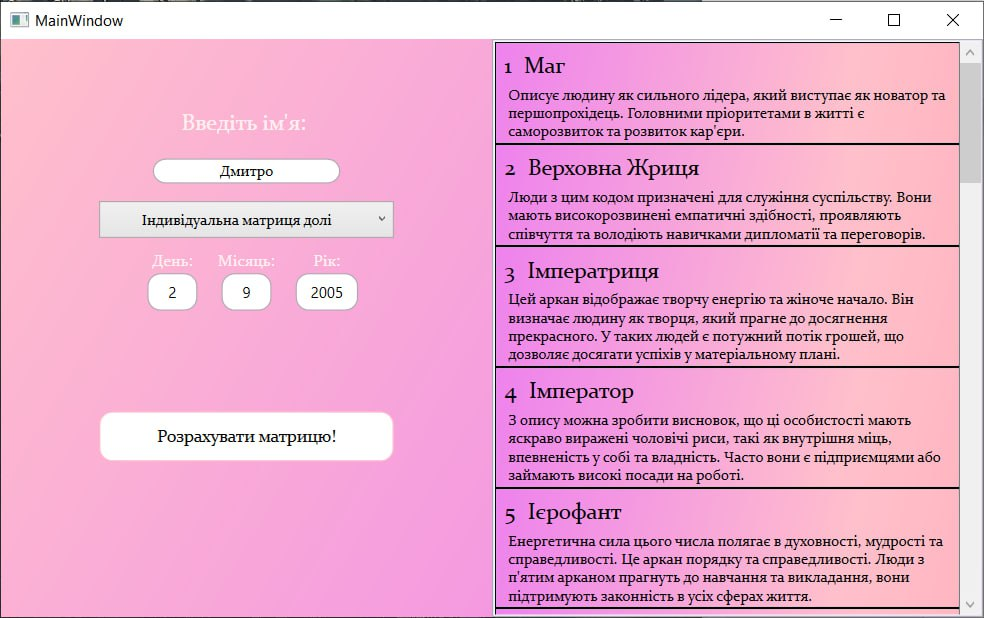


Рисунок В.2 – початкове вікно індивідуальна матриця

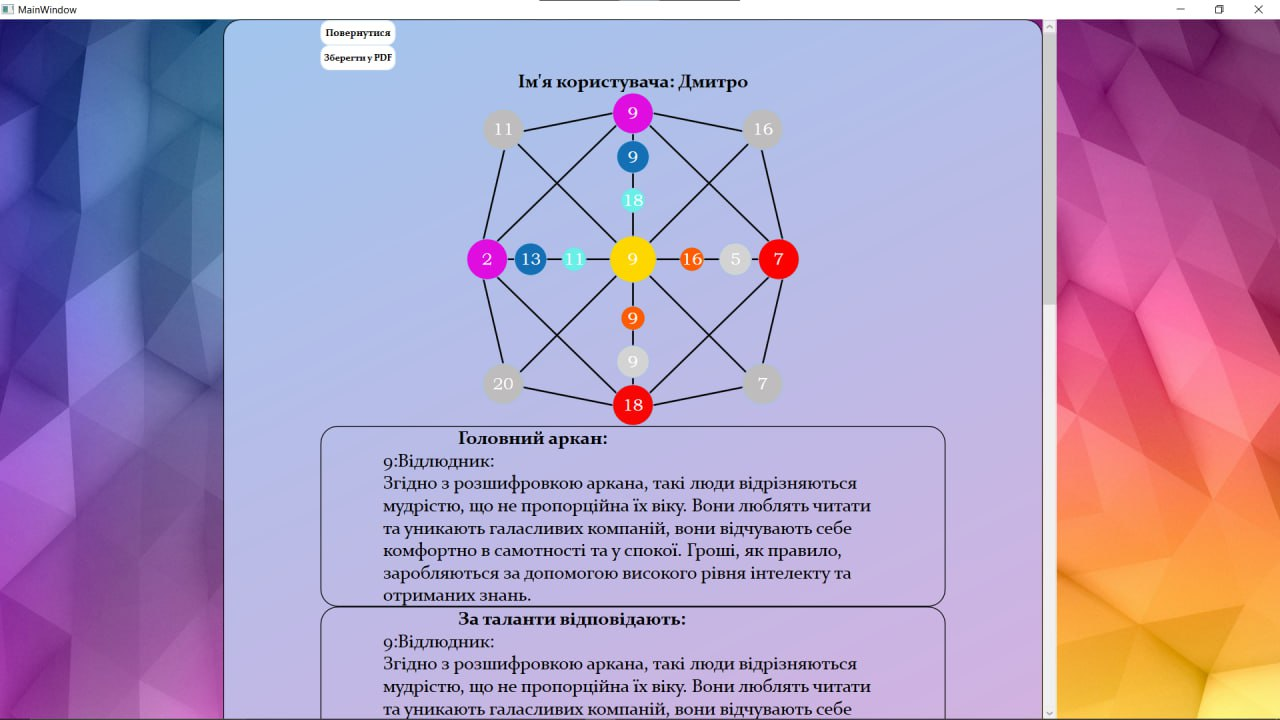


Рисунок В.3 – вікно інформації



Рисунок В.4 – вікно детальної інформації

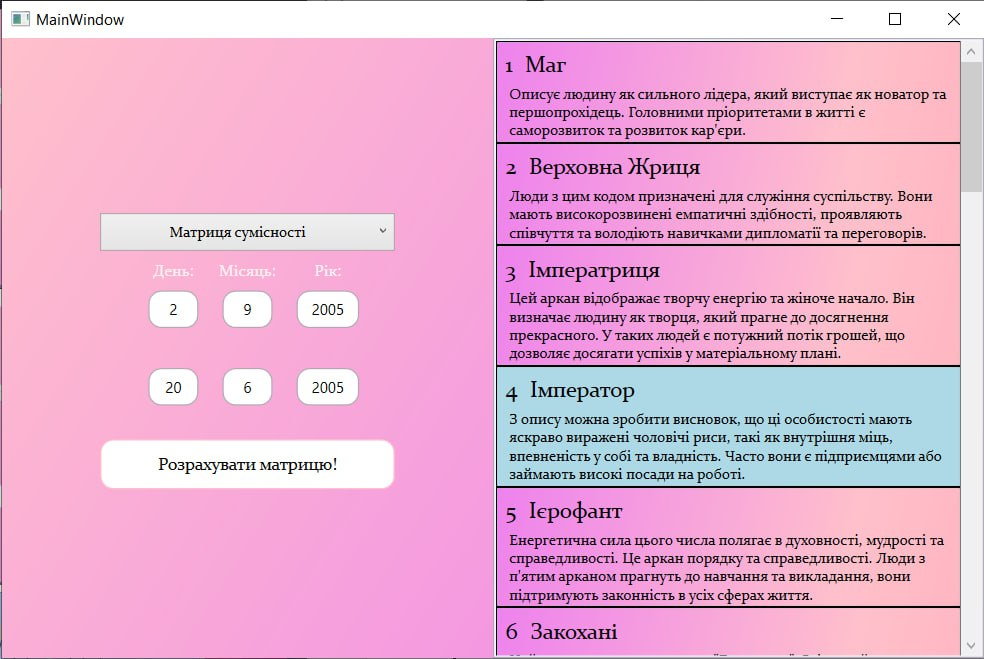


Рисунок В.5 – початкове вікно матриця сумісності

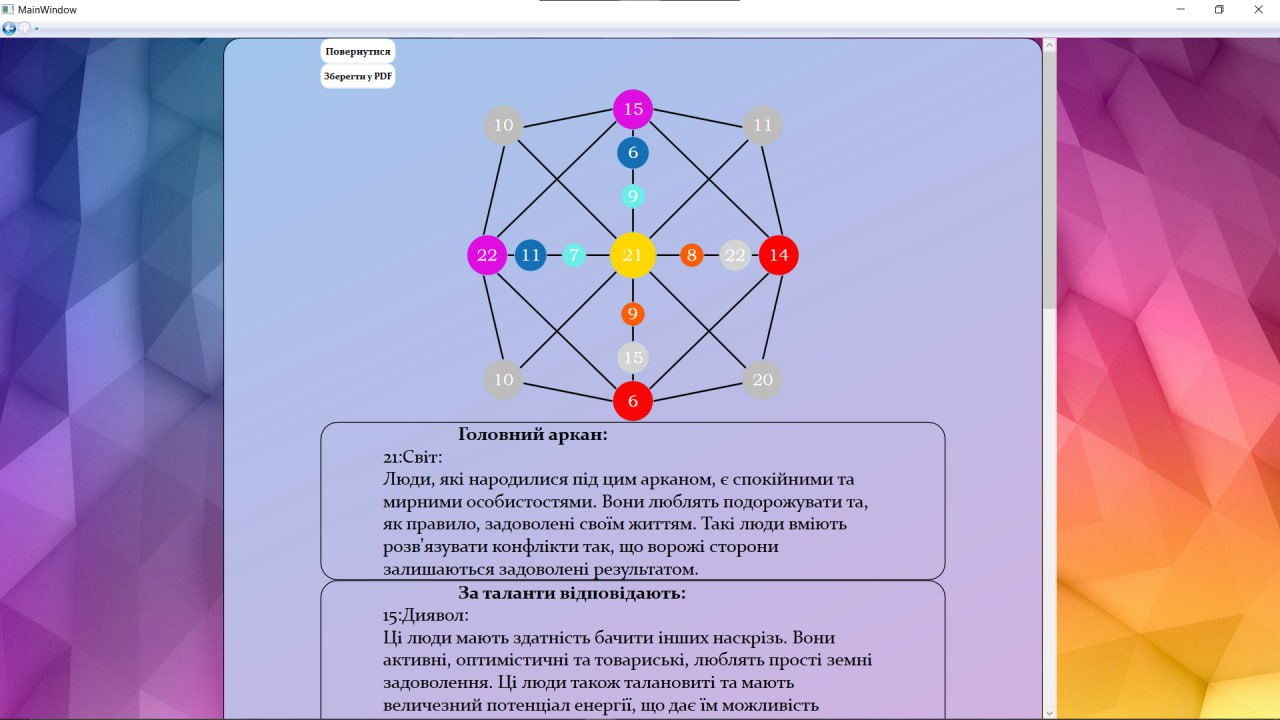


Рисунок В.6 – вікно інформації матриця сумісності

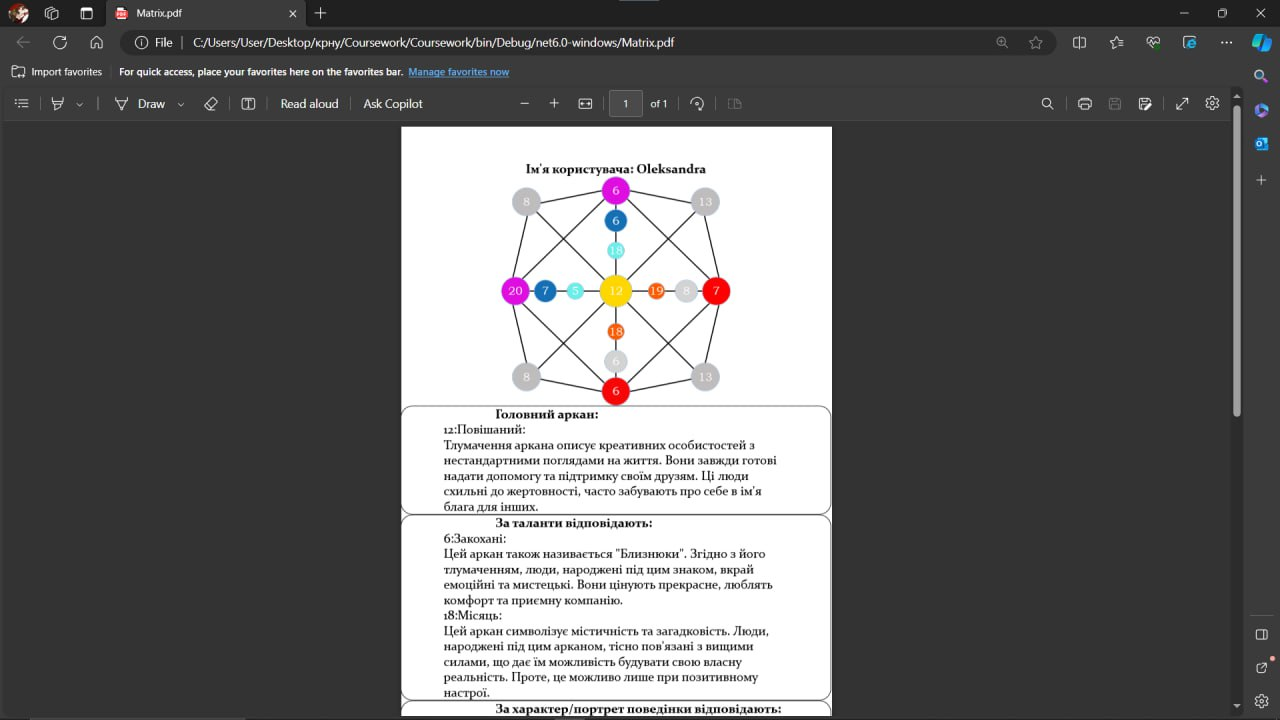


Рисунок В.7 – збережена у PDF матриця

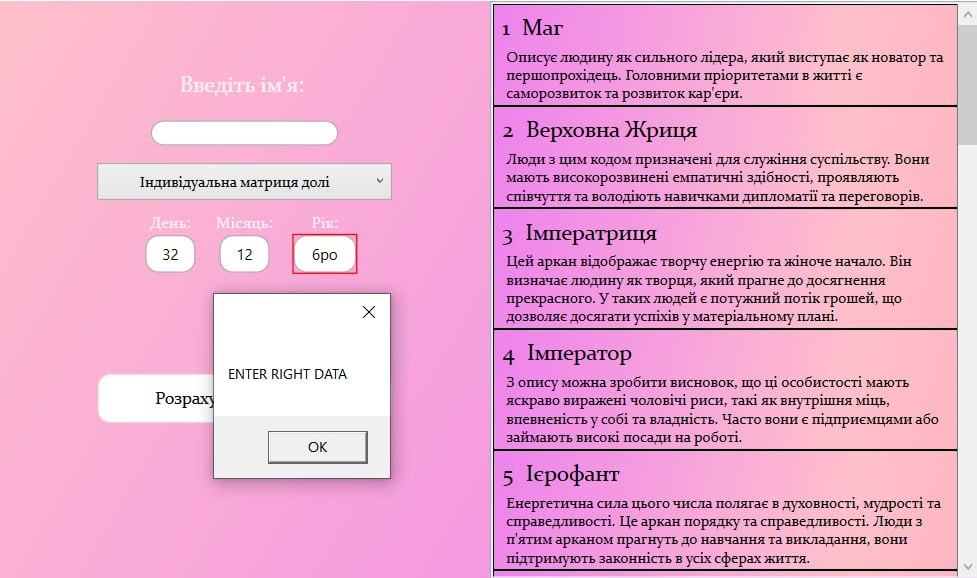


Рисунок В.8 – помилка при вводі невалідних даних