**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ**

Навчально-науковий інститут Інформаційних технологій

Кафедра інженерії програмного забезпечення

**Пояснювальна записка**

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: «**Розробка автоматизованої системи пост-реєстраційної обробки користувачів на мові С#**»

Виконав: студент 4 курсу, групи ПД-44 спеціальності

121 Інженерія програмного забезпечення

(шифр і назва спеціальності)

Скуратовський А.В.

(прізвище та ініціали)

Керівник Дібрівний О.А.

(прізвище та ініціали)

Рецензент

(прізвище та ініціали)

Київ – 2022

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ**

**Навчально-науковий інститут інформаційних технологій**

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти – «Бакалавр»

Напрям підготовки – 121 – Інженерія програмного забезпечення

**ЗАТВЕРДЖУЮ**

Завідувач кафедри

Інженерії програмного забезпечення

О.В. Негоденко

“ ” 2022 року

**ЗАВДАННЯ**

**НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ**

Скуратовському Андрію Володимировичу

(прізвище, ім’я, по батькові)

1. Тема роботи: «Розробка автоматизованої системи пост-реєстраційної обробки користувачів на мові C#»

Керівник роботи доцент кафедри, доктор філософії ІПЗ Дібрівний О. А. ,

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від 18. 02. 2022 року № .

1. Строк подання студентом роботи «3» червня 2022 року
2. Вхідні дані до роботи:
   1. Середовище розробки Visual Studio 2019
   2. Науково-технічна література
   3. Windows Presentation Foundation, .NET Core
   4. Ручний алгоритм обробки даних
3. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
   1. Аналіз вимог до системи обробки даних
   2. Аналіз та порівняння існуючи типів систем обробки даних
   3. Дослідження програмних засобів реалізації додатку
   4. Розробити функціонал та інтерфейс відповідно до вимог застосунку
   5. Провести аналіз продуктивності застосунку
4. Перелік графічного матеріалу:
   1. Титульний слайд
   2. Мета, об’єкт та предмет дослідження
   3. Аналоги систем обробки даних
   4. Технічні завдання
   5. Програмні засоби реалізації
   6. Діаграма класів
   7. Діаграма варіантів використання
   8. Апробація результатів дослідження
   9. Висновки
5. Дата видачі завдання «11» квітня 2022 р.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів бакалаврської роботи | Строк виконання етапів роботи | Примітка |
| 1 | Підбір науково-технічної літератури | 11.04.2022 | Виконано |
| 2 | Дослідження існуючих типів систем обробки даних | 12.04 – 14.04.2022 | Виконано |
| 4 | Встановлення вимог до програмного забезпечення та вибір типу системи | 15.04 – 19.04.2022 | Виконано |
| 5 | Проєктування архітектури системи | 20.04 – 25.04.2022 | Виконано |
| 6 | Розробка системи для автоматизації обробки користувачів | 26.04 – 05.05.2022 | Виконано |
| 7 | Висновки, оформлення роботи | 06.05 – 11.05.2022 | Виконано |
| 8 | Розробка демонстраційних матеріалів | 12.05 – 15.05.2022 | Виконано |
| 9 | Попередній захист роботи | 16.05 – 30.06.2022 | Виконано |
| 10 | Здача роботи | 03.06.2022 |  |

Студент Скуратовський А.В.

(підпис) (прізвище та ініціали)

Керівник роботи Дібрівний О.А.

(підпис) (прізвище та ініціали)

# РЕФЕРАТ

Текстова частина дипломної роботи 66 с., 40 рис., 3 таблиці, 8 джерел.

Ключові слова: C#, Visual Studio, автоматизація, користувач, WPF, .NET

*Об’єкт дослідження* – поліпшення та прискорення ланки бізнес-процесу обробки даних користувачів.

*Предмет дослідження* – автоматизована система для перевірки та обробки користувацької інформації.

*Мета роботи* – підвищення ефективності бізнесу процесу пост-реєстраційної обробки даних шляхом розробки автоматизованої системи для обробки імпортованих файлів з даними користувачів.

*Методи дослідження* – методи сортування, групування інформації, методи тестування, валідації та верифікації програмного забезпечення.

Відповідно до поставлено мети, для розв’язання технічної проблеми в роботі необхідно виконати такі завдання:

* Аналіз вимог до систем обробки даних та методів збереження конфіденційності інформації
* Аналіз технічних засобів та вибір оптимальних рішень для розробки додатку автоматизованої системи
* Аналіз алгоритму обробки даних задля коректної програмної реалізації
* Аналіз продуктивності розробляємої автоматизованої системи обробки даних
* На основі результатів виконаних досліджень розробити автоматизовану систему пост-реєстраційної обробки даних користувачів

Упровадження розробленого застосунку підвищить ефективність та швидкість обробки даних користувачів.

В роботі виконано аналіз існуючих типів систем обробки даних.

Проаналізовано можливість середовища розробки Visual Studio. Розроблено логіку додатку та зручність користування інтерфейсом для користувачів.

Особливістю програмного продукту, є повна захищеність конфіденційних даних користувачів: електронна пошта, регіон, тип пристрою, з якого було розпочато процес реєстрації у сервісі. Також автоматизація більш ризикованої частини обробки даних, через присутність людського фактора. Конфіденційність забезпечується відсутністю апаратно запрограмованої змоги збереження даних в будь-яких фізичних, мережевих, хмарних сховищах. Дані зберігаються тимчасово лише в пам’яті додатку для подальшої її обробки. Після обробки та закриття програмного забезпечення, дані, що були збережені до, видаляються без подальшої змоги їх відновлення.

Галузь використання – обробка даних користувачів, що реєструються у системі компанії, яка потребує збереження цілісності та конфіденційності даних своїх користувачів.

**ЗМІСТ**

[РЕФЕРАТ 6](#_Toc102237279)

[ВСТУП 10](#_Toc102237280)

[1 ОГЛЯД ТА АНАЛІЗ ІНСУЮЧИХ СИСТЕМ ОБРОБКИ ДАНИХ 12](#_Toc102237281)

[1.1 Поняття обробки даних 12](#_Toc102237282)

[1.1.1 Збір інформації 13](#_Toc102237283)

[1.1.2 Підготовка 13](#_Toc102237284)

[1.1.3 Введення даних 13](#_Toc102237285)

[1.1.4 Обробка даних 13](#_Toc102237286)

[1.1.5 Відображення 14](#_Toc102237287)

[1.1.6 Зберігання 14](#_Toc102237288)

[1.2 Типи систем обробки даних 14](#_Toc102237289)

[1.2.1 Ручна обробка даних 14](#_Toc102237290)

[1.2.2 Механічна обробка даних 15](#_Toc102237291)

[1.2.3 Електронна обробка даних 15](#_Toc102237292)

[1.3 Аналіз та порівняння систем обробки даних 16](#_Toc102237293)

[2 ПОСТАНОВКА ТЕХНІЧНОГО ЗАВДАННЯ ТА ВИБІР ТИПУ СИСТЕМИ ОБРОБКИ ДАНИХ 20](#_Toc102237294)

[3 ОПИС ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ 23](#_Toc102237295)

[3.1 Visual Studio 23](#_Toc102237296)

[3.2 C# 26](#_Toc102237297)

[3.3 .NET 28](#_Toc102237298)

[3.4 WPF 30](#_Toc102237299)

[3.5 Adobe Photoshop 37](#_Toc102237300)

[4 ОПИС АЛГОРИТМУ ОБРОБКИ ДАНИХ 38](#_Toc102237301)

[5 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ 42](#_Toc102237302)

[5.1 Розробка додаткового модулю очищення файлів 45](#_Toc102237303)

[5.2 Розробка основного модуля обробки Landing файлу 52](#_Toc102237304)

[5.3 Розробка основного модуля обробки Typeform файлу 56](#_Toc102237305)

[5.4 Розробка основного модулю групування та розкидування даних 60](#_Toc102237306)

[5.5 Продуктивність застосунку 63](#_Toc102237307)

[ВИСНОВКИ 65](#_Toc102237308)

[ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 66](#_Toc102237309)

[Додаток A 67](#_Toc102237310)

[Додаток Б 73](#_Toc102237311)

# ВСТУП

*Актуальність роботи*. Нині все ще більша частина інформації обробляється людиною. Державні установи, паспортні сервіси та інше. Все це потребує людських зусиль, часу, настрою та емоційної рівноваги, що в наш час може бути недосяжним повною мірою. Тому через потенційний ріст тривожності та спаданню конвертованості працівників установ, що обробляють дані, відсоток помилок та затримок в обробці даних стрімко зростає. Через необхідність обробки даних у великій кількості, особливо у ланках реєстрацій користувачів соц. мереж, або ж реєстрацій житлового майна чи нерухомості, працівники беруть все більше перерв і все менше мають жаги до роботи. Для розв’язання цієї проблеми можна використовувати спеціальні системи, що зможуть оброблять інформацію замість людей.

Це дозволить мінімізувати ризики при обробці інформації й, що є плюсом, прискорити ланку обробки даних. Критичну зону обробляє додаток, а менш важливу частину задачі виконує сам користувач – працівник.

Для виконання поставленої мети слід виконати наступні завдання:

1. Проаналізувати переваги та недоліки типів систем обробки даних
2. Розробити вимоги до нового застосунку на основі аналізу алгоритму обробки даних і переваг та недоліків існуючих типів систем обробки даних.
3. Проаналізувати технічні засоби, що використовуються для розробки застосунку
4. Спроєктувати та розробити новий застосунок на основі алгоритму обробки даних.

*Об’єкт дослідження* – поліпшення та прискорення ланки бізнес-процесу обробки даних користувачів.

*Предмет дослідження* – автоматизована система для перевірки та обробки користувацької інформації.

*Мета роботи* – підвищення ефективності бізнесу процесу пост-реєстраційної обробки даних шляхом розробки автоматизованої системи для обробки імпортованих файлів з даними користувачів.

*Методи дослідження* – методи сортування, групування інформації, методи тестування, валідації та верифікації програмного забезпечення.

*Практичне значення одержаних результатів*. Система (застосунок) допоможе працівникам обробляти дані значно швидше та зберегти конфіденційність інформації користувачів.

*Особистий внесок*. Вдосконалено алгоритм обробки даних за допомогою скальпування (парсингу) сторінок користувачів.

*Результати роботи*. Матеріали дипломного проєкту можуть сприяти підвищенню ефективності обробки даних користувачів комерційної компанії askfm.

# ОГЛЯД ТА АНАЛІЗ ІНСУЮЧИХ СИСТЕМ ОБРОБКИ ДАНИХ

## Поняття обробки даних

Обробка даних — це будь-який процес, який виконує комп’ютерна програма для введення даних та узагальнення, аналізу чи іншого перетворення даних у інформацію, яку можна використовувати. Процес може бути автоматизований і запущений на комп’ютері. Він включає запис, аналіз, сортування, узагальнення, обчислення, поширення та зберігання даних. Оскільки дані є найбільш корисними, коли вони добре представлені та насправді інформативні, системи обробки даних часто називають інформаційними системами. Однак, терміни є приблизно синонімами, виконуючи подібні перетворення. Системи обробки даних, як правило, маніпулюють необробленими даними в інформацію, а також інформаційні системи зазвичай приймають необроблені дані як вхідні дані, щоб створити інформацію як вихідну інформацію. Обробку даних можна відрізнити або не відрізнити від перетворення даних, коли процес полягає лише в перетворенні даних в інший формат і не передбачає будь-яких маніпуляцій з даними. Компілятор програмного коду є прикладом програмної системи обробки даних. Програмна система обробки даних, як правило, є окремою одиницею програмного забезпечення, оскільки її вихідні дані можуть бути спрямовані на будь-яку кількість інших підсистем обробки інформації.

Цикл обробки даних складається з серії кроків, на яких необроблені дані (вхідні дані) надходять у процес (ЦП) для отримання корисної інформації (виведення). Кожен крок виконується в певному порядку, але весь процес повторюється циклічно. Вихідні дані першого циклу обробки даних можуть бути збережені та подані як вхідні дані для наступного циклу.

Загалом цикл обробки даних складається з шести основних етапів:

### Збір інформації

Збір необроблених даних є першим кроком циклу обробки даних. Тип зібраних необроблених даних має величезний вплив на вироблений продукт. Отже, необроблені дані повинні бути зібрані з визначених і точних джерел, щоб наступні висновки були дійсними та придатними для використання. Необроблені дані можуть включати грошові цифри, файли cookie вебсайту, звіти про прибутки/збитки компанії, поведінку користувачів тощо.

### Підготовка

Підготовка або очищення даних – це процес сортування та фільтрації необроблених даних для видалення непотрібних і неточних даних. Необроблені дані перевіряються на наявність помилок, дублювання, прорахунків або відсутніх даних, і перетворюються у відповідну форму для подальшого аналізу та обробки. Це робиться для того, щоб в блок обробки надходили дані лише найвищої якості.

### Введення даних

На цьому кроці необроблені дані перетворюються в машиночитну форму і надходять у блок обробки. Це може бути у вигляді введення даних за допомогою клавіатури, сканера або будь-якого іншого джерела введення.

### Обробка даних

На цьому етапі необроблені дані піддаються різним методам обробки даних з використанням алгоритмів машинного навчання та штучного інтелекту для отримання бажаного результату. Цей крок може дещо відрізнятися від процесу до процесу залежно від джерела даних, що обробляються (озера даних, онлайн-бази даних, підключені пристрої тощо) та передбачуваного використання результату.

### Відображення

Дані, нарешті, передаються та відображаються користувачеві у вигляді графіків, таблиць, векторних файлів, аудіо, відео, документів тощо. Цей результат можна зберегти та обробити в наступному циклі обробки даних.

### Зберігання

Останнім кроком циклу обробки даних є зберігання, де дані та метадані зберігаються для подальшого використання. Це забезпечує швидкий доступ до інформації, коли це необхідно, а також дозволяє використовувати її як вхідні дані в наступному циклі обробки даних.

## Типи систем обробки даних

Типи обробки даних феноменально зросли зі зростанням вимог від ручної обробки даних до автоматизації.

### Ручна обробка даних

Метод ручної обробки даних – це коли спеціалісти з введення даних, записують та обробляють дані вручну через реєстр, паперові системи запису та інші процеси введення даних вручну. Хоча це один з найперших методів обробки даних, ручне введення даних є дорогим, трудомістким, схильним до помилок і прямопропорційно залежить від людського фактора.

Наприклад бібліотечна система, яка складається з каталогу, що містить відсортовану та впорядковану інформацію про літературу, книжки, енциклопедії, інше, та їх місце зберігання, де література знаходиться на вказаних в каталозі позиціях, але пошук літератури і її доставляння виконується вручну, як приклад може слугувати бібліотека.

### Механічна обробка даних

Механічна система обробляє дані за допомогою механічних пристроїв, таких як друкарські машинки, механічні принтери та інші пристрої. Попри те, що механічна система швидша, ніж метод обробки даних вручну, вона почала зникати разом з майбутніми еволюціями. Наприклад всі перші лічильні машини від машини Шиккарда до арифмометра, машина Жаккарда.

### Електронна обробка даних

У 1980 році, з народженням комп’ютерів, електронна обробка даних (ЕОД) знаменувала своє існування. Комп’ютер безперебійно обробляє дані автоматично за попередньо визначеними інструкціями спеціалістів з даних.

Наприклад, у цей час було поширеним використання електронних таблиць для запису оцінок учнів.

Попри те, що цей метод обробки даних точний, надійний і швидший, ніж його попередник, він все ще потребує спеціалістів з даних для ручного введення даних і обчислень. Поділяється на 2 види :

1. **Автоматизована система обробки даних**

У розв’язанні проблеми обробки даних – інформаційної проблеми, бере участь система і людина. Система може виконувати якусь частину роботу, або повноцінно сама, під керівництвом та наглядом людини. Тобто, програма може обробляти інформацію, але запуск цієї програми виконує людина і покроково відповідно процесу, контролювати і регулювати ті чи інші дії.

Наприклад – виробництво продукції на фабриці. Запрограмована програма виконує алгоритм дій, замість людини – збирає комп’ютерні комплектуючі, або ж ноутбуки. Все це збирається під наглядом людини, якщо буде брак товару, то людина власноруч може виправити, або підправити недолік, усунувши його, прикладом може слугувати встановлення клавіш у рамку клавіатури. Встановлюючи клавішу, може виникнути проблема, що пластик виготовлений не зовсім якісно, і десь буде маленька задирка пластику і кнопка на клавіатурі буде стояти криво-косо. Працівник помітить цей дефект і вставить клавішу вірно, бо вона повинна бути згідно з стандартами.

1. **Автоматична система обробки даних**

Задача виконується без участі людини. Як чорний ящик, користувач передає вхідні дані програмі та програма сама виконує роботу, обробляє, перевіряє інформацію, дані. Користувач на виході отримує 100% готовий результат, який не потрібно доробляти чи знову закидувати дані у програму на повторний цикл. Якщо хоча б 1% потрібно доробляти користувачеві, то це вже автоматизована система.

Прикладом автоматичних систем можуть слугувати: системи протиповітряної оборони, радари, системи самонаведення.

## Аналіз та порівняння систем обробки даних

Отже, розглянувши основні типи інформаційних систем обробки даних, можемо тепер їх порівняти. Для порівняння візьмемо переваги та недоліки кожного варіанту систем. Розпочнемо з самого банального і простого у реалізації і продовжимо більш сучасними варіантами.

Ручний:

Переваги:

* Користувач повністю контролює процес
* Можливість вносити корективи в процес в реальному часі
* Можливість змінювати порядок виконання алгоритму обробки інформації, що дає право пропустити деякі процеси за необхідності.

Недоліки:

* Повний цикл та кожна ланка цього циклу обробки даних займає багато часу
* Ймовірність похибки при обробці дуже висока, що прямопропорційно залежить від людського фактора
* Час обробки даних завжди різниться в великих масштабах, від хвилини до декількох годин
* Якщо обробка даних займає багато часу, наприклад цілий день, то обов’язкова обідня перерва працівника лише збільшить загальний час циклу

Механізований :

Переваги:

* Загальний час циклу обробки менший ніж у ручної системи
* Система частково виконує людську роботу
* Контролюється людиною, тобто робота виконується під пильним наглядом людини

Недоліки:

* Великий ризик браку інформації у разі поломки системи
* Все ще процес майже повністю контролюється від людини
* Швидкість обробки все ще залежить від людини, наприклад від швидкості друку на печатній машинці, або від розумових здібностей для використання та запам’ятовування алгоритму взаємодії механізованої системи
* Відсутність повної адаптивної складової – не завжди буде можливість ввести корективи в процес обробки даних
* Обідня перерва все ще сповільнює обробку

Автоматизований:

Переваги:

* Можливість майже повністю перевести обробку даних в автоматичний режим
* Загальний час обробки значно зменшився в порівнянні з попередніми двома
* Процес можливо зупинити в екстреному випадку
* У разі виникнення помилки, користувач може вносити свої корективи у вихідні дані перед передачею інформації наступній ланці циклу
* Ризик помилки дуже низький
* Обідня перерва може не впливати на результати і тривалість роботи
* Вплив людського фактора майже не спостерігається у процесі обробки

Недоліки:

* Відсутність можливості динамічно вносити корективи у алгоритм обробки, потрібно зупиняти роботу і вносити корективи до початку обробки даних
* Потрібно враховувати ризики зупинки роботи, слід робити будь-яке резервне копіювання даних перед початком роботи
* Загальний час циклу, все ще залежить від людини, частково від людського фактора
* Через не повну автоматизацію процесу обробки, можемо отримати браковані дані на виході
* Вхідні і вихідні дані контролює людина, тобто випадково дані на вході або виході можуть бути подані у не правильному форматі або вигляді

Автоматичний:

Переваги:

* Весь процес обробки повністю автоматизований
* Людський фактор не має впливу на процес обробки
* Обідня перерва суттєво не змінює загальний час циклу обробки
* Цикл обробки в декілька разів прискорився, у порівнянні з ручним чи механізованим варіантом
* Можливість автоматичної передачі вихідних даних наступній ланці процесу
* Присутня можливість використання та(або) комбінування з іншими видами обробки даних

Недоліки:

* Повна відсутність введення коректив в алгоритм роботи
* Неможлива екстрена зупинка, якщо був початий процес обробки, то можливості скасування старту немає, що в у випадку помилкових вхідних даних призводить до пошкодження цілісності циклу обробки даних
* Через можливість помилки у разі скасування, або не вірних вхідних даних, потрібно додатково передбачати можливості виправлення вихідних даних
* Через відсутність екстреної зупинки, загальний час обробки може збільшитись в декілька разів : додатковий час на виправлення помилки та додатковий час на новий прогін циклу обробки.
* Відсутність контролю над ситуацією при обробці

# ПОСТАНОВКА ТЕХНІЧНОГО ЗАВДАННЯ ТА ВИБІР ТИПУ СИСТЕМИ ОБРОБКИ ДАНИХ

**Визначення цілей та вимог додатку**

Спочатку виділимо цілі додатку, задля подальшого коректного визначення вимог до системи.

**Цілі додатку**

1. Прискорення обробки вхідних даних.
2. Часткове або повне виключення впливу людського фактора при обробці вхідних даних
3. Часткова автоматизація процесу обробки та перевірки постреєстраційних даних користувачів

**Визначення вимог**

Встановлені вимоги повинні повністю, повною мірою відповідати цілям додатку. Вимоги встановлені для додатка. На основі наступних вимог додатку можливо буде присвоїти тип інформаційної системи обробки даних цьому додатку. Експлуатаційні та функціональні вимоги:

1. Можливість використання додатку не в повному обсязі – модульність
2. Можливість перетягування вхідних файлів у зону імпорту вхідних файлів
3. Можливість екстреної зупинки роботи додатку
4. Можливість редагування списку країн і відповідних мовних регіонів
5. Можливість обробки двох типів файлів: формату Excel (.xlsx, .xls), та формату текстового файлу – .csv
6. Можливість зміни вихідних даних (файлів) після їх обробки
7. Конфіденційність – будь-які дані, а саме: логіни, електронна пошта, регіон, тип платформи з якої була проведена реєстрація, не повинні зберігатись в будь-якому фізичному / хмарному / тимчасовому / постійному сховищі, окрім тимчасової пам’яті додатку у вигляді оперативної пам’яті пристрою на якому встановлений додаток
8. Після закриття роботи додатку, дані збережені всередині додатку повинні бути знищені, і не підлягають відновленню при будь-яких умовах.
9. Імена вхідних файлів не змінні
10. Перед початком обробки даних програмою, файли повинні пройти етап резервного копіювання, в той же каталог (папка) системи де містяться вхідні файли
11. Алгоритм перевірки та обробки даних програми, повинен в точності відображати та виконувати ручний варіант алгоритму обробки та перевірки даних.
12. Цілісність – дані, що не вимагають редагування, зміни, та (або) не підпадають під правила обробки даних ручним алгоритмом дій, не можуть мати ніяких змін в ході роботи додатку, та повинні бути відтворені у вихідних файлах у повному обсязі
13. Можливість використання, встановлення додатку на будь-якій версії системи Windows, а саме Windows 7, 8, 10, 11

Згідно з цими встановленими вимогами можливо визначитись з мовою програмування та середовищем розробки для створення додатку.

Технічні вимоги:

1. Мова програмування – C#
2. Технологія для розробки додатку та його інтерфейсу – WPF
3. Середовище розробки – Visual Studio 2019 Community Version
4. Застосунок для створення зображень для додатка – Adobe Photoshop
5. Застосунок для відкриття та взаємодії з файлами формату .xlsx / .xls / .csv – Microsoft Excel
6. Можливе використання безплатний додатків, технологій або розширень для застосунків задля виконання поставлених вимог

**Вибір типу системи обробки інформації**

Згідно з поставленими технічними, функціональними вимогами до застосунку можливо визначити та присвоїти тип інформаційної системи обробки даних для розробляємого програмного забезпечення.

Тип системи – Автоматизована інформаційна система обробки даних.

# ОПИС ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ

## Visual Studio

Visual Studio — це інтегроване середовище розробки (IDE), розроблене Microsoft для розробки графічного інтерфейсу користувача (GUI), консолі, веб-додатків, вебпрограм, мобільних додатків, хмарних програм і веб-сервісів тощо. За допомогою цієї IDE ви можете створювати керований код, а також рідний код. Воно використовує різні платформи програмного забезпечення для розробки програмного забезпечення Microsoft, наприклад Windows Store, Microsoft Silverlight, Windows API тощо. Це не специфічна для мови IDE, оскільки ви можете використовувати її для написання коду на C#, C++, VB (Visual Basic), Python, JavaScript та багато інших мов. Крім того, він підтримує 36 різних мов програмування. Він доступний як для Windows, так і для macOS.

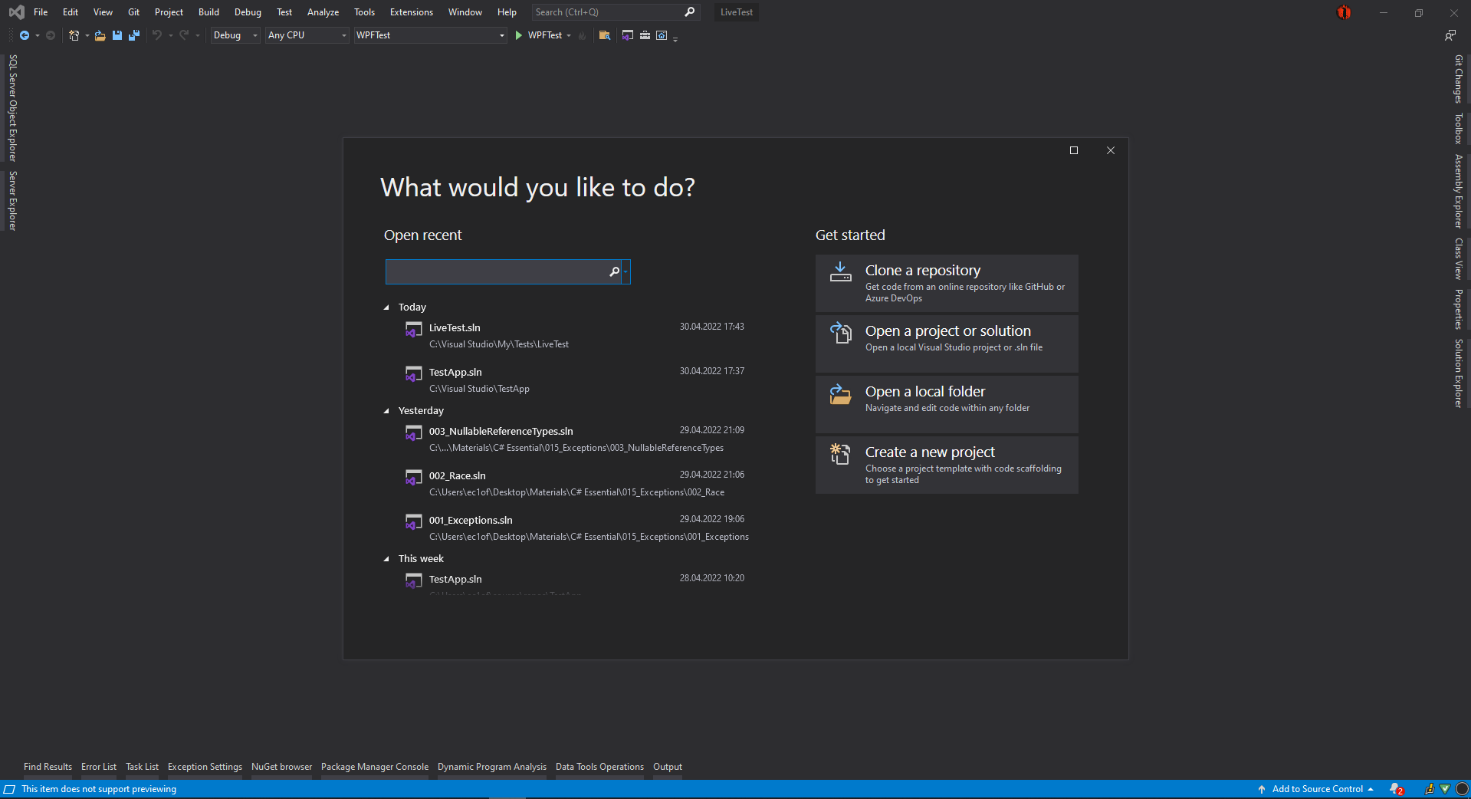


Рис. 1.1 Початкове вікно Visual Studio

На Рис. 1.1 можливо побачити початковий інтерфейс інтегрованого середовища розробки, а саме:

* Меню – верхня частина вікна
* Вікно вибору дії – можливість створити новий проєкт, або відкрити вже існуючий
* Нижня стрічка – інформаційні вкладки для діагностичних та даних які потрібні розробнику. Також на цій стрічці присутня консоль встановлення пакетів-розширень для проєктів.

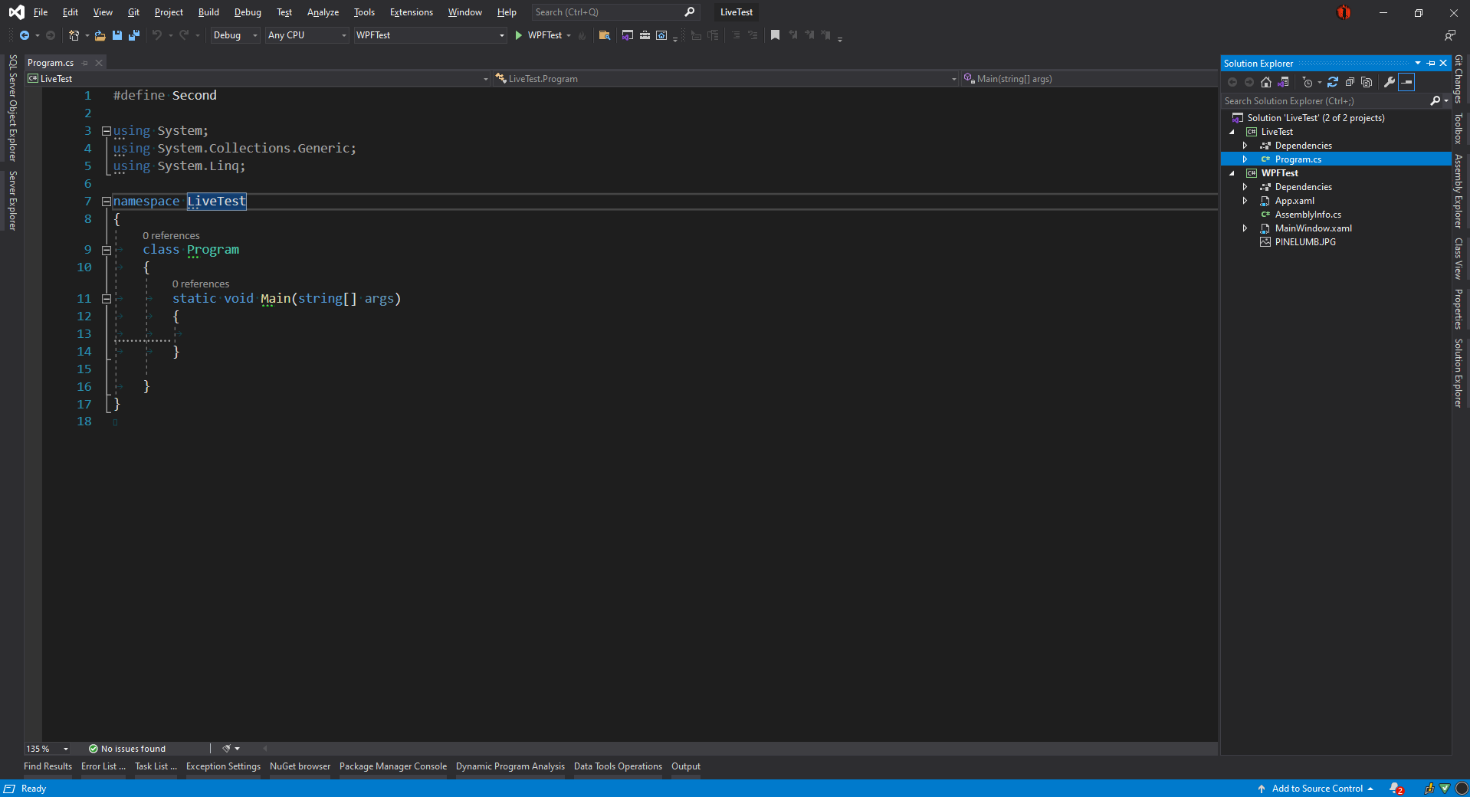


Рис. 1.2 Головний інтерфейс Visual Studio

Головний робочий інтерфейс інструменту розробника Visual Studio (рис. 1.2) складається з 3 елементів

* Меню – верхня стрічка, що складається з заголовків випадаючих підменю:
  + Файл (File) – меню керування файлом
  + Правка (Edit) – меню керування текстом
  + Вид (View) – меню керування видом вікна
  + Git – меню керування системою контролю версій Git
  + Проєкт (Project) – управління проєктом
  + Збірка (Build) – збірка файлів проєкту в єдине ціле
  + Відладка (Debug) – діагностичне меню для розробників
  + Тест (Test) – меню тестування
  + Аналіз (Analyze) – аналіз коду, тексту
  + Засоби (Tools) – налаштування
  + Розширення (Extensions) – додаткові можливості середовища розробки
  + Вікно (Window) – вид вікна Visual Studio
  + Довідка (Help) – інформація про програму
  + Стрічка з елементами керування, запуску та управлінням інтерфейсу, розташовується нижче стрічки з заголовками
* Головне вікно – редактор коду, інтерфейсна частина програма, яка відіграє роль текстового редактора коду, посередині екрану
* Права бокова частина – Оглядач рішення – дозволяє керувати файлами проєкту, додавати, або видаляти будь-які файли / каталоги у рішення (каталог який містить безліч проєктів)

Версії Visual Studio:

1. Community: Це безплатна версія, яка анонсована у 2014 році. Усі інші видання платні. Вона містить функції, подібні до Professional Edition. Використовуючи це видання, будь-який розробник може розробляти власні безплатні або платні програми, такі як програми .Net, вебдодатки та багато іншого. В організації підприємства це видання має деякі обмеження. Наприклад, якщо ваша організація має понад 250 комп’ютерів і річний дохід перевищує 1 мільйон доларів США, вам заборонено використовувати це видання. У непідприємницькій організації цим випуском можуть користуватися до п’яти користувачів. Його головна мета — забезпечити підтримку екосистеми (доступ до тисяч розширень) і мов (ви можете кодувати на C#, VB, F#, C++, HTML, JavaScript, Python тощо).
2. Professional: Це комерційний випуск Visual Studio. Він доступний у Visual Studio 2010 і новіших версіях. Він забезпечує підтримку редагування XML і XSLT і містить такий інструмент, як Server Explorer та інтеграцію з Microsoft SQL Server. Корпорація Майкрософт надає безплатну пробну версію цього видання, і після пробного періоду користувач повинен заплатити, щоб продовжити його використання. Його головна мета — забезпечити гнучкість (професійні інструменти розробника для створення будь-якого типу додатків), продуктивність (потужні функції, такі як CodeLens підвищують продуктивність вашої команди), співпрацю (інструменти Agile планування проєктів, діаграми тощо) і переваги для передплатників, плюс Azure, Pluralsight тощо.
3. Enterprise: Це інтегроване, наскрізне рішення для команд будь-якого розміру з вимогливими потребами до якості та масштабу. Корпорація Майкрософт надає 90-денну безоплатна пробну версію цього видання, і після пробного періоду користувач повинен заплатити, щоб продовжити його використання. Основна перевага цього видання полягає в тому, що воно має високу масштабованість і забезпечує високоякісне програмне забезпечення.

## C#

C# (вимовляється як «See Sharp») — сучасна, об’єктноорієнтована та безпечна для типів даних мова програмування. C# дозволяє розробникам створювати багато різноманітних захищених та надійних програм, які працюють у рамках .NET.

Вона була розроблена компанією Microsoft під керівництвом Андерса Хейлсберга та його команди в рамках ініціативи .NET і схвалений Європейською асоціацією виробників комп’ютерів (ECMA) та Міжнародною організацією зі стандартів (ISO). C# є однією з мов для загальномовної інфраструктури. Синтаксично C# дуже схожий на Java і є простим для користувачів, які знають C, C++ або Java.

Ця мова програмування є доволі популярною серед кіл розробників як пет-проєктів, так і повноцінних комерційних / корпоративних систем або додатків. Внаслідок такої неспадної популярності та розповсюдженості, компанія Microsoft (головний розробник) підтримує мову програмування, та періодично випускає й світ все нові й нові версії мови. У 2022 вже у світ може вийти версія C# 10.0, що можливо рахувати ювілейною версією. Хоча ця версія за цифрою є ювілейною, але за рахунком не зовсім. На початку створення, C# поділявся на 2 види - .NET Framework та на .NET Core. Загалом це два схожі між собою різновиди мови, але концепція і цілі були різними:

1. .NET Framework – початковий різновид, націлений на розробку додатків виключно під операційну систему Windows, що зі свого боку дає переваги «з коробки», такі як пряма сумісність з Windows, та більший діапазон можливостей для взаємодії з ОС. Розробка цього різновиду зупинилась на версії .NET Framework 4.8. Надалі є лише підтримка, на невизначений час.
2. .NET Core (надалі став просто .NET) – наразі основний різновид який підтримується та оновлюється. Від початку став набирати популярності шляхом можливості розробки застосунків під різні види систем – Android, IOS, Windows, Linux, MacOS. І зараз займає більшу частину ринку розробки C#.

В додаток тому, що ця мова є популярна, це дозволяє оперативно знайти відповідь на майже будь-яке питання, що може турбувати розробника. Для цього можливо використовувати різні форуми, такі як StackOverflow, CyberForum та інші, або ж використовувати офіційну документацію, що можливо знайти на сайті Microsoft.

C# популярний не лише за його особливості, безпеку та сучасність, а й за різновид типів додатків які можливо створити цією мовою:

1. Windows-застосунки
2. Мобільні додатки (Andoid, IOS)
3. Вебдодатки
4. Ігри (Unity)
5. Інші види додатків, котрі можливо створювати шляхом використання додаткових фреймворків / розширень, таких як AvaloniaUI, Xamarin.

Так, кожен вид цих застосунків можливо створювати і іншими методами, але на це потрібні інші ресурси, інші середовища розробки, котрі не завжди поставляються разом з підтримкою всіх типів, або всіх платформ. Тому вибір мови C# для створення автоматизованої системи, що буде використовуватись на операційній системі Windows, є доволі гарним рішенням, завдяки додатковій опції C# - застосунки, котрі були створені цією мовою програмування, майже завжди можливо використати без додаткових інструментів в майбутньому – зібрав, архівував файли, передав другу – той користується повноцінно так, як і розробник.

## .NET

.NET — це програмна платформа, спланована і розроблена компанією Microsoft. Першою версією фреймворку .Net була версія 1.0, яка з’явилася в 2002 році. Простіше кажучи, це віртуальна машина для компіляції та виконання програм, написаних різними мовами, такими як C#, VB.Net тощо. На основі .Net існують різноманітні мови програмування, найпоширенішими є VB.Net та C#. Вона використовується для створення додатків для Windows, мобільних пристроїв, Інтернету тощо. Вона надає багато функцій, а також підтримує галузеві стандарти. На даний час, присутні два різновиди фреймворку .NET :

.NET Core — це безплатна платформа з відкритим кодом, універсальна платформа розробки для розробки сучасних хмарних програмних додатків на операційних системах Windows, Linux і macOS. Вона працює на кількох платформах і була оновлена, щоб зробити .NET швидшим, масштабованим і сучасним. .NET Core є одним із великих внесків Microsoft, випущеним під ліцензією MIT. Вона пропонує такі функції:

* Кросплатформенність
* Відритий вихідний код
* Висока ефективність
* Велика кількість платформ під розробку та режим розробника тощо.

.NET Framework — підтримує понад 60 мов програмування, серед яких 11 були спроєктовані та розроблені компанією Microsoft. Інші мови, які не належать Microsoft підтримують використання .NET Framework, але не є спроєктованими чи розробленими Microsoft’ом.

Нижче наведена порівняльна таблиця (табл. 1) .NET Core та .NET Framework:

Таблиця 1 – Порівняння .NET Core та .NET Framework

|  |  |  |
| --- | --- | --- |
| Характеристика | .NET Core | .NET Framework |
| Відкритий код | Є повністю відкритим для розробників | Лише деякі компоненти є відкритими |
| Кроссплатформенність (сумісність) | Працює за принципом «збери один раз, запускай будь де». Сумісний з різними операційними системами – Windows, Linux, Mac OS. | Сумісний лише з операційною системою Windows, хоча розроблявся для підтримки програмного забезпечення та додатків на усіх операційних системах. |
| Продуктивність та масштабованість | Забезпечує високу продуктивність і масштабованість | Є менш ефективним в порівнянні з .NET Core з точки зору продуктивності та масштабованості програм |

Продовження таблиці 1 – Порівняння .NET Core та .NET Framework

|  |  |  |
| --- | --- | --- |
| Розробка під мобільні пристрої | Сумісний з платформами мобільних додатків з відкритим вихідним кодом, наприклад Xamarin, через стандартну бібліотеку .NET. Розробники використовують інструменти Xamarin для налаштування мобільного додатку для певних мобільних пристроїв, таких як телефони iOS, Android, Windows. | Не підтримує жодної платформи для розробки мобільних додатків. |
| Модульність | Можливе модульне встановлення – економія пам’яті | Усі бібліотеки та пакети зібрані в єдине ціле, не розділяються |
| Підтримка нових версій | При оновленні, користувачу застосунку не потрібно встановлювати власноруч нову версію бібліотеки для коректної роботи ПЗ. | При оновленні, для коректної роботи застосунку на базі цієї компоненти, користувачу потрібно додатково до ПЗ встановити нову версію .NET Framework. |

## WPF

Windows Presentation Foundation (WPF), фреймворк інтерфейсу користувача, який не залежить від роздільної здатності та використовує векторний механізм візуалізації, створений для використання переваг сучасного графічного обладнання. WPF надає повний набір функцій розробки додатків, які включають розширювану мову розмітки додатків (XAML), елементи керування, прив’язку даних, макет, 2D та 3D-графіку, анімацію, стилі, шаблони, документи, медіа, текст і типографіку.

WPF дає змогу розробляти програму з використанням як розмітки, так і коду, з яким повинні бути знайомі розробники ASP.NET. Зазвичай ви використовуєте розмітку XAML для реалізації зовнішнього вигляду програми, а для реалізації її поведінки використовуєте керовані мови програмування (кодовий код). Такий поділ графічного вигляду та функціональної поведінки має такі переваги:

* Витрати на розробку та обслуговування знижуються, оскільки розмітка, специфічна для зовнішнього вигляду, не тісно пов’язана з кодом, що залежить від поведінки.
* Розробка ефективніша, оскільки дизайнери можуть реалізувати зовнішній вигляд програми одночасно з розробниками, які реалізують поведінку програми.
* Глобалізація та локалізація для додатків WPF спрощена.

Графічна та частково логічна частина WPF додатку може бути написана за допомогою XAML розмітки, що є універсальною не тільки для WPF додатку, а також, наприклад для інтерфейсу мобільних застосунків, розроблених на основі Xamarin.

XAML є декларативною мовою розмітки. Що стосується моделі програмування .NET Core, то XAML спрощує створення інтерфейсу користувача, в порівнянні з альтернативним методом – створення інтерфейсу з використанням коду, де потрібно власноруч розміщувати елементи не маючи передпоказу інтерфейсу, та навпомацки вивіряти розміри елементів на формах.

XAML розмітка у WPF версії, дозволяє попередньо зрозуміти та побачити як буде виглядати інтерфейс, який створюється (рис. 2.1) у верхній частині, та можливість написання XAML розмітки у нижній частині вікна (рис. 2.1).

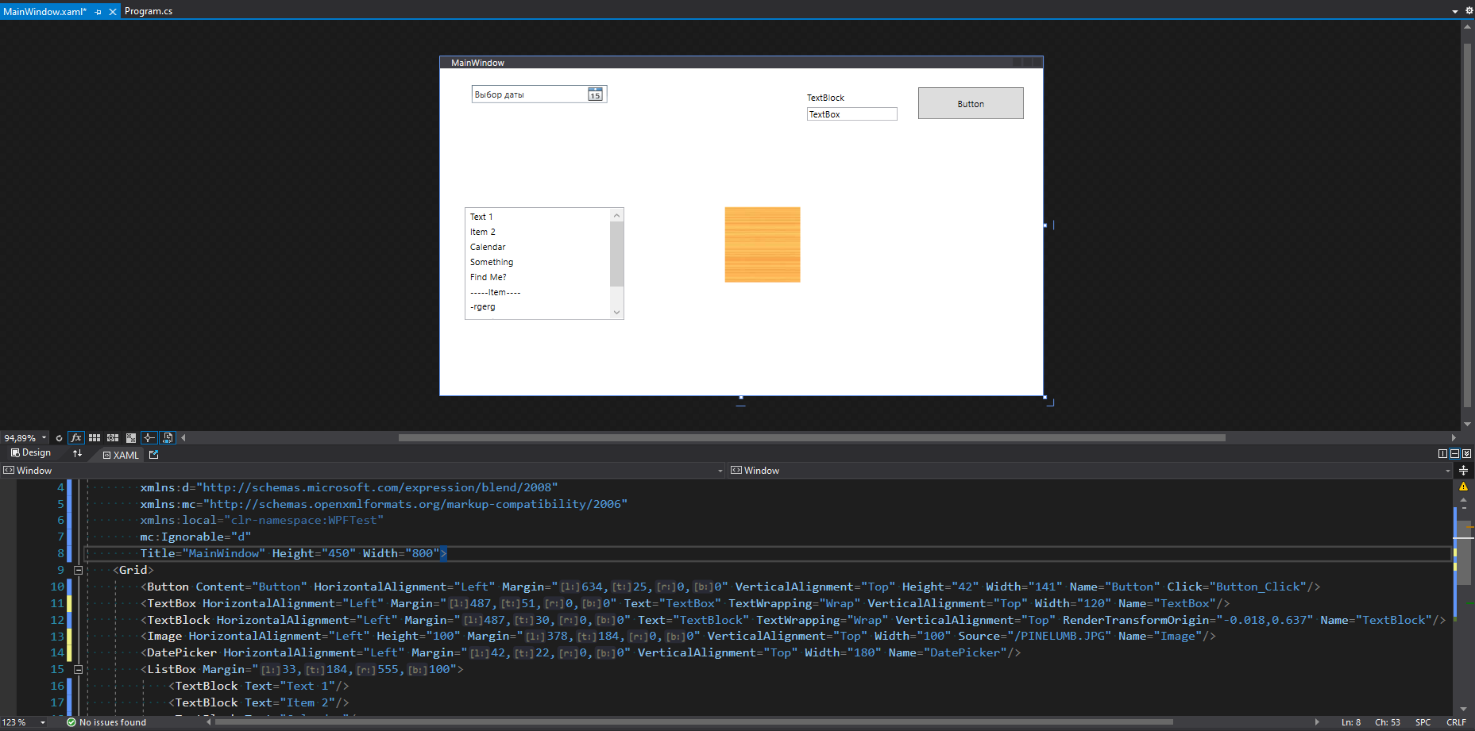


Рис. 2.1 – XAML розмітка та її передпоказ

Основними, але не всіма елементами користувацького є:

1. Button
2. DatePicker
3. ListBox
4. TextBlock
5. TextBox
6. Image
7. MessageBox

Елемент керування Button (рис. 2.2) використовується для взаємодії користувача з елементом інтерфейсу у вигляді кнопки на яку можливо натиснути, та завдяки закріпленню логіки при натисканні на елемент керування Button відбудеться якась подія. Прикладом події є виклик інформаційного текстового вікна MessageBox, що дозволяє вивести якусь інформацію на екран шляхом появи віконця посеред екрану (рис. 2.3)

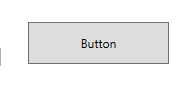


Рис. 2.2 – Елемент керування Button

Елемент інтерфейсу MessageBox (рис. 2.3) можливо використовувати заради інформування користувача текстовим віконцем. У цього елемента є декілька властивостей, наприклад ми можемо задати умовний тип вікна – помилка, попередження, або просто якась інформація, що змінить іконку всередині вікна, відповідно до його типу: помилка – хрестик в червоному кільці, попередження – знак оклику у помаранчевому трикутнику, інформація – знак оклику у синьому трикутнику. Також можливо вказати чи будуть елементи керування у цього вікна, і які саме – кнопки : Ок, Ні, Так, Скасувати, або ж ніяких. Відповідно до того яку кнопку натисне користувач, то такий результат повернеться у вигляді елементу списку результатів форми вікна.

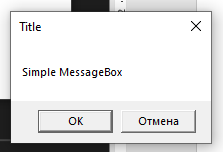


Рис. 2.3 – Елемент інтерфейсу MessageBox

Згідно з офіційною документацією, викликати таке вікно показу повідомлення можливо за допомогою виконання команди Show() на об’єкті MessageBox (рис. 2.4). Скріншот коду відображає логіку натискання елементу керування Button (рис. 2.2). Тобто, при натисненні кнопки буде викликане вікно показу інформації (рис. 2.3). В параметрах (дужках) метода Show, ми можемо вказати такі дані, як: текст повідомлення, заголовок вікна, тип кнопок на формі, тип іконки повідомлення (помилка, попередження і т.д.), очікуваний результат натискання кнопки за замовчуванням.

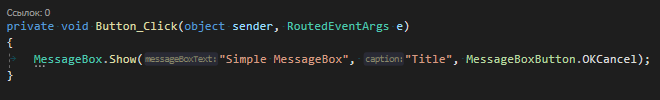


Рис. 2.4 Виклик методу Show на об’єкті типу MessageBox

Елемент керування DatePicker (рис. 2.5) можливо використовувати для вибору: конкретної дати, діапазону дат, або ж вибору дня, місяця, року.

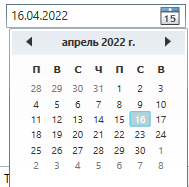


Рис. 2.5 Приклад використання DatePicker

Елемент розмітки – ListBox (рис. 2.6) являє собою список з елементів, які можливо продивлятись, прокручувати видимий діапазон за допомогою каретки навігації у правій частині списку серед наявних. Є можливість додавання, видалення, редагування елементів у списку при необхідності. Також є можливість збільшення або зменшення розмірів елемента, за допомогою зміни відповідних параметрів ширини та висоти елемента на формі, або у коді.

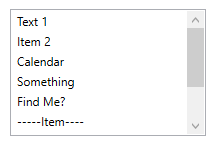


Рис. 2.6 Елемент розмітки – ListBox

TextBlock – елемент інтерфейсу, що являє собою текстовий блок, який не має можливості редагування при запуску додатку. Може бути використаний, як заголовок чого-небудь, наприклад кнопки, або текстового поля TextBox (рис. 2.7), що в свою чергу можливо змінювати – редагувати текст всередині блоку. На рисунку 2.7, можливо побачити тестове поле та заголовок до нього.



Рис. 2.7 Текстовий блок TextBox у вигляді заголовка до текстового поля TextBox

Елемент інтерфейсу Image може містити в собі зображення (рис. 2.8) таких розширень як: .jpg, .jpeg, .png та інші. При встановленні зображення у вигляді джерела, ми можемо задати властивості цього зображення, наприклад розтягнути його на всю площину елемента інтерфейсу Image, або ж не розтягувати та розмістити по центру елемента, що дозволить центрувати внутрішнє зображення в рамках контейнеру Image.

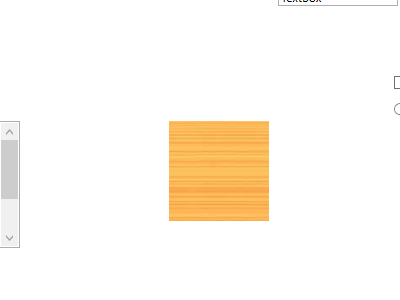


Рис. 2.8 Елемент інтерфейсу Image – контейнер для зображень

За допомогою цих вищезазначених елементів можливо створити застосунок на основі коду C#, та розмітки XAML, з використанням фреймворку WPF і запустити його для тестування та демонстрації отриманого (рис. 2.9).

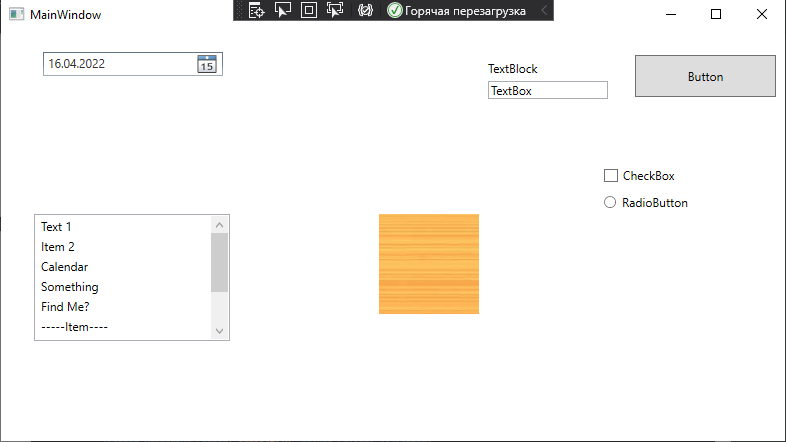


Рис. 2.9 Приклад інтерфейсу додатку WPF

## Adobe Photoshop

Photoshop — програмне забезпечення для редагування фотографій та растрового графічного дизайну, яке дозволяє користувачам створювати, редагувати й керувати різними графічними зображеннями, а також цифровим мистецтвом. Він також дозволяє створювати та редагувати растрові зображення з кількома шарами та імпортувати зображення в різних форматах файлів. Photoshop розроблений Adobe Systems для Windows і MacOS.

На даний час є лідером на ринку галузі роботи з растровими форматами зображень. У порівнянні з іншими додатками роботи з растровим зображенням, Фотошоп (назва у простонародді) є найзручнішим. Тому він буде використаний для створення зображень для додатка. Його сумісність з різними типами зображень (формати розширення файлів зображень) дозволяє створювати майже будь-якого роду картинки, фото, рисунки.

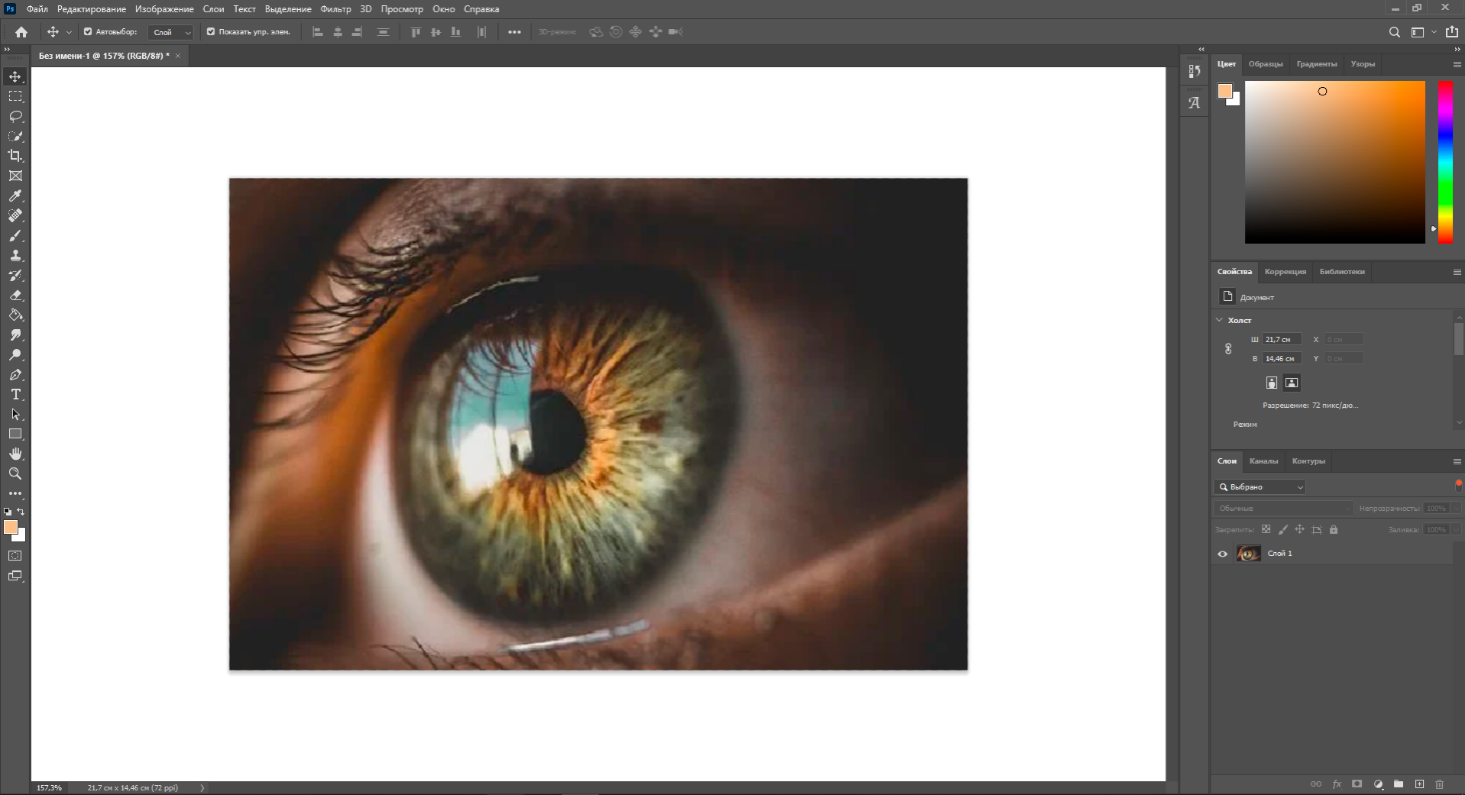


Рис. 3 Інтерфейс застосунку Adobe Photoshop

# ОПИС АЛГОРИТМУ ОБРОБКИ ДАНИХ

Перед початком самої розробки застосунку, потрібно визначитись, що потрібно автоматизувати в обробці даних. Згідно з поставленими вимогами та алгоритмом, що наведений нижче, можливо виявити зони та кроки, що підлягають автоматизації, а що ні.

Ручний алгоритм обробки даних – умовно для роботи алгоритму візьмемо сайт ask.fm. У наступних описах алгоритму роботи, автоматизування в дані які можуть розкрити або порушити конфіденційність користувачів умовної системи, будуть згенеровані випадковим чином. Всі збіги даних з реальними користувачами є випадковими.

Отож, сам алгоритм роботи:

1. **Отримання вхідних файлів:**
   1. **Landing.csv** - файл з даними реєстрації одного типу (реєстрація з мобільного пристрою), з такими даними як:
      * **id** - персональний ідентифікатор в системі
      * **login** - логін користувача у системі
      * **email** - адреса електронної пошти, яка вказана при реєстрації у систему
      * **country** - країна звідки користувач реєструється (надалі, дані в цьому полі будуть замінені на відповідні мовні регіони)
      * **platform** - тип пристрою з якого реєструвався користувач
   2. **Typeform.xlsx** - також файл з даними користувачів які реєструвались, але дані отримані з варіанту реєстрації в браузері, з такими полями:
      * login та email
      * geo – в цьому випадку, вже автоматично вказаний мовний регіон реєстрації (не країна)
   3. **total.xlsx** - файл зі списком логінів, що вже успішно зареєстровані в системі, потрібен для запобігання повторної реєстрації
2. **Обробка Landing файлу**
   1. Поле адреси електронної пошти - перевірка адрес на правильність доменів (права частина адреси, після знаку “@”), у випадку виявлення помилок типу невірного порядку символів, на кшталт “@gamil.com”, що дає зрозуміти працівнику (той хто обробляє дані), що цей домен пошти є помилковим, і найближчим по подібності є “@gmail.com”. Відбувається редагування поля даних пошти, а саме - заміна домену на коректний варіант. Список відповідних варіацій помилкових доменів та їх виправлення наведені у таблиці 2.1 (країна з кодом RU замінюється окремо на PBKIH, далі в обробці не використовується цей шифр)
   2. Поле країни (надалі поле стане регіоном) - дані в цьому полі замінюються відповідно до таблиці 2.2
   3. Поле логінів та пошт - відбувається ручний пошук логінів / пошт, що можуть розглядатись як потенційні дублікати (боти - створені заради наживи сторінки користувачів), такі прецеденти видаляють з файлу (весь рядок з даним користувача видаляється з файлу). Цей етап виконується виключно вручну, та рішення приймає працівник, що виконує обробку файлу.
   4. Поля ідентифікатора та платформи не підлягають перевірки та редагування
3. **Обробка Typeform файлу**
   1. Поле логіну - є критичним місцем, а саме тут спостерігається найбільша тенденція впливу людського фактора. При наявності символу пробілу або “@”, їх видаляють. Дублікатів логінів не повинно бути, тому всі повторення того чи іншого логіну в файлі потребують видалення всього рядка даних. Більш ніж у 35% випадках користувачі які реєструються вказують невірний логін у системі. Тому поле логіну підлягає перевірці на існування сторінки користувача, а саме логін копіюється та комбінується з адресою вебсторінки соціальної мережі, приклад посилання буде виглядати наступним чином - “ask .fm/Askfm”, де “ask .fm/” - це є частиною посилання на сам сайт, а “Askfm” є логіном користувача. Якщо при переході по цьому посиланні існує сторінка користувача, то беремо його фактичний логін зі сторінки (присутнє таке поле на сторінці), оскільки подальші кроки циклу обробки даних є чуттєві до регістру логіну та колір заливки у файлі змінюють на жовтий у відповідній комірці. Якщо такої сторінки не існує, поле помічається помаранчевим кольором фону заливки у файлі.
   2. Поле логіну звіряється зі списком наявних зареєстрованих логінів з файлу Total, якщо логін файлу Typeform наявний у файлі Total, то рядок з даними з файлу Typeform.xlsx видаляють.
   3. Поле пошти - дублікати видаляють (весь рядок з даними)
   4. Поле регіону не підлягає перевірки та редагування
4. **Групування даних**
   1. Дані з файлів Landing та Typeform групуються за наявними регіонами у таблиці 2.2, та дані копіюють у вихідні файли, що мають таке оформлення:
      * Файли формату .xlsx - шаблон назви файлу - “регіон”\_funnel.xlsx, де “регіон” замінюється на відповідну назву регіону групи даних. Під кожен мовний регіон створюється окремий файл, де перший рядок є заголовками полів - login, email. А під ними в комірках розташовуються відповідні дані кожного користувача відповідно регіону.
      * Файли формату .csv - шаблон назви файлу - “регіон”\_funnel.csv, де “регіон” замінюється на відповідну назву регіону групи даних. Під кожен мовний регіон створюється окремий файл, куди копіюється лише логіни користувачів того чи іншого регіону.

Згідно з описаним алгоритмом та встановленми вимогами, тепер визначимо які саме пункти цього алгоритму підлягають автоматизації: 2.1-2.2, 3-й пункт повністю, 4й пункт повністю.

Перевіряючи вимоги слід зазначити, що до цього звичайного алгоритму, додатково буде доданий пункт про резервне копіювання файлів Landing та Typeform перед початком їх обробки, задля збереження можливості відновлення даних у критичному випадку.

Таблиця 2.1 Вірні домени електронної пошти та їх помилкові варіанти

|  |  |
| --- | --- |
| Вірний домен | Невірні варіації |
| gmail.com | gamil.com, gmil.com, gimal.com, gmal.com, gmile.com, gmila.com, gmai.com, gmsil.com, gimel.com, gail.com, gmoil.com, gmill.com, gamail.com, gamil.com, gamail.com, gmqil.com |
| icloud.com | icoud.com, iclud.com, iloud.com |
| yahoo.com | yaho.com, ahoo.com, ayhoo.com, yahooo.com |
| outlook.com | outlok.com, oultook.com, ouylook.com |
| hotmail.com | homail.com, hotmil.com, htmail.com, hotnail.com |

Таблиця 2.2 Мовні регіони та їх групи країн (коди країн)

|  |  |
| --- | --- |
| Мовний регіон | Група країн (коди країн) |
| ARAB | AE, EG, IQ, JO, KW, PS, SA, SY |
| DE | DE, AT, CH |
| MX | AR, BO, CL, CO, CR, EC, ES, HN, MX, NI, PE, SV, VE |
| ENG | AU, BD, BR, CA, CN, FR, GB, IL, IN, MY, NL, PH, PK, RO, SG, US |
| RU | KZ, MD, AM |
| Інші регіони (регіон = країна) | UA, IT, ID, PL, TH, TR |

# ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Завдання котре повинен вирішувати застосунок – автоматизація процесу обробки даних на основі ручного алгоритму. Пункти, шо повинні бути автоматизовані та можливості додатку були описані вище, відповідно можна розпочинати розробку.

Згідно з вимогами середовище розробки – Visual Studio Community 2019. Оскільки це є безоплатна версія додатку, завантажити її не є складним процесом. Завантаження з офіційного сайту Microsoft (рис. 4.1).

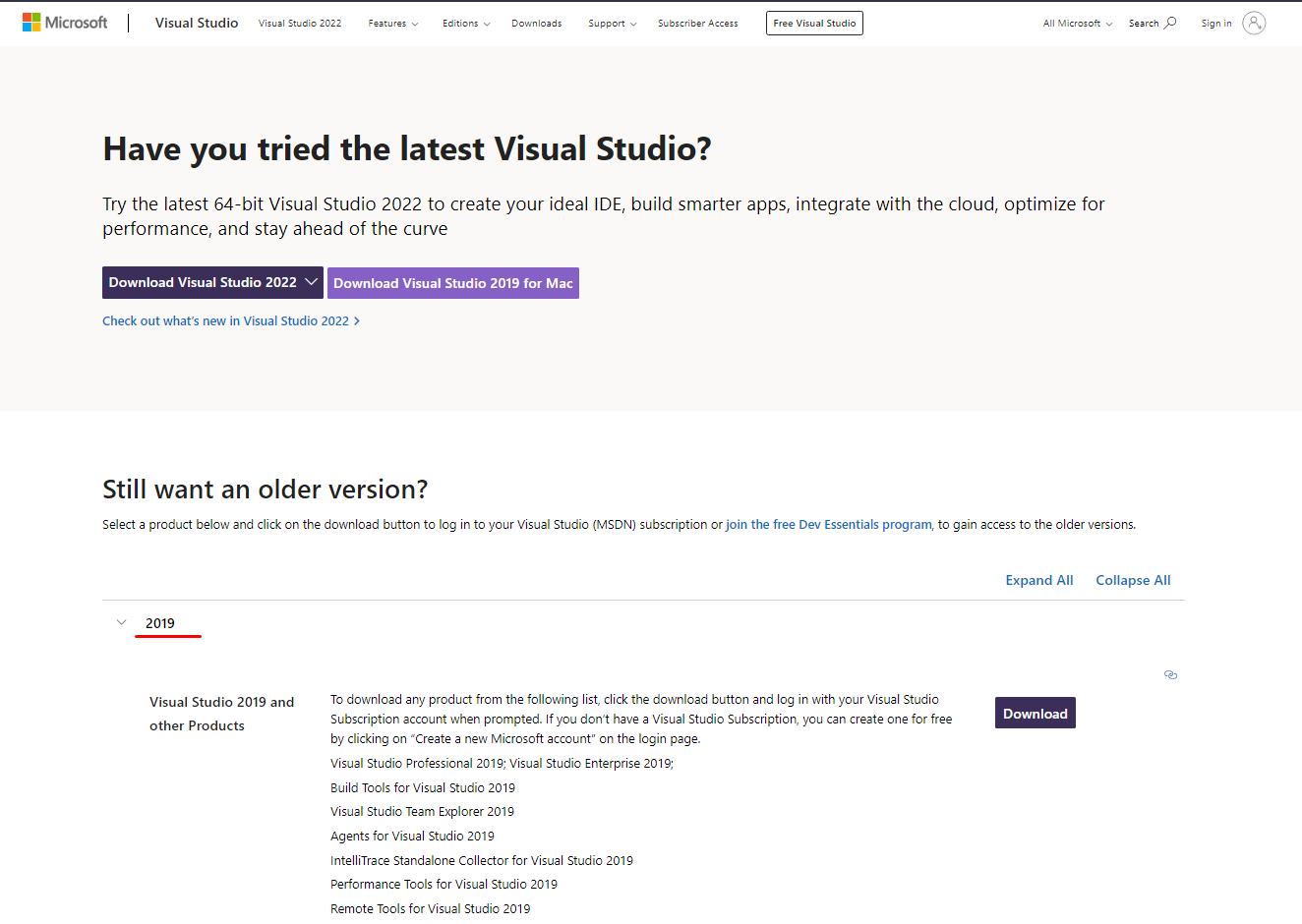


Рис. 4.1 Завантаження Visual Studio 2019

Далі, створюємо наш проєкт, користуючись вбудованим шаблоном WPF. В меню Файл (File) вибираємо Створити (New), далі Проєкт (Project). У діалоговому віконці, що відкриється (рис. 4.2) обираємо шаблон WPF Application без приставки .NET Framework.

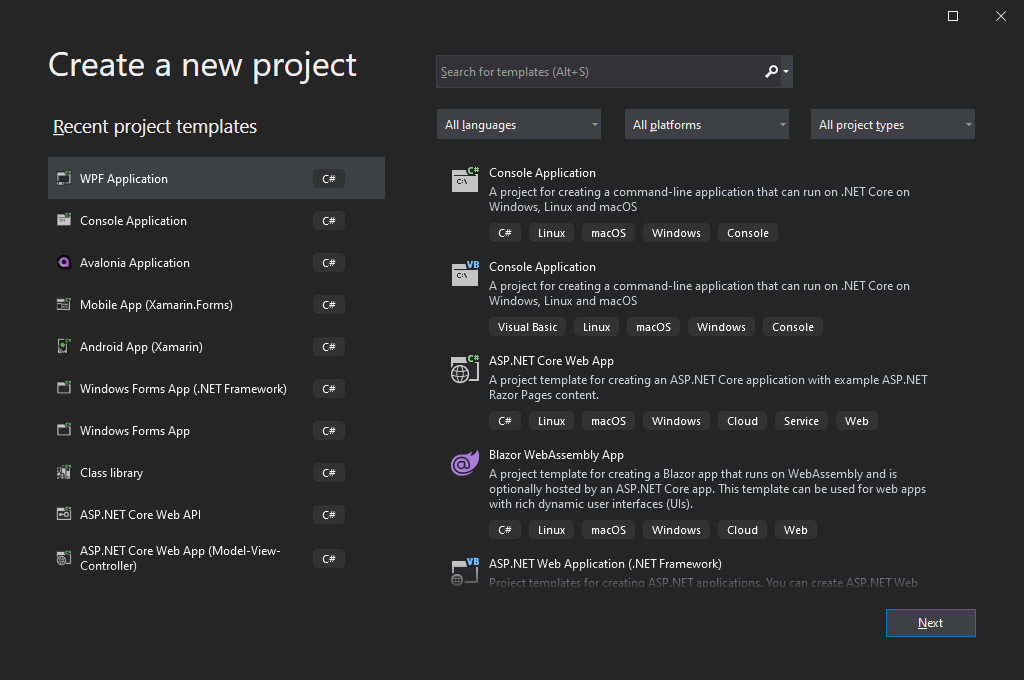


Рис. 4.2 Створення проєкту

Далі надаємо ім’я проєкту (застосунку в майбутньому) (рис. 4.3) та його місцезнаходження на локальному диску.

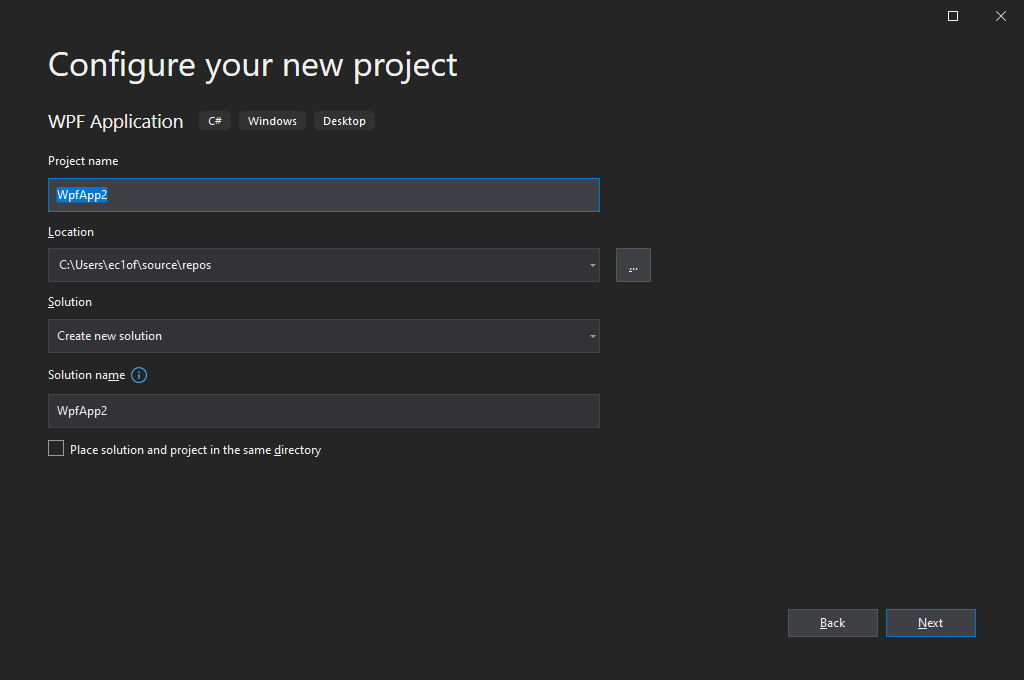


Рис. 4.3 Задання ім’я проєкту та вибір місцезнаходження

Після задання імені проєкту і перейшовши до наступного кроку (натискання кнопки Далі (Next)), Visual Studio автоматично створить проєкт з заданим ім’ям, та стандартним набором файлів, головними з яких є MainWindow.xaml – файл з розміткою (інтерфейсом) та MainWindow.xaml.cs – файл розширення C#. Ці 2 файли є взаємопов’язаними між собою – C# файл фактично керує та відповідає за логіку інтерфейсу описаного у XAML файлі.

Згідно з вимогами, додаток повинен взаємодіяти з файлами формату Excel, тобто з табличними файлами, що не є укомплектованою можливістю студії при встановленні. Для коректної роботи з файлами потрібно встановити умовно-безкоштовне розширення NuGet – EPPlus (рис. 4.4), для нашого проєкту, що дасть змогу охопити майже весь спектр можливостей роботи з таблицями Excel, що і в ручному режимі.

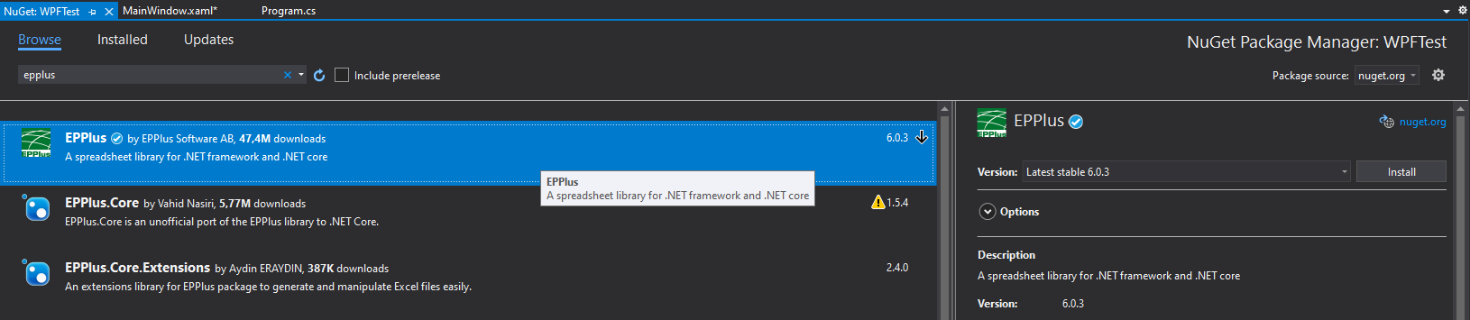


Рис. 4.4 Встановлення EPPlus

У списку вимог є пункт про модульність. Він визначає, що не всі компоненти програми будуть використовуватись. Відповідно, поділимо додаток на 3 основні і 1 додаткову частину : обробка Landing файлу, обробка Typeform, групування та розкидування даних по файлах, додатковою частиною додатку (як бонус для користувача) можемо викрити можливість очистки файлів формату Excel. Додаткова частина міститиме поле для імпорту файлів формату Excel, та при натисненні її імпортовані файли будуть очищені повністю. Обґрунтування цієї можливості можливо вибрати те, що система є автоматизованою і модульною, тому не всі етапи можуть виконуватись, деякі сам працівник вирішує виконувати автоматично чи ні. Якщо ж користувач вирішить власноруч згрупувати та розкласти дані по файлах, то відкривати близько 22 файлів (кількість регіонів \* 2 (файл .xlsx та .csv) ) і власноруч їх очищати є доволі неприємним заняттям.

Розташуємо наші компоненти програми у відповідному порядку складності виконання (від легкого рівня до складного) : очищення файлів, обробка Landing, обробка Typeform, а от компонента групування даних є тривіальною задачею, тому робити будемо її в останню чергу.

Отож, розпочнемо розробку додаткового модуля.

## Розробка додаткового модуля очищення файлів

Цей етап є найпростішим, тому що весь процес створення автоматизованого додатка базується на вже відточеному алгоритмі ручної обробки. Відповідно, всі файли для розкидання та групування даних заздалегідь підготовлені, та очікують пасивно в папці (каталозі) (рис. 4.5). На фото можемо побачити 22 файли, кожен з яких відповідає своєму регіону. Ці файли можуть містити в собі різну кількість даних, які нам не потрібні, тому створимо алгоритм, що очищує їх.



Рис. 4.5 Файли призначенні для очищення

WPF застосунок складається по суті з файлів що йдуть парами – інтерфейс + логіка. Тому на основі цього створюємо новий файл CleanerPage.xaml та автоматично разом з ним створюється файл CleanerPage.xaml.cs. (рис. 4.6). Ці два файли і будуть містити в собі логіку та інтерфейс алгоритму очищення файлів.

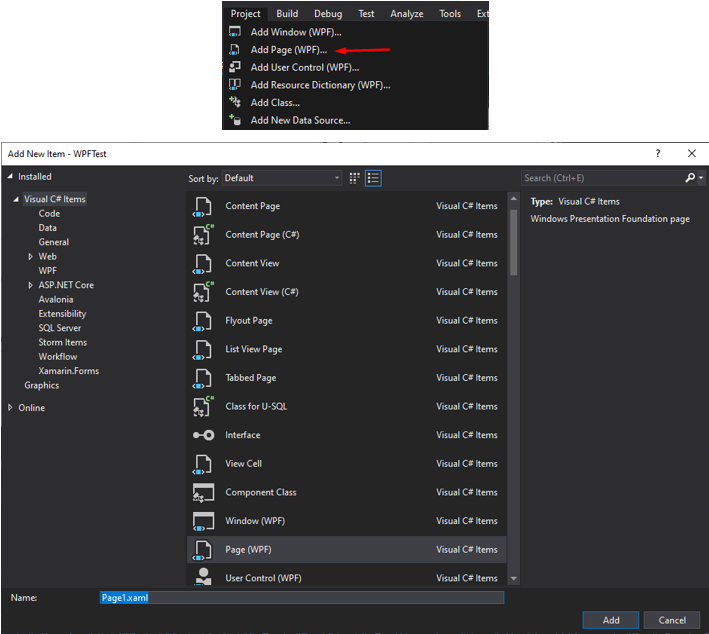


Рис. 4.6 Створення нової сторінки CleanerPage

Після створення нової сторінки, вона буде пустою, незаповненою компонентами (рис. 4.7), тобто не має інтерфейсу взагалі. Будемо створювати додаток сторінкового типу – тобто кожна сторінка додатку відповідає за свою ділянку логіки застосунку, та містить свій власний інтерфейс. Сторінки будуть розташовуватись на головному вікні програми – MainWindow.xaml.

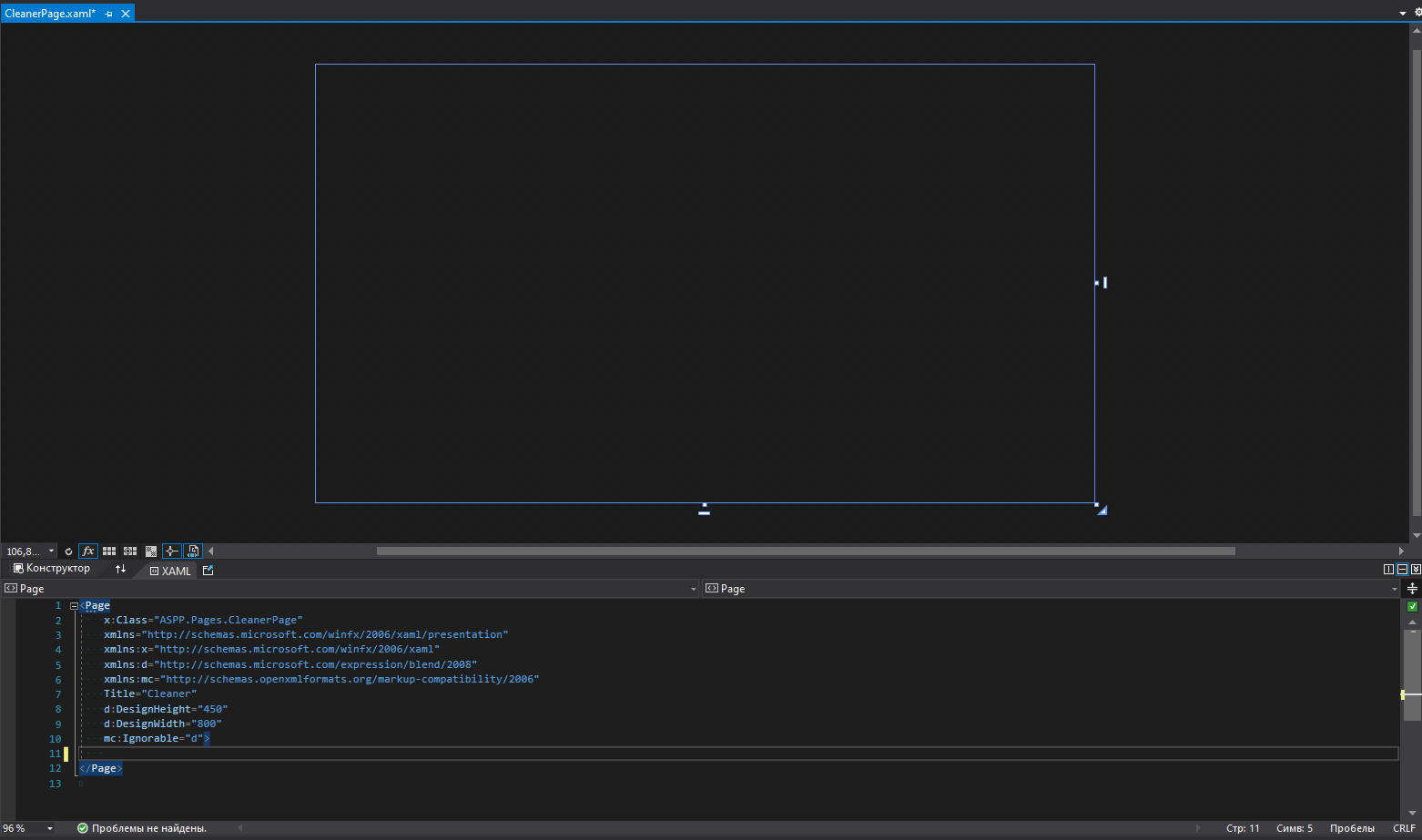


Рис. 4.7 Вигляд новоствореної сторінки додатку

Тепер слід заповнити цю сторінку елементами, що дозволять користувачу взаємодіяти з додатком :

1. Grid – Таблиця розташування елементів – поділяється на рядки та стовпчики, в які можливо розташувати елементи.
2. TextBox – буде являти собою зону імпорту файлів (перетягування файлів у вікно програми в цю зону, та відпусканні клавіші миші запустить процес імпорту файлів у застосунок), контейнер для файлів
3. Button – кнопка, що запустить процес очищення файлів у випадку лише, якщо TextBox не порожній (тобто там є імпортовані файли), та файли всередині є потрібного формату.

Також на всіх сторінках додатку буде наявним рядок стану – StatusBar, в якому будуть відображатись інформація щодо результатів алгоритму, його автора та час за який була виконана дія для замірів продуктивності роботи додатку. Порівняння продуктивності буде в розділі з висновками.

Попередній вигляд інтерфейсу сторінки очищення файлів виглядає так – рис. 4.8. У TextBox’і є картинка, зроблена власноруч, за допомогою Photoshop.

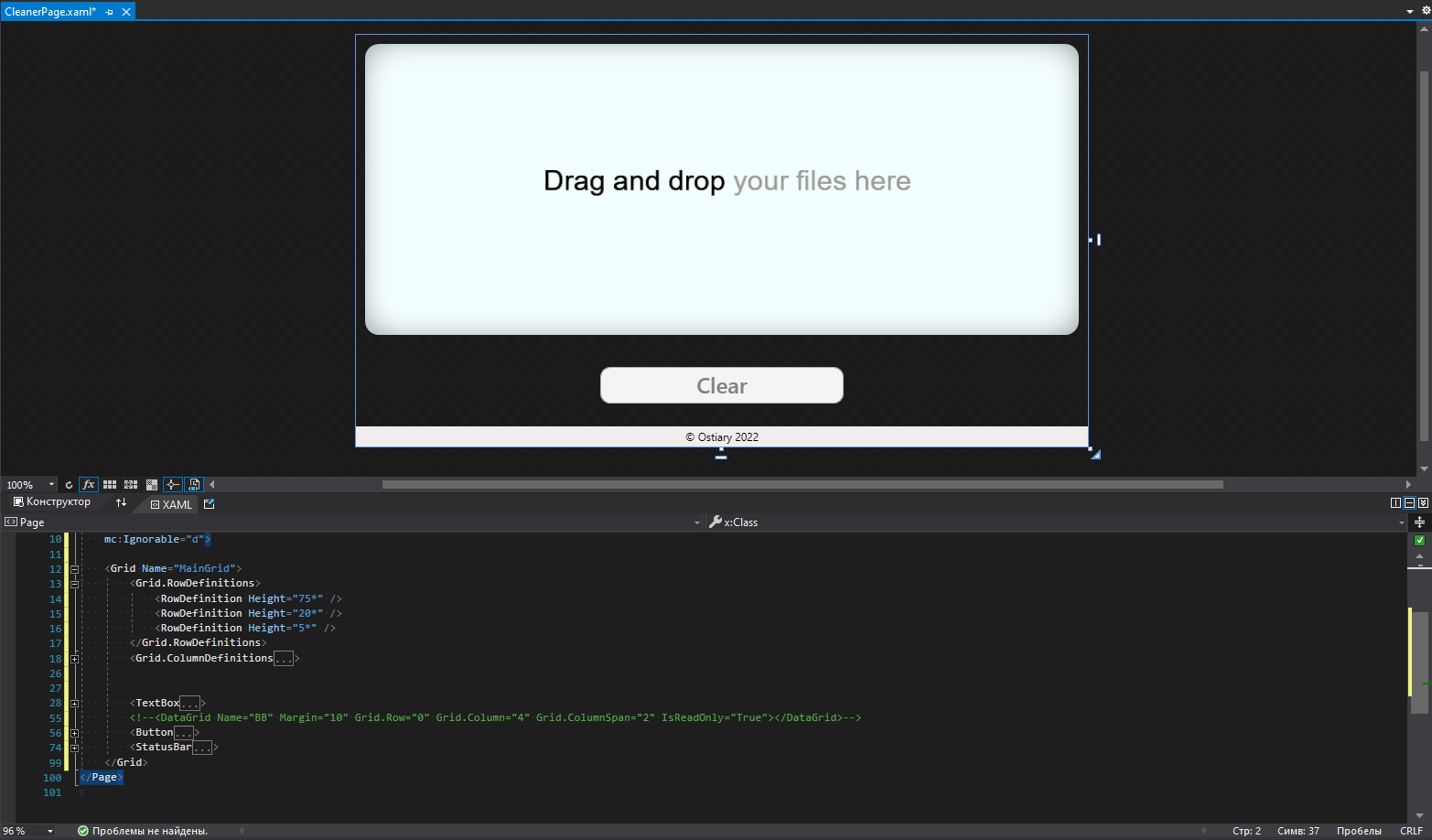


Рис. 4.8 Попередній вигляд сторінки очищення файлів

Логіка сторінки, яка описана у code-behind (коду схованому за інтерфейсом) виглядає наступним чином :

* Користувач перетягуючи файли у зону імпорту, запускає алгоритм-подію, що отримує список файлів, що був перетягнутий та формує список файлів у вигляді рядків з повними кваліфікаційними іменами-шляхами файлів.
* Надалі користувач натискає кнопку очищення файлів, запускається відповідний алгоритм-подія очищення файлів, котрі лежать у контейнері (текстовому полі). Формується список (рис. 4.9) файлів типу класу ListInUse (рис. 4.10)

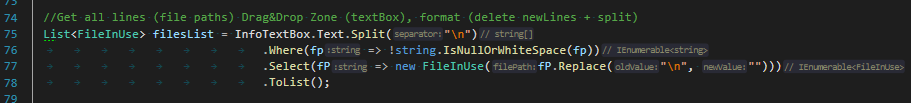


Рис. 4.9 Формування списку файлів типу класу FileInUse

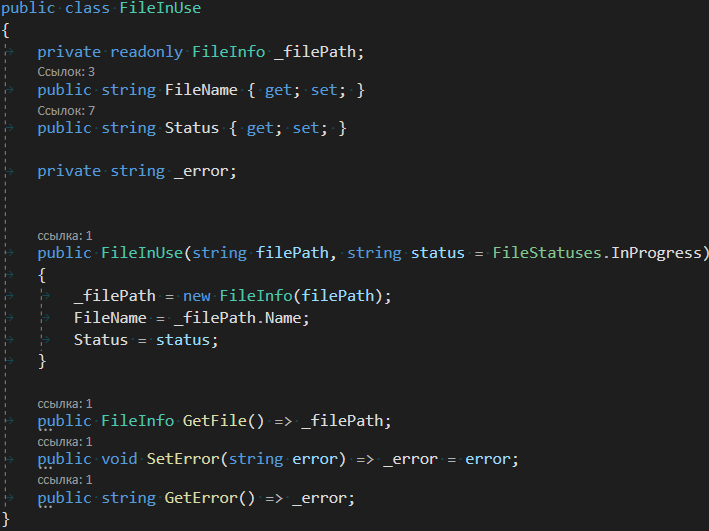


Рис. 4.10 Класс FileInUse

Описова частина класу **FileInUse**:

* **\_filePath** – приватне поле класу, що вміщує повну інформацію про файл
* **FileName** – публічне ім’я файлу
* **Status** – публічний статус обробки файлу
* **\_error** – приватне поле, що вмішує інформацію про помилку
* **FileInUse (string filePath, string status)** – публічний конструктор класу, що приймає повний шлях до файлу, статус файлу (за замовчуванням – В процесі обробки). Конструктор заповнює поле відповідно кожному аргументу. Ім’я файлу береться на основі самого файлу, точніше його властивості Name
* **GetFile ()** – метод який повертає при виклику, повну інформацію про файл
* **SetError (string Error)** – приймає рядок з інформацією про помилку та записує його у відповідне поле класу
* **GetError ()** – повертає рядок з текстом помилки
* Після формування списку, відбувається сам процес очищення файлів та виводу інформації в окрему новостворену таблицю DataGrid, в якій в поля: ім’я файлу та статус файлу, вносяться дані. Весь список формується та виводиться відразу ж в таблицю, а згодом поступово в мірі очищення кожного файлу, таблиця заповнюється згідно з статусом очищення файлу. У випадку невідповідності формату файлу, або іншої помилки, поле статусу відповідного файлу заповнюється червоним кольором. Успішно очищені файли жовтим кольором, і файли котрі знаходяться в процесі обробки помаранчевим. Відповідні статуси файлів – Done (Успішно), In Progress (В процесі), Failed (Помилка). На рисунку 4.11 представлений варіант коли виникає помилка.

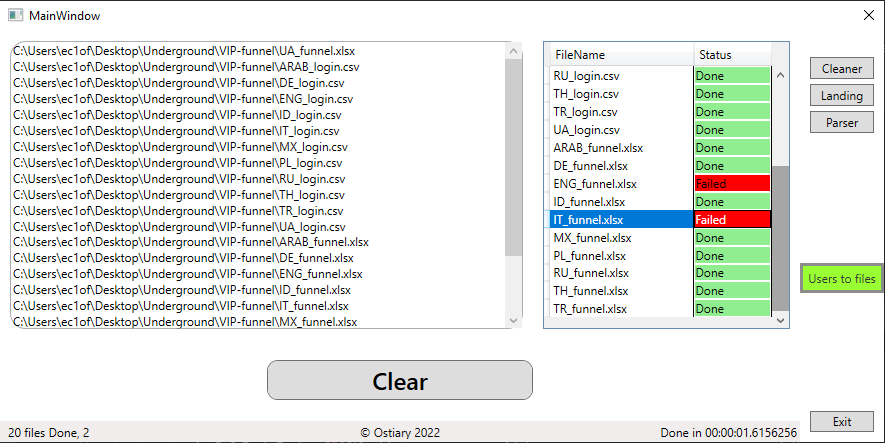


Рис. 4.11 Приклад виникнення помилки (симуляція)

Помилка була симульована тим, що деякі файли були відриті у додатку Excel. Як відомо, то редагувати файли, що відкриті в іншому додатку заборонено. Також, є додаткова функція. При натисканні на поле з помилкою, відкривається віконце з деталями помилки (рис. 4.12).

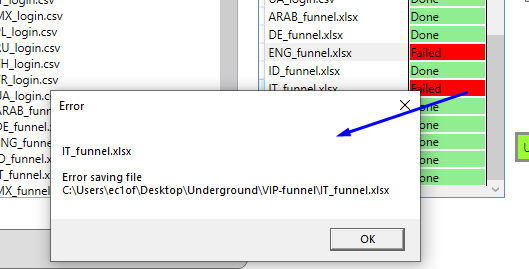


Рис. 4.12 Відображення деталей помилки

## Розробка основного модуля обробки Landing файлу

Цей модуль буде теж як окрема сторінка з файлами LandingPage.xaml та LandingPage.xaml.cs (інтерфейс і логіка відповідно).

Інтерфейс по структурі майже схожий на CleanerPage.xaml, але відрізняється тим, що у поле імпорту файлу, можливий імпорт лише Landing.csv файлу (одного). Вигляд інтерфейсу на рис. 4.13

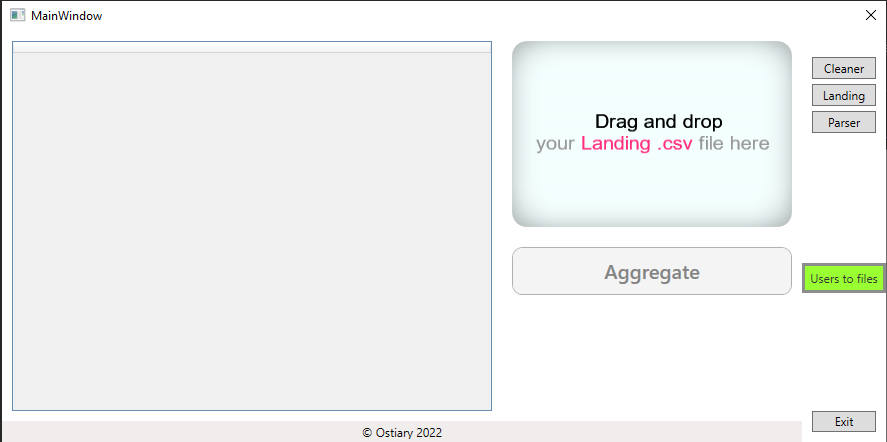


Рис. 4.13 Структура інтерфейсу Landing частини

При спробі імпорту невірного формату файлу, буде показане вікно з помилкою (рис. 4.14). Симуляція проводилась з імпортом файлу .xlsx формату.

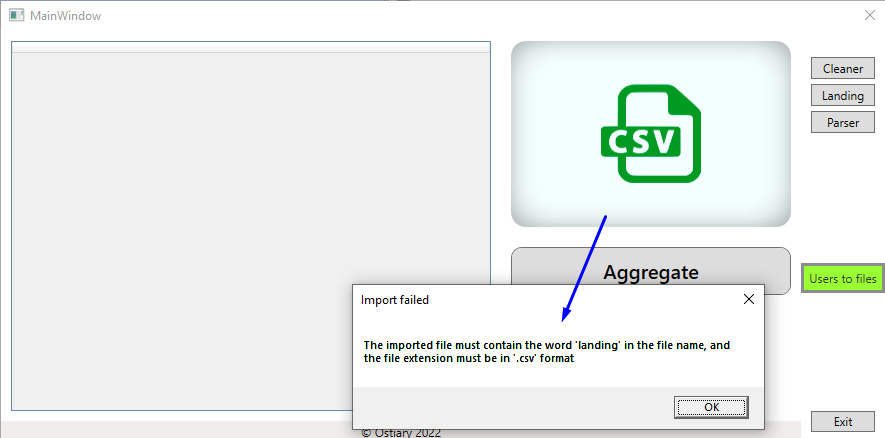


Рис. 4.14 Симуляція помилки імпорту невірного файлу

При успішному імпорті файлу, кнопка обробки активується і вона може бути натиснута. При активації, буде відображене вікно про те що стовпці у файлі потрібно розташувати у певному порядку, задля коректної роботи алгоритму (рис. 4.15)

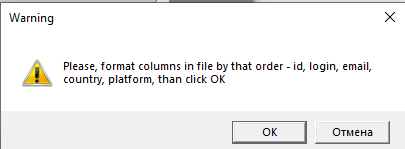


Рис. 4.15 Інформування користувача про відповідність стовпців

ОК – запустить алгоритм до виконання, Скасувати – дозволить знову імпортувати файл новий.

Сам алгоритм роботи складається з одного методу **AggregateLandingButton\_Click()**, його структура :

* Зчитується перший рядок файлу – заголовки стовпчиків з даними, повинні бути у такому порядку – id, login, email, geo, platform. Якщо це не так, то видається помилка, що заголовки знаходяться не в тому порядку. (рис. 4.16.1) Процес імпорту, активації потрібно робити знову.

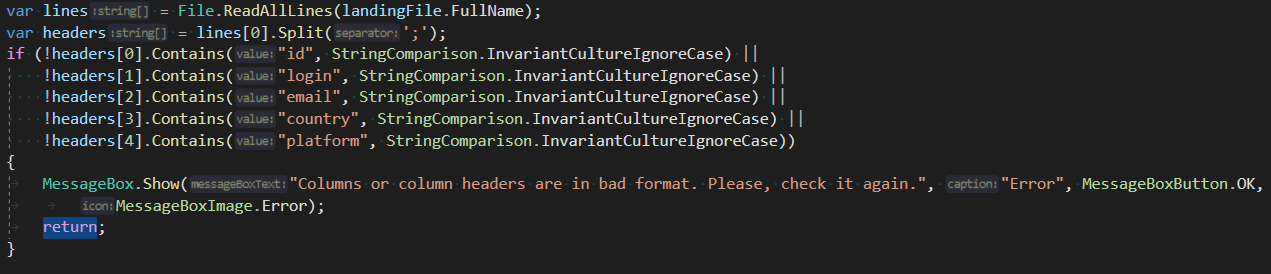


Рис. 4.16.1 Зчитування заголовків та їх перевірка

* Всі активні рядки файлу (окрім першого) формують список з інформацією про користувача типу User з відповідними назвами полів як у заголовків стовпчиків з даними.
* Файл копіюється в окремий каталог – резервне копіювання
* Країна кожного користувача замінюється на відповідний регіон (таблиця 2.2) (рис. 4.16.2)

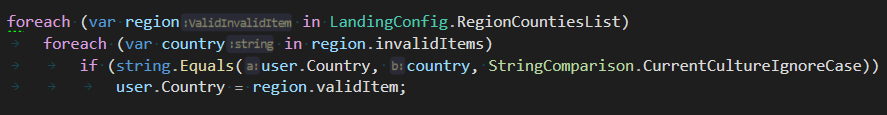


Рис. 4.16.2 Заміна країн на регіони

* Помилкові домени електронної пошти замінюються на відповідні вірні домени (таблиця 2.1)

Згідно з вимогами, був створений файл **landingSettings.json**, котрий можливо змінювати за потреби. Там знаходяться список вірних та невірних доменів, список регіонів та їх країн. На основі цього файлу й відбувається заміна країн та доменів пошт.

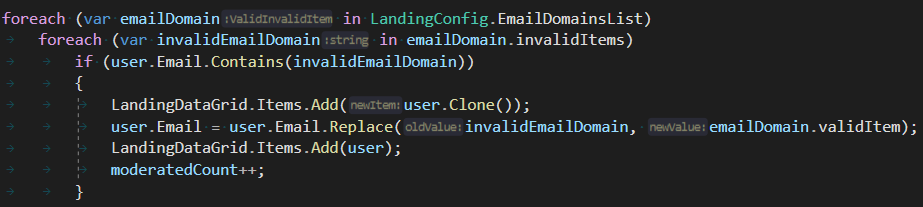


Рис. 4.16.3 Заміна помилкових доменів

* Оброблені дані файлу Landing.csv зберігаються назад у файл.
* Також оброблені рядки файлу (користувачі) виводяться у таблицю результатів. Червоний рядок – первинний вид (той, що потрібно обробити), зелений – фінальний (оброблений). Зміни відображаються лише у доменах пошти, щоб розуміти які помилки були знайдені у файлі. (рис. 4.17)

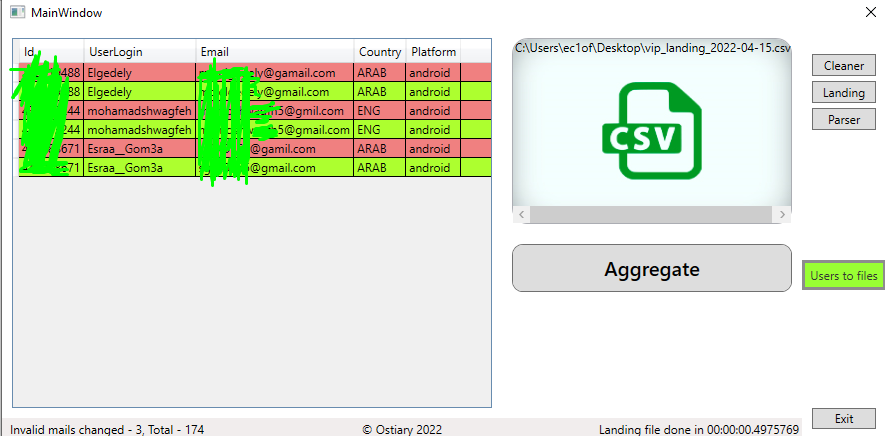


Рис. 4.17 Результати обробки Landing файлу

Деякі дані були замальовані, задля запобігання збігів конфіденційних даних.

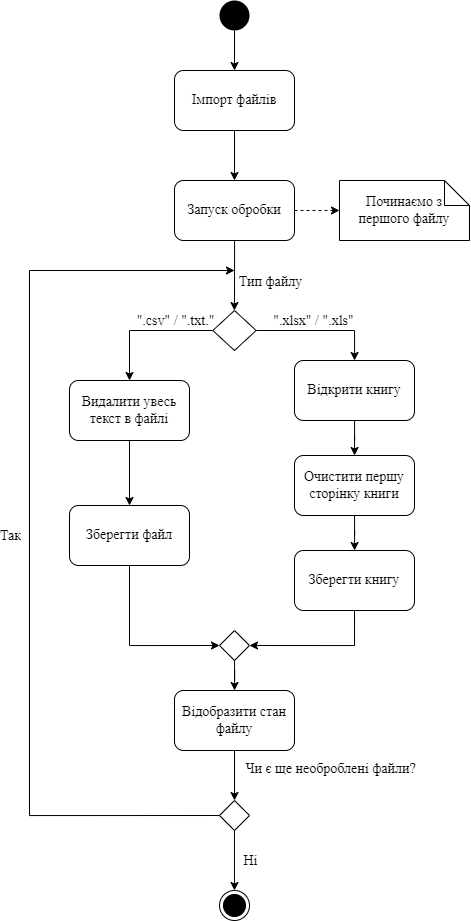


Рис. 4.18 Діаграма діяльності очищення файлів

## Розробка основного модуля обробки Typeform файлу

На цьому етапі розробки постає задача роботи з 2-ма файлами. Оскільки, задля коректної роботи алгоритму, нам потрібен сам файл Typeform.xlsx і файл Total.xlsx, тоді вони є обов’язковими (обидва) для нашої сторінки. Саме тому інтерфейс нашої сторінки змінився.

Сторінка має 2 зони імпорту – файл Typeform, та файл Total. Відповідно перевірки на тип та ім’я файлу присутні. При спробі імпорту не Excel файлу, або ж файлу з іншим ім’ям, то буде виведена помилка про невірний імпорт (рис. 4.19). Приклад – імпорт файлу Landing у зону імпорту файлу Total.

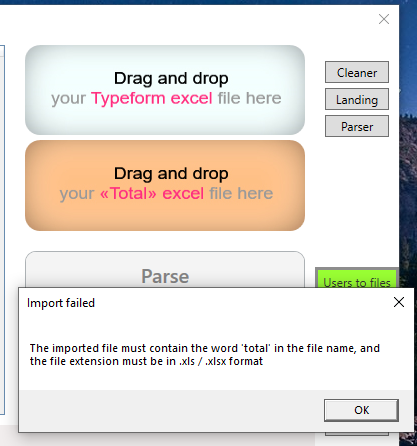


Рис. 4.19 Симуляція помилки невірного імпорту

Як і на інших сторінках, при вірному імпорті, кнопка активації обробки стає активною, і її можливо натиснути.

Додатково для відображення дій алгоритму та результатів роботи, на сторінці присутні два DataGrid – ліва таблиця відображає вхідні дані з файлу – сирий формат. Права таблиця відображає готовність кожного користувача (точніше його логінів). Дані в правій таблиці відображаються парами – логін + статус логіну. Статус логіну проставляється відповідно до перевірки:

* Якщо, вебсторінка користувача існує, результат – ОК, заливка жовтим кольором
* Якщо користувача не існує, або його сторінка заблокована в системі, то в полі статусу логіну буде відображена помилка, та код помилки. Заливка помаранчевим кольором.

Приклад результатів роботи представлений на рис. 4.20

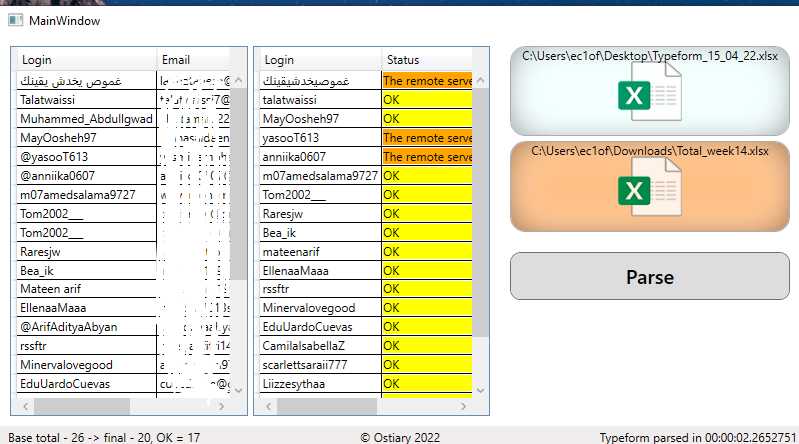


Рис. 4.20 Приклад результатів роботи алгоритму обробки Typeform файлу

Опис алгоритму обробки файлу Typeform.xlsx:

* Імпорт файлу Typeform. Якщо ім’я файлу не містить слово “typeform” та(або) не є Excel файлом (розширення файлу .xlsx, .xls), то буде виведена помилка. Потрібен повторний імпорт
* Імпорт файлу Total працює по ти же правилам, що і імпорт файлу Typeform, але замість слова “typeform”, ім’я файлу повинно містити слово “total”.
* Після успішного імпорту файлів, користувач натискає на кнопку опрацювання даних. Починає свою роботу алгоритм обробки.
* Файли проходять етап резервного копіювання
* Формується список користувачів типу TypeformUser. У класі користувача є 3 поля, що відповідають ім’ям стовпчиків у файлі Typeform, а саме: login, email, geo.
* Кожен логін користувача форматується згідно з правилами – видалити зайві символи: символ “@”, та усі пробільні символи з логіну.
* Дані у напів-сирому вигляді відображаються у лівій таблиці DataGrid, що дозволяє порахувати скільки користувачів заповнили форму реєстрації (хоча це також можливо проглянути у самому файлі Typeform, відкривши його за допомогою Excel додатку)
* Видаляються дублікати (весь рядок даних) у випадку якщо адреса електронної пошти або логін дублюються
* Видалення користувачів, що вже зареєстровані, тобто їх логіни наявні у файлі Total
* Далі для правої таблиці формується окремий список типу LoginStatus. Цей клас містить всього два текстових поля – логін і його статус.
* У коді створюється віртуальний браузер (вебклієнт) для роботи з вебсторінками.
* Формується посилання на сторінку користувача і відбувається перехід до неї. І відповідно тому чи існує сторінка чи ні, проставляється статус користувача і виводиться у таблицю.
* У випадку існування сторінки, з неї витягується логін користувача, задля забезпечення коректної співпраці наступним ланок циклу обробки даних. Вже наявні логіни користувачів замінюються на витягнуті зі сторінки користувача.
* Після всього етапу перевірки, список користувачів формується остаточно і записується назад у файл Typeform.xlsx

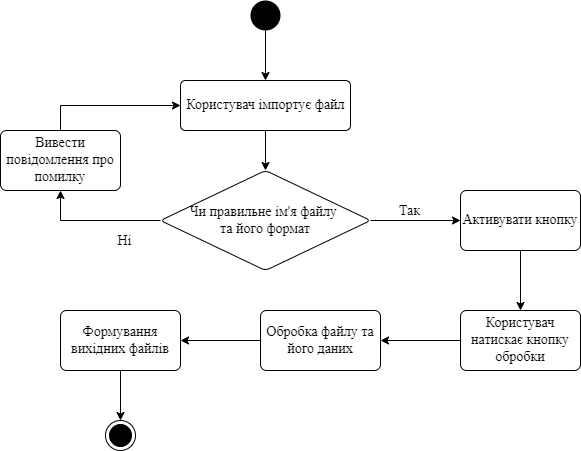


Рис. 4.21 Діаграма діяльності обробки файлів Landing та Typeform

## Розробка основного модуля групування та розкидування даних

Перед початком розробки цього модуля, хотілося б приділити увагу інтерфейсу додатку, що не належить жодній зі сторінок, або модулю, що були описані вище. Тобто маю на увазі, головний інтерфейс додатку.

Головний інтерфейс складається з 3 кнопок перемикання сторінок (модулів описаних вище), при натисненні на кожну кнопку, в контейнері сторінки, що займає більшу частину інтерфейсу, буде відображена необхідна нам сторінка.

Також для забезпечення вимог про екстрену зупинку, у правому нижньому кутку додатка існує кнопка виходу. Після підтвердження виходу з додатку, застосунок припинить свою роботу і закриється. Інтерфейс виглядає наступним чином (рис. 4.22). Червоним прямокутник виділено контейнер для сторінок – Frame.

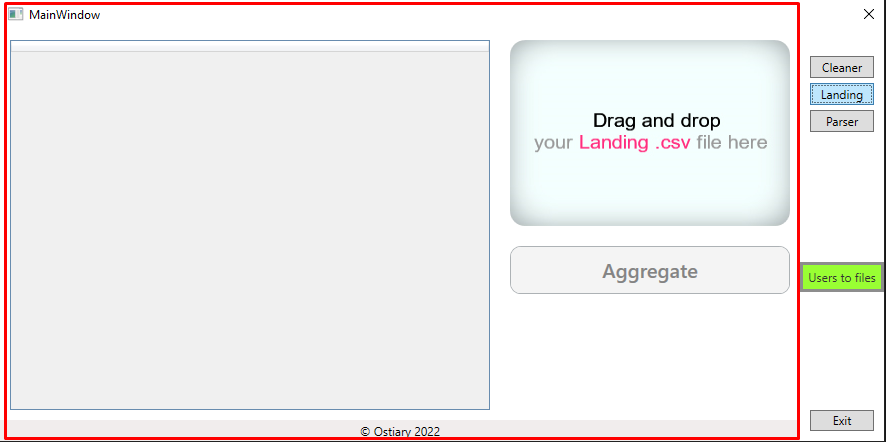


Рис. 4.22 Інтерфейс додатку загалом

А тепер перейдемо до групування.

Процес групування даних базується на файлах Landing і Typeform. Процес буде відбуватись навіть при наявності всього одного з них файлів, що знову забезпечує модульність. Кнопка, що відповідає за процес виділена зеленим кольором на рисунку 4.22. Сам процес групування та розкидання (рис. 4.24, 4.25):

* Якщо процес групування відбувається без попереднього імпорту Landing та(або) Typeform файлу, то буде виведена помилка з інформацією про те що потрібно спочатку імпортувати файли необхідні для групування. (рис. 4.23)

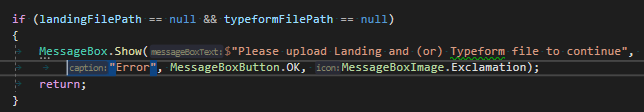


Рис. 4.23 Перевірка на наявність хоча б одного файлу для групування

* У випадку якщо ж один, або обидва файли імпортовані, починається процес групування
* З файлів Typeform та Landing зчитуються дані (логін, пошта та регіон) і формується список користувачів типу Credentials, де є 3 відповідних поля – Login, Email, Region
* Далі, користувачі формуються у групи по відповідності до їх регіону. Регіони, що підлягають групуванню та розкидуванню представлені у таблиці 2.2. Регіони які не присутні у цій таблиці, не обробляються
* Для кожного регіону створюється окремий файл Excel формату з назвою «регіон»\_funnel.xlsx, де «регіон» - назва відповідного регіону з таблиці. В файл записуються дані кожного користувача відповідного регіону у форматі – логін + пошта. Перший рядок файлу має заголовки login та email.
* Також для кожного регіону створюється текстовий файл з ім’ям «регіон»\_funnel.csv. В файл записується лише логін кожного користувача.
* Всі сформовані (створені) файли записуються в окрему папку, де зберігаються для передачі цих вихідних даних у наступну частину циклу обробки даних



Рис. 4.24 Процес групування та розкидування даних

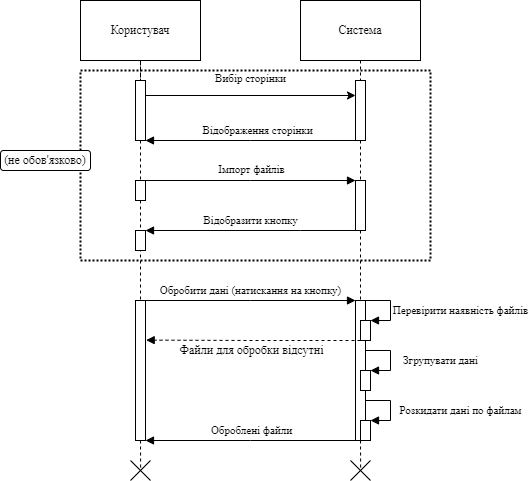


Рис. 4.25 Діаграма послідовності групування та розкидування

## Продуктивність застосунку

Задля того, щоб додатково переконатись у актуальності та необхідності застосунку, проведемо заміри часу обробки користувачів. Заміри будуть проводитись на однаковій групі файлів.

Ручний алгоритм обробки – Landing – ~3 хвилини, Typeform – 5 хвилин,

групування та розкидування – ~4 хвилини (у випадку якщо вихідні файли не потрібно створювати, а лише очищувати) + ~3 хвилини на створення файлів для вихідних даних = приблизно 15 хвилин часу

Автоматизований алгоритм (розроблений застосунок) – всі ті самі кроки, менше ніж за хвилину. З урахуванням імпорту файлів та деякими затримками – приблизно 33 секунди. Сам же алгоритм відпрацював так:

* Landing – менше ніж за секунду
* Typeform – 3.27 секунди
* Групування – менше ніж за 2 секунди

Тому враховуючи всі обставини, 15 хвилин на ручну обробку, було автоматизовано до приблизно 33-х секунд. Відповідно можливо порахувати у скільки разів було збільшено ефективність процесу обробки користувачів.

~15 хвилин = 900 секунд, 900 / 33 = 27,27

Тобто, приблизно у 25-27 разів автоматизований застосунок швидший ніж ручний метод обробки.

Щодо того як багато ресурсів використовує додаток, то на рис. 4.25, видно, що наш додаток можливо використовувати навіть на слабких системах. 25% від усіх процесорів (у додатку був задіяний лише 1 із 8 процесорів) та 168 МБ оперативної пам’яті, що загалом показує нам, що додаток є легким для системи.

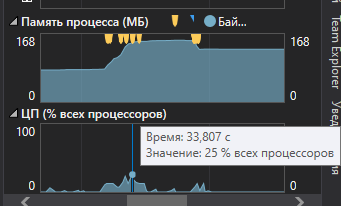


Рис. 4.25 Використання ресурсів системи.

# ВИСНОВКИ

У результаті виконання даної дипломної роботи було розроблено десктопний додаток задля автоматизації процесу постреєстраційної обробки даних користувачів. У роботі зазначені основні принципи та кроки розробки додатку.

Було проведено дослідження, що виявляють актуальність даної роботи. А саме рівень та відсоток помилок та кількість часу на обробку даних, проаналізовано типи систем обробки даних.

На основі переваг та недоліків існуючих типів систем обробки даних і алгоритму обробки даних, було проаналізовано вимоги та спроєктовано новий застосунок для забезпечення автоматизації обробки користувачів. Спроєктований додаток відповідає усім встановленим вимогам.

Проведено аналіз існуючих програмних засобів для реалізації розробки застосунку під ОС Windows та роботи з табличними файлами Excel. Таким чином було обрано мову програмування C# та середовище розробки Visual Studio, користувацький інтерфейс додатку було створено за допомогою технології WPF.

Для перевірки правильності та продуктивності роботи було проведено порівняльний аналіз роботи застосунку.

Розроблено автоматизовану систему яка дозволяє обробляти постреєстраційні дані користувачів з урахуванням та відповідності до встановлених вимог. Розроблений додаток може використовувати будь-який кваліфікований працівник комерційної компанії, що знає алгоритм обробки даних. Користування додатком можливо простим розташування файлів додатку у зручному місці.

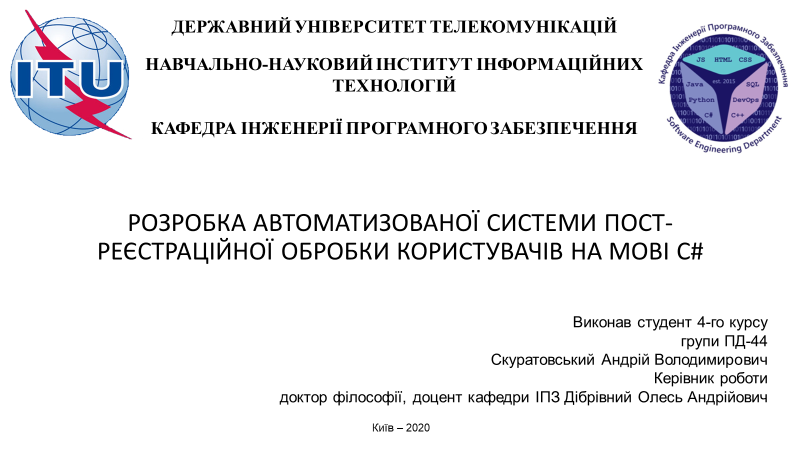
Надалі планується розробити:

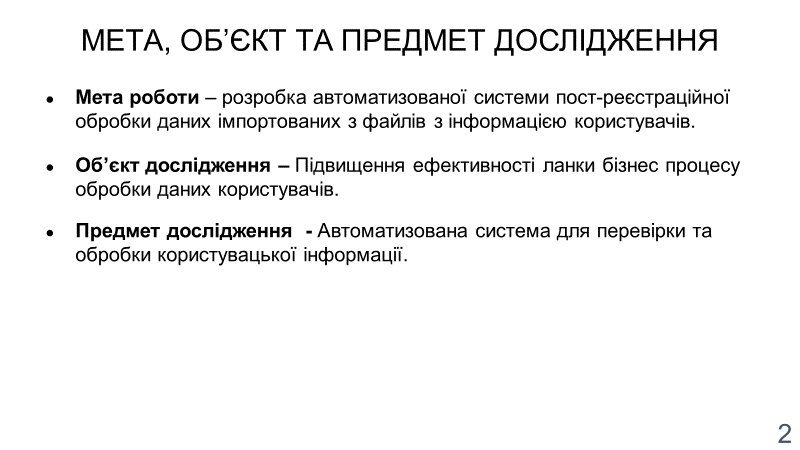
1. Взаємодію з Google Таблицями
2. Імпорт вже існуючих вихідних файлів, задля їх подальшої обробки
3. Архівування вихідних файлів
4. Можливість очищення файлів більшої кількості форматів.

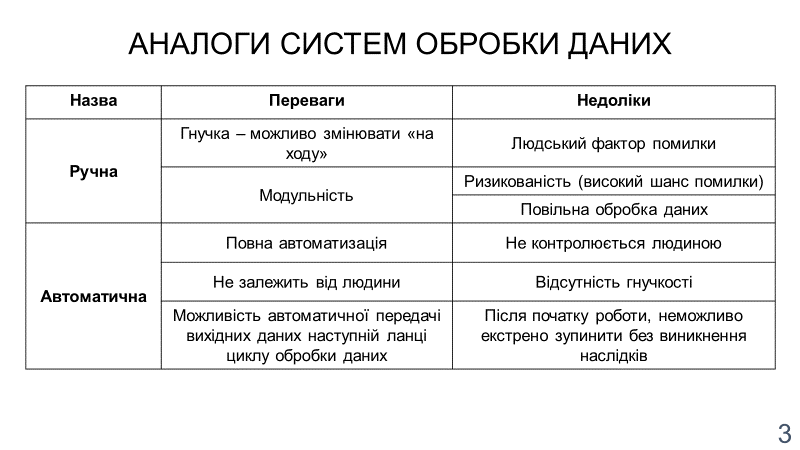
# ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

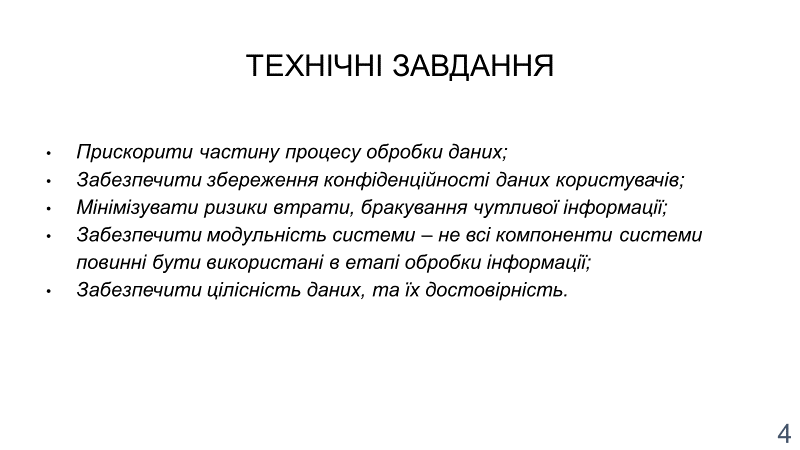
1. Коноваленко І.В. Програмування мовою C# 6.0 / – Тернопіль, ТНТУ, 2016. с. 227.
2. ANDREW TROELSEN, Philip Japikse, C# 6.0 and the .NET 4.6 Framework /- Apress, 2015 – c. 1083.
3. Desktop Guide for .NET, .NET Core, and .NET Framework [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.microsoft.com/en-us/dotnet/desktop/?view=netdesktop-6.0>.
4. Differences Between .NET Core and .NET Framework [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/differences-between-net-core-and-net-framework/>.
5. A tour of the C# language [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: <https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>.
6. Developers Tutorials, documentation and other developer resources [Електронний ресурс] – Режим доступу до ресурсу: <https://epplussoftware.com/en/Developers/>.
7. Duggal N. What Is Data Processing: Cycle, Types, Methods, Steps and Examples [Електронний ресурс] / Nikita Duggal. – 2022. – Режим доступу до ресурсу: <https://www.simplilearn.com/what-is-data-processing-article>.
8. What is Data Processing: Definition, Cycle and Types [Електронний ресурс] – Режим доступу до ресурсу: <https://itechdata.ai/types-of-data-processing/>.

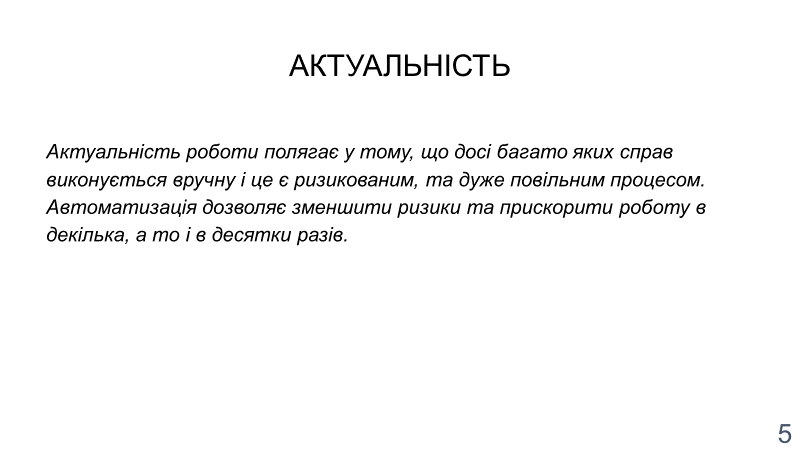
# Додаток A

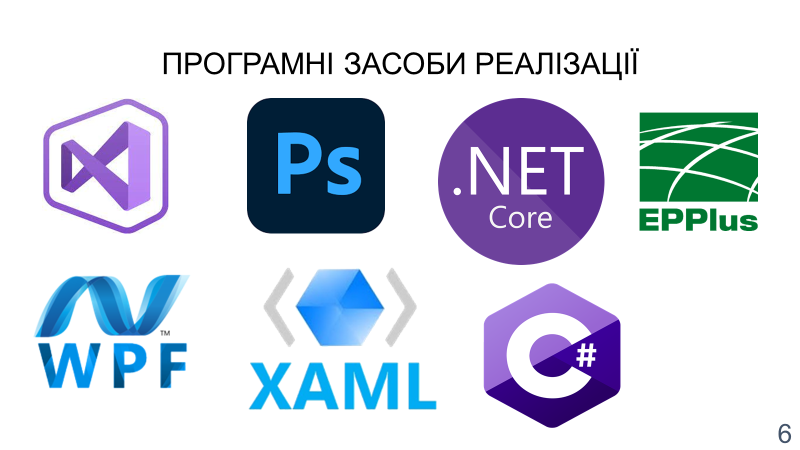


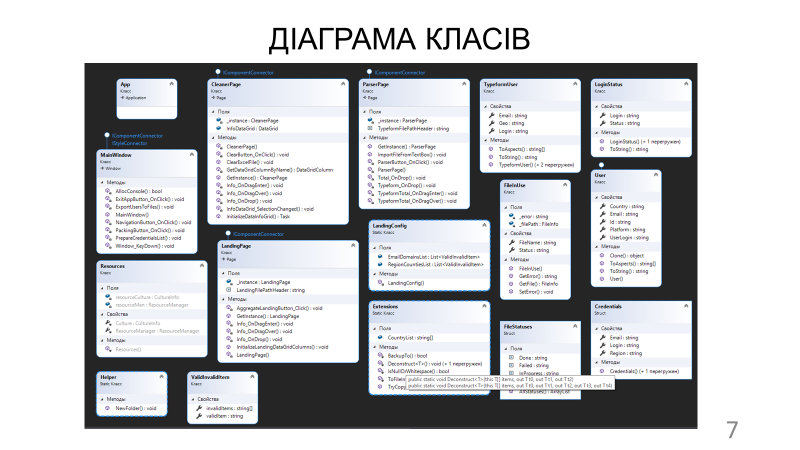




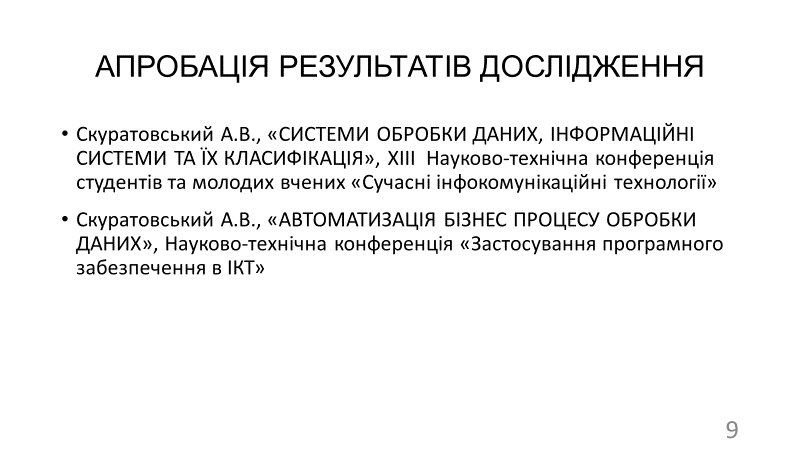


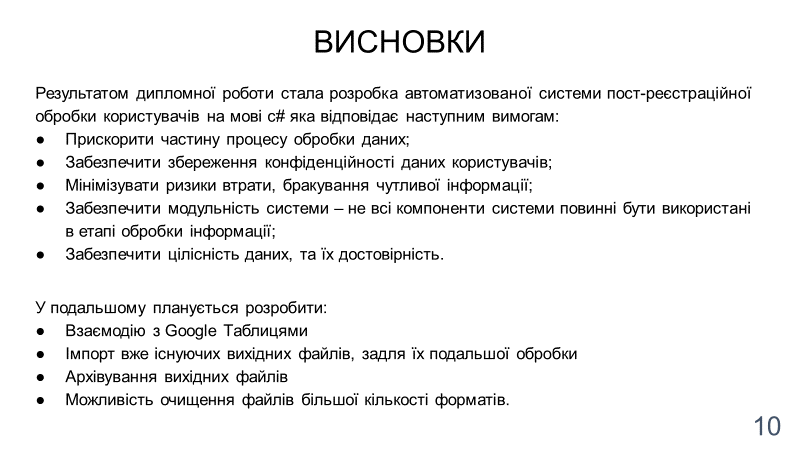


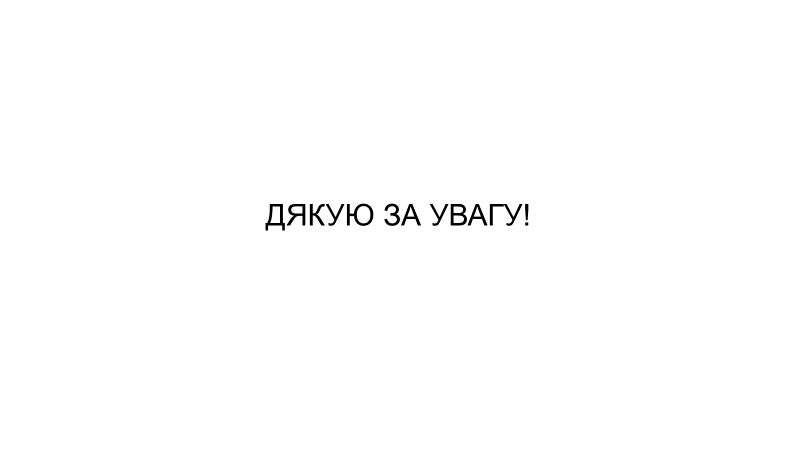












# Додаток Б

Увесь код застосунку:

**MainWindow.xaml**

<Window

x:Class="ASPP.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"

Title="MainWindow"

Width="900"

Height="450"

Left="0"

ResizeMode="NoResize"

Top="0"

WindowStartupLocation="Manual"

mc:Ignorable="d">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="800" />

<ColumnDefinition Width="1\*" />

</Grid.ColumnDefinitions>

<Frame

Name="MainFrame"

Grid.Column="0"

NavigationUIVisibility="Hidden" />

<Grid Grid.Column="1">

<Grid.RowDefinitions>

<RowDefinition Height="3\*" />

<RowDefinition Height="6\*" />

<RowDefinition Height="1\*" />

</Grid.RowDefinitions>

<StackPanel

Name="NavigationPanel"

Grid.Row="0"

VerticalAlignment="Center">

<StackPanel.Resources>

<Style TargetType="Button">

<Setter Property="Margin" Value="10,5,10,0" />

<Setter Property="Padding" Value="0,2" />

<EventSetter Event="Click" Handler="NavigationButton\_OnClick" />

</Style>

</StackPanel.Resources>

<Button Name="CleanerPageNavigationButton" Content="Cleaner" />

<Button Name="MainPageNavigationButton" Content="Landing" />

<Button Name="ParserPageNavigationButton" Content="Parser" />

</StackPanel>

<StackPanel Grid.Row="1" VerticalAlignment="Center">

<Button

Name="PackingButton"

Grid.Row="1"

Height="30"

Background="Chartreuse"

BorderThickness="3"

Click="PackingButton\_OnClick"

Content="Users to files"

Opacity="0.8" />

</StackPanel>

<Button

Name="ExitAppButton"

Grid.Row="2"

Margin="10"

Click="ExitAppButton\_OnClick"

Content="Exit" />

</Grid>

</Grid>

</Window>

**MainWindow.xaml.cs**

using ASPP.Pages;

using OfficeOpenXml;

using System;

using System.Collections.Generic;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Runtime.InteropServices;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using Path = System.IO.Path;

namespace ASPP

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

[DllImport("kernel32.dll", SetLastError = true)]

[return: MarshalAs(UnmanagedType.Bool)]

static extern bool AllocConsole();

public MainWindow()

{

InitializeComponent();

ExcelPackage.LicenseContext = LicenseContext.NonCommercial;

}

private async void Window\_KeyDown(object sender, KeyEventArgs e)

{

await Dispatcher.InvokeAsync(() =>

{

if (Keyboard.IsKeyDown(Key.LeftCtrl) && Keyboard.IsKeyDown(Key.W))

this.Close();

});

}

private void NavigationButton\_OnClick(object sender, RoutedEventArgs e)

{

var navigationButton = sender as Button;

switch (navigationButton!.Content)

{

case "Cleaner":

this.MainFrame.Navigate(CleanerPage.GetInstance());

break;

case "Landing":

this.MainFrame.Navigate(LandingPage.GetInstance());

break;

case "Parser":

this.MainFrame.Navigate(ParserPage.GetInstance());

break;

}

}

private void ExitAppButton\_OnClick(object sender, RoutedEventArgs e)

{

if (MessageBox.Show("Confirm exit", "Exit ?", MessageBoxButton.OKCancel) == MessageBoxResult.OK)

this.Close();

}

private void PackingButton\_OnClick(object sender, RoutedEventArgs e)

{

var landingFilePath = Environment.GetEnvironmentVariable(LandingPage.LandingFilePathHeader);

var typeformFilePath = Environment.GetEnvironmentVariable(ParserPage.TypeformFilePathHeader);

var usersList = new List<Credentials>();

if (landingFilePath == null && typeformFilePath == null)

{

MessageBox.Show($"Please upload Landing and (or) Typeform file to continue",

"Error", MessageBoxButton.OK, MessageBoxImage.Exclamation);

return;

}

PrepareCredentialsList(landingFilePath, typeformFilePath, usersList);

ExportUsersToFiles(usersList, Path.Combine(Directory.GetParent(landingFilePath ?? typeformFilePath)!.FullName, $"VIP-funnel\_{DateTime.Now:dd\_MM}"));

MessageBox.Show("Done", "Users successfully exported to funnel files", MessageBoxButton.OK, MessageBoxImage.None, MessageBoxResult.OK, MessageBoxOptions.RightAlign);

}

/// <summary>

/// Export users credentials by countries, such as : logins + emails to excel funnel files and logins to csv files

/// </summary>

/// <param name="usersList">Users credentials list</param>

/// <param name="funnelFolderPath">Folder path to store funnel files</param>

private static void ExportUsersToFiles(IEnumerable<Credentials> usersList, string funnelFolderPath)

{

var countryList = usersList.GroupBy(u => u.Region)

.Where(g => Extensions.CountryList.Any(g.Key.Contains)).ToList();

Helper.NewFolder(funnelFolderPath); // Folder for funnel files

foreach (var country in countryList)

{

// Logins+Emails into excel file

using (var excelFile = new ExcelPackage())

{

excelFile.Workbook.Worksheets.Add("Лист1")

.Cells.LoadFromCollection(country.Select(c => new { login = c.Login, email = c.Email }), true);

excelFile.SaveAsAsync(NewFunnelFile(".xlsx"));

}

// Logins into csv file

var csvFile = NewFunnelFile(".csv");

File.WriteAllLinesAsync(csvFile.FullName, country.Select(c => c.Login).ToArray());

//local method for creating new funnel file

FileInfo NewFunnelFile(string extension) => new (Path.Combine(funnelFolderPath, $"{country.Key}\_funnel{extension}"));

}

}

/// <summary>

/// Import users credentials from Landing and(or) Typeform file(-s) into list of

/// </summary>

/// <param name="landingFilePath"></param>

/// <param name="typeformFilePath"></param>

/// <param name="usersList"></param>

private void PrepareCredentialsList(string? landingFilePath, string? typeformFilePath, List<Credentials> usersList)

{

if (landingFilePath != null)

File.ReadAllLines(landingFilePath)[1..].ToList().ForEach(l =>

{

var ls = l.Split(';');

usersList.Add(new Credentials(ls[1..4]));

});

if (typeformFilePath == null) return;

using var typeformSheet = new ExcelPackage(new FileInfo(typeformFilePath)).Workbook.Worksheets[0];

int startColumn = typeformSheet.Dimension.Start.Column;

string filterColor = Color.Yellow.ToArgb().ToString("X2");

for (int i = typeformSheet.Dimension.Start.Row + 1; i < typeformSheet.Dimension.Rows; i++)

{

if (typeformSheet.Cells[i, startColumn] is var loginCell &&

loginCell.Style.Fill.BackgroundColor.Rgb == filterColor)

usersList.Add(

new Credentials(

loginCell.Value.ToString(),

typeformSheet.Cells[i, startColumn + 1].Value.ToString(),

typeformSheet.Cells[i, startColumn + 2].Value.ToString()

)

);

}

}

}

}

**CleanerPage.xaml**

<Page

x:Class="ASPP.Pages.CleanerPage"

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"

Title="Cleaner"

d:DesignHeight="450"

d:DesignWidth="800"

mc:Ignorable="d">

<Grid Name="MainGrid">

<Grid.RowDefinitions>

<RowDefinition Height="75\*" />

<RowDefinition Height="20\*" />

<RowDefinition Height="5\*" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

</Grid.ColumnDefinitions>

<TextBox

Name="InfoTextBox"

Grid.Row="0"

Grid.Column="0"

Grid.ColumnSpan="6"

Margin="10"

AllowDrop="True"

BorderThickness="0"

Cursor="Arrow"

HorizontalScrollBarVisibility="Auto"

IsReadOnly="True"

PreviewDragEnter="Info\_OnDragEnter"

PreviewDragOver="Info\_OnDragOver"

PreviewDrop="Info\_OnDrop"

VerticalScrollBarVisibility="Auto">

<TextBox.Background>

<ImageBrush

AlignmentX="Left"

ImageSource="../Images/CleanerWatermarkImage.jpg"

Stretch="Fill" />

</TextBox.Background>

<TextBox.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="15" />

</Style>

</TextBox.Resources>

</TextBox>

<!--<DataGrid Name="BB" Margin="10" Grid.Row="0" Grid.Column="4" Grid.ColumnSpan="2" IsReadOnly="True"></DataGrid>-->

<Button

Name="ClearButton"

Grid.Row="1"

Grid.Column="2"

Grid.ColumnSpan="2"

Height="40"

Click="ClearButton\_OnClick"

Content="Clear"

Cursor="Hand"

FontSize="24"

FontWeight="DemiBold"

IsEnabled="False">

<Button.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="10" />

</Style>

</Button.Resources>

</Button>

<StatusBar

Grid.Row="2"

Grid.Column="0"

Grid.ColumnSpan="6">

<StatusBar.ItemsPanel>

<ItemsPanelTemplate>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

</Grid.ColumnDefinitions>

</Grid>

</ItemsPanelTemplate>

</StatusBar.ItemsPanel>

<StatusBarItem Grid.Column="0">

<TextBlock Name="AppInfoText" Margin="5,0,0,0" />

</StatusBarItem>

<StatusBarItem Grid.Column="1" HorizontalContentAlignment="Center">

<TextBlock Text="© Ostiary 2022" />

</StatusBarItem>

<StatusBarItem Grid.Column="2" HorizontalContentAlignment="Right">

<TextBlock Name="TimerInfoText" TextAlignment="Right" />

</StatusBarItem>

</StatusBar>

</Grid>

</Page>

**CleanerPage.xaml.cs**

using ASPP.Pages.CleanerPageElements;

using OfficeOpenXml;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

namespace ASPP.Pages

{

/// <summary>

/// Логика взаимодействия для CleanerPage.xaml

/// </summary>

public partial class CleanerPage : Page

{

private static CleanerPage \_instance;

public static CleanerPage GetInstance() => \_instance ??= new CleanerPage();

private CleanerPage()

{

InitializeComponent();

}

}

public partial class CleanerPage : Page

{

public DataGrid InfoDataGrid;

#region Drag&Drop\_Events

private void Info\_OnDrop(object sender, DragEventArgs e)

{

var textBox = sender as TextBox;

Dispatcher.InvokeAsync(() =>

{

Mouse.OverrideCursor = Cursors.Wait;

textBox.Clear();

string[] imports = (string[])e.Data.GetData(DataFormats.FileDrop);

foreach (var fileInfo in imports)

InfoTextBox.Text += fileInfo + "\n";

AppInfoText.Text = imports.Length + " files imported";

textBox.Background.Opacity = 0;

textBox.BorderThickness = new Thickness(1);

Mouse.OverrideCursor = null;

ClearButton.IsEnabled = !InfoTextBox.Text.IsNullOrWhitespace();

});

//Button cursor hand ?

}

private void Info\_OnDragEnter(object sender, DragEventArgs e)

{

if (e.Data.GetDataPresent(DataFormats.FileDrop, false))

e.Effects = DragDropEffects.All;

}

private void Info\_OnDragOver(object sender, DragEventArgs e) => e.Handled = true;

#endregion

private void ClearButton\_OnClick(object sender, RoutedEventArgs e)

{

Dispatcher.InvokeAsync(async () =>

{

ClearButton.IsEnabled = false;

DateTime st = DateTime.Now;

//Get all lines (file paths) Drag&Drop Zone (textBox), format (delete newLines + split)

List<FileInUse> filesList = InfoTextBox.Text.Split("\n")

.Where(fp => !string.IsNullOrWhiteSpace(fp))

.Select(fP => new FileInUse(fP.Replace("\n", "")))

.ToList();

//Initialize DataGrid

await InitializeDataInfoGrid(filesList);

InfoDataGrid.Cursor = Cursors.Wait;

var isAnyErrorFired = false;

foreach (var elementFileInUse in filesList)

{

FileInfo file = elementFileInUse.GetFile();

try

{

switch (file.Extension)

{

case ".txt":

case ".csv":

await File.WriteAllTextAsync(file.FullName, string.Empty);

break;

case ".xls":

case ".xlsx":

await Task.Run(() => ClearExcelFile(file));

break;

default:

throw new Exception("Incorrect extension of the file");

}

elementFileInUse.Status = FileStatuses.Done;

}

catch (Exception exception)

{

elementFileInUse.Status = FileStatuses.Failed;

elementFileInUse.SetError(exception.Message);

isAnyErrorFired = true;

}

InfoDataGrid.Items.Refresh();

}

if (isAnyErrorFired)

InfoDataGrid.SelectionChanged += InfoDataGrid\_SelectionChanged;

AppInfoText.Text = $"{filesList.Count(f => f.Status == FileStatuses.Done)} files Done" + (isAnyErrorFired ? ", " + filesList.Count(f => f.Status == FileStatuses.Failed) : "");

TimerInfoText.Text = "Done in " + (DateTime.Now - st);

ClearButton.IsEnabled = true;

InfoDataGrid.Cursor = null;

});

}

public static void ClearExcelFile(FileInfo file)

{

using var excel = new ExcelPackage(file);

foreach (var worksheet in excel.Workbook.Worksheets)

{

worksheet.Cells.Clear();

worksheet.View.ActiveCell = worksheet.View.TopLeftCell;

}

excel.Save();

}

public async Task InitializeDataInfoGrid(IEnumerable<FileInUse> source)

{

#region Initialization

// Make space for datagrid

InfoTextBox.SetValue(Grid.ColumnSpanProperty, 4);

InfoDataGrid = new DataGrid()

{

Margin = new Thickness(10),

IsReadOnly = true,

GridLinesVisibility = DataGridGridLinesVisibility.Vertical,

AutoGenerateColumns = true,

//RowDetailsTemplate = new DataTemplate(){} - How ?

};

await Dispatcher.InvokeAsync(() => MainGrid.Children.Add(InfoDataGrid));

InfoDataGrid.ItemsSource = source;

InfoDataGrid.UpdateLayout(); // Initializing columns

InfoDataGrid.Items.Refresh(); // If remove this line - will be raised exception - "TriggerCollection is sealed"

Grid.SetRow(InfoDataGrid, 0);

Grid.SetColumn(InfoDataGrid, 4);

Grid.SetColumnSpan(InfoDataGrid, 2);

var fileNameColumn = GetDataGridColumnByName(InfoDataGrid, nameof(FileInUse.FileName)) as DataGridTextColumn;

var statusColumn = GetDataGridColumnByName(InfoDataGrid, nameof(FileInUse.Status)) as DataGridTextColumn;

fileNameColumn.Width = new DataGridLength(65, DataGridLengthUnitType.Star);

statusColumn.Width = new DataGridLength(35, DataGridLengthUnitType.Star);

#endregion

#region Triggers

statusColumn!.ElementStyle = new Style(typeof(TextBlock));

foreach (var s in FileStatuses.AllStatuses())

statusColumn.ElementStyle.Triggers.Add(new Trigger()

{

Property = TextBlock.TextProperty,

Value = s,

Setters = { new Setter(

TextBlock.BackgroundProperty,

s switch

{

FileStatuses.Done => Brushes.LightGreen,

FileStatuses.InProgress => Brushes.DarkOrange,

FileStatuses.Failed => Brushes.Red,

\_ => Brushes.White

})

}

});

#endregion

}

private DataGridColumn GetDataGridColumnByName(DataGrid dtGrid, string header)

{

return dtGrid.Columns.FirstOrDefault(c => c.Header.ToString() == header);

}

private void InfoDataGrid\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

var item = (sender as DataGrid).SelectedItem as FileInUse;

if (item.Status == FileStatuses.Failed)

MessageBox.Show($"{item.FileName}{Environment.NewLine}{Environment.NewLine}{item.GetError()}", "Error", MessageBoxButton.OK);

}

}

}

**LandingPage.xaml**

<Page

x:Class="ASPP.Pages.LandingPage"

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"

Title="Landing"

d:DesignHeight="450"

d:DesignWidth="800"

mc:Ignorable="d">

<Grid>

<Grid.RowDefinitions>

<RowDefinition />

<RowDefinition />

<RowDefinition />

<RowDefinition />

<RowDefinition Height="1.7\*" />

<RowDefinition Height="0.3\*" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

</Grid.ColumnDefinitions>

<TextBox

Name="LandingTextBox"

Grid.Row="0"

Grid.RowSpan="3"

Grid.Column="5"

Grid.ColumnSpan="3"

Margin="10"

AllowDrop="True"

BorderThickness="0"

Cursor="Arrow"

HorizontalScrollBarVisibility="Auto"

IsReadOnly="True"

PreviewDragEnter="Info\_OnDragEnter"

PreviewDragOver="Info\_OnDragOver"

PreviewDrop="Info\_OnDrop"

VerticalScrollBarVisibility="Auto">

<TextBox.Background>

<ImageBrush

AlignmentX="Left"

ImageSource="../Images/LandingWatermarkImage.jpg"

Stretch="Fill" />

</TextBox.Background>

<TextBox.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="15" />

</Style>

</TextBox.Resources>

</TextBox>

<DataGrid

Name="LandingDataGrid"

Grid.Row="0"

Grid.RowSpan="5"

Grid.Column="0"

Grid.ColumnSpan="5"

Margin="10"

AlternatingRowBackground="GreenYellow"

IsReadOnly="True"

RowBackground="LightCoral" />

<Button

Name="AggregateLandingButton"

Grid.Row="3"

Grid.Column="5"

Grid.ColumnSpan="3"

Margin="10"

Click="AggregateLandingButton\_Click"

Content="Aggregate"

FontSize="20"

FontWeight="DemiBold"

IsEnabled="False">

<Button.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="10" />

</Style>

</Button.Resources>

</Button>

<StatusBar

Grid.Row="6"

Grid.Column="0"

Grid.ColumnSpan="8">

<StatusBar.ItemsPanel>

<ItemsPanelTemplate>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

</Grid.ColumnDefinitions>

</Grid>

</ItemsPanelTemplate>

</StatusBar.ItemsPanel>

<StatusBarItem Grid.Column="0">

<TextBlock Name="AppInfoText" Margin="5,0,0,0" />

</StatusBarItem>

<StatusBarItem Grid.Column="1" HorizontalContentAlignment="Center">

<TextBlock Text="© Ostiary 2022" />

</StatusBarItem>

<StatusBarItem Grid.Column="2" HorizontalContentAlignment="Right">

<TextBlock Name="TimerInfoText" TextAlignment="Right" />

</StatusBarItem>

</StatusBar>

</Grid>

</Page>

**LandingPage.xaml.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Reflection;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using ASPP.Pages.LandingPageElements;

using Path = System.IO.Path;

namespace ASPP.Pages

{

/// <summary>

/// Логика взаимодействия для LandingPage.xaml

/// </summary>

public partial class LandingPage : Page

{

public const string LandingFilePathHeader = "landingFilePath";

private static LandingPage \_instance;

public static LandingPage GetInstance() => \_instance ??= new LandingPage();

private LandingPage()

{

InitializeComponent();

}

}

public partial class LandingPage : Page

{

private void Info\_OnDrop(object sender, DragEventArgs e)

{

var landingTextBox = sender as TextBox;

Dispatcher.InvokeAsync(() =>

{

landingTextBox.Clear();

landingTextBox.Cursor = Cursors.Wait;

var importedPaths = (string[])e.Data.GetData(DataFormats.FileDrop);

var landingFilePath = importedPaths.FirstOrDefault(iP =>

iP.Contains("landing", StringComparison.InvariantCultureIgnoreCase) && iP.Contains(".csv"));

string landingWatermarkImageUriPath;

if (!landingFilePath.IsNullOrWhitespace())

{

landingTextBox.TextAlignment = TextAlignment.Center;

landingTextBox.Text = landingFilePath;

landingTextBox.BorderThickness = new Thickness(1);

landingWatermarkImageUriPath = "pack://application:,,,/Images/LandingImportedWatermarkImage.jpg";

AggregateLandingButton.IsEnabled = true;

}

else

{

landingTextBox.BorderThickness = new Thickness(0);

landingWatermarkImageUriPath = "pack://application:,,,/Images/LandingWatermarkImage.jpg";

MessageBox.Show(

"The imported file must contain the word 'landing' in the file name, and the file extension must be in '.csv' format",

"Import failed", MessageBoxButton.OK);

AggregateLandingButton.IsEnabled = false;

}

landingTextBox.Background = new ImageBrush()

{

AlignmentX = AlignmentX.Left,

ImageSource = new BitmapImage(new Uri(landingWatermarkImageUriPath)),

Stretch = Stretch.Fill

};

landingTextBox.Cursor = null;

Environment.SetEnvironmentVariable(LandingFilePathHeader, landingFilePath);

});

}

private void Info\_OnDragEnter(object sender, DragEventArgs e)

{

if (e.Data.GetDataPresent(DataFormats.FileDrop, false))

e.Effects = DragDropEffects.All;

}

private void Info\_OnDragOver(object sender, DragEventArgs e) => e.Handled = true;

private void AggregateLandingButton\_Click(object sender, RoutedEventArgs e)

{

if (LandingTextBox.Text.IsNullOrWhitespace())

return;

var landingFile = new FileInfo(Environment.GetEnvironmentVariable(LandingFilePathHeader));

if (MessageBox.Show("Please, format columns in file by that order - id, login, email, country, platform, than click OK",

"Warning", MessageBoxButton.OKCancel, MessageBoxImage.Warning) != MessageBoxResult.OK)

return;

Dispatcher.InvokeAsync(() =>

{

var startTime = DateTime.Now;

var lines = File.ReadAllLines(landingFile.FullName);

var headers = lines[0].Split(';');

if (!headers[0].Contains("id", StringComparison.InvariantCultureIgnoreCase) ||

!headers[1].Contains("login", StringComparison.InvariantCultureIgnoreCase) ||

!headers[2].Contains("email", StringComparison.InvariantCultureIgnoreCase) ||

!headers[3].Contains("country", StringComparison.InvariantCultureIgnoreCase) ||

!headers[4].Contains("platform", StringComparison.InvariantCultureIgnoreCase))

{

MessageBox.Show("Columns or column headers are in bad format. Please, check it again.", "Error", MessageBoxButton.OK,

MessageBoxImage.Error);

return;

}

var listOfUsers = lines[1..].Select(line => new User(line.Split(';'))).ToList();

var moderatedCount = 0;

landingFile.BackupTo(Path.Combine(landingFile.Directory.FullName, @$"Backup-{DateTime.Now:dd\_MM\_yyyy}"));

InitializeLandingDataGridColumns();

foreach (var user in listOfUsers)

{

foreach (var region in LandingConfig.RegionCountiesList)

foreach (var country in region.invalidItems)

if (string.Equals(user.Country, country, StringComparison.CurrentCultureIgnoreCase))

user.Country = region.validItem;

foreach (var emailDomain in LandingConfig.EmailDomainsList)

foreach (var invalidEmailDomain in emailDomain.invalidItems)

if (user.Email.Contains(invalidEmailDomain))

{

LandingDataGrid.Items.Add(user.Clone());

user.Email = user.Email.Replace(invalidEmailDomain, emailDomain.validItem);

LandingDataGrid.Items.Add(user);

moderatedCount++;

}

}

var moderatedUsersList = new List<string> { string.Join(';', headers) };

moderatedUsersList.AddRange(listOfUsers.Select(x => string.Join(';', x.ToAspects())).ToList());

File.WriteAllText(landingFile.FullName, String.Empty);

File.WriteAllLinesAsync(landingFile.FullName, moderatedUsersList).Wait();

AppInfoText.Text = $"Invalid mails changed - {moderatedCount}, Total - {moderatedUsersList.Count}";

TimerInfoText.Text = $"Landing file done in {DateTime.Now - startTime}";

});

}

public void InitializeLandingDataGridColumns()

{

foreach (var propertyName in typeof(User).GetProperties(BindingFlags.Public | BindingFlags.Instance).Select(p => p.Name).ToList())

LandingDataGrid.Columns.Add(new DataGridTextColumn() { Header = propertyName, Binding = new Binding(propertyName) });

}

}

}

**ParserPage.xaml**

<Page

x:Class="ASPP.Pages.ParserPage"

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"

Title="Parser"

d:DesignHeight="450"

d:DesignWidth="800"

Background="White"

mc:Ignorable="d">

<Grid>

<Grid.RowDefinitions>

<RowDefinition />

<RowDefinition />

<RowDefinition />

<RowDefinition />

<RowDefinition Height="1.7\*" />

<RowDefinition Height="0.3\*" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

</Grid.ColumnDefinitions>

<Grid

Grid.Row="0"

Grid.RowSpan="3"

Grid.Column="5"

Grid.ColumnSpan="3"

Margin="10">

<Grid.RowDefinitions>

<RowDefinition />

<RowDefinition />

</Grid.RowDefinitions>

<TextBox

Name="TypeformTextBox"

Grid.Row="0"

Margin="0,0,0,2.5"

AllowDrop="True"

BorderThickness="0"

Cursor="Arrow"

HorizontalScrollBarVisibility="Auto"

IsReadOnly="True"

PreviewDragEnter="TypeformTotal\_OnDragEnter"

PreviewDragOver="TypeformTotal\_OnDragOver"

PreviewDrop="Typeform\_OnDrop"

VerticalScrollBarVisibility="Auto">

<TextBox.Background>

<ImageBrush

AlignmentX="Left"

ImageSource="../Images/TypeformWatermarkImage.jpg"

Stretch="Fill" />

</TextBox.Background>

<TextBox.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="15" />

</Style>

</TextBox.Resources>

<UIElement.RenderTransform>

<MatrixTransform />

</UIElement.RenderTransform>

</TextBox>

<TextBox

Name="TotalTextBox"

Grid.Row="1"

Margin="0,2.5,0,0"

AllowDrop="True"

BorderThickness="0"

Cursor="Arrow"

HorizontalScrollBarVisibility="Auto"

IsReadOnly="True"

PreviewDragEnter="TypeformTotal\_OnDragEnter"

PreviewDragOver="TypeformTotal\_OnDragOver"

PreviewDrop="Total\_OnDrop"

VerticalScrollBarVisibility="Auto">

<TextBox.Background>

<ImageBrush

AlignmentX="Left"

ImageSource="../Images/TotalWatermarkImage.jpg"

Stretch="Fill" />

</TextBox.Background>

<TextBox.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="15" />

</Style>

</TextBox.Resources>

<UIElement.RenderTransform>

<MatrixTransform />

</UIElement.RenderTransform>

</TextBox>

</Grid>

<Grid

Name="ParsingGrid"

Grid.Row="0"

Grid.RowSpan="5"

Grid.Column="0"

Grid.ColumnSpan="5"

Margin="10">

<Grid.ColumnDefinitions>

<ColumnDefinition />

<ColumnDefinition />

</Grid.ColumnDefinitions>

<DataGrid

Name="BeforeParsingDataGrid"

Grid.Column="0"

Margin="0,0,2.5,0"

IsReadOnly="True" />

<DataGrid

Name="AfterParsingDataGrid"

Grid.Column="1"

Margin="2.5,0,0,0"

IsReadOnly="True" />

</Grid>

<Button

Name="ParserButton"

Grid.Row="3"

Grid.Column="5"

Grid.ColumnSpan="3"

Margin="10"

Click="ParserButton\_OnClick"

Content="Parse"

FontSize="20"

FontWeight="DemiBold"

IsEnabled="False">

<Button.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="10" />

</Style>

</Button.Resources>

</Button>

<StatusBar

Grid.Row="6"

Grid.Column="0"

Grid.ColumnSpan="8">

<StatusBar.ItemsPanel>

<ItemsPanelTemplate>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

</Grid.ColumnDefinitions>

</Grid>

</ItemsPanelTemplate>

</StatusBar.ItemsPanel>

<StatusBarItem Grid.Column="0">

<TextBlock Name="AppInfoText" Margin="5,0,0,0" />

</StatusBarItem>

<StatusBarItem Grid.Column="1" HorizontalContentAlignment="Center">

<TextBlock Text="© Ostiary 2022" />

</StatusBarItem>

<StatusBarItem Grid.Column="2" HorizontalContentAlignment="Right">

<TextBlock Name="TimerInfoText" TextAlignment="Right" />

</StatusBarItem>

</StatusBar>

</Grid>

</Page>

**ParserPage.xaml.cs**

using System;

using ASPP.Pages.ParserPageElements;

using OfficeOpenXml;

using System.Collections.Generic;

using System.Globalization;

using System.IO;

using System.Linq;

using System.Net;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using Color = System.Drawing.Color;

namespace ASPP.Pages

{

/// <summary>

/// Логика взаимодействия для ParserPage.xaml

/// </summary>

public partial class ParserPage : Page

{

public const string TypeformFilePathHeader = "typeformFilePath";

private static ParserPage \_instance;

public static ParserPage GetInstance() => \_instance ??= new ParserPage();

private ParserPage()

{

InitializeComponent();

}

}

public partial class ParserPage

{

private void Typeform\_OnDrop(object sender, DragEventArgs e)

{

var textBox = sender as TextBox;

Dispatcher.InvokeAsync(() =>

{

ImportFileFromTextBox(textBox, e, "typeform", ".xls", ".xlsx");

ParserButton.IsEnabled = !textBox.Text.IsNullOrWhitespace() && !TotalTextBox.Text.IsNullOrWhitespace();

});

}

private void Total\_OnDrop(object sender, DragEventArgs e)

{

var textBox = sender as TextBox;

Dispatcher.InvokeAsync(() =>

{

ImportFileFromTextBox(textBox, e, "total", ".xls", ".xlsx");

ParserButton.IsEnabled = !textBox.Text.IsNullOrWhitespace() && !TypeformTextBox.Text.IsNullOrWhitespace();

});

}

public void ImportFileFromTextBox(TextBox textBox, DragEventArgs e, string baseFileName, params string[] extensions)

{

textBox.Clear();

textBox.Cursor = Cursors.Wait;

var importedPaths = (string[])e.Data.GetData(DataFormats.FileDrop);

var filePath = importedPaths.FirstOrDefault(iP => iP.Contains(baseFileName, StringComparison.InvariantCultureIgnoreCase) && extensions.Any(iP.Contains));

string waterMarkImageUriPath;

if (!filePath.IsNullOrWhitespace())

{

textBox.TextAlignment = TextAlignment.Center;

textBox.Text = filePath;

textBox.BorderThickness = new Thickness(1);

waterMarkImageUriPath = $"pack://application:,,,/Images/{baseFileName}ImportedWatermarkImage.jpg";

Environment.SetEnvironmentVariable($"{baseFileName}FilePath", filePath);

}

else

{

textBox.BorderThickness = new Thickness(0);

waterMarkImageUriPath = $"pack://application:,,,/Images/{baseFileName}WatermarkImage.jpg";

MessageBox.Show(

$"The imported file must contain the word '{baseFileName}' in the file name, and the file extension must be in {string.Join(" / ", extensions)} format",

"Import failed", MessageBoxButton.OK);

}

textBox.Background = new ImageBrush()

{

AlignmentX = AlignmentX.Left,

ImageSource = new BitmapImage(new Uri(waterMarkImageUriPath)),

Stretch = Stretch.Fill

};

textBox.Cursor = null;

}

private void TypeformTotal\_OnDragEnter(object sender, DragEventArgs e)

{

if (e.Data.GetDataPresent(DataFormats.FileDrop, false))

e.Effects = DragDropEffects.All;

}

private void TypeformTotal\_OnDragOver(object sender, DragEventArgs e) => e.Handled = true;

private void ParserButton\_OnClick(object sender, RoutedEventArgs e)

{

Dispatcher.InvokeAsync(async () =>

{

var st = DateTime.Now;

if (TypeformTextBox.Text.IsNullOrWhitespace())

return;

ParsingGrid.Cursor = Cursors.Wait;

var typeformFile = new FileInfo(Environment.GetEnvironmentVariable("typeformFilePath")!);

var totalFile = new FileInfo(Environment.GetEnvironmentVariable("totalFilePath")!);

typeformFile.BackupTo(System.IO.Path.Combine(typeformFile.Directory.FullName, @$"Backup-{DateTime.Now:dd\_MM\_yyyy}"));

string[] typeformHeaders;

List<TypeformUser> typeformUsersList = new List<TypeformUser>();

using (var typeformSheet = new ExcelPackage(typeformFile).Workbook.Worksheets[0])

{

int startRow = typeformSheet.Dimension.Start.Row;

int startColumn = typeformSheet.Dimension.Start.Column;

typeformHeaders = typeformSheet.Cells[startRow, startColumn, startRow, startColumn + 2]

.Select(c => c.Value.ToString()).ToArray();

for (var i = startRow + 1; i <= typeformSheet.Dimension.Rows; i++)

{

typeformUsersList.Add(new TypeformUser(typeformSheet.Cells[i, startColumn].Value,

typeformSheet.Cells[i, startColumn + 1].Value,

typeformSheet.Cells[i, startColumn + 2].Value));

}

await BeforeParsingDataGrid.Dispatcher.InvokeAsync(() =>

{

BeforeParsingDataGrid.ItemsSource = typeformUsersList.ToList();

BeforeParsingDataGrid.Items.Refresh();

BeforeParsingDataGrid.UpdateLayout();

});

typeformUsersList.ForEach(u => u.Login = u.Login.Replace("@", "").Replace(" ", ""));

typeformUsersList.Reverse();

typeformUsersList = typeformUsersList.GroupBy(u => u.Email).Select(u => u.First()).GroupBy(u => u.Login).Select(u => u.First()).ToList();

typeformUsersList.Reverse();

using var totalSheet = new ExcelPackage(totalFile).Workbook.Worksheets[0];

var totalLoginList = new List<string>();

for (int i = totalSheet.Dimension.Start.Row; i <= totalSheet.Dimension.Rows; i++)

totalLoginList.Add(totalSheet.Cells[i, totalSheet.Dimension.Start.Column].Value.ToString());

typeformUsersList = typeformUsersList.Where(u => !totalLoginList.Contains(u.Login, StringComparer.InvariantCultureIgnoreCase)).ToList();

}

ParsingGrid.Cursor = null;

AfterParsingDataGrid.Cursor = Cursors.Wait;

List<LoginStatus> loginStatuses = new List<LoginStatus>();

loginStatuses.AddRange(typeformUsersList.Select(u => new LoginStatus(u.Login)));

await AfterParsingDataGrid.Dispatcher.InvokeAsync(() =>

{

AfterParsingDataGrid.ItemsSource = loginStatuses;

(AfterParsingDataGrid.Columns[1] as DataGridTextColumn)!.ElementStyle = new Style(typeof(TextBlock))

{

Setters =

{

new Setter()

{

Property = TextBlock.BackgroundProperty,

Value = Brushes.Orange

}

},

Triggers =

{

new Trigger()

{

Property = TextBlock.TextProperty,

Value = "OK",

Setters =

{ new Setter(TextBlock.BackgroundProperty, Brushes.Yellow) }

}

}

};

});

var loginStatusesTasks = loginStatuses.Select(async userLoginStatus =>

{

WebClient webClient = new WebClient { Proxy = null };

webClient.Headers.Add("user-agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.79 Safari/537.36");

try

{

string url = "https://ask.fm/" + userLoginStatus.Login;

string htmlPage =await webClient.DownloadStringTaskAsync(url); //Html page (code) of user`s page

Console.WriteLine($@"{userLoginStatus.Login} - Downloaded");

var isUserActive = true;

Match usernameRegexMatch = Regex.Match(htmlPage,

"(<span class=\"(.+|) dir=\"ltr\">@)(.+)(<\\/span>)",

RegexOptions.IgnoreCase); //Проверка на существование профиля

if (!usernameRegexMatch.Success) //If there is no field with username

throw new Exception("No such user");

Match uDateRegexMatch = Regex.Match(htmlPage, "(<time datetime=\")(.+)(<\\/time>)");

if (!isUserActive && !uDateRegexMatch.Success)

throw new Exception("No answers");

var uPositionRegex = "dir=\"ltr\">@";

var uPositionRegexMatch =

usernameRegexMatch.Value.IndexOf(uPositionRegex,

StringComparison.CurrentCultureIgnoreCase) + uPositionRegex.Length;

var parsedUsername = "";

while (usernameRegexMatch.Value[uPositionRegexMatch] != '<')

parsedUsername += usernameRegexMatch.Value[uPositionRegexMatch++];

userLoginStatus.Login = parsedUsername;

userLoginStatus.Status = "OK";

}

catch (Exception exception)

{

userLoginStatus.Status = exception.Message;

Console.WriteLine($@"{userLoginStatus.Login} - Failed");

}

webClient.Dispose();

AfterParsingDataGrid.Dispatcher.Invoke(() =>

{

AfterParsingDataGrid.Items.Refresh();

AfterParsingDataGrid.UpdateLayout();

});

Console.WriteLine($@"{userLoginStatus.Login} - Refreshed");

});

await Task.WhenAll(loginStatusesTasks);

AfterParsingDataGrid.Items.Refresh();

AfterParsingDataGrid.UpdateLayout();

//typeformUsersList.ForEach(u => u.Login = loginStatuses.First(ls => string.Compare(u.Login, ls.Login, CultureInfo.CurrentCulture, CompareOptions.IgnoreNonSpace) == 0).Login);

//Code above is screwed up, but code below this line is working fine, i don't know why.

foreach (var typeformUser in typeformUsersList)

{

foreach (var loginStatus in loginStatuses)

{

if (string.Compare(typeformUser.Login, loginStatus.Login, CultureInfo.CurrentCulture, CompareOptions.IgnoreNonSpace) == 0)

{

typeformUser.Login = loginStatus.Login;

break;

}

}

}

using (var typeformExcelFile = new ExcelPackage(typeformFile))

{

var typeformSheet = typeformExcelFile.Workbook.Worksheets[0];

typeformSheet.Cells.Clear();

for (int i = 0; i < typeformHeaders.Length; i++)

typeformSheet.Cells[1, i+1].Value = typeformHeaders[i];

int row = 2;

foreach (var user in typeformUsersList)

{

typeformSheet.Cells[row, 1].Value = user.Login;

typeformSheet.Cells[row, 1].Style.Fill.SetBackground(loginStatuses.First(ls => ls.Login.Equals(user.Login, StringComparison.InvariantCultureIgnoreCase)).Status == "OK"

? Color.Yellow : Color.Orange);

typeformSheet.Cells[row, 2].Value = user.Email;

typeformSheet.Cells[row, 3].Value = user.Geo;

row++;

}

await typeformExcelFile.SaveAsync();

}

TimerInfoText.Text = $"Typeform parsed in {DateTime.Now - st}";

AppInfoText.Text = $"Base total - {BeforeParsingDataGrid.Items.Count} -> final - {typeformUsersList.Count}, OK = {loginStatuses.Count(ls => ls.Status == "OK")}";

AfterParsingDataGrid.Cursor = null;

});

}

}

}

**FileInUse.cs**

using System.IO;

namespace ASPP.Pages.CleanerPageElements

{

public class FileInUse

{

private readonly FileInfo \_filePath;

public string FileName { get; set; }

public string Status { get; set; }

private string \_error;

public FileInUse(string filePath, string status = FileStatuses.InProgress)

{

\_filePath = new FileInfo(filePath);

FileName = \_filePath.Name;

Status = status;

}

public FileInfo GetFile() => \_filePath;

public void SetError(string error) => \_error = error;

public string GetError() => \_error;

}

}

**FileStatuses.cs**

using System.Collections;

namespace ASPP.Pages.CleanerPageElements

{

public struct FileStatuses

{

public const string Done = "Done";

public const string InProgress = "In Progress";

public const string Failed = "Failed";

public static ArrayList AllStatuses() => new() { Done, InProgress, Failed };

}

}

**LandingConfig.cs**

using System.Collections.Generic;

using Microsoft.Extensions.Configuration;

// ReSharper disable StringLiteralTypo

namespace ASPP.Pages.LandingPageElements

{

public static class LandingConfig

{

public static List<ValidInvalidItem> EmailDomainsList;

public static List<ValidInvalidItem> RegionCountiesList;

static LandingConfig()

{

var landingConfiguration = new ConfigurationBuilder().AddJsonFile("landingSettings.json").Build();

EmailDomainsList = new List<ValidInvalidItem>();

RegionCountiesList = new List<ValidInvalidItem>();

landingConfiguration.GetSection("Emails").Bind(EmailDomainsList);

landingConfiguration.GetSection("Regions").Bind(RegionCountiesList);}

}

public class ValidInvalidItem

{

public string validItem { get; set; }

public string[] invalidItems { get; set; }

}

}

**User.cs**

using System;

using System.Text;

namespace ASPP.Pages.LandingPageElements

{

class User : ICloneable

{

public string Id { get; set; }

public string UserLogin { get; set; }

public string Email { get; set; }

public string Country { get; set; }

public string Platform { get; set; }

public override string ToString()

=> new StringBuilder().AppendJoin('|', Id, UserLogin,Email,Country,Platform).ToString();

public string[] ToAspects()

=> new []{ Id, UserLogin, Email, Country, Platform };

public User(string[] rawUser) => (Id, UserLogin, Email, Country, Platform) = rawUser;

public object Clone() => new User(this.ToAspects());

}

}

**LoginStatus.cs**

namespace ASPP.Pages.ParserPageElements

{

public class LoginStatus

{

public string Login { get; set; }

public string Status { get; set; }

public override string ToString() => $"{Login,-30} | {Status}";

public LoginStatus(string login)

{

Login = login;

}

public LoginStatus(string login, string status) : this(login)

{

Status = status;

}

}

}

**TypeformUser.cs**

namespace ASPP.Pages.ParserPageElements

{

public class TypeformUser

{

public string Login { get; set; }

public string Email { get; set; }

public string Geo { get; set; }

public override string ToString() => $"{Login,-30} | {Email,-40} | {Geo}";

public string[] ToAspects() => new[] { Login, Email, Geo };

public TypeformUser(string[] rawUser) => (Login, Email, Geo) = rawUser;

public TypeformUser(object login, object email, object geo) => (Login, Email, Geo) = (login.ToString(), email.ToString(), geo.ToString());

public TypeformUser(string login, string email, string geo) => (Login, Email, Geo) = (login, email, geo);

}

}

**Extensions.cs**

using System;

using System.IO;

using System.Windows;

namespace ASPP.Pages

{

public static class Extensions

{

public static readonly string[] CountryList = { "ARAB", "DE", "ENG", "ID", "IT", "MX", "PL", "RU", "TH", "TR", "UA" };

public static bool IsNullOrWhitespace(this string s) => string.IsNullOrWhiteSpace(s);

public static FileInfo ToFileInfo(this string filePath)

=> new FileInfo(filePath);

public static bool TryCopyTo(string sourceFileName, string destFileName)

{

if (!File.Exists(sourceFileName))

throw new FileNotFoundException("Source file for copy wasn't found!");

try

{

File.Copy(sourceFileName, destFileName, true);

if (!File.Exists(destFileName))

throw new FileNotFoundException("File copy failed, destination file was not found");

return true;

}

catch (Exception e)

{

MessageBox.Show(e.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

return false;

}

}

public static bool BackupTo(this FileInfo file, string backupPathFolder)

{

//var backupPath = @$"{file.Directory!.FullName}\{Programsssss.BackupPathFolder}";

var backupPath = Path.Combine(file.Directory!.FullName, backupPathFolder);

try

{

if (!Directory.Exists(backupPath))

Directory.CreateDirectory(backupPath);

}

catch (Exception e)

{

MessageBox.Show(e.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

return false;

}

return TryCopyTo(file.FullName, Path.Combine(backupPath, file.Name));

}

public static void Deconstruct<T>(this T[] items, out T t0, out T t1, out T t2, out T t3, out T t4)

{

t0 = items.Length > 0 ? items[0] : default;

t1 = items.Length > 1 ? items[1] : default;

t2 = items.Length > 2 ? items[2] : default;

t3 = items.Length > 3 ? items[3] : default;

t4 = items.Length > 4 ? items[4] : default;

}

public static void Deconstruct<T>(this T[] items, out T t0, out T t1, out T t2)

{

t0 = items.Length > 0 ? items[0] : default;

t1 = items.Length > 1 ? items[1] : default;

t2 = items.Length > 2 ? items[2] : default;

}

}

}

**Credentials.cs**

using System;

namespace ASPP

{

public struct Credentials

{

public string Login { get; }

public string Email { get; }

public string Region { get; }

public Credentials(string login, string email, string country)

{

this.Login = login;

this.Email = email;

this.Region = country.Contains("AR",StringComparison.InvariantCultureIgnoreCase) ? "ARAB" : country;

}

public Credentials(string[] credentials) : this(credentials[0], credentials[1], credentials[2])

{}

}

}

**Helper.cs**

using System.IO;

namespace ASPP

{

public static class Helper

{

public static void NewFolder(string folderFullPath)

{

if (Directory.Exists(folderFullPath))

Directory.Delete(folderFullPath, true);

Directory.CreateDirectory(folderFullPath);

}

}

}

**landingSettings.json**

{

"Emails": [

{

"validItem": "@gmail.com",

"invalidItems": [ "@gamil.com", "@gmil.com", "@gimal.com", "@gmal.com", "@gmile.com", "@gmila.com", "@gmai.com", "@gmsil.com", "@gimel.com", "@gail.com", "@gmoil.com", "@gmill.com", "@gamail.com", "@gamil.com", "@gamail.com", "@gmqil.com" ]

},

{

"validItem": "@icloud.com",

"invalidItems": [ "@icoud.com", "@iclud.com", "@iloud.com" ]

},

{

"validItem": "@yahoo.com",

"invalidItems": [ "@yaho.com", "@ahoo.com", "@ayhoo.com", "@yahooo.com" ]

},

{

"validItem": "@outlook.com",

"invalidItems": [ "@outlok.com", "@oultook.com", "@ouylook.com" ]

},

{

"validItem": "@hotmail.com",

"invalidItems": [ "@homail.com", "@hotmil.com", "@htmail.com", "@hotnail.com" ]

}

],

"Regions": [

{

"validItem": "PBKIH",

"invalidItems": ["RU"]

},

{

"validItem": "RU",

"invalidItems": ["KZ", "MD", "AM"]

},

{

"validItem": "ARAB",

"invalidItems": ["AE", "EG", "IQ", "JO", "KW", "PS", "SA", "SY"]

},

{

"validItem": "DE",

"invalidItems": ["DE", "AT", "CH"]

},

{

"validItem":"MX",

"invalidItems": ["AR", "BO", "CL", "CO", "CR", "EC", "ES", "HN", "MX", "NI", "PE", "SV", "VE"]

},

{

"validItem": "ENG",

"invalidItems": ["AU", "BD", "BR", "CA", "CN", "FR", "GB", "IL", "IN", "MY", "NL", "PH", "PK", "RO", "SG", "US"]

}

]

}