Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра автоматизованих систем управління

**Курсова робота**

**з дисципліни "Об’єктно-орієнтоване програмування "**

**на тему**

**"Програмне забезпечення для організації картотеки Інтерполу"**

**Виконав:**

студент гр. ОІ-26

Лабунський Я. А.

**Керівник:**

Асистент кафедри АСУ

Івасів С. С.

Львів – 2024

Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра автоматизованих систем управління

**Завдання до курсової роботи**

з дисципліни **“Об’єктно-орієнтоване програмування”**

**Прізвище, ім’я студента** Лабунський Ярослав

**Група** ОІ-26

**Тема курсової роботи** Програмне забезпечення для

організації картотеки Інтерполу

**Спеціальна частина завдання:**

1. В огляді літератури проаналізувати відкриті джерела з питань керування картотеками, зокрема офіційного вебсайту.
2. Дослідити застосування структур даних та алгоритмів сортування для побудови логіки взаємодії даних з бази у середовищах програмування.
3. Розробити та реалізувати:

3.1) зручний, ефективний, інтуітивно зрозумілий користувацький інтерфейс програми на мові Python з використанням бібліотеки "PyQt".

3.2) клас “Інтерпол” для отримання інформації про об'єкти з бази даних, та реалізація основних методів: додання, пошук та сортування об’єктів за параметрами. Реалізація методів буде реалізована но доречних мовах. Основні структура і сортування на С++.

3.3) Для сховища даних буде використано SQL server management studio

3.4) методи для пошуку даних об’єкти класа.

3.5) організацію файлів програми.

1. Для програмних рішень використати мови C++ та Python.
2. Термін завершення роботи – 10 грудя 2024 р.

Завдання видано 09 вересня 2024 р.

Керівник асистент каф. АСУ Івасів С. С.

**ЗМІСТ**

[ВСТУП 3](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809074)

[1 ПОСТАНОВКА ЗАДАЧІ 4](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809075)

[2 ОГЛЯД ТА АНАЛІЗ ЛІТЕРАТУРНИХ ДЖЕРЕЛ **Ошибка! Закладка не определена.**](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809076)

[2.1 Огляд технологій і засобів для програмної реалізації……………..**Ошибка! Закладка не определена.**](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809077)

[2.2Аналіз та вибір алгоритмів **Ошибка! Закладка не определена.**](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809078)

[3 ОПИС СТРУКТУРИ ПРОГРАМНОГО РІШЕННЯ, МЕТОДІВ, АЛГОРИТМІВ І ДАНИХ 7](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809081)

[3.1 Діаграма USE-CASE 7](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809082)

[3.2 Опис структури програми 8](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809083)

[3.3 Опис алгоритмів 9](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809084)

[4 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЄКТУ **Ошибка! Закладка не определена.**](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809085)

[4.1 Діаграма компонентів **Ошибка! Закладка не определена.**](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809086)

[4.2 Опис структури програмного проєкту **Ошибка! Закладка не определена.**](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809087)

[5 ОПИС ПРОВОДЕНИХ ЕКСПЕРИМЕНТІВ 17](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809088)

[5.1 Вимоги до запуску програми 17](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809089)

[5.2 Тестові сценарії: 18](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809090)

5.3 Вказівки для встановлення п 19

[ВИСНОВОК 19](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809091)

[ЛІТЕРАТУРНІ ДЖЕРЕЛА 20](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809092)

[ДОДАТКИ 22](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809093)

[Додаток 1 22](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809094)

[Додаток 2 **Ошибка! Закладка не определена.**](file:///C:\Users\Admin\Downloads\2.%20КР%20(Д.%20М.).docx#_Toc184809095)

**ВСТУП**

У сучасному світі інформаційні технології стали невід'ємною частиною багатьох сфер людської діяльності, включаючи боротьбу зі злочинністю та забезпечення безпеки. Однією з найважливіших організацій, що займаються координацією міжнародних зусиль у боротьбі з транснаціональною злочинністю, є Інтерпол. Для ефективного виконання завдань, таких як обробка даних про розшукуваних осіб, потрібні сучасні програмні рішення.

Метою даної курсової роботи є розробка прототипу програмної системи-картотеки для Інтерполу з використанням мов програмування Python та C++. Ця система дозволяє організувати інформацію про розшукуваних осіб у структурованому вигляді, реалізувати функції пошуку, сортування та управління даними.

У ході роботи буде розглянуто сучасні підходи до розробки програмного забезпечення, методи взаємодії між компонентами системи, а також особливості роботи з базами даних для зберігання інформації. Проєкт має на меті створити інструмент, що сприятиме покращенню ефективності управління інформацією в діяльності правоохоронних органів.

Таким чином, актуальність роботи обумовлена потребою у створенні програмного забезпечення, здатного обробляти значні обсяги даних, забезпечуючи при цьому зручність використання та високу швидкодію

# **ПОСТАНОВКА ЗАДАЧІ**

Мета розробки програмного продукту полягає у створені програмної версії сайта “Інтерпол” відтворюючи аспект пошуку міжнародних злочинців, засвоєння навичок з об'єктно-орієнтованого програмування, алгоритмів та структур даних, а також правильного налагодження програми.

Результатом роботи програми є зручний, інтуітивно зрозумілий інтерфейс для користувача, швидкий пошук об’єктів бази даних за параметрами. Для цього можна використати такі параметри вхідних дані: ім’я, фамілія, національність, стать, вік, країна, яка розшукує, та ключево слово.

* 1. Функціональні можливості даного програмного рішення:
     1. Пошук за параметрами
     2. Сортування результата вибріки за ім’ям або віком
     3. Отримання повної існуючої інформації об'єктів
  2. Функціональні обмеження даного програмного рішення :
     1. Фіковасний розмір головного вікна
     2. На сторінці максимум 8 об'єктів
  3. Встановлені умови з боку користувача:
     1. Зручний інтерфейс
     2. Консистентність даних об'єктів
     3. Швидкодія програми навіть з великою кількість данах
  4. Встановлені вимоги до програми з боку апаратних та програмних платформ:
     1. Платформа Windows із підтримкою Python версії 3+ та C++ версії 11+
     2. Необхідна реляційна база даних SQL server management studio для роботи з об’єктами
     3. Додатково потрібно встановити бібліотеки pybind11 та pyodbc
     4. Місце на жорсткому диску 500мб.

# **ОГЛЯД ТА АНАЛІЗ ЛІТЕРАТУРНИХ ДЖЕРЕЛ**

* 1. **Огляд технологій і засобів для програмної реалізації**

Для ефективної роботи з об’єктами з бази даних необхідна швидкість виконання алгоритмів. Тому для реалізації структури даних та алгоритму сортування кращим вибором стане С++ за певним критеріями:

* Висока швидкість компіляції
* Ефективне управління ресурсами
* Ефективність при низькорівневих операціях
* Підтримка складних структур даних

Для для зручного введення, обробки та виведення інформації про об’єкти, сформованої алгоритмами необхідно створити інтерфейс. Для його реалізації краще підійде Python через:

* Легкість і швидкість розробки
* Потужні бібліотеки для роботи з інтерфейсами
* Легка інтеграція з базами даних
* Інтерактивність

Для під’єднання створених бібліотек на С++ у Python буде використано модуль pybind11

* 1. **Аналіз та вибір алгоритмів**

Для занесення об’єктів із бази даних у програму необхідно знайти найкращу структуру даних під потребу програми. Такі структури, як масиви, списки, хеш-таблиці, стеки та черги, зразу відкидаємо, оскільки вони не забезпечують необхідної ефективності для пошуку, вставки чи видалення даних при великому обсязі або складній структурі зв'язків між об'єктами.

Натомість, оптимальними є дерева зі складністю операцій O(log n) (наприклад, AVL чи червоне-чорне дерево), оскільки вони дозволяють ефективно працювати з впорядкованими даними та забезпечують швидкий доступ до елементів.

Так як ключем вставки буде айді об'єкта, то просте двійкове дерево не підійде, тому що воно буде “рости” в один бік, так як айді починається з 1 і буде постійно збільшуватись, то двійкове дерево перетвориться у зв'язковий список зі складністю операцій O(n). Тому потрібно вибрати якесь з самобалансуючих дерев, а саме червоно-чорне або AVL:

* AVL-дерево: Забезпечує більш строгий баланс, що гарантує оптимальнішу продуктивність для операцій пошуку. Але це також означає, що для вставки та видалення може знадобитися більше обертів для балансування.
* Червоного-чорне дерево: Операції вставки та видалення зазвичай працюють швидше, оскільки балансування менш жорстке. Проте через це в деяких випадках висота дерева може бути більша, що може вплинути на швидкість пошуку.

Так як виконуваитись будуть тільки операції вставки та пошуку, що важливо, то краще обрати AVL-дерево.

Щодо алгоритмів сортування простіше і швидше буде використати стандартне сортування для мови Python **- Timsort** (гібрид MergeSort і InsertionSort), але в програмному рішені буде реалізовано ідею алгоритму гібридного сортування QuickSort та InsertionSort з кількома оптимізаціями:

* Перевірка чи масив уже відсортований за зростанням чи спаданням (це важливо, так як це буде часто використовуватись саме в програмі)
* Оптимізація вибору опорного елемента (перший, середній, останній)
* Замість рекурсії використано стек
* Для малих підмасивів використовується сортування вставками (поріг 10 елементів)

# **ОПИС СТРУКТУРИ ПРОГРАМНОГО РІШЕННЯ, МЕТОДІВ, АЛГОРИТМІВ І ДАНИХ**

* 1. **Діаграма USE-CASE**

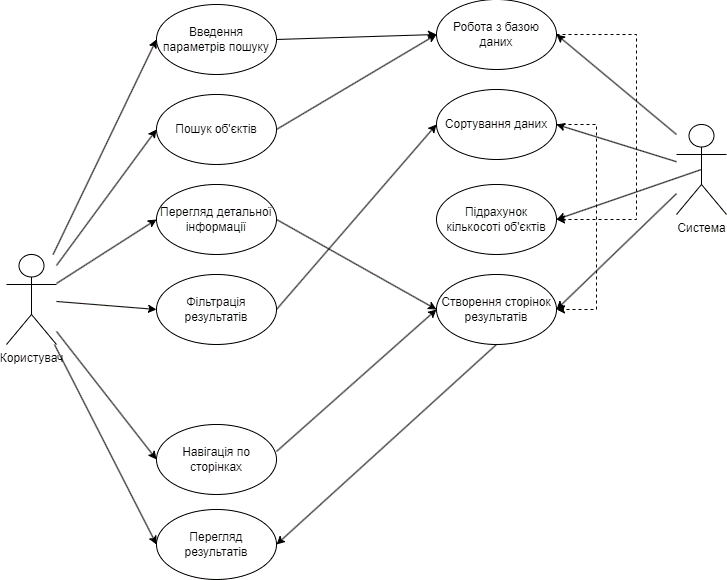


Рис. 3.1. Діаграма USE-CASE

* 1. **Опис програмних компонентів системи, діаграма класів**

Клас діаграма яка включає в себе основні класи програми рис. 3.2 додаток А

**Класи:**

Ui\_MainWindow– відповідає за головне вікно інтерфейсу користувача. На ньому відображено: інформацію для введення параметрів, самі вікна для введення, загальна в базі даних та для вибірки кількість об’єктів, сторінки для розміщення даних злочинців, кнпоки пошуку, руху по сторінкам, сортування.

Ui\_PersonInfoWindow – відповідає за додаткове вікно інтерфейсу користувача. На ньому відображено детально відому інформацію про об’єкт, та додаткові фото, якщо такі інсують.

SearchButton – передача отриманих параметрів пошуку із інтерфейса до класу WantedPerosonSearcher.

PageNavigationButtons– перемикання і оновлення стану сторінок відповідного віджета.

PersonSortButtons – сортування об’єктів вибірки за іменем, прізвищем та віком.

InterpolDB – підключення програми до бази даних SQL server management studio (Включає у себе WantedPerosonSearcherіCountOfWantedPersons)

WantedPerosonSearcher – виклик процедури та отримання її результату у вигляді об’єктів та додання його у структуру даних (AVL дерево)

CountOfWantedPersons – виклик підрахунку бази даних загальної кількості злочинців.

**Файли:**

GlobalValues – файл для збереження заповненого дерева та загальної кількості злочинців.

**Переліки**(**enum**):

Countries **–** зберігає 193 країн для комбобоксів при виборі національності та країни, яка подала на розшук злочинців.

* 1. **Опис і схеми використаних алгоритмів загальної роботи програми**

Схема ідеї побудови програми наведена на рис 3.3 додатка А

**Алгоритм підрахунку загальної кількості об’єктів у базі даних:**

1. Користувач відкриває програму.
2. У базу даних відправляється запит на прорахунок
3. Повертається кількість, вона заноситься у глобальну змінну та виводиться у інтерфейс

**Алгоритм введення, отримання та зберігання даних:**

1. Користувач вводить у спеціальні поля параметри, за якими потрібно знайти злочинців, або не вводить нічого для отримання одразу всіх об’єктів.
2. Параметри відправляються у підготовлену процедуру в базі даних, яка повертає дані про об’єкти.
3. Функція форматування приводить ці дані у необхідний вигляд.
   1. Якщо типом є рядок, і він не None (NULL з бази даних у python перетворюється у None), то він не змінюється, інакше – порожній рядок ‘’.
   2. Якщо це число цілого типу, і воно не None, то його залишаємо, інакше – -1 (таке значення вибрано, бо воно зазвичай означає помилку, у нашому випадку це тимчасове значення для подальшого опрацювання).
   3. Якщо ж це дата, то її також треба привести у правильну форму, а саме у вигляд день/місяць/рік (в SQL інакше форматування, яке тут не так доречно), якщо ж None, то так само у порожній рядок.
   4. Ну і перевірка чи фото існує і папці, яка їх зберігає, вона отримує лист з назв фото (у базі вигляд surname\_name1.png, surname\_name2.png), додає до них шлях і перевіряє чи існує такий файл за посиланням. В результаті неіснуючі або неправильні пропускаються, а інші додаються у такому ж форматі рядка через кому.
4. Відформатовані дані заносяться у збалансоване дерево в глобальну змінну, ключем якого є айді об’єктів.

**Алгоритм виведення на екран:**

1. Перевірка чи кнопка пошуку була нажата.
2. Оновлення інформації про кількість об’єктів у вибірці:
   1. Якщо ця кількість не нуль, то оновлюємо статус
   2. Якщо ж нуль, то повідомляємо що об’єктів за заданим параметрами не знайдено
3. Очищення, якщо попередньо уже був створений віджет, для уникнення дублювання.
4. Дістаємо дерево з глобальної змінної і сортуємо його.
   1. Якщо це перше зчитування даних, то воно зажди буде невідсортоване, але при подальших діях все буде працювати коректно.
   2. Натиснувши на одну з кнопок сортування повернеться відсортований список з інформацією про об’єкти.
5. Вираховуємо необхідну кількість сторінок для виведення усіх об’єктів.
   1. Для цього використаємо формулу 3.1 додаток А округлення числа вгору до найбільшого цілого.
6. Створимо словник для передачі даних злочинців та формування об’єктів сторінки, які включають у себе: фото, фамілія, ім’я, вік, національність, айді. Останній не виводиться у інтерфейс, але важливий для подальшої роботи програми.
7. Для розташування об’єкта у сітці сторінці буде використано формули 3.2 і 3.3 додаток А, де у нашому випадку n = 4.
8. Викликаємо методи для розміщення інформації на сторінку з алгоритмом:
   1. Фото, якщо не існує, то заміняємо його на спеціальне з відображенням “photo not available”, якщо ж є кілька або одне, то вибираємо перше, як головне.
   2. Далі йдуть ім’я та прізвище, які йдуть як гіперпосилання, натиснувши які відкривається нове вікно, в якому відображено усю існуючу додаткову інформацію про злочинця (тут якраз і знадобиться айді, яке ми передавали об’єктам сторінок, тому що за ним буде пошук у дереві).
   3. Ну і останніми будуть вік і національність, і якщо він існує, то до нього додається рядок “years old”.
   4. Якщо ж певного або певних параметрів не існує, то текст підтягується по висоті для гарного і зручного вигляду.
   5. Також для оформлення додано лінії, які розмежовують рядки сторінок. Вони будуються для індексів у межах від 4 до 7.
9. Так продовжується заповнення сторінок, поки не закінчаться об’єкти у словнику.

**Алгоритм відкриття нових вікон з детальною інформацією:**

1. Спочатку відбувається пошук у дереві за айді об’єкта, на який веде гіперпосилання.
2. Далі створюється вікно з інформацією, яка включає у себе:
   1. Заголовок, у якому є повне ім’я та прізвище злочинця (якщо не існує то замінюється на “Unknown”) та разом з надписом “wanted by” країна, яка розшукує.
   2. Зліва під заголовком будуть розміщенні фотографії у обмеженні 3 штук.
   3. Справа від фотографій буде назва відділу інформації, назва параметру та його значення.
   4. Якщо ж не існує значення, тобто немає даних, то рядок пропускається. Таке ж правило працює і для рядків відділів.
3. Таких вікон можна відкрити безліч, що зручно.

**Алгоритм дерева:**

1. Структури даних
   1. Person: Структура для зберігання персональних даних про людину. Включає такі поля, як ідентифікатор (id), прізвище, ім'я, стать, дата народження, вік, місце народження, національність, характеристики, фото, мови та інші параметри.
   2. TreeNode: Структура для вузлів AVL-дерева, яка зберігає об'єкт типу Person, а також лівий і правий дочірні вузли (для AVL-дерева) та висоту вузла.
2. Алгоритм AVL-дерева
   1. PersonTree: Клас для реалізації AVL-дерева, який містить корінь дерева та методи для додавання осіб, пошуку особи за ідентифікатором (ID), виведення всіх осіб в порядку зростання їх ID, а також для балансування дерева після кожної операції вставки.
   2. Вставка: При додаванні нового елемента в дерево, проводиться перевірка на баланс дерева. Якщо баланс дерева порушується (різниця висот лівого та правого піддерева більше ніж 1), виконується обертання вузлів (ліве або праве обертання) для відновлення балансу
   3. Пошук за ID: Метод findById використовує рекурсивний пошук для знаходження особи з заданим ідентифікатором. Якщо особу не знайдено, кидається виняток.
   4. Отримання всіх осіб: Метод getAllPersons рекурсивно збирає всіх осіб дерева і повертає їх у вигляді вектора.

**Алгоритм сортування:**

1. Перевірка напрямку сортування:
   1. Функція check\_sort\_direction перевіряє, чи вже відсортований масив. Вона повертає true, якщо масив відсортований в порядку зростання або спадання, та вказує на напрямок сортування.
2. Розділення масиву:
   1. Функція partition вибирає опорний елемент для сортування (за допомогою методу "медіана трьох") і розбиває масив на дві частини, у залежності від напрямку сортування: зліва елементи менші (для зростання) або більші (для спадання) за опорний елемент, справа — більші або менші відповідно.
3. Алгоритм QuickSort:
   1. Функція quicksort реалізує ітеративну версію алгоритму QuickSort, що використовує стек для обробки підмасивів замість рекурсії. Для малих масивів застосовується InsertionSort (з порогом 10 елементів), щоб уникнути накладних витрат на рекурсивні виклики.
   2. Окрім того, алгоритм оптимізований для пам'яті, оскільки стек для зберігання діапазонів масивів виділяється заздалегідь.
4. Перевірка вже відсортованих масивів:
   1. Якщо масив вже відсортований, і його напрямок не відповідає бажаному, відбувається просто перевертання масиву.

# **ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЄКТУ**

* 1. **Загальна стркутура програмного проєкту**

На рис. 4.1 у додатку А зображена стркутура програмного проєкту у PyCharm. У папці venv окрім основних файлів є додаткові папки. Cpp, для коду на мові C++ та setups, для налаштувань компіляції й підключення їх до пайтона. Також є папки interpol\_logos, в якій є фото для встановлення іконок програми, та InterpolDB, де зберігаються файли бази даних.

Для створення інтерфейсу буде використано PyQt, ось його важливі переваги над tkinter:

* Більша функціональність
* Інструменти розробки (Qt Designer)
* Кросплатформеність
* Широкі можливості роботи з даними
* Підтримка багатовіконних додатків
* Підтримка складної логіки

Щодо взаємодії з базою даних було обрано pyodbc. Хоча це не найкращий вибір з існуючих, але він добре оптимізований під моє середовище, а саме SQL server management studio, і для такого проєкту цього буде достатньо.

* 1. **Опис структури бази даних програмного проєкту**

Додатково опишемо створення бази даних. Так як об’єктів може бути 5 000 (на офіційному сайті 6668), то кращим вибором буде використати ненормалізовану базу даних по таким причинам:

* Швидкість читання
* Швидкість розробки
* Простота запису та обслуговування
* Легкість внесення змін
* Гнучкість у зберіганні даних
* Менше ресурсів на з’єднання

Щодо створення таблиці. Для айді візьмемо стандартний розмір int, для прізвища, ім’я, міста та країни народження, національності, країни, яка розшукує вистачить 100 байтів. Для кольорів очей та волосся вистачить 50 байтів. Для статі – 10, звинувачення – 512, мови – 255, фото – 127 байт. Щодо роста і ваги можна взяти tiny int, розмір якого 1 беззнаковий байт. Ну і для дати народження спеціальний тип date.

CREATE TABLE WantedPersons (

id INT IDENTITY(1,1) PRIMARY KEY,

family\_name NVARCHAR(100) DEFAULT NULL,

forename NVARCHAR(100) DEFAULT NULL,

gender NVARCHAR(10) DEFAULT NULL,

date\_of\_birth DATE DEFAULT NULL,

place\_of\_birth\_city NVARCHAR(100) DEFAULT NULL,

place\_of\_birth\_country NVARCHAR(100) DEFAULT NULL,

nationality NVARCHAR(100) DEFAULT NULL,

distinguishing\_marks NVARCHAR(255) DEFAULT NULL,

wanted\_by NVARCHAR(100) DEFAULT NULL,

height TINYINT DEFAULT NULL,

[weight] TINYINT DEFAULT NULL,

hair\_color NVARCHAR(50) DEFAULT NULL,

eye\_color NVARCHAR(50) DEFAULT NULL,

charges NVARCHAR(512) DEFAULT NULL,

languages NVARCHAR(255) DEFAULT NULL,

photo NVARCHAR(127) DEFAULT NULL

);

Далі потрібно створити процедуру, яка буде шукати об’єкти за параметрами. Ідея якої що об’єкт може мати або NULL або порожній рядок, або підбирається через like % парам %. Також є ключеве слово, яке шукає одразу за всіма параметрами. Отже, у вибірку коректно додаються необхідні об’єкти.

CREATE PROCEDURE GetWantedPersons

@family\_name NVARCHAR(100) = NULL,

@forename NVARCHAR(100) = NULL,

@nationality NVARCHAR(100) = NULL,

@gender NVARCHAR(10) = NULL,

@min\_age TINYINT = NULL,

@max\_age TINYINT = NULL,

@wanted\_by NVARCHAR(50) = NULL,

@keyword NVARCHAR(255) = NULL

AS

BEGIN

SELECT wp.id,

wp.family\_name,

wp.forename,

wp.gender,

wp.date\_of\_birth,

DATEDIFF(YEAR, wp.date\_of\_birth, GETDATE()) AS current\_age,

wp.place\_of\_birth\_city,

wp.place\_of\_birth\_country,

wp.nationality,

wp.distinguishing\_marks,

wp.wanted\_by,

wp.height,

wp.[weight],

wp.hair\_color,

wp.eye\_color,

wp.charges,

wp.photo,

wp.languages

FROM WantedPersons wp

WHERE

(@family\_name IS NULL OR wp.family\_name LIKE '%' + @family\_name + '%' OR @family\_name = '')

AND (@forename IS NULL OR wp.forename LIKE '%' + @forename + '%' OR @forename = '')

AND (@nationality IS NULL OR wp.nationality LIKE '%' + @nationality + '%' OR @nationality = '')

AND (

@gender IS NULL OR @gender = '' OR wp.gender = @gender OR (@gender = 'unknown' AND wp.gender IS NULL)

)

AND (@min\_age IS NULL OR wp.date\_of\_birth IS NULL OR DATEDIFF(YEAR, wp.date\_of\_birth, GETDATE()) >= @min\_age)

AND (@max\_age IS NULL OR wp.date\_of\_birth IS NULL OR DATEDIFF(YEAR, wp.date\_of\_birth, GETDATE()) <= @max\_age)

AND (@wanted\_by IS NULL OR wp.wanted\_by LIKE '%' + @wanted\_by + '%' OR @wanted\_by = '')

AND (

@keyword IS NULL OR

wp.family\_name LIKE '%' + @keyword + '%'

OR wp.forename LIKE '%' + @keyword + '%'

OR wp.nationality LIKE '%' + @keyword + '%'

OR wp.distinguishing\_marks LIKE '%' + @keyword + '%'

OR wp.gender LIKE '%' + @keyword + '%'

OR wp.charges LIKE '%' + @keyword + '%'

OR wp.hair\_color LIKE '%' + @keyword + '%'

OR wp.eye\_color LIKE '%' + @keyword + '%'

OR wp.height = @keyword

OR wp.weight] = @keyword

OR wp.place\_of\_birth\_city LIKE '%' + @keyword + '%'

OR wp.place\_of\_birth\_country LIKE '%' + @keyword + '%'

OR wp.wanted\_by LIKE '%' + @keyword + '%'

)

END;

GO

Також потрібно надати облікову запису користувача доступ до використання цих таблиці та процедури.

USE InterpolDB;

GRANT SELECT ON dbo.WantedPersons TO [user];

GRANT EXECUTE ON dbo.GetWantedPersons TO [user];

# **ОПИС ПРОВЕДЕНИХ ЕКСПЕРИМЕНТІВ**

* 1. **Вимоги до запуску програми**

**Адміністратору:**

1. Переконатись що встановлено Python версії 3.7 або новішої, C++ версії 11 або новішої.
2. Переконатись що запущений сервер бази даних.
3. Переконатись у встановленні актуальних версій всіх необхідних бібліотек. До них відносяться і власноскомпільовані файли cpp.

**Користувачеві:**

1. Правильно завантажити програму.
2. Запустити програму через термінал або інтегроване середовище (IDE)
   1. **Тестові сценарії та показ роботи програми**

Тест 1: нічого не вводити у параметри пошуку.

Результат: рис. 5.1 додаток А виведуться усі існуючі об’єкти. Також видно роботу форматування інформації, якщо не існує певного параметра, то він упускається і підтягується інший.

Тест 2: Введення одного неповного параметра.

Результат: рис. 5.2 додаток А вибірка з об’єктів, які мають у прізвищі “GAR”. Також видно роботу ліній розділення.

Тест 3: пошук за багатьма параметрами.

Результат: рис. 5.3 додаток А все працює коректно.

Тест 4: перевірка роботи додаткового вікна.

Результат: рис. 5.4 додаток А коректне відображення об’єкта у додатковому вікні.

Тест 5: перевірка ключового слова.

Результат: Додаток А рис. 5.5, вибірка з 5 людей з ростом 177, для приклада показано тільки 2.

Тест 6: перевірка відображення фото.

Результат: Додаток А рис. 5.6, рис. 5.7, рис. 5.8, рис. 5.9. На рис. 5.7 відображено 2 фото, у базі даних спеціально вказано одного одного фото неправильно, тому замість 3 відображається тільки 2.

Тест 7: Кнопки перегортання сторінок та сосртування.

Результат: Додаток А рис 5.10, рис 5.11, рис 5.12, рис 5.13, коректна робота усіх кнопок.

Тест 8: Форматування інформації при відсутності певних даних.

Результат: Додаток А рис. 5.15. З відсутністю “Charges” та кольорів волосся та очей табло зберігає приємний підтянутий формат.

* 1. **Вказівки для встановлення програми**

Розташування проєкта у студента: C:\Users\PC\PycharmProjects\INTERPOL

Розташування бібліотек типу WHL та WinRAR: C:\Users\PC\dist

Ручне встановлення та компіляція бібліотек:

Для налаштування бібілотеки

pip install setuptools wheel

Бібліотека з деревом

python C:\Users\PC\PycharmProjects\INTERPOL\setup\_tree.py sdist bdist\_wheel

pip install dist\person\_tree-1.0.2-cp311-cp311-win\_amd64.whl

Бібліотека з сортуванням

python C:\Users\PC\PycharmProjects\INTERPOL\setup\_sort.py sdist bdist\_wheel

pip install C:\Users\PC\dist\quicksort-1.0.0-cp311-cp311-win\_amd64.whl

.mdf та .ldf файли і фото розташовані у папці interpolDB

# **ВИСНОВКИ**

У рамках виконання курсової роботи я розробив програму для обробки та пошуку інформації про злочинців в базі даних. Програма використовує сучасні принципи об’єктно-орієнтованого програмування та інтерфейси для ефективного управління даними. Основною метою стало створення системи, яка дозволяє знаходити злочинців за різними параметрами, такими як ім'я, вік, стать, національність, тощо.

Програма працює на основі реляційної бази даних, в якій зберігається детальна інформація про осіб, яких розшукує міжнародна поліція (INTERPOL). Для цього була створена структура даних, що включає особисті дані, фотографії, особливі прикмети та інші важливі атрибути, що допомагають ідентифікувати особу.

Для зручності пошуку реалізовано інтерфейс, який дозволяє користувачеві здійснювати запити за різними параметрами. Це дозволяє не лише знаходити конкретних осіб, а й отримувати більш детальну інформацію для подальших дій.

Програма підтримує різні механізми пошуку, зокрема за числовими та текстовими параметрами, а також забезпечує перевірку наявності необхідної інформації у базі даних.

Робота також включає обробку виняткових ситуацій, що дозволяє запобігати помилкам при виконанні запитів і надає користувачеві корисні повідомлення про помилки чи відсутність результатів.

Це рішення сприяє оптимізації роботи правоохоронних органів та забезпечує швидкий доступ до важливої інформації при пошуку розшукуваних осіб у міжнародних базах даних.

# **ЛІТЕРАТУРНІ ДЖЕРЕЛА**

1. Офіційний сайт INTERPOL. [Електронний ресурс]. – Режим доступу: <https://www.interpol.int>.
2. Qt for Python Documentation. [Електронний ресурс]. – Режим доступу: <https://doc.qt.io>.
3. Qt Designer Documentation. [Електронний ресурс]. – Режим доступу: <https://doc.qt.io/qtforpython-6.7/overviews/qtdesigner-manual.html>.
4. Pyodbc Documentation. [Електронний ресурс]. – Режим доступу: <https://github.com/mkleehammer/pyodbc/wiki>.
5. Pybind11 Documentation. [Електронний ресурс]. – Режим доступу: <https://pybind11.readthedocs.io/en/stable/>.
6. AVL Tree. [Електронний ресурс]. – Режим доступу: <https://en.wikipedia.org/wiki/AVL_tree>.
7. Quick Sort. [Електронний ресурс]. – Режим доступу: <https://en.wikipedia.org/wiki/Quicksort>.
8. Insertion Sort. [Електронний ресурс]. – Режим доступу: <https://en.wikipedia.org/wiki/Insertion_sort>.

# **ДОДАТКИ**

**Додаток А**

**Формули, фото та рисунки**

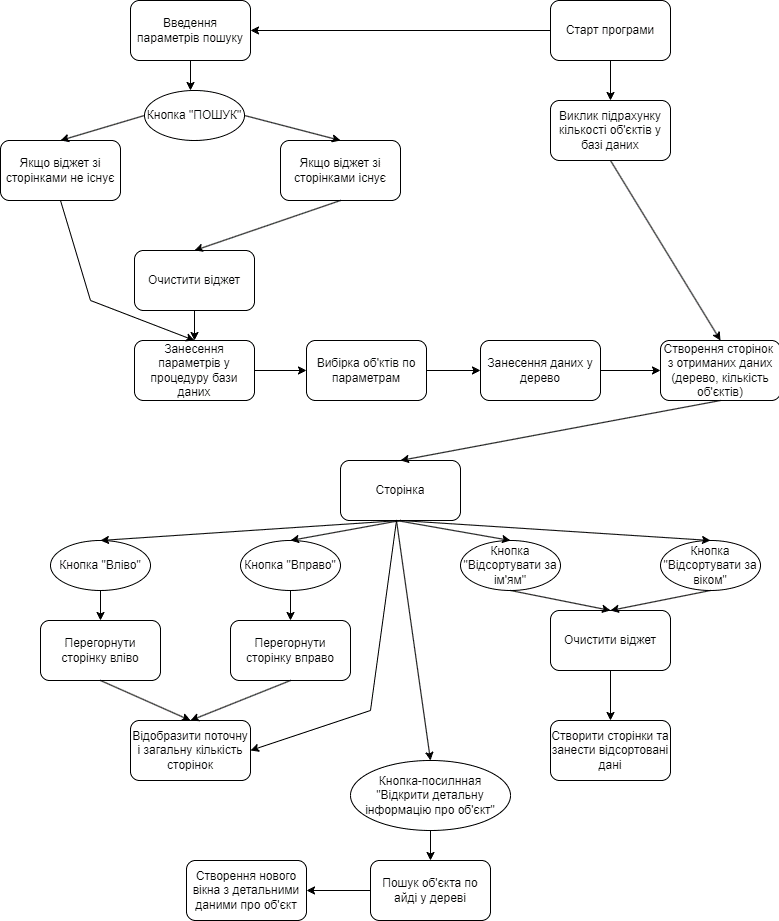


Рис 3.1. Схема ідеї побудови програми

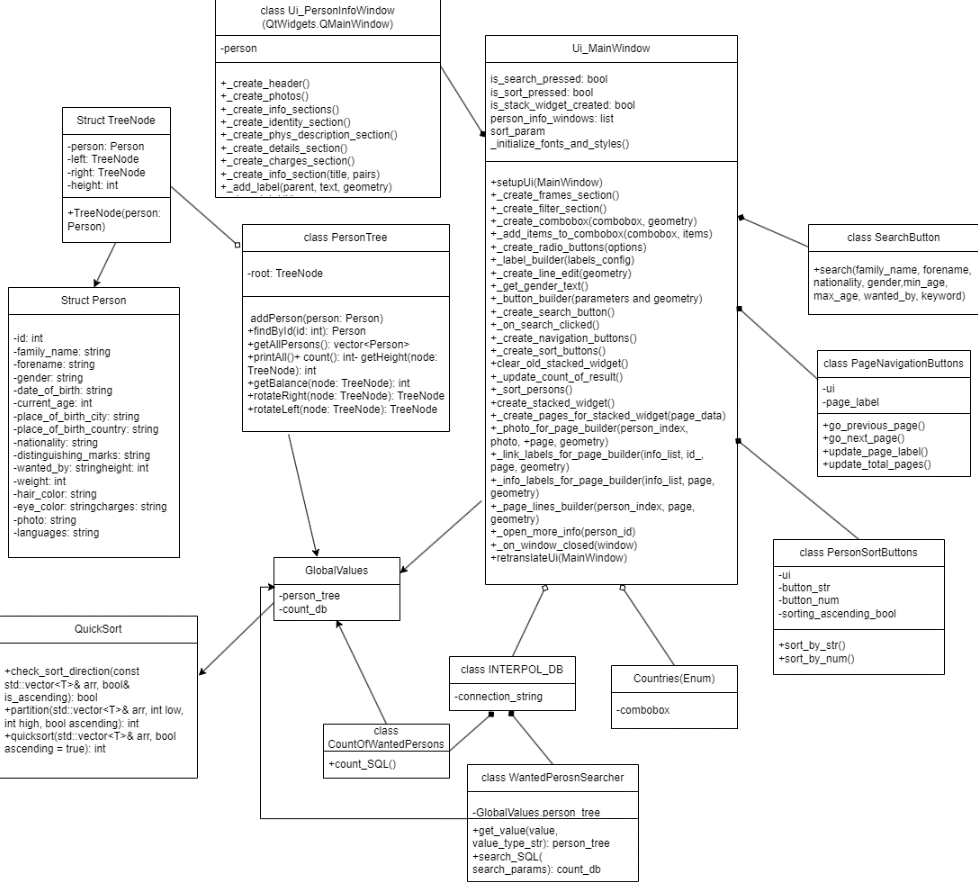


Рис 3.2. UML діаграма

|  |  |
| --- | --- |
|  | (3.1) |

Де

C – кількість сторінок,

N – загальна кільсть осіб,

M – максимальна кількість фото на сторінці

|  |  |
| --- | --- |
|  | (3.2) |

Де

row – рядок,

i – індекс елемента у списку,

n – кількість колонок у сітці

|  |  |
| --- | --- |
|  | (3.3) |

Де

col – колонка,

i – індекс елемента у списку,

n – кількість колонок у сітці,

Mod – залишок від ділення

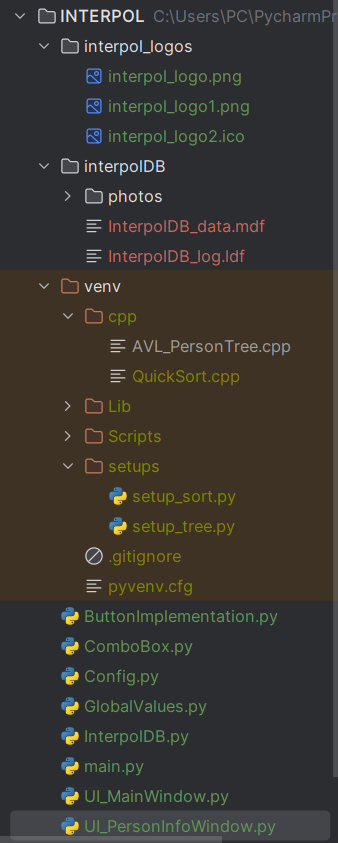


Рис 4.1. Розташування файлів у програмі

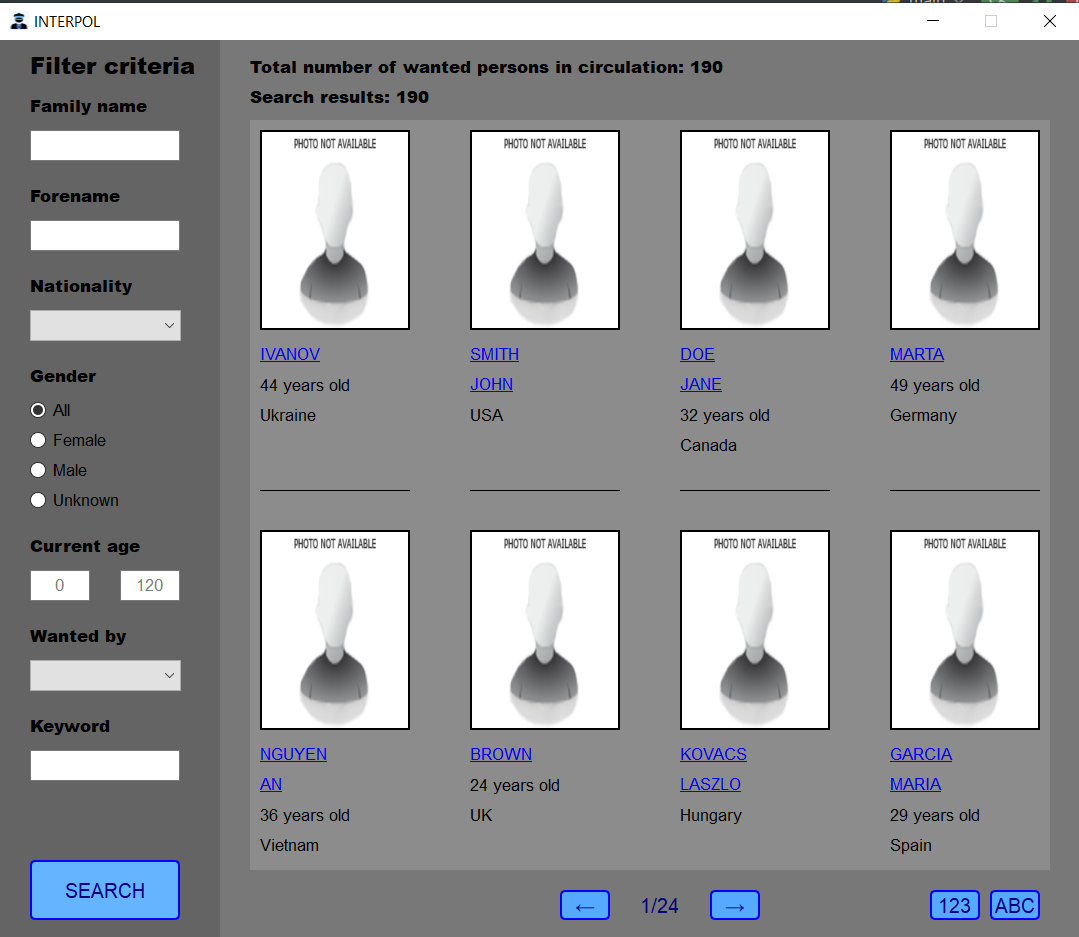


Рис. 5.1. Вигляд програми

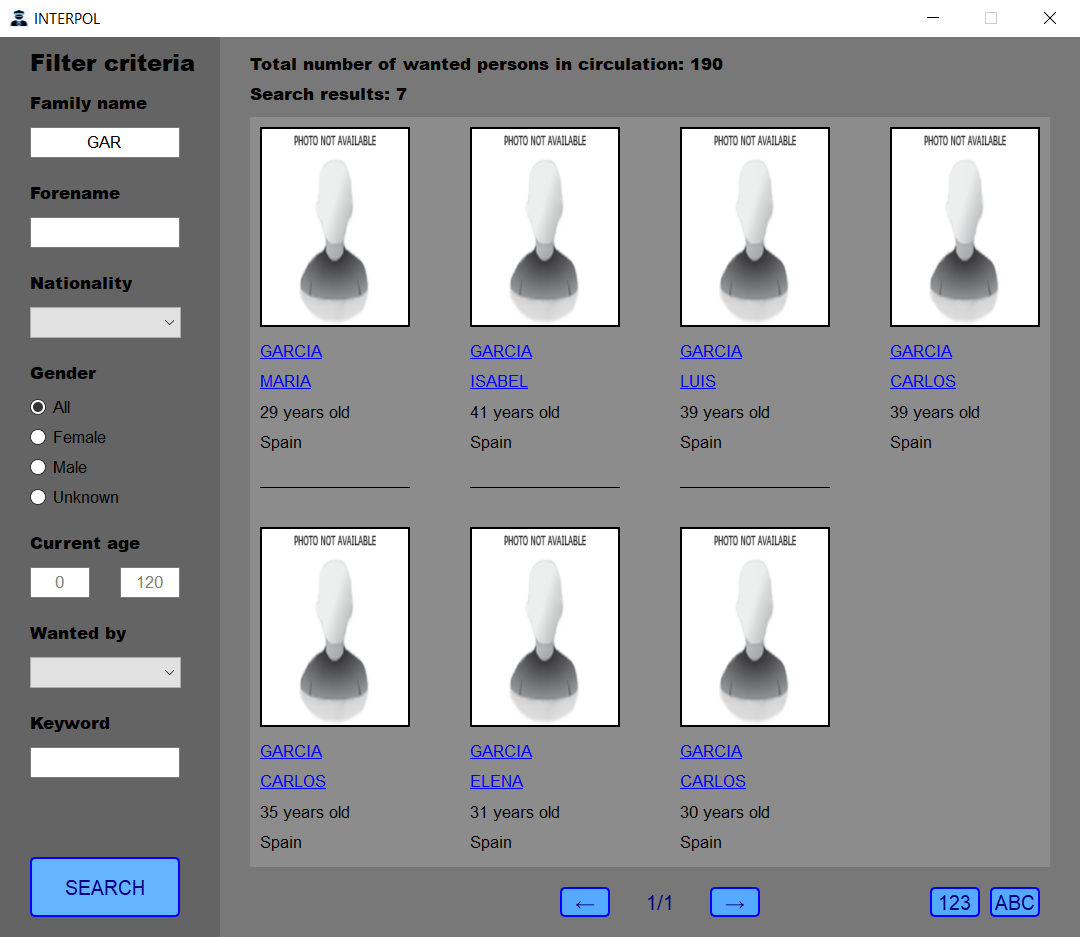


Рис. 5.2. Вибірка за одним параметром

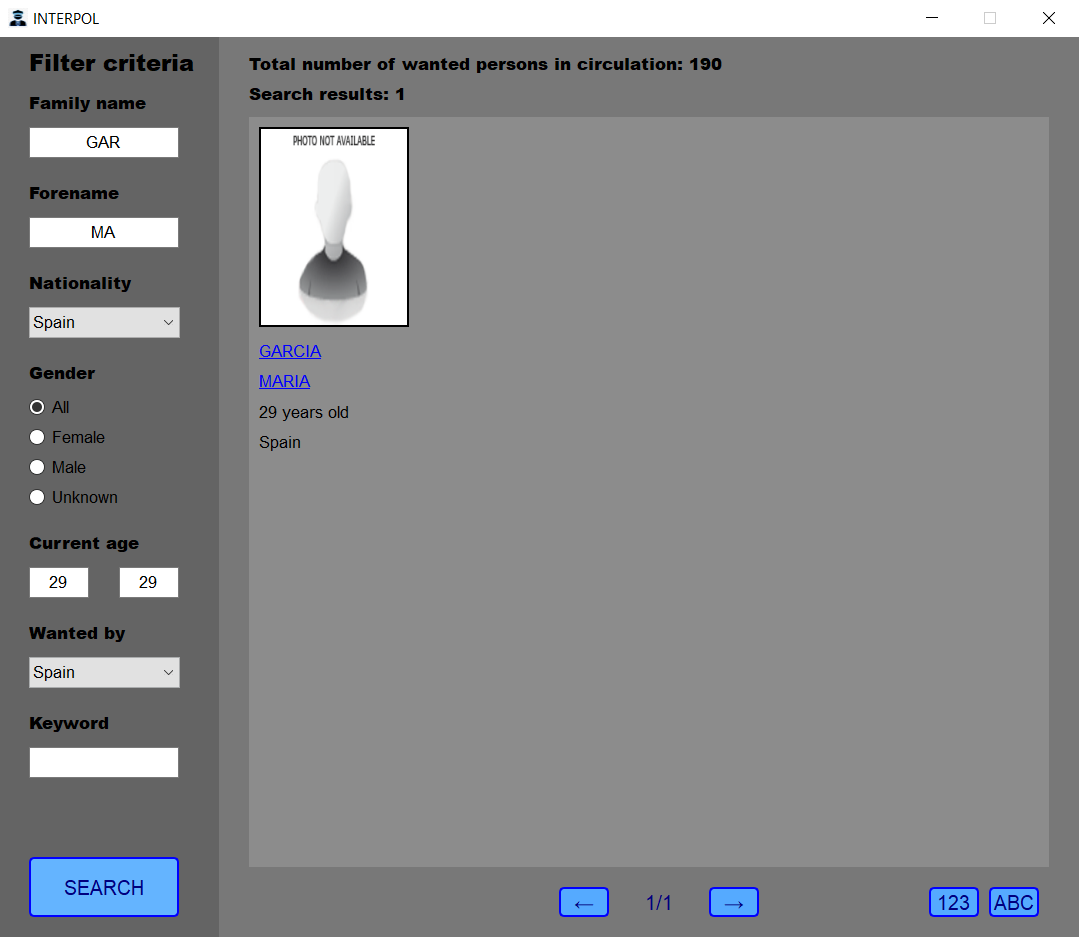


Рис. 5.3. Вибірка за всіма параметрами

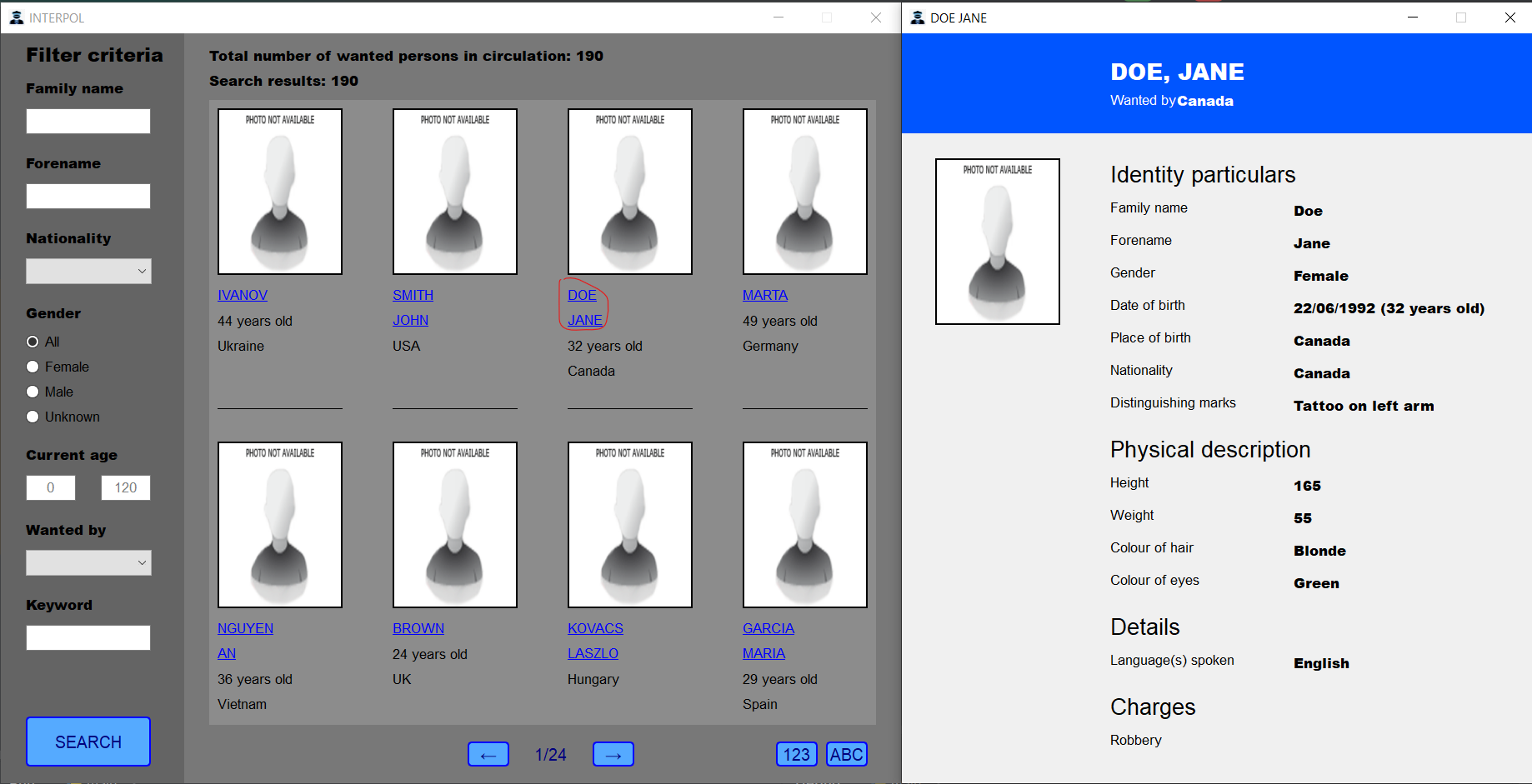


Рис. 5.4. Взаємодія головного і додаткового вікон

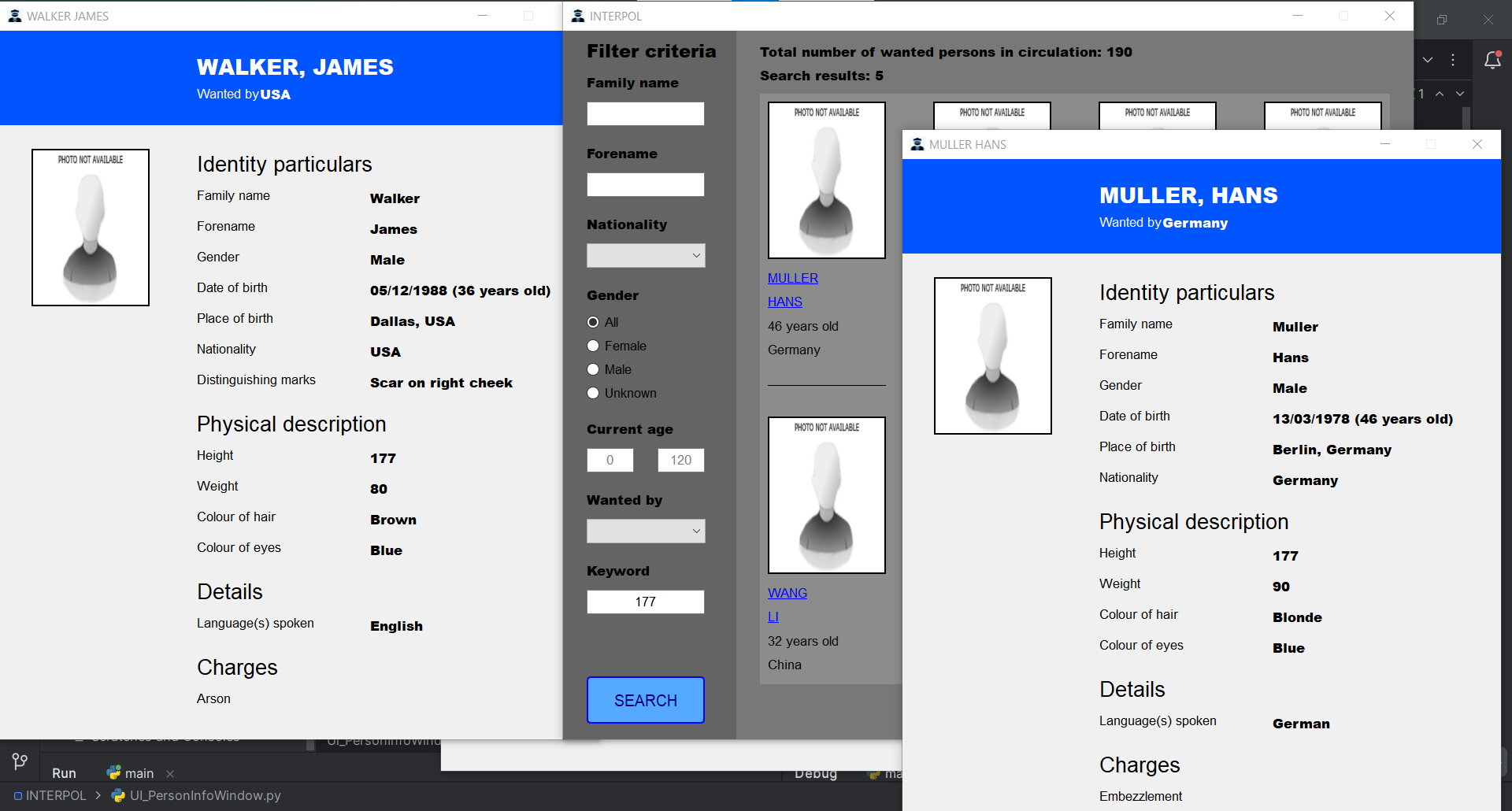


Рис 5.5. Одразу кілька відкритих вікон

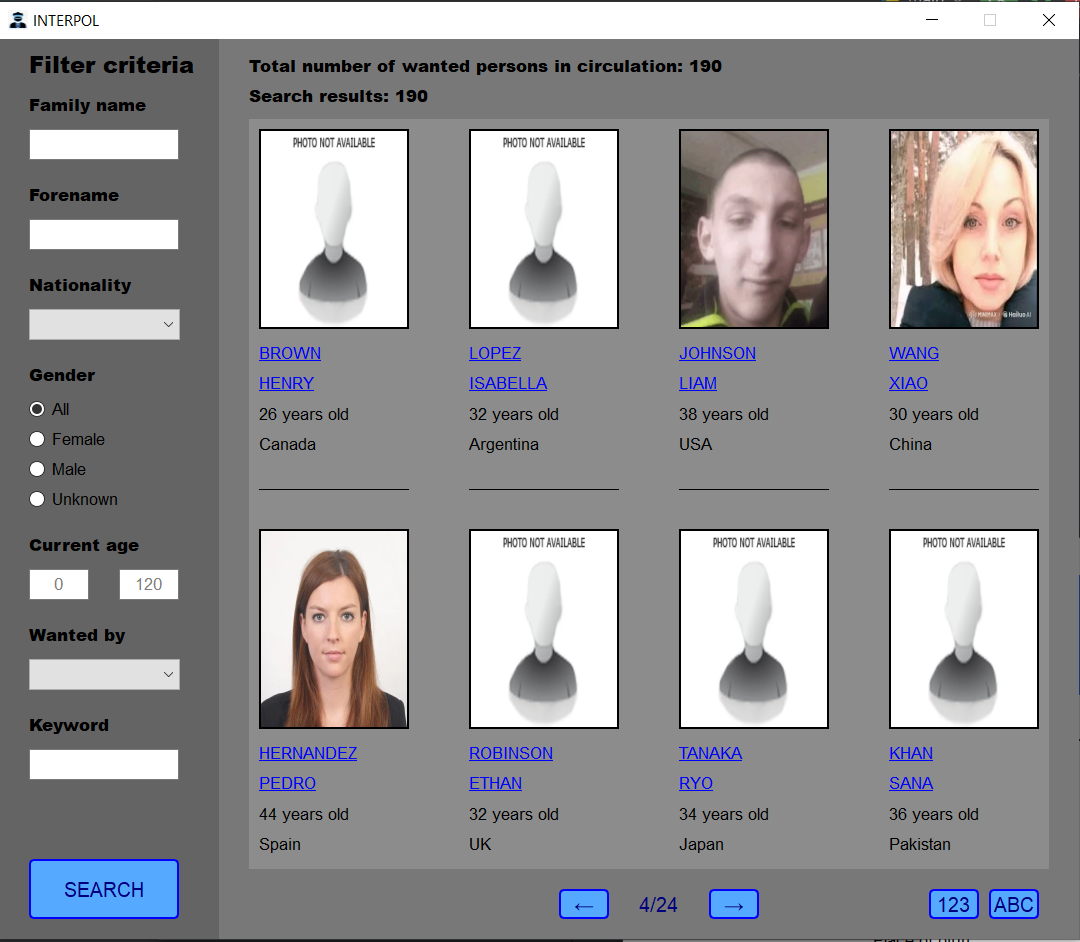


Рис 5.6. Відображення фото

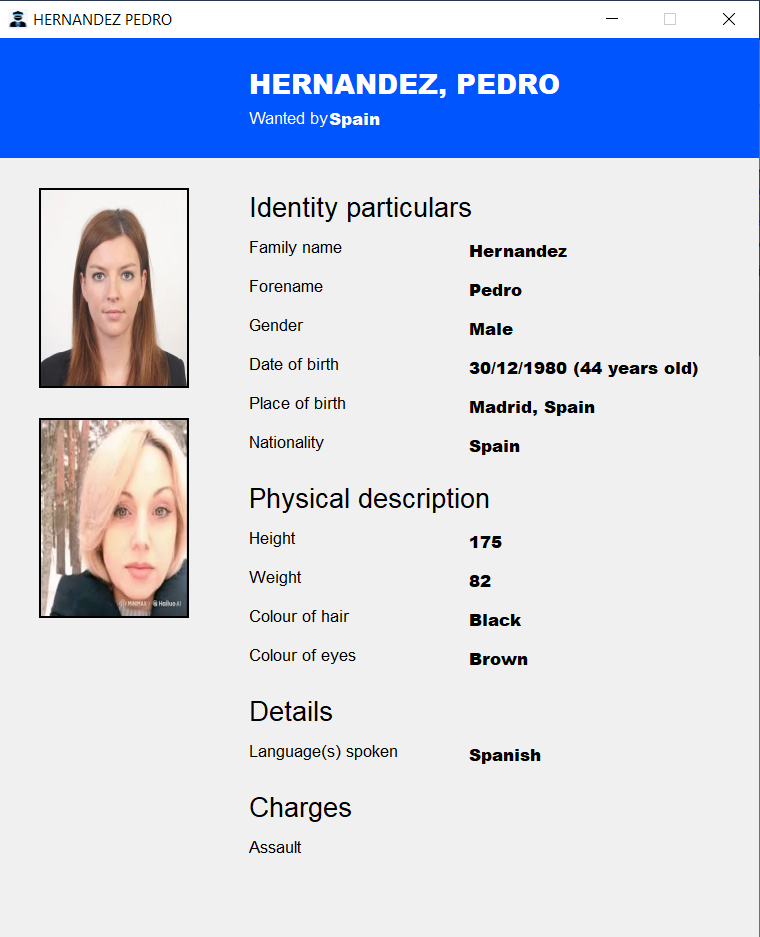


Рис 5.7. Вікно з детальною інформацією

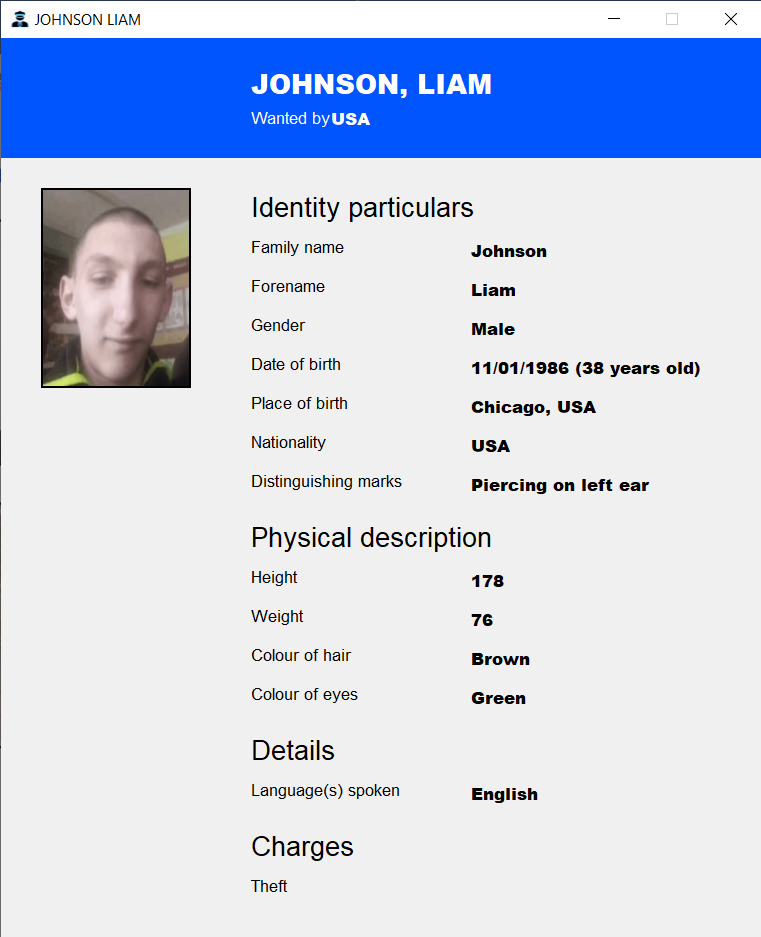


Рис 5.8. Одне фото

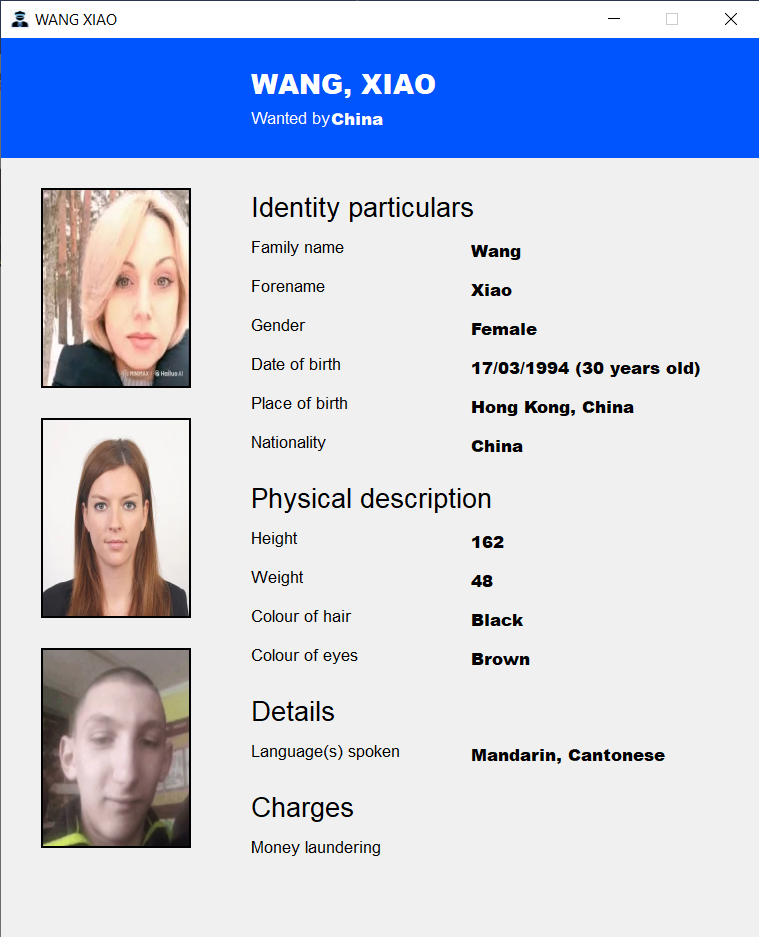


Рис 5.9. Максимальна кількість фото

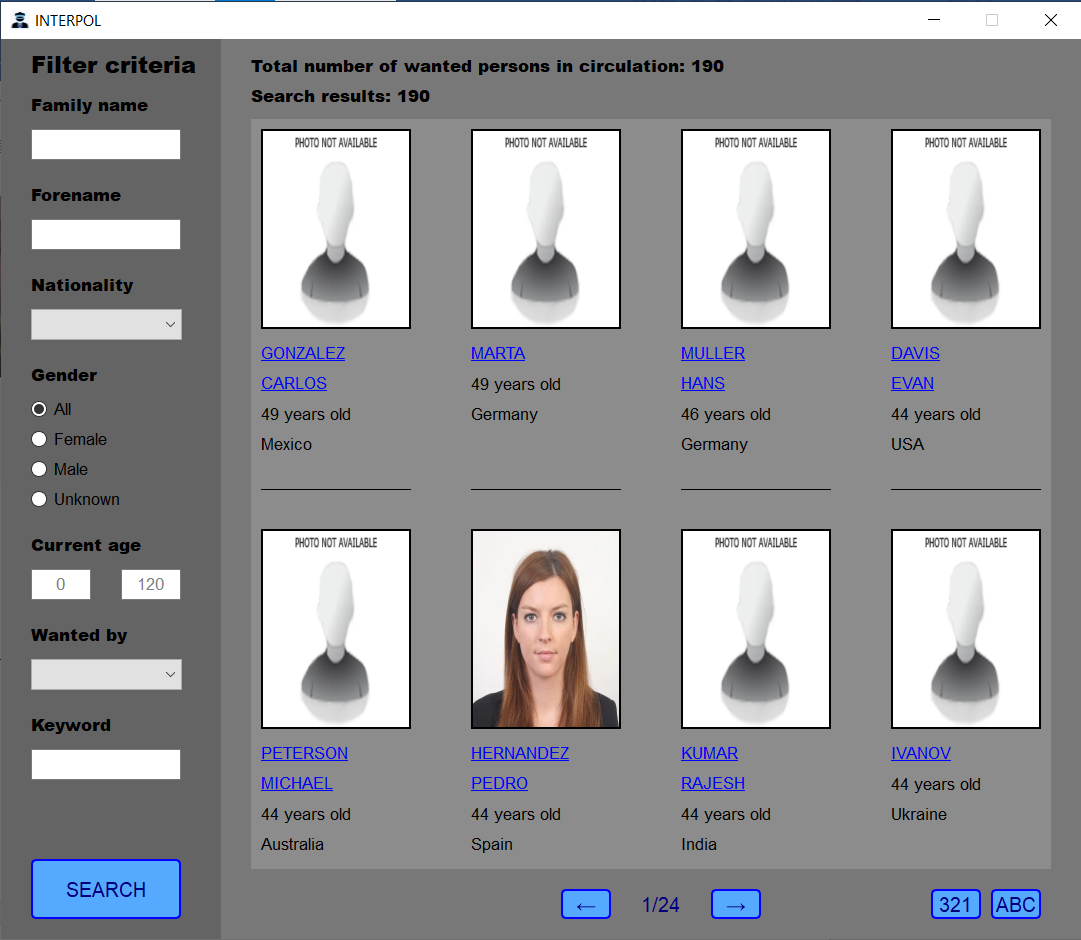


Рис 5.10. Сортування за віком від старшого до молодшого

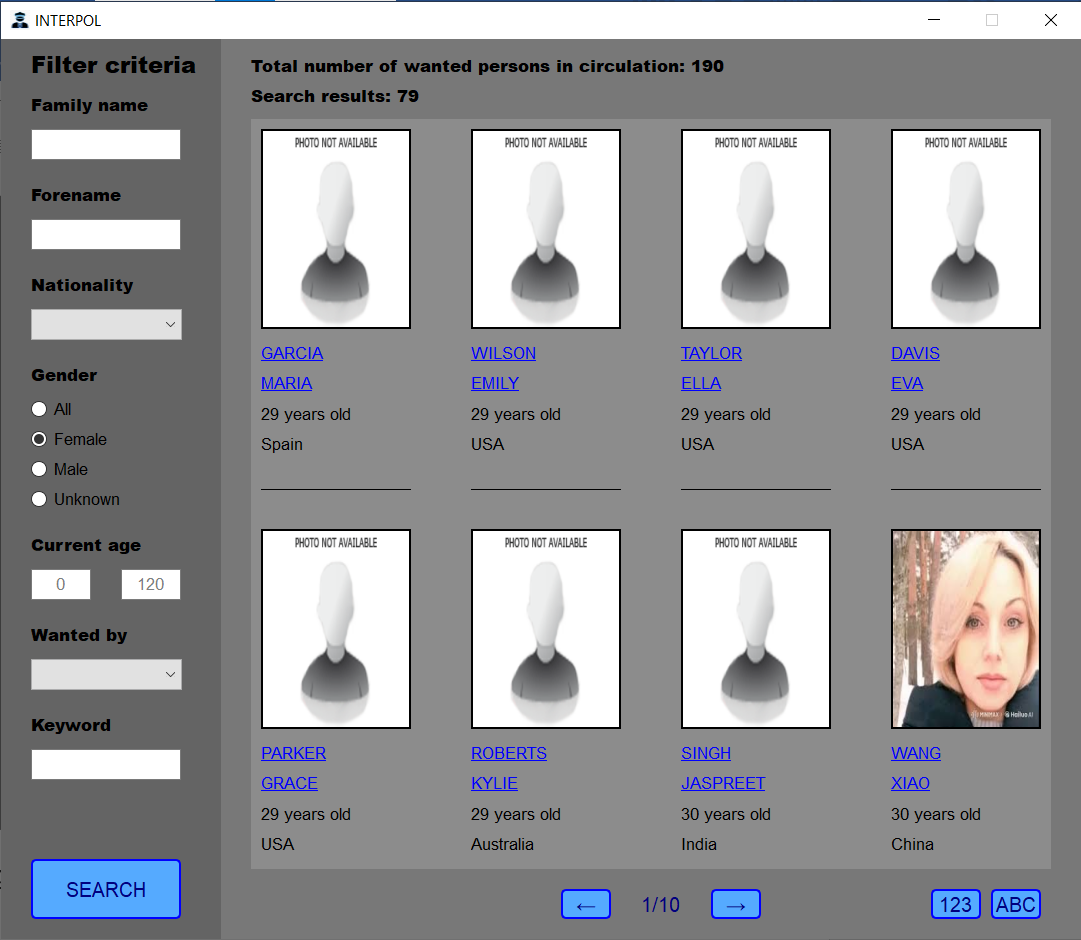


Рис. 5.11. Сортування з вибраним параметром від наймолодшого до найстаршого

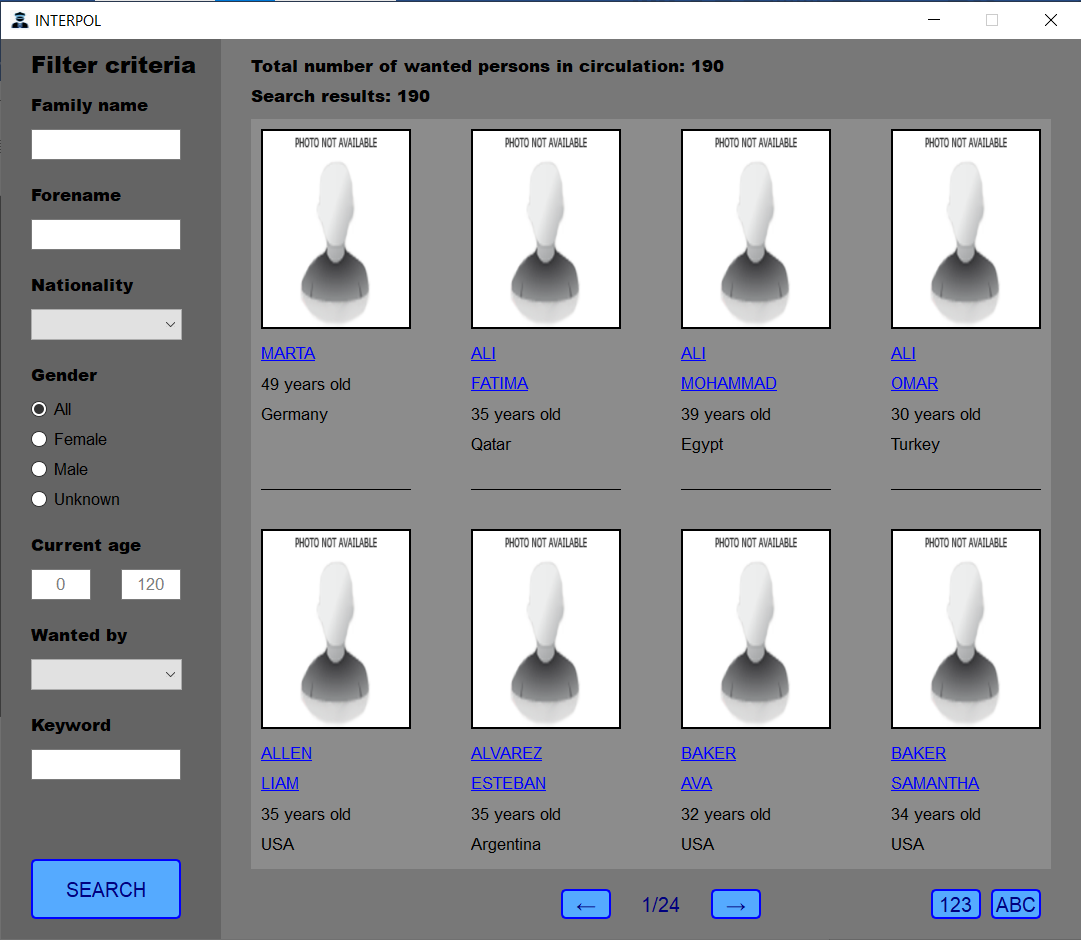


Рис 5.12. Сортування за алфавітом від а до я за ім'ям та прізвищем

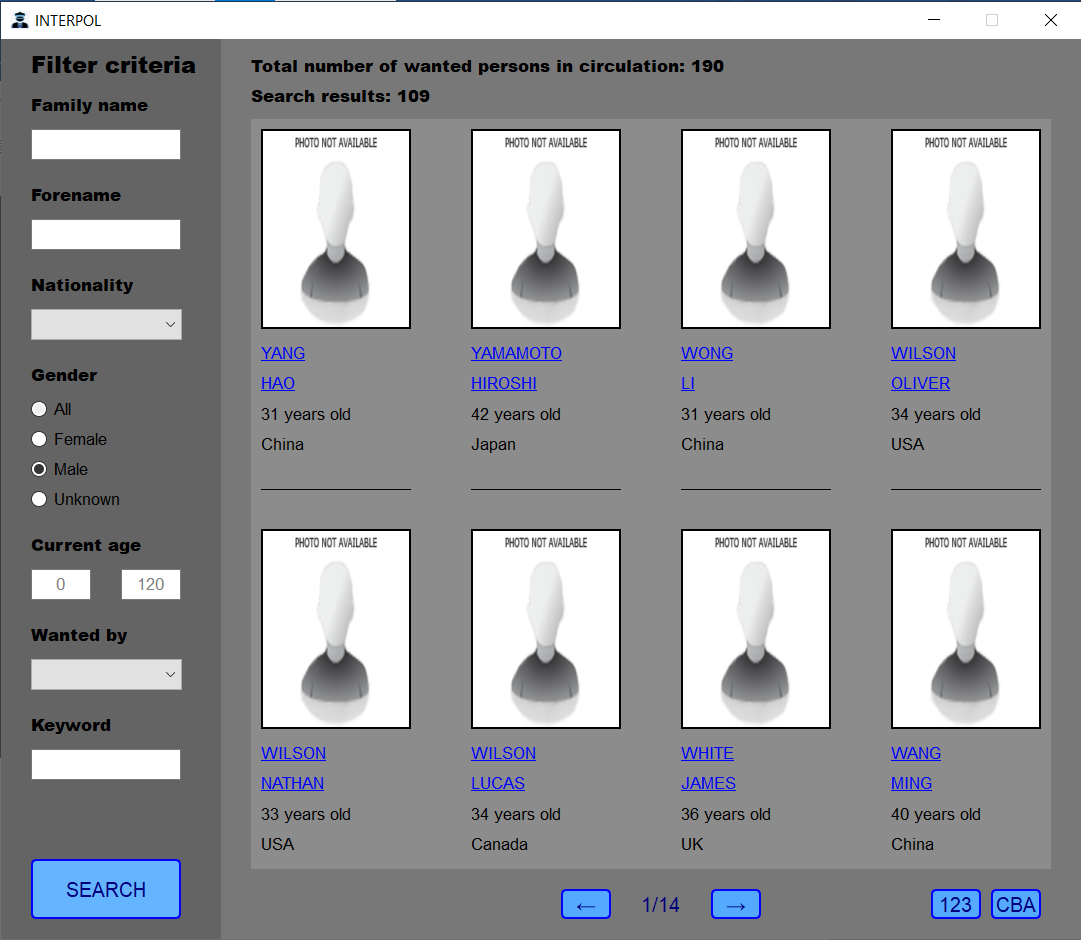


Рис. 5.13. Сортування за алфавітом від я до а з вибраним параметром

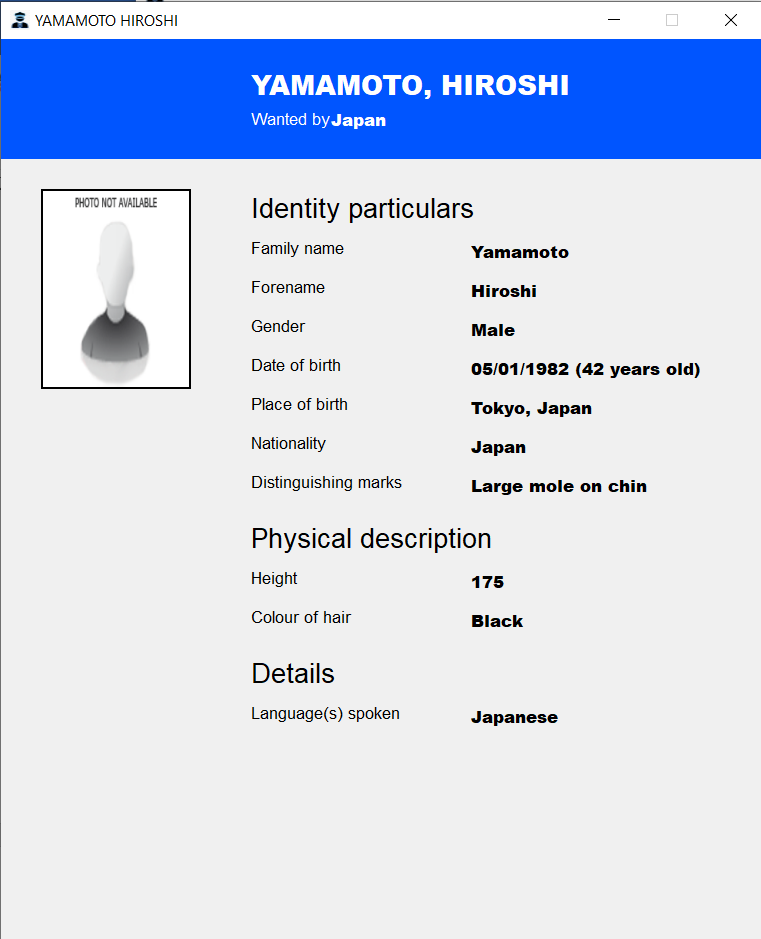


Рис. 5.14. Форматування вікна з відсутнім блоком інформації

# **Додаток Б**

**Код програми**

Файл main.py:

import sys  
from PyQt5.QtWidgets import QApplication, QMainWindow  
from UI\_MainWindow import Ui\_MainWindow  
  
  
class MainWindow(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.ui = Ui\_MainWindow()  
 self.ui.setupUi(self)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = QApplication(sys.argv)  
 main\_window = MainWindow()  
 main\_window.show()  
 sys.exit(app.exec\_())

Файл Ui\_MainWindow.py:

from PyQt5 import QtCore, QtGui, QtWidgets  
from PyQt5.QtGui import QIcon  
from UI\_PersonInfoWindow import Ui\_PersonInfoWindow  
from ButtonImplementation import SearchButton, PageNavigationButtons, PersonsSortButtons  
  
import ComboBox  
import Config as cfg  
import GlobalValues  
import InterpolDB  
  
import quicksort  
import time  
  
  
class Ui\_MainWindow:  
 def \_\_init\_\_(self):  
 self.is\_search\_pressed = False  
 self.is\_sort\_pressed = False  
 self.is\_stack\_widget\_created = False  
 self.person\_info\_windows = []  
 self.sort\_param = None  
 self.\_initialize\_fonts\_and\_styles()  
  
 def \_initialize\_fonts\_and\_styles(self):  
 # Fonts  
 self.header\_font = QtGui.QFont('Arial Black', 14)  
 self.button\_font = QtGui.QFont('Arial', 12)  
 self.info\_font = QtGui.QFont('Arial', 10)  
 self.info\_bold\_font = QtGui.QFont('Arial Black', 10)  
  
 # colors  
 self.link\_color = 'color: rgb(0, 0, 80);'  
 self.photo\_border = 'border: 2px solid black;'  
  
 self.frame\_1\_color = 'background-color: rgb(100, 100, 100);'  
 self.frame\_2\_color = 'background-color: rgb(120, 120, 120);'  
 self.frame\_3\_color = 'background-color: rgb(140, 140, 140);'  
  
 def setupUi(self, MainWindow):  
 MainWindow.setObjectName('MainWindow')  
 MainWindow.setFixedSize(1080, 900)  
 MainWindow.setWindowIcon(QIcon(r'interpol\_logos\interpol\_logo.png'))  
  
 self.central\_widget = QtWidgets.QWidget(MainWindow)  
 self.central\_widget.setObjectName('central\_widget')  
  
 # Styles and labels  
 self.\_create\_frames\_section()  
 self.\_create\_filter\_section()  
  
 # Buttons  
 self.\_create\_search\_button()  
 self.\_create\_navigation\_buttons()  
 self.\_create\_sort\_buttons()  
  
 MainWindow.setCentralWidget(self.central\_widget)  
 self.retranslateUi(MainWindow)  
 QtCore.QMetaObject.connectSlotsByName(MainWindow)  
  
 def \_create\_frames\_section(self):  
 # Geometry for frames  
 frame\_1\_geometry = QtCore.QRect(0, 0, 220, 900)  
 frame\_2\_geometry = QtCore.QRect(220, 0, 861, 900)  
 frame\_3\_geometry = QtCore.QRect(250, 80, 800, 750)  
  
 # Frame for input  
 self.frame\_1 = QtWidgets.QFrame(self.central\_widget)  
 self.frame\_1.setGeometry(frame\_1\_geometry)  
 self.frame\_1.setStyleSheet(self.frame\_1\_color)  
  
 # Frame for main  
 self.frame\_2 = QtWidgets.QFrame(self.central\_widget)  
 self.frame\_2.setGeometry(frame\_2\_geometry)  
 self.frame\_2.setStyleSheet(self.frame\_2\_color)  
  
 # Frame for data  
 self.frame\_3 = QtWidgets.QFrame(self.central\_widget)  
 self.frame\_3.setGeometry(frame\_3\_geometry)  
 self.frame\_3.setStyleSheet(self.frame\_3\_color)  
  
 def \_create\_filter\_section(self):  
 # Lines for input  
 self.m\_family\_name\_input = self.\_create\_line\_edit((30, 90, 150, 31))  
 self.m\_forename\_input = self.\_create\_line\_edit((30, 180, 150, 31))  
 self.m\_min\_age\_input = self.\_create\_line\_edit((30, 530, 60, 31), placeholder='0')  
 self.m\_max\_age\_input = self.\_create\_line\_edit((120, 530, 60, 31), placeholder='120')  
 self.m\_keyword\_input = self.\_create\_line\_edit((30, 710, 150, 31))  
  
 # Validators for age  
 self.m\_min\_age\_input.setValidator(QtGui.QIntValidator(0, 120))  
 self.m\_max\_age\_input.setValidator(QtGui.QIntValidator(0, 120))  
  
 # Combobox central widget  
 self.m\_nationality\_combobox = QtWidgets.QComboBox(self.central\_widget)  
 self.m\_wanted\_by\_comboBox = QtWidgets.QComboBox(self.central\_widget)  
  
 # Creating a Combobox  
 self.\_create\_combobox(self.m\_nationality\_combobox, (30, 270, 151, 31))  
 self.\_create\_combobox(self.m\_wanted\_by\_comboBox, (30, 620, 151, 31))  
  
 # Initialize the Combobox  
 self.\_add\_items\_to\_combobox(self.m\_nationality\_combobox, ComboBox.Countries)  
 self.\_add\_items\_to\_combobox(self.m\_wanted\_by\_comboBox, ComboBox.Countries)  
  
 # Radio button style  
 gender\_options = [  
 ('m\_radioButton\_1', 'All', (30, 360, 95, 20)),  
 ('m\_radioButton\_2', 'Female', (30, 390, 95, 20)),  
 ('m\_radioButton\_3', 'Male', (30, 420, 95, 20)),  
 ('m\_radioButton\_4', 'Unknown', (30, 450, 95, 20)),  
 ]  
 self.\_create\_radio\_buttons(gender\_options)  
 self.m\_radioButton\_1.setChecked(True)  
  
 # Getting persons from the database  
 try:  
 InterpolDB.CountOfWantedPersons().count\_SQL()  
 except Exception as e:  
 print(f'Error in counting wanted persons: {e}')  
  
 # Input info styles  
 labels\_cfg = [  
 ('m\_filter\_criteria', 'Filter criteria', (30, 10, 201, 30), self.header\_font),  
 ('m\_family\_name', 'Family name', (30, 50, 150, 30), self.info\_bold\_font),  
 ('m\_forename', 'Forename', (30, 140, 150, 30), self.info\_bold\_font),  
 ('m\_nationality', 'Nationality', (30, 230, 150, 30), self.info\_bold\_font),  
 ('m\_gender', 'Gender', (30, 320, 150, 30), self.info\_bold\_font),  
 ('m\_current\_age', 'Current age', (30, 490, 150, 30), self.info\_bold\_font),  
 ('m\_wanted\_by', 'Wanted by', (30, 580, 150, 30), self.info\_bold\_font),  
 ('m\_keyword', 'Keyword', (30, 670, 150, 30), self.info\_bold\_font),  
 ('m\_tot\_num\_in\_circ', f'Total number of wanted persons in circulation: {GlobalValues.count\_db}',  
 (250, 10, 600, 31), self.info\_bold\_font),  
 ('m\_search\_results', '', (250, 40, 800, 31), self.info\_bold\_font),  
 ]  
 self.\_label\_builder(labels\_cfg)  
  
 def \_create\_combobox(self, combobox, geometry):  
 combobox.setGeometry(QtCore.QRect(\*geometry))  
 combobox.setFont(self.info\_font)  
  
 def \_add\_items\_to\_combobox(self, combobox, items):  
 for item in items:  
 combobox.addItem(item.value)  
  
 def \_create\_radio\_buttons(self, options):  
 for name, text, geometry in options:  
 radio\_btn = QtWidgets.QRadioButton(self.central\_widget)  
 radio\_btn.setObjectName(name)  
 radio\_btn.setGeometry(QtCore.QRect(\*geometry))  
 radio\_btn.setFont(self.info\_font)  
 radio\_btn.setText(text)  
 setattr(self, name, radio\_btn)  
  
 def \_label\_builder(self, labels\_cfg):  
 for name, text, geometry, font in labels\_cfg:  
 label = QtWidgets.QLabel(self.central\_widget)  
 label.setObjectName(name)  
 label.setGeometry(QtCore.QRect(\*geometry))  
 label.setFont(font)  
 label.setText(text)  
 setattr(self, name, label)  
  
 def \_create\_line\_edit(self, geometry, placeholder=''):  
 line\_edit = QtWidgets.QLineEdit(self.central\_widget)  
 line\_edit.setGeometry(QtCore.QRect(\*geometry))  
 line\_edit.setFont(self.info\_font)  
 line\_edit.setPlaceholderText(placeholder)  
 line\_edit.setAlignment(QtCore.Qt.AlignCenter)  
 return line\_edit  
  
 def \_get\_gender\_text(self):  
 if self.m\_radioButton\_2.isChecked():  
 return self.m\_radioButton\_2.text() # 'Female'  
 elif self.m\_radioButton\_3.isChecked():  
 return self.m\_radioButton\_3.text() # 'Male'  
 elif self.m\_radioButton\_4.isChecked():  
 return self.m\_radioButton\_4.text() # 'Unknown'  
 else:  
 return '' # 'All'  
  
 def \_button\_builder(self, name, text, geometry, font, style):  
  
 button = QtWidgets.QPushButton(self.central\_widget)  
 button.setObjectName(name)  
 button.setGeometry(QtCore.QRect(\*geometry))  
 button.setFont(font)  
 button.setText(text)  
 button.setStyleSheet(style)  
 return button  
  
 def \_create\_search\_button(self):  
 self.m\_btn\_search = self.\_button\_builder('m\_btn\_search', 'SEARCH',  
 (30, 820, 150, 60), self.button\_font,  
 cfg.style\_sheet\_btn)  
  
 # From button implementation  
 self.button = SearchButton()  
  
 self.m\_btn\_search.clicked.connect(self.\_on\_search\_clicked)  
  
 def \_on\_search\_clicked(self):  
 # Тепер пошук здійснюється лише після натискання кнопки  
 self.button.search(  
 self.m\_family\_name\_input.text(),  
 self.m\_forename\_input.text(),  
 self.m\_nationality\_combobox.currentText(),  
 self.\_get\_gender\_text(),  
 self.m\_min\_age\_input.text(),  
 self.m\_max\_age\_input.text(),  
 self.m\_wanted\_by\_comboBox.currentText(),  
 self.m\_keyword\_input.text(),  
 )  
  
 self.is\_search\_pressed = True  
  
 if self.is\_stack\_widget\_created:  
 self.clear\_old\_stacked\_widget()  
  
 if not self.is\_stack\_widget\_created:  
 self.create\_stacked\_widget()  
 self.is\_stack\_widget\_created = True  
  
 def \_create\_navigation\_buttons(self):  
 x\_pos = 635  
 y\_pos = 850  
 x\_width = 50  
 y\_height = 30  
  
 x\_gap = 25  
 x\_pos\_left = x\_pos - x\_width - x\_gap  
 x\_pos\_right = x\_pos + x\_width + x\_gap  
  
 # <- Button style for going to the previous page  
 self.m\_btn\_prev = self.\_button\_builder('m\_btn\_prev', '←',  
 (x\_pos\_left, y\_pos, x\_width, y\_height),  
 self.button\_font, cfg.style\_sheet\_btn)  
  
 # -> Button style for going to the next page  
 self.m\_btn\_next = self.\_button\_builder('m\_btn\_next', '→',  
 (x\_pos\_right, y\_pos, x\_width, y\_height),  
 self.button\_font, cfg.style\_sheet\_btn)  
  
 # Label style to show the current page number  
 self.page\_label = QtWidgets.QLabel(self.central\_widget)  
 self.page\_label.setFont(self.button\_font)  
 self.page\_label.setGeometry(QtCore.QRect(x\_pos, y\_pos, x\_width, y\_height))  
 self.page\_label.setAlignment(QtCore.Qt.AlignCenter)  
 self.page\_label.setStyleSheet(cfg.style\_sheet\_page\_num)  
  
 # From button implementation  
 self.buttons\_page = PageNavigationButtons(self, self.page\_label)  
  
 self.m\_btn\_prev.clicked.connect(self.buttons\_page.go\_previous\_page)  
 self.m\_btn\_next.clicked.connect(self.buttons\_page.go\_next\_page)  
  
 def \_create\_sort\_buttons(self):  
 x\_pos\_left = 930  
 y\_pos = 850  
 x\_width = 50  
 y\_height = 30  
  
 x\_gap = 10  
 x\_pos\_right = x\_pos\_left + x\_width + x\_gap  
 # Button style for sorting by family name and forename  
 self.m\_btn\_sort\_str = self.\_button\_builder('m\_btn\_sort\_str', 'ABC',  
 (x\_pos\_right, y\_pos, x\_width, y\_height),  
 self.button\_font, cfg.style\_sheet\_btn)  
  
 # Button style for sorting by age  
 self.m\_btn\_sort\_num = self.\_button\_builder('m\_btn\_sort\_num', '123',  
 (x\_pos\_left, y\_pos, x\_width, y\_height),  
 self.button\_font, cfg.style\_sheet\_btn)  
  
 # From button implementation  
 self.buttons = PersonsSortButtons(self, self.m\_btn\_sort\_str, self.m\_btn\_sort\_num)  
  
 self.m\_btn\_sort\_str.clicked.connect(self.buttons.sort\_by\_str)  
 self.m\_btn\_sort\_num.clicked.connect(self.buttons.sort\_by\_num)  
  
 def clear\_old\_stacked\_widget(self):  
 if hasattr(self, 'stackedWidget'):  
 # Видаляємо всі старі сторінки зі stackedWidget  
 for i in range(self.stackedWidget.count() - 1, -1, -1):  
 widget = self.stackedWidget.widget(i)  
 if widget != None:  
 self.stackedWidget.removeWidget(widget)  
 widget.deleteLater()  
  
 self.is\_stack\_widget\_created = False  
  
 def \_update\_count\_of\_result(self):  
 if GlobalValues.person\_tree.count() != 0:  
 new\_value = f'Search results: {GlobalValues.person\_tree.count()}'  
 else:  
 new\_value = f'There are no results for your search. Please select different criteria.'  
 # Update the new value for the result label  
 self.m\_search\_results.setText(new\_value)  
 self.m\_search\_results.update()  
  
 def \_sort\_persons(self):  
 # Persons from AVL tree  
 persons = GlobalValues.person\_tree.getAllPersons()  
  
 if not self.is\_sort\_pressed or not self.sort\_param:  
 return persons  
  
 sort\_type, ascending = self.sort\_param  
  
 start\_time = time.time()  
  
 if sort\_type == 'age':  
 sort\_keys = [f"{person.current\_age:05d}\_{person.id:05d}" for person in persons]  
 elif sort\_type == 'name':  
 sort\_keys = [f"{person.family\_name.lower()}\_{person.forename.lower()}\_{person.id:05d}"  
 for person in persons]  
 else:  
 return persons  
  
 sorted\_keys = quicksort.sort\_strings(sort\_keys, ascending)  
  
 if sort\_type == 'age':  
 key\_to\_person = {f"{person.current\_age:05d}\_{person.id:05d}": person  
 for person in persons}  
 else:  
 key\_to\_person = {f"{person.family\_name.lower()}\_{person.forename.lower()}\_{person.id:05d}": person  
 for person in persons}  
  
 end\_time = time.time()  
 execution\_time = end\_time - start\_time  
 print(f"Час виконання сортування: {execution\_time:.10f} секунд")  
 return [key\_to\_person[key] for key in sorted\_keys]  
  
 # Widget for pages  
 def create\_stacked\_widget(self):  
 if not self.is\_search\_pressed:  
 return  
  
 self.\_update\_count\_of\_result() # Update info about the result  
 self.clear\_old\_stacked\_widget() # Clear the last widget before creating a new one  
  
 self.stackedWidget = QtWidgets.QStackedWidget(self.central\_widget)  
 self.stackedWidget.setGeometry(QtCore.QRect(220, 20, 841, 800))  
  
 all\_persons = self.\_sort\_persons() # sorted list  
  
 max\_photos\_per\_page = 8  
 # Whole number of pages + remainder from division  
 max\_pages = ((len(all\_persons) // max\_photos\_per\_page)  
 + (1 if len(all\_persons) % max\_photos\_per\_page else 0))  
  
 self.page\_data = {} # Dictionary for displaying information on the page  
  
 for person in all\_persons:  
 photo = person.photo.split(',')[0] if person.photo else r'interpolDB\photos\unknown.png'  
 family\_name = person.family\_name  
 forename = person.forename  
 age = f'{person.current\_age} years old' if person.current\_age != -1 else ''  
 nationality = person.nationality  
 id\_ = person.id  
  
 for page\_key in range(1, max\_pages + 1):  
 if page\_key not in self.page\_data:  
 self.page\_data[page\_key] = []  
 if len(self.page\_data[page\_key]) < max\_photos\_per\_page:  
 self.page\_data[page\_key].append((  
 photo,  
 family\_name,  
 forename,  
 age,  
 nationality,  
 id\_))  
 break  
  
 self.\_create\_pages\_for\_stacked\_widget(self.page\_data)  
  
 def \_create\_pages\_for\_stacked\_widget(self, page\_data):  
  
 for page\_key, page\_photo\_position in page\_data.items():  
 page = QtWidgets.QWidget()  
 page.setObjectName(f'm\_page\_{page\_key}')  
  
 for person\_index, (photo, family\_name, forename, age, nationality, id\_) in enumerate(page\_photo\_position):  
 # Info position options  
 row = person\_index // 4  
 col = person\_index % 4  
  
 x\_offset = 40 + col \* 210  
 y\_offset = 70 + row \* 400  
  
 # Photo 3 by 4  
 x\_photo\_width = 150  
 y\_photo\_height = 200  
  
 y\_std\_pos = 210  
 y\_step = 30  
 x\_width = 150  
 y\_height = 30  
  
 y\_line\_shift = 40  
 y\_line\_std\_pos = y\_offset - y\_line\_shift  
 y\_line\_height = 1  
  
 # Call builder for photo  
 self.\_photo\_for\_page\_builder(person\_index, photo, x\_offset, y\_offset,  
 x\_photo\_width, y\_photo\_height, page)  
  
 # Create name links  
 labels\_first\_data\_res = [  
 ('m\_family\_name\_res', f'{family\_name.upper()}', self.info\_font),  
 ('m\_forename\_res', f'{forename.upper()}', self.info\_font),  
 ]  
  
 # New position for next labels & Call builder for links labels  
 y\_new\_pos = self.\_link\_labels\_for\_page\_builder(labels\_first\_data\_res, x\_offset,  
 y\_offset, y\_std\_pos, y\_step, x\_width,  
 y\_height, id\_, page)  
  
 labels\_second\_data\_res = [  
 ('m\_age\_res', f'{age}', self.info\_font),  
 ('m\_nationality\_res', f'{nationality}', self.info\_font)  
 ]  
  
 # Call builder for info labels  
 self.\_info\_labels\_for\_page\_builder(labels\_second\_data\_res, x\_offset, y\_offset,  
 y\_new\_pos, y\_step, x\_width, y\_height, page)  
  
 self.\_page\_lines\_builder(x\_offset, y\_line\_std\_pos, x\_width,  
 y\_line\_height, person\_index, page)  
  
 self.stackedWidget.addWidget(page)  
  
 self.buttons\_page.update\_total\_pages()  
 self.stackedWidget.show()  
 self.stackedWidget.setCurrentIndex(0)  
  
 def \_photo\_for\_page\_builder(self, person\_index, photo, x\_offset, y\_offset,  
 x\_photo\_width, y\_photo\_height, page):  
 photo\_label = QtWidgets.QLabel(page)  
 photo\_label.setObjectName(f'm\_photo\_{person\_index}')  
 photo\_label.setGeometry(QtCore.QRect(x\_offset, y\_offset, x\_photo\_width, y\_photo\_height))  
 photo\_label.setPixmap(QtGui.QPixmap(photo))  
 photo\_label.setScaledContents(True)  
 photo\_label.setStyleSheet(self.photo\_border)  
  
 def \_link\_labels\_for\_page\_builder(self, info\_list, x\_offset, y\_offset, y\_std\_pos, y\_step,  
 x\_width, y\_height, id\_, page):  
 step = 0  
 for name, text, font in info\_list:  
 if text:  
 link\_label = QtWidgets.QLabel(page)  
 link\_label.setObjectName(name)  
 link\_label.setGeometry(QtCore.QRect(x\_offset, y\_offset + y\_std\_pos + step,  
 x\_width, y\_height))  
 link\_label.setText(f'<a href="{id\_}">{text}</a>')  
 link\_label.setStyleSheet(self.link\_color)  
 link\_label.setFont(font)  
 link\_label.setOpenExternalLinks(False)  
 link\_label.linkActivated.connect(self.\_open\_more\_info)  
 step += y\_step  
 setattr(self, name, link\_label)  
 return step + y\_std\_pos # New position for next labels  
  
 def \_info\_labels\_for\_page\_builder(self, info\_list, x\_offset, y\_offset, y\_std\_pos, y\_step,  
 x\_width, y\_height, page):  
 step = 0  
 for name, text, font in info\_list:  
 if text:  
 label = QtWidgets.QLabel(page)  
 label.setObjectName(name)  
 label.setGeometry(QtCore.QRect(x\_offset, y\_offset + y\_std\_pos + step,  
 x\_width, y\_height))  
 label.setFont(font)  
 label.setText(text)  
 step += y\_step  
 setattr(self, name, label)  
 return step + y\_std\_pos # New position for next labels  
  
 def \_page\_lines\_builder(self, x\_offset, y\_line\_std\_pos, x\_width,  
 y\_line\_height, person\_index, page):  
 if person\_index in range(4, 8):  
 line = QtWidgets.QFrame(page)  
 line.setGeometry(QtCore.QRect(x\_offset, y\_line\_std\_pos, x\_width, y\_line\_height))  
 line.setFrameShape(QtWidgets.QFrame.HLine)  
 line.setFrameShadow(QtWidgets.QFrame.Raised)  
 line.setStyleSheet(self.photo\_border)  
  
 def \_open\_more\_info(self, person\_id):  
 person\_id = int(person\_id)  
 person = GlobalValues.person\_tree.findById(person\_id)  
  
 if person:  
 new\_window = Ui\_PersonInfoWindow(person)  
 self.person\_info\_windows.append(new\_window)  
 new\_window.destroyed.connect(lambda: self.\_on\_window\_closed(new\_window))  
 new\_window.show()  
  
 def \_on\_window\_closed(self, window):  
 if window in self.person\_info\_windows:  
 self.person\_info\_windows.remove(window)  
  
 def retranslateUi(self, MainWindow):  
 \_translate = QtCore.QCoreApplication.translate  
 MainWindow.setWindowTitle(\_translate('MainWindow', 'INTERPOL'))

Файл Ui\_PersonInfoWindow.py:

from PyQt5 import QtCore, QtGui, QtWidgets  
from PyQt5.QtGui import QIcon, QPixmap, QFontMetrics  
  
class Ui\_PersonInfoWindow(QtWidgets.QMainWindow):  
 def \_\_init\_\_(self, person):  
 super().\_\_init\_\_()  
 self.person = person  
 self.\_setup\_window()  
 self.\_initialize\_fonts\_and\_styles()  
 self.\_setup\_ui()  
  
 def \_setup\_window(self):  
 self.setWindowTitle(f"{self.person.family\_name.upper()} {self.person.forename.upper()}")  
 self.setFixedSize(760, 900)  
 self.setWindowIcon(QIcon(r'interpol\_logos\interpol\_logo.png'))  
  
 def \_initialize\_fonts\_and\_styles(self):  
 self.header\_color\_text = "color: white;"  
 self.header\_font = QtGui.QFont("Arial Black", 16)  
 self.section\_font = QtGui.QFont("Arial", 16)  
 self.info\_font = QtGui.QFont("Arial", 10)  
 self.info\_bold\_font = QtGui.QFont("Arial Black", 10)  
  
 def \_setup\_ui(self):  
 self.central\_widget = QtWidgets.QWidget(self)  
 self.setCentralWidget(self.central\_widget)  
 self.current\_y = 0  
  
 self.\_create\_header()  
 self.\_create\_photos()  
 self.\_create\_info\_sections()  
  
 def \_create\_header(self):  
 self.header = QtWidgets.QWidget(self.central\_widget)  
 self.header.setGeometry(0, 0, 760, 120)  
 self.header.setStyleSheet("background-color: rgb(0, 85, 255);")  
  
 name = ", ".join(filter(None, [self.person.family\_name.upper(),  
 self.person.forename.upper()])) or '"Unknown"'  
 self.\_add\_label(self.header, name, 250, 30, 470, 30,  
 self.header\_font, self.header\_color\_text)  
  
 self.\_add\_label(self.header, "Wanted by", 250, 70, 80, 20,  
 self.info\_font, self.header\_color\_text)  
 self.\_add\_label(self.header, self.person.wanted\_by, 330, 70, 390, 20,  
 self.info\_bold\_font, self.header\_color\_text)  
  
 self.current\_y = 120  
  
 def \_create\_photos(self):  
 photos = self.person.photo.split(',') if self.person.photo else [r'interpolDB\photos\unknown.png']  
 x\_offset, y\_offset, width, height, y\_shift = 40, 150, 150, 200, 230  
  
 for i, photo in enumerate(photos[:3]):  
 label = QtWidgets.QLabel(self.central\_widget)  
 label.setGeometry(x\_offset, y\_offset + i \* y\_shift, width, height)  
 label.setPixmap(QPixmap(photo))  
 label.setScaledContents(True)  
 label.setStyleSheet("border: 2px solid black;")  
  
 self.current\_y = y\_offset  
  
 def \_create\_info\_sections(self):  
 self.\_create\_identity\_section()  
 self.\_create\_phys\_description\_section()  
 self.\_create\_details\_section()  
 self.\_create\_charges\_section()  
  
 def \_create\_identity\_section(self):  
 place = ", ".join(filter(None, [self.person.place\_of\_birth\_city,  
 self.person.place\_of\_birth\_country]))  
 date\_of\_birth = f"{self.person.date\_of\_birth} ({self.person.current\_age} years old)" if self.person.date\_of\_birth else ''  
 pairs = [  
 ("Family name", self.person.family\_name),  
 ("Forename", self.person.forename),  
 ("Gender", self.person.gender),  
 ("Date of birth", date\_of\_birth),  
 ("Place of birth", place),  
 ("Nationality", self.person.nationality),  
 ("Distinguishing marks", self.person.distinguishing\_marks),  
 ]  
 self.\_create\_info\_section("Identity particulars", pairs)  
  
 def \_create\_phys\_description\_section(self):  
 pairs = [  
 ("Height", f"{self.person.height}" if self.person.height != -1 else ''),  
 ("Weight", f"{self.person.weight}" if self.person.weight != -1 else ''),  
 ("Colour of hair", self.person.hair\_color),  
 ("Colour of eyes", self.person.eye\_color)  
 ]  
 self.\_create\_info\_section("Physical description", pairs)  
  
 def \_create\_details\_section(self):  
 if self.person.languages:  
 self.\_create\_info\_section("Details", [("Language(s) spoken",  
 self.person.languages)])  
  
 def \_create\_charges\_section(self):  
 if self.person.charges:  
 title\_height = self.\_add\_label(self.central\_widget, "Charges", 250,  
 self.current\_y, 470, None, self.section\_font)  
 self.current\_y += title\_height + 10  
  
 charges\_height = self.\_add\_label(self.central\_widget, self.person.charges, 250,  
 self.current\_y, 470, None, self.info\_font)  
 self.current\_y += charges\_height + 10  
  
 def \_create\_info\_section(self, title, pairs):  
 if not any(value for \_, value in pairs):  
 return  
  
 x\_offset, x\_label\_width, x\_value\_width = 250, 200, 270  
 x\_value\_offset = x\_offset + x\_label\_width + 20  
  
 title\_height = self.\_add\_label(self.central\_widget, title, x\_offset,  
 self.current\_y, 470, None, self.section\_font)  
 self.current\_y += title\_height + 10  
  
 for label, value in pairs:  
 if value:  
 label\_height = self.\_add\_label(self.central\_widget, label, x\_offset,  
 self.current\_y, x\_label\_width, None,  
 self.info\_font)  
 value\_height = self.\_add\_label(self.central\_widget, str(value), x\_value\_offset,  
 self.current\_y, x\_value\_width, None,  
 self.info\_bold\_font)  
 self.current\_y += max(label\_height, value\_height) + 10  
  
 self.current\_y += 10  
  
 def \_add\_label(self, parent, text, x, y, width, height=None, font=None, stylesheet=None):  
 label = QtWidgets.QLabel(text, parent)  
 if font:  
 label.setFont(font)  
 if stylesheet:  
 label.setStyleSheet(stylesheet)  
 label.setWordWrap(True)  
 metrics = QFontMetrics(label.font())  
 if not height:  
 height = metrics.boundingRect(0, 0, width, 1000, QtCore.Qt.TextWordWrap, text).height() + 5  
 label.setGeometry(x, y, width, height)  
 return height  
  
 def retranslateUi(self):  
 self.setWindowTitle(f"{self.person.family\_name.upper()} {self.person.forename.upper()}")

Файл InterpolDB.py:

import pyodbc  
import person\_tree  
import GlobalValues  
import os  
  
connection\_string = (  
 "Driver={ODBC Driver 17 for SQL Server};"  
 "Server=DESKTOP-8NKLDQT\\MSSQLSERVER02;"  
 "Database=InterpolDB;"  
 "Trusted\_Connection=no;"  
 "UID=user;"  
 "PWD=1468;"  
 "Encrypt=no;"  
)  
  
  
class WantedPersonSearcher:  
 def \_\_init\_\_(self):  
 GlobalValues.person\_tree = person\_tree.PersonTree()  
  
 def get\_value(self, value, value\_type='str'):  
 if value is None:  
 if value\_type == 'str' or value\_type == 'date' or value\_type == 'photo':  
 return ''  
 elif value\_type == 'int':  
 return -1  
  
 if value\_type == 'str':  
 return str(value)  
 elif value\_type == 'int':  
 return int(value)  
 elif value\_type == 'date':  
 return value.strftime('%d/%m/%Y') if hasattr(value, 'strftime') else str(value)  
 elif value\_type == 'photo':  
  
 photo\_paths = value.split(',')  
 valid\_paths = []  
  
 for path in photo\_paths:  
 full\_path = rf'interpolDB\photos\{path.strip()}'  
 if os.path.isfile(full\_path):  
 valid\_paths.append(full\_path)  
  
 if valid\_paths:  
 return ','.join(valid\_paths)  
 else:  
 return ''  
  
  
 def search\_SQL(self, search\_params):  
 try:  
 with pyodbc.connect(connection\_string) as conn:  
 with conn.cursor() as cursor:  
 cursor.execute("""EXEC GetWantedPersons  
 @family\_name = ?,  
 @forename = ?,  
 @nationality = ?,  
 @gender = ?,  
 @min\_age = ?,  
 @max\_age = ?,  
 @wanted\_by = ?,  
 @keyword = ?""",  
 search\_params['family\_name'],  
 search\_params['forename'],  
 search\_params['nationality'],  
 search\_params['gender'],  
 search\_params['min\_age'],  
 search\_params['max\_age'],  
 search\_params['wanted\_by'],  
 search\_params['keyword'])  
  
 rows = cursor.fetchall()  
 columns = [column[0] for column in cursor.description]  
  
 for row in rows:  
 row\_dict = dict(zip(columns, row))  
  
 try:  
 person = person\_tree.Person(  
 self.get\_value(row\_dict.get('id'), 'int'),  
 self.get\_value(row\_dict.get('family\_name'), 'str'),  
 self.get\_value(row\_dict.get('forename'), 'str'),  
 self.get\_value(row\_dict.get('gender'), 'str'),  
 self.get\_value(row\_dict.get('date\_of\_birth'), 'date'),  
 self.get\_value(row\_dict.get('current\_age'), 'int'),  
 self.get\_value(row\_dict.get('place\_of\_birth\_city'), 'str'),  
 self.get\_value(row\_dict.get('place\_of\_birth\_country'), 'str'),  
 self.get\_value(row\_dict.get('nationality'), 'str'),  
 self.get\_value(row\_dict.get('distinguishing\_marks'), 'str'),  
 self.get\_value(row\_dict.get('wanted\_by'), 'str'),  
 self.get\_value(row\_dict.get('height'), 'int'),  
 self.get\_value(row\_dict.get('weight'), 'int'),  
 self.get\_value(row\_dict.get('hair\_color'), 'str'),  
 self.get\_value(row\_dict.get('eye\_color'), 'str'),  
 self.get\_value(row\_dict.get('charges'), 'str'),  
 self.get\_value(row\_dict.get('photo'), 'photo'),  
 self.get\_value(row\_dict.get('languages'), 'str')  
  
 )  
 GlobalValues.person\_tree.addPerson(person)  
 except Exception as e:  
 print(f"Error creating Person object: {e}")  
 print(f"Row data: {row\_dict}")  
  
 print(f"Total persons in the tree: {GlobalValues.person\_tree.count()}")  
  
 except pyodbc.Error as e:  
 print(f"SQL error when getting people: {e}")  
  
  
class CountOfWantedPersons:  
 def count\_SQL(self):  
 try:  
 with pyodbc.connect(connection\_string) as conn:  
 with conn.cursor() as cursor:  
 cursor.execute("SELECT COUNT(\*) FROM WantedPersons;")  
 result = cursor.fetchone()  
 GlobalValues.count\_db = result[0]  
  
 print(f"Total wanted persons in database: {GlobalValues.count\_db}")  
 except pyodbc.Error as e:  
 print(f"SQL error in counting wanted persons: {e}")

Файл GlobalValues.py:

person\_tree = None # AVL TREE  
count\_db = None

Файл Config.py:

### Style sheet for buttons###  
style\_sheet\_btn = """  
 QPushButton {  
 color: rgb(0, 0, 127);  
 background-color: rgb(85, 170, 255);  
 border: 2px solid blue;  
 border-radius: 5px;  
 }  
 QPushButton:pressed {  
 background-color: rgb(0, 100, 255);   
 border: 2px solid white;   
 padding-left: 3px;   
 padding-top: 3px;  
 }  
 QPushButton:hover {  
 background-color: rgb(100, 180, 255);   
 }  
 """  
  
### Style sheet for page number ###  
style\_sheet\_page\_num = """  
 color: rgb(0, 0, 127);  
 """

Файл ComboBox.py:

from enum import Enum  
  
  
class Countries(Enum):  
 NONE = ""  
 AFGHANISTAN = "Afghanistan"  
 ALGERIA = "Algeria"  
 ANDORRA = "Andorra"  
 ANGOLA = "Angola"  
 ANTIGUA\_AND\_BARBUDA = "Antigua and Barbuda"  
 ARGENTINA = "Argentina"  
 ARMENIA = "Armenia"  
 AUSTRALIA = "Australia"  
 AUSTRIA = "Austria"  
 AZERBAIJAN = "Azerbaijan"  
 BAHAMAS = "Bahamas"  
 BAHRAIN = "Bahrain"  
 BANGLADESH = "Bangladesh"  
 BARBADOS = "Barbados"  
 BELARUS = "Belarus"  
 BELGIUM = "Belgium"  
 BELIZE = "Belize"  
 BENIN = "Benin"  
 BHUTAN = "Bhutan"  
 BOLIVIA = "Bolivia"  
 BOSNIA\_AND\_HERZEGOVINA = "Bosnia and Herzegovina"  
 BOTSWANA = "Botswana"  
 BRAZIL = "Brazil"  
 BRUNEI = "Brunei"  
 BULGARIA = "Bulgaria"  
 BURKINA\_FASO = "Burkina Faso"  
 BURUNDI = "Burundi"  
 CABO\_VERDE = "Cabo Verde"  
 CAMBODIA = "Cambodia"  
 CAMEROON = "Cameroon"  
 CANADA = "Canada"  
 CENTRAL\_AFRICAN\_REPUBLIC = "Central African Republic"  
 CHAD = "Chad"  
 CHILE = "Chile"  
 CHINA = "China"  
 COLOMBIA = "Colombia"  
 COMOROS = "Comoros"  
 CONGO = "Congo"  
 COSTA\_RICA = "Costa Rica"  
 CROATIA = "Croatia"  
 CUBA = "Cuba"  
 CYPRUS = "Cyprus"  
 CZECH\_REPUBLIC = "Czech Republic"  
 DENMARK = "Denmark"  
 DJIBOUTI = "Djibouti"  
 DOMINICA = "Dominica"  
 DOMINICAN\_REPUBLIC = "Dominican Republic"  
 ECUADOR = "Ecuador"  
 EGYPT = "Egypt"  
 EL\_SALVADOR = "El Salvador"  
 EQUATORIAL\_GUINEA = "Equatorial Guinea"  
 ERITREA = "Eritrea"  
 ESTONIA = "Estonia"  
 ESWATINI = "Eswatini"  
 ETHIOPIA = "Ethiopia"  
 FIJI = "Fiji"  
 FINLAND = "Finland"  
 FRANCE = "France"  
 GABON = "Gabon"  
 GAMBIA = "Gambia"  
 GEORGIA = "Georgia"  
 GERMANY = "Germany"  
 GHANA = "Ghana"  
 GREECE = "Greece"  
 GRENADA = "Grenada"  
 GUATEMALA = "Guatemala"  
 GUINEA = "Guinea"  
 GUINEA\_BISSAU = "Guinea Bissau"  
 GUYANA = "Guyana"  
 HAITI = "Haiti"  
 HONDURAS = "Honduras"  
 HUNGARY = "Hungary"  
 ICELAND = "Iceland"  
 INDIA = "India"  
 INDONESIA = "Indonesia"  
 IRAN = "Iran"  
 IRAQ = "Iraq"  
 IRELAND = "Ireland"  
 ISRAEL = "Israel"  
 ITALY = "Italy"  
 JAMAICA = "Jamaica"  
 JAPAN = "Japan"  
 JORDAN = "Jordan"  
 KAZAKHSTAN = "Kazakhstan"  
 KENYA = "Kenya"  
 KIRIBATI = "Kiribati"  
 KOREA\_NORTH = "North Korea"  
 KOREA\_SOUTH = "South Korea"  
 KUWAIT = "Kuwait"  
 KYRGYZSTAN = "Kyrgyzstan"  
 LAOS = "Laos"  
 LATVIA = "Latvia"  
 LEBANON = "Lebanon"  
 LESOTHO = "Lesotho"  
 LIBERIA = "Liberia"  
 LIBYA = "Libya"  
 LIECHTENSTEIN = "Liechtenstein"  
 LITHUANIA = "Lithuania"  
 LUXEMBOURG = "Luxembourg"  
 MADAGASCAR = "Madagascar"  
 MALAWI = "Malawi"  
 MALAYSIA = "Malaysia"  
 MALDIVES = "Maldives"  
 MALI = "Mali"  
 MALTA = "Malta"  
 MARSHALL\_ISLANDS = "Marshall Islands"  
 MAURITANIA = "Mauritania"  
 MAURITIUS = "Mauritius"  
 MEXICO = "Mexico"  
 MICRONESIA = "Micronesia"  
 MOLDOVA = "Moldova"  
 MONACO = "Monaco"  
 MONGOLIA = "Mongolia"  
 MONTENEGRO = "Montenegro"  
 MOROCCO = "Morocco"  
 MOZAMBIQUE = "Mozambique"  
 MYANMAR = "Myanmar"  
 NAMIBIA = "Namibia"  
 NAURU = "Nauru"  
 NEPAL = "Nepal"  
 NETHERLANDS = "Netherlands"  
 NEW\_ZEALAND = "New Zealand"  
 NICARAGUA = "Nicaragua"  
 NIGER = "Niger"  
 NIGERIA = "Nigeria"  
 NORTH\_MACEDONIA = "North Macedonia"  
 NORWAY = "Norway"  
 OMAN = "Oman"  
 PAKISTAN = "Pakistan"  
 PALAU = "Palau"  
 PANAMA = "Panama"  
 PAPUA\_NEW\_GUINEA = "Papua New Guinea"  
 PARAGUAY = "Paraguay"  
 PERU = "Peru"  
 PHILIPPINES = "Philippines"  
 POLAND = "Poland"  
 PORTUGAL = "Portugal"  
 QATAR = "Qatar"  
 ROMANIA = "Romania"  
 RUSSIA = "Russia"  
 RWANDA = "Rwanda"  
 SAINT\_KITTS\_AND\_NEVIS = "Saint Kitts and Nevis"  
 SAINT\_LUCIA = "Saint Lucia"  
 SAINT\_VINCENT\_AND\_THE\_GRENADINES = "Saint Vincent and the Grenadines"  
 SAMOA = "Samoa"  
 SAN\_MARINO = "San Marino"  
 SAO\_TOME\_AND\_PRINCIPE = "Sao Tome and Principe"  
 SAUDI\_ARABIA = "Saudi Arabia"  
 SENEGAL = "Senegal"  
 SERBIA = "Serbia"  
 SEYCHELLES = "Seychelles"  
 SIERRA\_LEONE = "Sierra Leone"  
 SINGAPORE = "Singapore"  
 SINT\_MAARTEN = "Sint Maarten"  
 SLOVAKIA = "Slovakia"  
 SLOVENIA = "Slovenia"  
 SOLOMON\_ISLANDS = "Solomon Islands"  
 SOMALIA = "Somalia"  
 SOUTH\_AFRICA = "South Africa"  
 SOUTH\_SUDAN = "South Sudan"  
 SPAIN = "Spain"  
 SRI\_LANKA = "Sri Lanka"  
 SUDAN = "Sudan"  
 SURINAME = "Suriname"  
 SWEDEN = "Sweden"  
 SWITZERLAND = "Switzerland"  
 SYRIA = "Syria"  
 TAIWAN = "Taiwan"  
 TANZANIA = "Tanzania"  
 THAILAND = "Thailand"  
 TOGO = "Togo"  
 TONGA = "Tonga"  
 TRINIDAD\_AND\_TOBAGO = "Trinidad and Tobago"  
 TUNISIA = "Tunisia"  
 TURKMENISTAN = "Turkmenistan"  
 TUVALU = "Tuvalu"  
 UGANDA = "Uganda"  
 UKRAINE = "Ukraine"  
 URUGUAY = "Uruguay"  
 UZBEKISTAN = "Uzbekistan"  
 VANUATU = "Vanuatu"  
 VATICAN\_CITY = "Vatican City"  
 VENEZUELA = "Venezuela"  
 VIETNAM = "Vietnam"  
 YEMEN = "Yemen"  
 ZAMBIA = "Zambia"  
 ZIMBABWE = "Zimbabwe"

Файл ButtonImplementation.py:

import InterpolDB  
  
class SearchButton:  
 def search(self, family\_name, forename, nationality, gender,  
 min\_age, max\_age, wanted\_by, keyword):  
 print(f"Пошук за прізвищем: {family\_name}, ім'ям: {forename}, "  
 f"національністю: {nationality}, статтю: {gender}, "  
 f"вік: від {min\_age} до {max\_age}, бажано за: {wanted\_by}, "  
 f"ключове слово: {keyword}")  
  
 search\_params = {  
 'family\_name': family\_name or '',  
 'forename': forename or '',  
 'nationality': nationality or '',  
 'gender': gender or '',  
 'min\_age': min\_age or 0,  
 'max\_age': max\_age or 120,  
 'wanted\_by': wanted\_by or '',  
 'keyword': keyword or '',  
 }  
  
 searcher = InterpolDB.WantedPersonSearcher()  
 try:  
 searcher.search\_SQL(search\_params)  
 except Exception as e:  
 print(f"Error during search: {e}")  
  
class PageNavigationButtons:  
 def \_\_init\_\_(self, ui, page\_label):  
 self.ui = ui  
 self.page\_label = page\_label  
  
 def go\_previous\_page(self):  
 # Ensure the stackedWidget exists  
 if hasattr(self.ui, 'stackedWidget'):  
 current\_index = self.ui.stackedWidget.currentIndex()  
 if current\_index > 0:  
 self.ui.stackedWidget.setCurrentIndex(current\_index - 1)  
 self.update\_page\_label()  
  
 def go\_next\_page(self):  
 # Ensure the stackedWidget exists  
 if hasattr(self.ui, 'stackedWidget'):  
 current\_index = self.ui.stackedWidget.currentIndex()  
 if current\_index < self.ui.stackedWidget.count() - 1:  
 self.ui.stackedWidget.setCurrentIndex(current\_index + 1)  
 self.update\_page\_label()  
  
 def update\_page\_label(self):  
 if hasattr(self.ui, 'stackedWidget'):  
 current\_index = self.ui.stackedWidget.currentIndex() + 1  
 total\_pages = self.ui.stackedWidget.count()  
 self.page\_label.setText(f"{current\_index}/{total\_pages}")  
  
 def update\_total\_pages(self):  
 if hasattr(self.ui, 'stackedWidget'):  
 total\_pages = self.ui.stackedWidget.count()  
 if total\_pages == 0:  
 total\_pages = 1  
 self.page\_label.setText(f"1/{total\_pages}")  
  
class PersonsSortButtons:  
 def \_\_init\_\_(self, ui, button\_str, button\_num):  
 self.ui = ui  
 self.str\_button = button\_str  
 self.num\_button = button\_num  
 self.str\_ascending = True  
 self.num\_ascending = True  
  
 def sort\_by\_str(self):  
 if hasattr(self.ui, 'stackedWidget'):  
 self.str\_ascending = not self.str\_ascending  
 self.str\_button.setText('ABC' if self.str\_ascending else 'CBA')  
 self.ui.is\_sort\_pressed = True  
 self.ui.sort\_param = ('name', self.str\_ascending)  
 self.ui.clear\_old\_stacked\_widget()  
 self.ui.create\_stacked\_widget()  
  
 def sort\_by\_num(self):  
 if hasattr(self.ui, 'stackedWidget'):  
 self.num\_ascending = not self.num\_ascending  
 self.num\_button.setText('123' if self.num\_ascending else '321')  
 self.ui.is\_sort\_pressed = True  
 self.ui.sort\_param = ('age', self.num\_ascending)  
 self.ui.clear\_old\_stacked\_widget()  
 self.ui.create\_stacked\_widget()

Файл QuickSort.cpp:

#include <pybind11/pybind11.h>  
#include <pybind11/stl.h>  
#include <vector>  
#include <string>  
#include <algorithm>  
#include <stdexcept>  
  
namespace py = pybind11;  
  
// Check if the array is already sorted (in either direction)  
template <typename T>  
bool check\_sort\_direction(const std::vector<T>& arr, bool& is\_ascending) {  
 if (arr.size() <= 1) return true;  
  
 bool ascending = true, descending = true;  
  
 for (size\_t i = 1; i < arr.size() && (ascending || descending); ++i) {  
 if (arr[i] < arr[i-1]) ascending = false;  
 if (arr[i] > arr[i-1]) descending = false;  
 }  
  
 is\_ascending = ascending;  
 return ascending || descending;  
}  
  
template <typename T>  
int partition(std::vector<T>& arr, int low, int high, bool ascending) {  
 // Selecting the median of the three elements as the reference element  
 int mid = low + (high - low) / 2;  
 T pivot;  
  
 if ((arr[low] <= arr[mid] && arr[mid] <= arr[high]) ||  
 (arr[high] <= arr[mid] && arr[mid] <= arr[low])) {  
 pivot = arr[mid];  
 std::swap(arr[mid], arr[high]);  
 } else if ((arr[mid] <= arr[low] && arr[low] <= arr[high]) ||  
 (arr[high] <= arr[low] && arr[low] <= arr[mid])) {  
 pivot = arr[low];  
 std::swap(arr[low], arr[high]);  
 } else {  
 pivot = arr[high];  
 }  
  
 int i = low - 1;  
  
 for (int j = low; j < high; ++j) {  
 if ((ascending && arr[j] < pivot) || (!ascending && arr[j] > pivot)) {  
 ++i;  
 std::swap(arr[i], arr[j]);  
 }  
 }  
  
 std::swap(arr[i + 1], arr[high]);  
 return i + 1;  
}  
  
// Iterative QuickSort and InsertionSort  
template <typename T>  
void quicksort(std::vector<T>& arr, bool ascending = true) {  
 if (arr.size() <= 1) return;  
  
 const int INSERTION\_SORT\_THRESHOLD = 10;  
 std::vector<std::pair<int, int>> stack;  
 stack.reserve(2 \* log2(arr.size())); // Memory allocation optimization  
  
 stack.push\_back({0, static\_cast<int>(arr.size() - 1)});  
  
 while (!stack.empty()) {  
 int low = stack.back().first;  
 int high = stack.back().second;  
 stack.pop\_back();  
  
 // Use insertion sort for small subarrays  
 if (high - low < INSERTION\_SORT\_THRESHOLD) {  
 for (int i = low + 1; i <= high; ++i) {  
 T key = arr[i];  
 int j = i - 1;  
  
 while (j >= low && ((ascending && arr[j] > key) ||  
 (!ascending && arr[j] < key))) {  
 arr[j + 1] = arr[j];  
 --j;  
 }  
 arr[j + 1] = key;  
 }  
 continue;  
 }  
  
 if (low < high) {  
 int pi = partition(arr, low, high, ascending);  
  
 // First, process the smaller subarray  
 if (pi - low < high - pi) {  
 stack.push\_back({pi + 1, high});  
 stack.push\_back({low, pi - 1});  
 } else {  
 stack.push\_back({low, pi - 1});  
 stack.push\_back({pi + 1, high});  
 }  
 }  
 }  
}  
  
// Python wrappers with checks and error handling  
std::vector<int> sort\_numbers(const std::vector<int>& input, bool ascending = true) {  
 if (input.empty()) {  
 return input;  
 }  
  
 std::vector<int> arr = input;  
 bool current\_ascending;  
  
 // Check if the array is already sorted  
 if (check\_sort\_direction(arr, current\_ascending)) {  
 // If the array is sorted, but in the wrong direction  
 if (current\_ascending != ascending) {  
 std::reverse(arr.begin(), arr.end());  
 }  
 return arr;  
 }  
  
 try {  
 quicksort(arr, ascending);  
 } catch (const std::exception& e) {  
 throw std::runtime\_error("Error while sorting numbers: " + std::string(e.what()));  
 }  
  
 return arr;  
}  
  
std::vector<std::string> sort\_strings(const std::vector<std::string>& input, bool ascending = true) {  
 if (input.empty()) {  
 return input;  
 }  
  
 std::vector<std::string> arr = input;  
 bool current\_ascending;  
  
 if (check\_sort\_direction(arr, current\_ascending)) {  
 if (current\_ascending != ascending) {  
 std::reverse(arr.begin(), arr.end());  
 }  
 return arr;  
 }  
  
 try {  
 quicksort(arr, ascending);  
 } catch (const std::exception& e) {  
 throw std::runtime\_error("Error while sorting strings: " + std::string(e.what()));  
 }  
  
 return arr;  
}  
  
PYBIND11\_MODULE(quicksort, m) {  
 m.doc() = "Optimized QuickSort iterative method for numbers and strings";  
  
 m.def("sort\_numbers", &sort\_numbers,  
 py::arg("input"),  
 py::arg("ascending") = true,  
 "Sorts a list of numbers in ascending or descending order.");  
  
 m.def("sort\_strings", &sort\_strings,  
 py::arg("input"),  
 py::arg("ascending") = true,  
 "Sorts a list of strings in ascending or descending order.");  
}

Файл AVL\_PersonTree.cpp:

#include <pybind11/pybind11.h>  
#include <pybind11/stl.h> // std::vector to Python  
#include <string>  
#include <vector>  
#include <memory>  
#include <algorithm>  
#include <iostream>  
  
namespace py = pybind11;  
  
// Structure for storing personal data  
struct Person {  
 int id;  
 std::string family\_name;  
 std::string forename;  
 std::string gender;  
 std::string date\_of\_birth;  
 int current\_age;  
 std::string place\_of\_birth\_city;  
 std::string place\_of\_birth\_country;  
 std::string nationality;  
 std::string distinguishing\_marks;  
 std::string wanted\_by;  
 int height;  
 int weight;  
 std::string hair\_color;  
 std::string eye\_color;  
 std::string charges;  
 std::string photo;  
 std::string languages;  
  
 Person(int id, const std::string& family\_name, const std::string& forename,  
 const std::string& gender, const std::string& date\_of\_birth, int current\_age,  
 const std::string& place\_of\_birth\_city, const std::string& place\_of\_birth\_country,  
 const std::string& nationality, const std::string& distinguishing\_marks,  
 const std::string& wanted\_by, int height,  
 int weight, const std::string& hair\_color, const std::string& eye\_color,  
 const std::string& charges, const std::string& photo, const std::string& languages)  
 : id(id), family\_name(family\_name), forename(forename), gender(gender),  
 date\_of\_birth(date\_of\_birth), current\_age(current\_age),  
 place\_of\_birth\_city(place\_of\_birth\_city), place\_of\_birth\_country(place\_of\_birth\_country),  
 nationality(nationality), distinguishing\_marks(distinguishing\_marks),wanted\_by(wanted\_by),  
 height(height), weight(weight), hair\_color(hair\_color), eye\_color(eye\_color),  
 charges(charges), photo(photo), languages(languages) {}  
};  
  
// Structure for the tree node  
struct TreeNode {  
 Person person;  
 std::shared\_ptr<TreeNode> left;  
 std::shared\_ptr<TreeNode> right;  
 int height;  
  
 TreeNode(const Person& p) : person(p), left(nullptr), right(nullptr), height(1) {}  
};  
  
// Class for AVL-tree construction  
class PersonTree {  
private:  
 std::shared\_ptr<TreeNode> root;  
  
 int getHeight(const std::shared\_ptr<TreeNode>& node) {  
 return node ? node->height : 0;  
 }  
  
 int getBalance(const std::shared\_ptr<TreeNode>& node) {  
 return node ? getHeight(node->left) - getHeight(node->right) : 0;  
 }  
  
 std::shared\_ptr<TreeNode> rotateRight(std::shared\_ptr<TreeNode> y) {  
 auto x = y->left;  
 auto T2 = x->right;  
  
 x->right = y;  
 y->left = T2;  
  
 y->height = std::max(getHeight(y->left), getHeight(y->right)) + 1;  
 x->height = std::max(getHeight(x->left), getHeight(x->right)) + 1;  
  
 return x;  
 }  
  
 std::shared\_ptr<TreeNode> rotateLeft(std::shared\_ptr<TreeNode> x) {  
 auto y = x->right;  
 auto T2 = y->left;  
  
 y->left = x;  
 x->right = T2;  
  
 x->height = std::max(getHeight(x->left), getHeight(x->right)) + 1;  
 y->height = std::max(getHeight(y->left), getHeight(y->right)) + 1;  
  
 return y;  
 }  
  
 std::shared\_ptr<TreeNode> insert(std::shared\_ptr<TreeNode> node, const Person& person) {  
 if (!node) {  
 return std::make\_shared<TreeNode>(person);  
 }  
  
 // Comparison by ID for insertion  
 if (person.id == node->person.id) {  
 return node;  
 }  
  
 if (person.id < node->person.id) {  
 node->left = insert(node->left, person);  
 } else {  
 node->right = insert(node->right, person);  
 }  
  
 node->height = 1 + std::max(getHeight(node->left), getHeight(node->right));  
  
 int balance = getBalance(node);  
  
 // LL  
 if (balance > 1 && person.id < node->left->person.id) {  
 return rotateRight(node);  
 }  
 // RR  
 if (balance < -1 && person.id > node->right->person.id) {  
 return rotateLeft(node);  
 }  
 // LR  
 if (balance > 1 && person.id > node->left->person.id) {  
 node->left = rotateLeft(node->left);  
 return rotateRight(node);  
 }  
 // RL  
 if (balance < -1 && person.id < node->right->person.id) {  
 node->right = rotateRight(node->right);  
 return rotateLeft(node);  
 }  
  
 return node;  
 }  
  
 void printInOrder(const std::shared\_ptr<TreeNode>& node) const {  
 if (!node) return;  
 printInOrder(node->left);  
 std::cout << "ID: " << node->person.id << "\n";  
 std::cout << "Name: " << node->person.family\_name << " " << node->person.forename << "\n";  
 std::cout << "Gender: " << node->person.gender << "\n";  
 std::cout << "Date of Birth: " << node->person.date\_of\_birth << "\n";  
 std::cout << "Current Age: " << node->person.current\_age << "\n";  
 std::cout << "Place of Birth: " << node->person.place\_of\_birth\_city << ", " << node->person.place\_of\_birth\_country << "\n";  
 std::cout << "Nationality: " << node->person.nationality << "\n";  
 std::cout << "Distinguishing marks: " << node->person.distinguishing\_marks << "\n";  
 std::cout << "Wanted By: " << node->person.wanted\_by << "\n";  
 std::cout << "Height: " << node->person.height << " cm\n";  
 std::cout << "Weight: " << node->person.weight << " kg\n";  
 std::cout << "Hair Color: " << node->person.hair\_color << "\n";  
 std::cout << "Eye Color: " << node->person.eye\_color << "\n";  
 std::cout << "Charges: " << node->person.charges << "\n";  
 std::cout << "Photo: " << node->person.photo << "\n";  
 std::cout << "Languages: " << node->person.languages << "\n";  
 std::cout << "------------------------------------\n";  
 printInOrder(node->right);  
 }  
  
 // Recursive method for getting all persons  
 std::vector<Person> getAllPersonsHelper(const std::shared\_ptr<TreeNode>& node) const {  
 std::vector<Person> persons;  
 if (node) {  
 auto left\_persons = getAllPersonsHelper(node->left);  
 persons.insert(persons.end(), left\_persons.begin(), left\_persons.end());  
  
 persons.push\_back(node->person);  
  
 auto right\_persons = getAllPersonsHelper(node->right);  
 persons.insert(persons.end(), right\_persons.begin(), right\_persons.end());  
 }  
 return persons;  
 }  
  
 Person\* findPersonById(const std::shared\_ptr<TreeNode>& node, int id) const {  
 if (!node) {  
 return nullptr;  
 }  
  
 if (id == node->person.id) {  
 return &node->person;  
 }  
  
 if (id < node->person.id) {  
 return findPersonById(node->left, id);  
 }  
  
 return findPersonById(node->right, id);  
 }  
  
public:  
 void addPerson(const Person& person) {  
 root = insert(root, person);  
 }  
  
 std::vector<Person> getAllPersons() const {  
 return getAllPersonsHelper(root);  
 }  
  
 py::object findById(int id) const {  
 Person\* person = findPersonById(root, id);  
 if (person) {  
 return py::cast(\*person);  
 } else {  
 throw std::runtime\_error("Person with the given ID not found.");  
 }  
 }  
  
 void printAll() const {  
 printInOrder(root);  
 }  
  
 int count() const {  
 return getAllPersons().size();  
 }  
};  
  
PYBIND11\_MODULE(person\_tree, m) {  
 m.doc() = "AVL person tree for INTERPOL database";  
  
 py::class\_<Person>(m, "Person")  
 .def(py::init<int, std::string, std::string, std::string, std::string, int,  
 std::string, std::string, std::string, std::string, std::string, int, int,  
 std::string, std::string, std::string, std::string, std::string>())  
 .def\_readwrite("id", &Person::id)  
 .def\_readwrite("family\_name", &Person::family\_name)  
 .def\_readwrite("forename", &Person::forename)  
 .def\_readwrite("gender", &Person::gender)  
 .def\_readwrite("date\_of\_birth", &Person::date\_of\_birth)  
 .def\_readwrite("current\_age", &Person::current\_age)  
 .def\_readwrite("place\_of\_birth\_city", &Person::place\_of\_birth\_city)  
 .def\_readwrite("place\_of\_birth\_country", &Person::place\_of\_birth\_country)  
 .def\_readwrite("nationality", &Person::nationality)  
 .def\_readwrite("distinguishing\_marks", &Person::distinguishing\_marks)  
 .def\_readwrite("wanted\_by", &Person::wanted\_by)  
 .def\_readwrite("height", &Person::height)  
 .def\_readwrite("weight", &Person::weight)  
 .def\_readwrite("hair\_color", &Person::hair\_color)  
 .def\_readwrite("eye\_color", &Person::eye\_color)  
 .def\_readwrite("charges", &Person::charges)  
 .def\_readwrite("photo", &Person::photo)  
 .def\_readwrite("languages", &Person::languages);  
  
 py::class\_<PersonTree>(m, "PersonTree")  
 .def(py::init<>())  
 .def("addPerson", &PersonTree::addPerson)  
 .def("findById", &PersonTree::findById)  
 .def("getAllPersons", &PersonTree::getAllPersons)  
 .def("printAll", &PersonTree::printAll)  
 .def("count", &PersonTree::count);  
}