

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, інтуїтивно 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 на доречних мовах. Основні структура і сортування на C++.
 - 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.
4. Для програмних рішень використати мови C++ та Python.
5. Термін завершення роботи – 10 грудня 2024 р.

Завдання видано

09 вересня 2024 р.

Керівник

асистент каф. АСУ Івасів С. С.

ЗМІСТ

ВСТУП	3
1 ПОСТАНОВКА ЗАДАЧІ	4
2 ОГЛЯД ТА АНАЛІЗ ЛІТЕРАТУРНИХ ДЖЕРЕЛ..	Ошибка! Закладка не определена.
2.1 Огляд технологій і засобів для програмної реалізації.....	Ошибка! Закладка не определена.
2.2 Аналіз та вибір алгоритмів.....	Ошибка! Закладка не определена.
3 ОПИС СТРУКТУРИ ПРОГРАМНОГО РІШЕННЯ, МЕТОДІВ, АЛГОРИТМІВ І ДАНИХ	7
3.1 Діаграма USE-CASE	7
3.2 Опис структури програми	8
3.3 Опис алгоритмів	9
4 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЄКТУ	Ошибка! Закладка не определена.
4.1 Діаграма компонентів.....	Ошибка! Закладка не определена.
4.2 Опис структури програмного проєкту	Ошибка! Закладка не определена.
5 ОПИС ПРОВОДЕНИХ ЕКСПЕРИМЕНТІВ	17
5.1 Вимоги до запуску програми	17
5.2 Тестові сценарії:	18
5.3 Вказівки для встановлення п	19
ВИСНОВОК.....	19
ЛІТЕРАТУРНІ ДЖЕРЕЛА.....	20
ДОДАТКИ.....	22
Додаток 1.....	22
Додаток 2.....	Ошибка! Закладка не определена.

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

1. ПОСТАНОВКА ЗАДАЧІ

Мета розробки програмного продукту полягає у створенні програмної версії сайту “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, інтуїтивно зрозумілий інтерфейс для користувача, швидкий пошук об'єктів бази даних за параметрами. Для цього можна використати такі параметри вхідних даних: ім'я, фамілія, національність, стать, вік, країна, яка розшукує, та ключове слово.

1.1. Функціональні можливості даного програмного рішення:

- 1.1.1. Пошук за параметрами

- 1.1.2. Сортування результату вибірки за ім'ям або віком

- 1.1.3. Отримання повної існуючої інформації об'єктів

1.2. Функціональні обмеження даного програмного рішення :

- 1.2.1. Фіксований розмір головного вікна

- 1.2.2. На сторінці максимум 8 об'єктів

1.3. Встановлені умови з боку користувача:

- 1.3.1. Зручний інтерфейс

- 1.3.2. Консистентність даних об'єктів

- 1.3.3. Швидкодія програми навіть з великою кількістю даних

1.4. Встановлені вимоги до програми з боку апаратних та програмних платформ:

- 1.4.1. Платформа Windows із підтримкою Python версії 3+ та C++ версії 11+

- 1.4.2. Необхідна реляційна база даних SQL server management studio для роботи з об'єктами

- 1.4.3. Додатково потрібно встановити бібліотеки pybind11 та pyodbc

- 1.4.4. Місце на жорсткому диску 500мб.

2. ОГЛЯД ТА АНАЛІЗ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

2.1. 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 C++ за певним критеріями:

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

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 C++ у Python буде використано модуль `pybind11`

2.2. 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 виконуватись 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 рішенні 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)

3. ОПИС СТРУКТУРИ ПРОГРАМНОГО РІШЕННЯ, МЕТОДІВ, АЛГОРИТМІВ І ДАНИХ

3.1. Діаграма USE-CASE



Рис. 3.1. Діаграма USE-CASE

3.2. 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 кількості 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 WantedPerosonSearcher і CountOfWantedPersons)

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.

3.3. 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.
 - 3.1. Якщо типом є рядок, і він не None (NULL з бази даних у python перетворюється у None), то він не змінюється, інакше – порожній рядок ''.
 - 3.2. 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).
 - 3.3. 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.

- 3.4. 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:
 - 2.1. Якщо ця кількість не нуль, то оновлюємо статус
 - 2.2. 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.
 - 4.1. Якщо це перше зчитування даних, то воно завжди буде невідсортоване, але при подальших діях все буде працювати коректно.
 - 4.2. Натиснувши на одну з кнопок сортування повернеться відсортований список з інформацією про об'єкти.
5. Вираховуємо необхідну кількість сторінок для виведення усіх об'єктів.
 - 5.1. 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:
 - 8.1. 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.
 - 8.2. 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).
 - 8.3. Ну і останніми будуть вік і національність, і якщо він існує, то до нього додається рядок “years old”.
 - 8.4. Якщо ж певного або певних параметрів не існує, то текст підтягується по висоті для гарного і зручного вигляду.
 - 8.5. 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 закінчатся 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:

- 2.1. 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.
- 2.2. Зліва під заголовком будуть розміщені фотографії у обмеженні 3 штук.
- 2.3. Справа від фотографій буде назва відділу інформації, назва параметру та його значення.
- 2.4. 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
 - 1.1. 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.
 - 1.2. 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
 - 2.1. 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.
 - 2.2. 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

2.3.Пошук за ID: Метод `findById` використовує рекурсивний пошук для знаходження особи з заданим ідентифікатором. Якщо особу не знайдено, кидається виняток.

2.4.Отримання всіх осіб: Метод `getAllPersons` рекурсивно збирає всіх осіб дерева і повертає їх у вигляді вектора.

Алгоритм сортування:

1. Перевірка напрямку сортування:

1.1. 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:

1.1. 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:

1.1. 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.

1.2. Окрім того, алгоритм оптимізований для пам'яті, оскільки стек для зберігання діапазонів масивів виділяється заздалегідь.

4. Перевірка вже відсортованих масивів:

1.1. Якщо масив вже відсортований, і його напрямок не відповідає бажаному, відбувається просто перевертання масиву.

4. ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЄКТУ

4.1. Загальна структура програмного проєкту

На рис. 4.1 у додатку А зображена структура 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.

4.2. 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];
```

5. ОПИС ПРОВЕДЕНИХ ЕКСПЕРИМЕНТІВ

5.1. 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 власнокомпільовані файли `сpp`.

Користувачеві:

1. Правильно завантажити програму.
2. Запустити програму через термінал або інтегроване середовище (IDE)

5.2. 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 пропускається 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 сортування.

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.

5.3. 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 бібліотеки

```
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

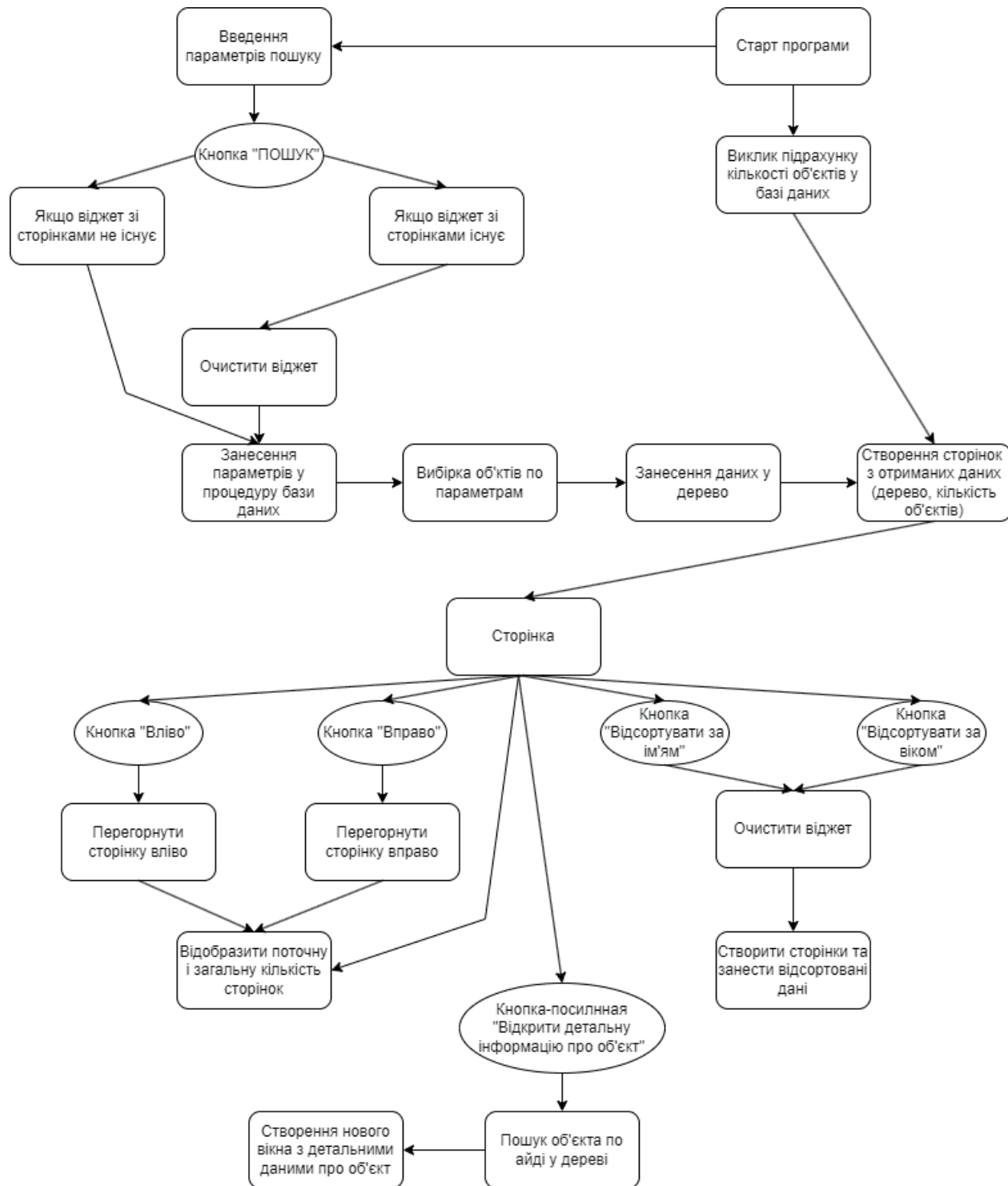


Рис 3.1. Схема ідеї побудови програми

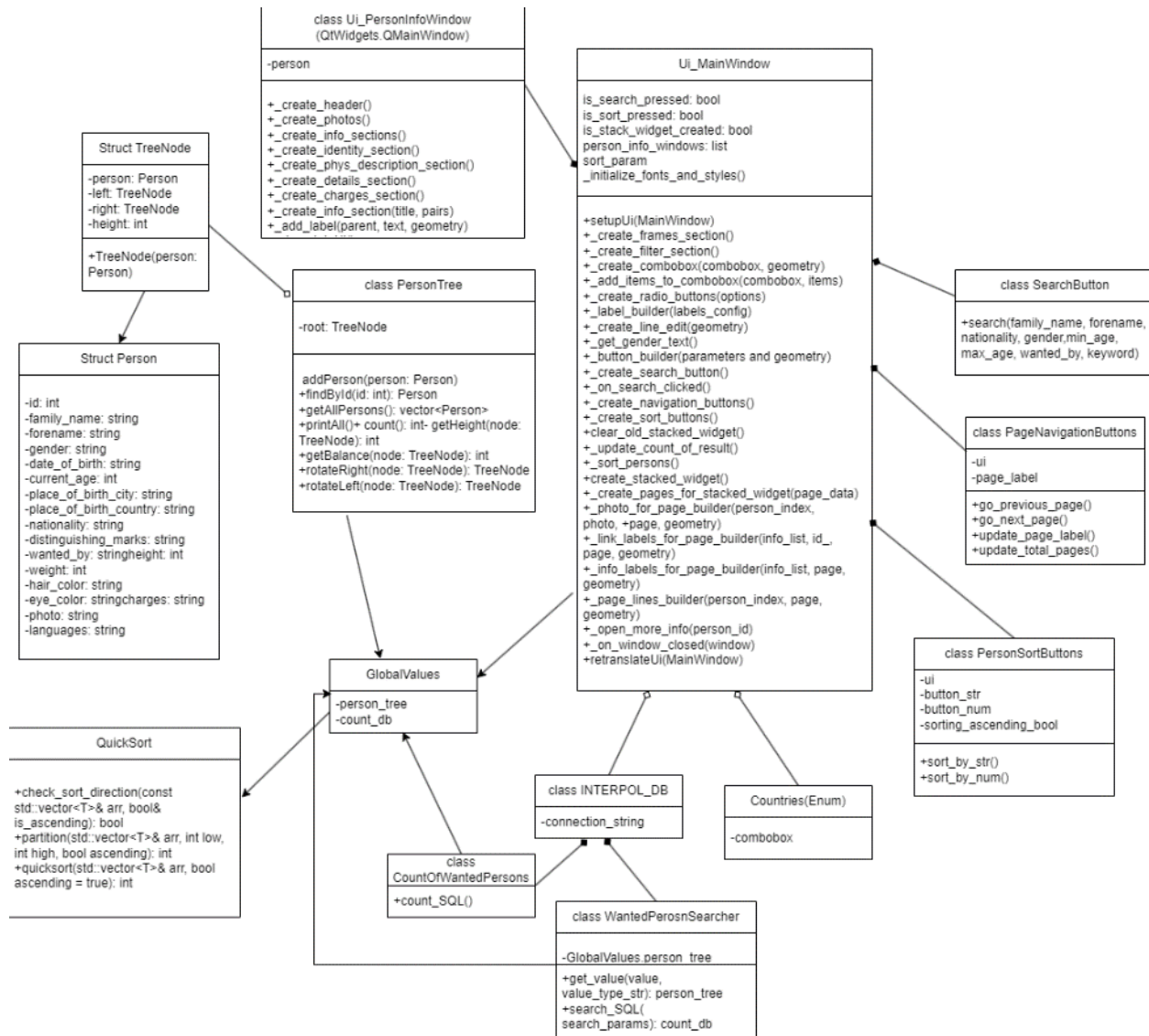


Рис 3.2. UML діаграма

$$C = \left\lceil \frac{N}{M} \right\rceil \quad (3.1)$$

Де

C – кількість сторінок,

N – загальна кількість осіб,

M – максимальна кількість фото на сторінці

$$\text{row} = \left\lfloor \frac{i}{n} \right\rfloor \quad (3.2)$$

Де

row – рядок,

i – індекс елемента у списку,

n – кількість колонок у сітці

$$\text{col} = i \bmod n \quad (3.3)$$

Де

col – колонка,

i – індекс елемента у списку,

n – кількість колонок у сітці,

Mod – залишок від ділення

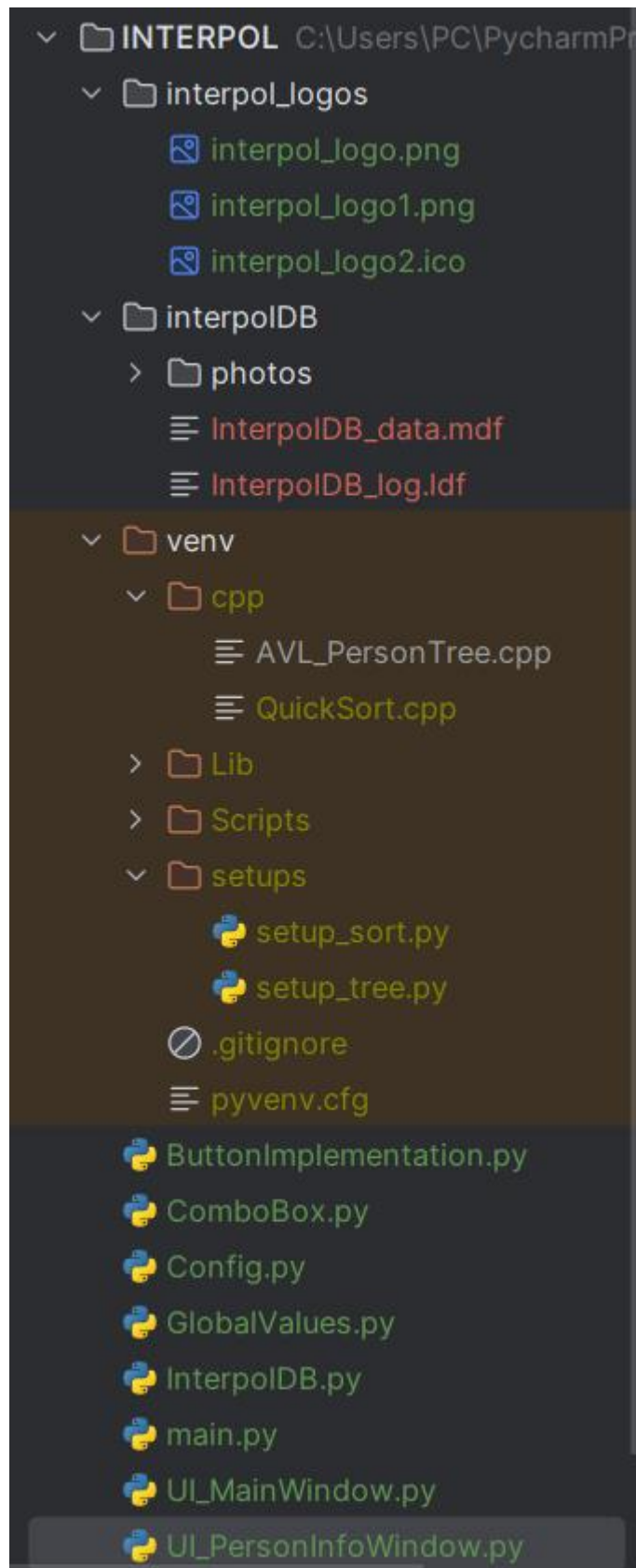


Рис 4.1. Розташування файлів у програмі

INTERPOL

Filter criteria

Family name

Forename

Nationality

Gender

All

Female

Male

Unknown

Current age

0

120

Wanted by

Keyword

SEARCH

Total number of wanted persons in circulation: 190

Search results: 190

PHOTO NOT AVAILABLE

[IVANOV](#)

44 years old

Ukraine

PHOTO NOT AVAILABLE

[SMITH](#)

[JOHN](#)

USA

PHOTO NOT AVAILABLE

[DOE](#)

[JANE](#)

32 years old

Canada

PHOTO NOT AVAILABLE

[MARTA](#)

49 years old

Germany

PHOTO NOT AVAILABLE

[NGUYEN](#)

[AN](#)

36 years old

Vietnam

PHOTO NOT AVAILABLE

[BROWN](#)

24 years old

UK

PHOTO NOT AVAILABLE

[KOVACS](#)

[LASZLO](#)

Hungary

PHOTO NOT AVAILABLE

[GARCIA](#)

[MARIA](#)

29 years old

Spain

←

1/24

→

123

ABC

Рис. 5.1. Вигляд програми

30

INTERPOL

Filter criteria

Family name

GAR

Forename

Nationality

Gender

All

Female

Male

Unknown

Current age

0

120

Wanted by

Keyword

SEARCH

Total number of wanted persons in circulation: 190

Search results: 7

PHOTO NOT AVAILABLE

[GARCIA MARIA](#)

29 years old

Spain

PHOTO NOT AVAILABLE

[GARCIA ISABEL](#)

41 years old

Spain

PHOTO NOT AVAILABLE

[GARCIA LUIS](#)

39 years old

Spain

PHOTO NOT AVAILABLE

[GARCIA CARLOS](#)

39 years old

Spain

PHOTO NOT AVAILABLE

[GARCIA CARLOS](#)

35 years old

Spain

PHOTO NOT AVAILABLE

[GARCIA ELENA](#)

31 years old

Spain

PHOTO NOT AVAILABLE

[GARCIA CARLOS](#)

30 years old

Spain

←

1/1

→

123

ABC

Рис. 5.2. Вибірка за одним параметром

31

INTERPOL

Filter criteria

Family name

GAR

Forename

MA

Nationality

Spain

Gender

All

Female

Male

Unknown

Current age

29

29

Wanted by

Spain

Keyword

SEARCH

Total number of wanted persons in circulation: 190

Search results: 1

PHOTO NOT AVAILABLE

GARCIA

MARIA

29 years old

Spain

←

1/1

→

123

ABC

Рис. 5.3. Вибірка за всіма параметрами

INTERPOL

Filter criteria

Family name

Forename

Nationality

Gender

All

Female

Male

Unknown

Current age

0

120

Wanted by

Keyword

SEARCH

Total number of wanted persons in circulation: 190

Search results: 190

PHOTO NOT AVAILABLE

IVANOV

44 years old

Ukraine

PHOTO NOT AVAILABLE

SMITH

JOHN

USA

PHOTO NOT AVAILABLE

DOE

JANE

32 years old

Canada

PHOTO NOT AVAILABLE

MARTA

49 years old

Germany

PHOTO NOT AVAILABLE

NGUYEN

AN

36 years old

Vietnam

PHOTO NOT AVAILABLE

BROWN

24 years old

UK

PHOTO NOT AVAILABLE

KOVACS

LASZLO

Hungary

PHOTO NOT AVAILABLE

GARCIA

MARIA

29 years old

Spain

←

1/24

→

123

ABC

DOE JANE

DOE, JANE

Wanted byCanada

PHOTO NOT AVAILABLE

Identity particulars

Family name

Doe

Forename

Jane

Gender

Female

Date of birth

22/06/1992 (32 years old)

Place of birth

Canada

Nationality

Canada

Distinguishing marks

Tattoo on left arm

Physical description

Height

165

Weight

55

Colour of hair

Blonde

Colour of eyes

Green

Details

Language(s) spoken

English

Charges

Robbery

32

Рис. 5.4. Взаємодія головного і додаткового вікон

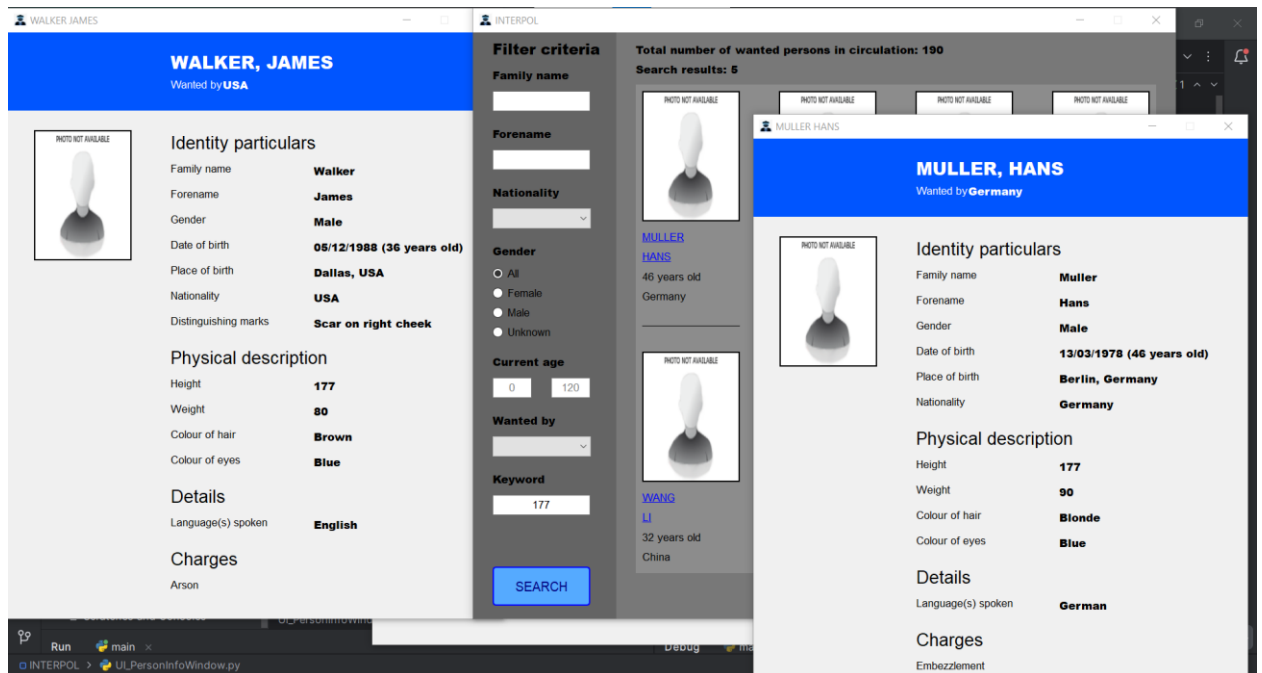


Рис 5.5. Одразу кілька відкритих вікон

INTERPOL

Filter criteria

Family name

Forename

Nationality

Gender

All

Female

Male

Unknown

Current age

0

120

Wanted by

Keyword

SEARCH

Total number of wanted persons in circulation: 190

Search results: 190

PHOTO NOT AVAILABLE

[BROWN HENRY](#)

26 years old

Canada

PHOTO NOT AVAILABLE

[LOPEZ ISABELLA](#)

32 years old

Argentina

[JOHNSON LIAM](#)

38 years old

USA

[WANG XIAO](#)

30 years old

China

[HERNANDEZ PEDRO](#)

44 years old

Spain

PHOTO NOT AVAILABLE

[ROBINSON ETHAN](#)

32 years old

UK

PHOTO NOT AVAILABLE

[TANAKA RYO](#)

34 years old

Japan

PHOTO NOT AVAILABLE

[KHAN SANA](#)

36 years old

Pakistan

←

4/24

→

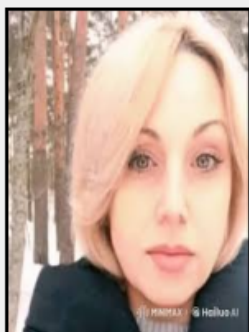
123

ABC

Рис 5.6. Відображення фото

34

Wanted by Spain



Family name	Hernandez
Forename	Pedro
Gender	Male
Date of birth	30/12/1980 (44 years old)
Place of birth	Madrid, Spain
Nationality	Spain

Height	175
Weight	82
Colour of hair	Black
Colour of eyes	Brown

Language(s) spoken **Spanish**

Assault

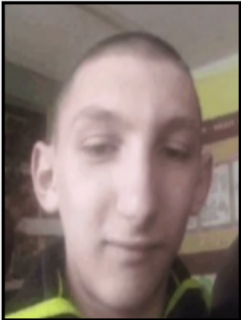
Рис 5.7. Вікно з детальною інформацією

JOHNSON LIAM

— □ ×

JOHNSON, LIAM

Wanted byUSA



Identity particulars

Family name	Johnson
Forename	Liam
Gender	Male
Date of birth	11/01/1986 (38 years old)
Place of birth	Chicago, USA
Nationality	USA
Distinguishing marks	Piercing on left ear

Physical description

Height	178
Weight	76
Colour of hair	Brown
Colour of eyes	Green

Details

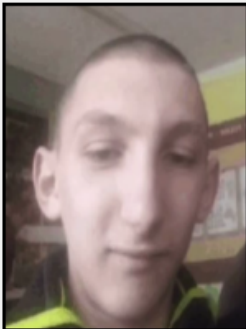
Language(s) spoken	English
--------------------	---------

Charges

Theft

Рис 5.8. Одне фото

Wanted by China



Family name	Wang
Forename	Xiao
Gender	Female
Date of birth	17/03/1994 (30 years old)
Place of birth	Hong Kong, China
Nationality	China

Height	162
Weight	48
Colour of hair	Black
Colour of eyes	Brown

Language(s) spoken **Mandarin, Cantonese**

Money laundering

37

INTERPOL

Filter criteria

Family name

Forename

Nationality

Gender

All

Female

Male

Unknown

Current age

0

120

Wanted by

Keyword

SEARCH

Total number of wanted persons in circulation: 190

Search results: 190

PHOTO NOT AVAILABLE

[GONZALEZ](#)
[CARLOS](#)
49 years old
Mexico

PHOTO NOT AVAILABLE

[MARTA](#)
49 years old
Germany

PHOTO NOT AVAILABLE


[MULLER](#)
[HANS](#)
46 years old
Germany

PHOTO NOT AVAILABLE

[DAVIS](#)
[EVAN](#)
44 years old
USA

PHOTO NOT AVAILABLE

[PETERSON](#)
[MICHAEL](#)
44 years old
Australia



[HERNANDEZ](#)
[PEDRO](#)
44 years old
Spain

PHOTO NOT AVAILABLE

[KUMAR](#)
[RAJESH](#)
44 years old
India

PHOTO NOT AVAILABLE

[IVANOV](#)
44 years old
Ukraine

←

1/24


→

321

ABC

Рис 5.10. Сортивання за віком від старшого до молодшого


38

 YAMAMOTO HIROSHI

— □ ×

YAMAMOTO, HIROSHI
Wanted by **Japan**

PHOTO NOT AVAILABLE



Identity particulars

Family name	Yamamoto
Forename	Hiroshi
Gender	Male
Date of birth	05/01/1982 (42 years old)
Place of birth	Tokyo, Japan
Nationality	Japan
Distinguishing marks	Large mole on chin

Physical description

Height	175
Colour of hair	Black

Details

Language(s) spoken	Japanese
--------------------	-----------------

Рис. 5.14. Форматування вікна з відсутнім блоком інформації

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,
                                                     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,
                                                     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 стопінки 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.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);
}

```