

Politechnika Warszawska  
Wydział Elektroniki i Technik Informacyjnych

Projekt z przedmiotu  
Systemy Informatyczne w Medycynie

## **STAT – Eksport informacji z bazy danych badań medycyny nuklearnej w formacie XML**

Zespół 4

Ika Gut 336356

Marta Kwiatkowska 318619

Weronika Pędzimąż 318623

Warszawa, 2025

## Spis treści:

1. Wprowadzenie.....	3
1.1. Opis projektu.....	3
1.2. Cel projektu.....	3
1.3. Model danych.....	3
1.4. Technologie i architektura.....	4
2. Struktura projektu.....	4
2.1. Struktura katalogów.....	4
3. Opis bazy danych.....	5
3.1. Tabela pacjentów.....	5
3.2. Tabela badań studies.....	5
3.3. Tabela serii.....	6
3.4. Tabela obrazów.....	6
3.5. Klucze główne i obce.....	7
4. Działanie systemu.....	8
4.1. Import metadanych DICOM.....	8
4.2. Uruchomienie aplikacji i nawiązanie połączenia.....	8
4.3. Przeglądanie i filtrowanie danych.....	9
4.4. Eksport i walidacja pliku XML.....	10
5. Implementacja - wybrane pliki.....	11
5.1. Baza Danych w języku SQL.....	11
5.2. Eksport danych do pliku XML.....	12

# 1. Wprowadzenie

## 1.1. Opis projektu

Projekt STAT to narzędzie stworzone do prostego i wygodnego eksportu oraz analizy danych z bazy obrazów scyntygraficznych stosowanych w medycynie nuklearnej. Aplikacja wykorzystuje hierarchiczny model DICOM, który uwzględnia cztery poziomy danych: pacjent, badanie, seria oraz obraz. Dzięki intuicyjnemu interfejsowi graficznemu opartemu na Qt, użytkownik może łatwo nawiązać połączenie z bazą przez ODBC, ustawić filtry (takie jak zakres dat, typ badania, modalność, wiek czy płeć) i podejrzeć wyniki przed eksportem. Finalne pliki XML odzwierciedlają pełną strukturę danych i zawierają sekcję statystyk z liczbą pacjentów, badań, faz, obrazów oraz sumarycznym rozmiarem danych. Projekt zapewnia także walidację wygenerowanych plików XML względem struktury XSD, w celu zapewnienia sprawdzenia poprawności wygenerowanych dokumentów.

## 1.2. Cel projektu

Głównym celem STAT jest umożliwienie szybkiego filtrowania i przeglądania rekordów z bazy danych oraz zapisania wyselekcjonowanych informacji w postaci spójnego pliku XML. Zależało nam na tym, by cała procedura – od zakładania nowej bazy, przez import metadanych DICOM, aż po wygenerowanie i automatyczną walidację dokumentu – była maksymalnie zautomatyzowana i odporna na błędy użytkownika.

## 1.3. Model danych

Projekt oparty jest na czteropoziomowym modelu danych zgodnym ze standardem DICOM, odwzorowanym w strukturze relacyjnej bazy PostgreSQL. Na najwyższym poziomie znajdują się dane pacjenta, takie jak jego unikalny identyfikator, imię i nazwisko, wiek oraz płeć. Każdy pacjent może mieć przypisane jedno lub więcej badań, które zawierają informacje takie jak swój unikalny identyfikator badania, data i godzina jego wykonania, opis, nazwisko lekarza kierującego, numer zlecenia oraz adres placówki wykonującej. Badanie może składać się z jednej lub więcej serii obrazów, z których każda posiada własny unikalny identyfikator, typ akwizycji, numer porządkowy oraz ewentualny dodatkowy opis. Każda seria składa się z pojedynczych obrazów, które opisane są przez rozdzielczość (liczbę wierszy i kolumn), liczbę bitów przypisanych do piksela, numer instancji obrazu oraz jego typ. Dane zawarte w tabeli obrazów są wykorzystywane podczas eksportu do formatu XML, co umożliwia pełne odwzorowanie hierarchii DICOM oraz przeprowadzenie szczegółowej analizy poszczególnych klatek obrazowych.

## 1.4. Technologie i architektura

Projekt został zrealizowany w języku C++, z wykorzystaniem biblioteki Qt jako fundamentu warstwy graficznej (GUI) oraz narzędzi do walidacji XML. Komunikacja z relacyjną bazą danych PostgreSQL odbywa się za pomocą interfejsu ODBC. Do zarządzania kompilacją wykorzystano system CMake. W celu ułatwienia wdrożeń środowisko zostało zcontainerowane przy pomocy Docker oraz Docker Compose. Aplikacja i baza danych działają w osobnych kontenerach, a ich uruchomienie sprowadza się do jednego polecenia. Walidacja plików XML oparta jest na wbudowanych mechanizmach Qt, co pozwala na szybkie i efektywne sprawdzanie poprawności danych względem schematów XSD bez konieczności stosowania zewnętrznych narzędzi.

## 2. Struktura projektu

### 2.1. Struktura katalogów

Projekt ma modułową strukturę – główny katalog zawiera konfigurację kontenerów Docker, a cały kod źródłowy aplikacji wraz z plikami SQL, XSD i GUI znajduje się w podkatalogu *sim/*. Dane wyjściowe (np. pliki XML) zapisywane są w katalogu *exports/*.

```
sim/
├── docker-compose.yml      # konfiguracja kontenerów
├── Dockerfile              # budowa obrazu aplikacji
├── exports/               # katalog na wygenerowane pliki XML
├──
├── sim/                   # kod źródłowy aplikacji
│   ├── CMakeLists.txt     # konfiguracja budowania
│   ├── *.cpp, *.h, *.ui   # pliki źródłowe C++ i Qt
│   └── export*.sql / *.xsd # pliki do eksportu i walidacji
├── sim_db.sql             # inicjalizacja danych PostgreSQL
└── README.md              # instrukcja uruchomienia projektu
```

### 3. Opis bazy danych

W projekcie zastosowano relacyjną bazę danych PostgreSQL zaprojektowaną tak, aby odzwierciedlać czteropoziomą hierarchię obiektów DICOM. Struktura składa się z czterech głównych tabel: patients, studies, series oraz images. Każda z nich reprezentuje odrębny poziom modelu świata rzeczywistego – od danych pacjenta, przez badania i serie, aż po poszczególne obrazy.

#### 3.1. Tabela pacjentów

Tabela pacjentów “patients” przechowuje podstawowe dane każdego badanego pacjenta. Każdy wiersz oznacza jednego pacjenta i posiada następujące kolumny:

- id (SERIAL, PRIMARY KEY) - Automatycznie inkrementowany wewnętrzny identyfikator rekordu.
- patient\_id (TEXT, NOT NULL, UNIQUE) - Unikalny identyfikator pacjenta pochodzący z systemu DICOM.
- name (TEXT) - Imię i nazwisko pacjenta
- age (INTEGER) - Wiek pacjenta wyrażony w latach.
- sex (CHAR(1)) - Płeć pacjenta zgodnie ze standardem DICOM (“M”, “F” lub “O”).

#### 3.2. Tabela badań studies

Na poziomie badań (“studies”) gromadzone są dane dotyczące poszczególnych badań wykonanych u pacjenta. Struktura tej tabeli jest następująca:

- id (SERIAL, PRIMARY KEY)
- patient\_id (INTEGER, NOT NULL) - Klucz obcy wskazujący na kolumnę patients.id; definiuje relację jeden-do-wielu (jeden pacjent może mieć wiele badań).
- study\_uid (TEXT, NOT NULL, UNIQUE) - Unikalny identyfikator badania (DICOM Study Instance UID).
- study\_date (DATE) - Data przeprowadzenia badania.
- study\_time (TIME) – godzina badania.
- description (TEXT) – opis badania.
- referring\_physician (TEXT) – lekarz kierujący.
- accession\_number (TEXT) – numer zlecenia.
- institution\_adress (TEXT) – adres instytucji.

Wszystkie wiersze w studies muszą być przypisane do odpowiedniego pacjenta.

### 3.3. Tabela serii

Tabela serii "series" odwzorowuje kolejne serie danego badania. Pola tej tabeli to:

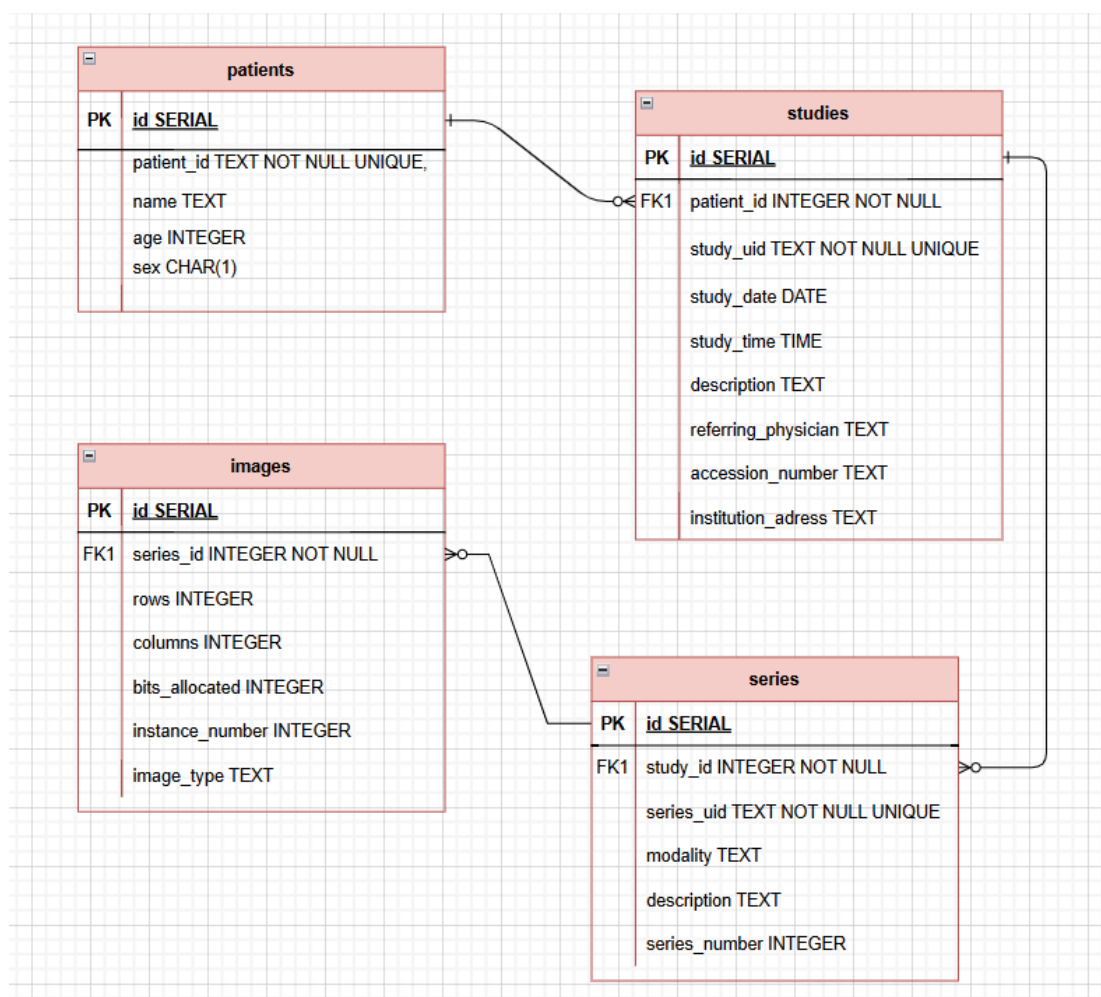
- id (SERIAL, PRIMARY KEY)
- patient\_id (INTEGER, NOT NULL) – klucz obcy do patients.id.
- study\_uid (TEXT, NOT NULL, UNIQUE) – unikalny identyfikator DICOM.
- study\_date (DATE) – data badania.
- study\_time (TIME) – godzina badania.
- description (TEXT) – opis badania.
- referring\_physician (TEXT) – lekarz kierujący.
- accession\_number (TEXT) – numer zlecenia.
- institution\_adress (TEXT) – adres instytucji.

### 3.4. Tabela obrazów

Choć w module eksportu XML tabela images nie jest wykorzystywana do zapisu szczegółów plików DICOM, jej obecność pozwala w przyszłości rozszerzyć aplikację o analizę każdego obrazu osobno. Zawiera ona:

- id (SERIAL, PRIMARY KEY)
- series\_id (INTEGER, NOT NULL) – klucz obcy do series.id.
- rows (INTEGER) – wysokość obrazu (w pikselach).
- columns (INTEGER) – szerokość obrazu (w pikselach).
- bits\_allocated (INTEGER) – liczba bitów na piksel.
- instance\_number (INTEGER) – numer instancji w serii.
- image\_type (TEXT) – typ obrazu.

Poniżej, na Rysunku 1 została przedstawiona struktura bazy danych.



Rysunek 1 - Struktura bazy danych wykorzystana w projekcie STAT

### 3.5. Klucze główne i obce

W każdej z głównych tabel — *patients*, *studies*, *series* i *images* — kolumna *id* pełni rolę klucza głównego, gwarantując unikalność i wydajną identyfikację rekordów. Relacje między tabelami realizowane są przy pomocy kluczy obcych:

- W tabeli *studies* kolumna *patient\_id* odwołuje się do *patients.id*, tworząc związek jeden-do-wielu (jeden pacjent może mieć wiele badań).
- W tabeli *series* kolumna *study\_id* odnosi się do *series.id*, co pozwala na grupowanie wielu serii w ramach pojedynczego badania.
- W tabeli *images* kolumna *series\_id* wskazuje na *series.id*, umożliwiając przypisanie każdego obrazu do konkretnej serii.

Dzięki temu możliwe jest odwzorowanie zależności:

Pacjent → Badania → Serie → Obrazy, z gwarancją spójności danych.

## 4. Działanie systemu

### 4.1. Import metadanych DICOM

Dane umieszczone w bazie danych PostgreSQL powstały w wykonanym wcześniej procesie importu przy pomocy programu *dicom\_to\_DB.cpp*. Program ten analizuje wskazany katalog z plikami DICOM, parsuje nagłówki i wykonuje odpowiednie zapytania SQL, aby uzupełnić cztery powiązane tabele: *patients*, *studies*, *series* i *images*. W ten sposób struktura bazy danych wiernie odwzorowuje hierarchię obiektów DICOM, uwzględniając m.in. identyfikatory i inne istotne atrybuty diagnostyczne.

Plik *dicom\_to\_DB.cpp* został zaprojektowany tak, by obsługiwać różne przypadki niekompletnych danych: w sytuacji, gdy pole *PatientID* jest puste, generowany jest unikalny identyfikator tymczasowy, aby zachować spójność rekordów. Dodatkowo, jeśli wiek pacjenta nie został bezpośrednio podany, program automatycznie oblicza go na podstawie daty urodzenia (*PatientBirthDate*). Mechanizm ten chroni również przed duplikacją danych – przed wstawieniem nowego rekordu sprawdzane są unikalne identyfikatory DICOM (*StudyInstanceUID*, *SeriesInstanceUID*).

Wypełniona w ten sposób baza danych została następnie zapisana do pliku *sim\_db.sql*, który jest automatycznie ładowany przez kontener *sim-db* podczas startu systemu. Oznacza to, że wszystkie dane są gotowe do użycia natychmiast po uruchomieniu projektu, bez potrzeby ponownego importu.

### 4.2. Uruchomienie aplikacji i nawiązanie połączenia

Po zainicjowaniu bazy danych uruchamiana jest aplikacja Qt, która działa w kontenerze *sim-app*. Proces ten odbywa się automatycznie podczas budowy i startu środowiska. Aplikacja wykorzystuje *ODBC* do połączenia z serwerem PostgreSQL działającym w kontenerze *sim-db*. Parametry połączenia (nazwa użytkownika, hasło, port i nazwa bazy) są przekazywane jako zmienne środowiskowe zdefiniowane w pliku *docker-compose.yml*.

Jeśli połączenie powiedzie się, aplikacja przechodzi do trybu interaktywnego, umożliwiając m.in. eksport danych do pliku XML oraz ich walidację względem schematu XSD. W przypadku niepowodzenia użytkownik zostaje poinformowany o błędzie w dedykowanym oknie statusu.

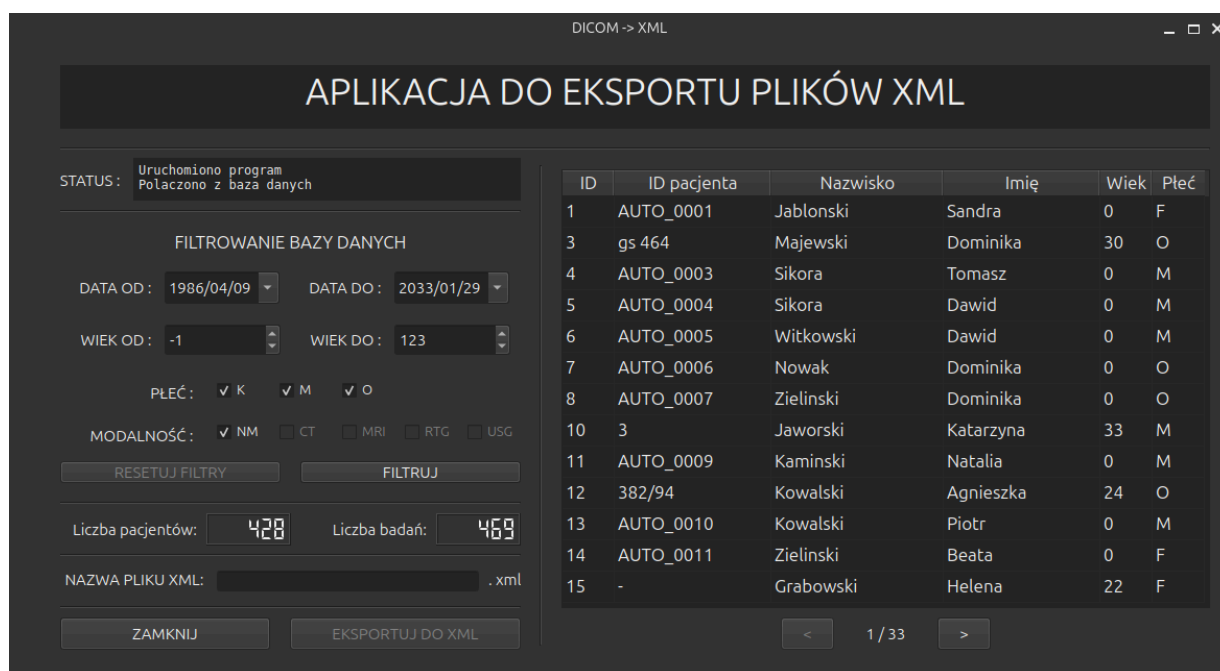
### 4.3. Przeglądanie i filtrowanie danych

Główne okno podzielone jest na dwie części, funkcyjną i wyświetlania.

Cześć wyświetlania składa się z widżetu tabeli oraz przycisków funkcyjnych do wyświetlania kolejnego (bądź poprzedniego) zestawu rekordów. Dostępność przycisków jest obsługiwana proceduralnie, tak jak wyznaczanie liczby dostępnych stron w widżecie.

Część funkcyjna składa się z modułu filtracji i modułu eksportu oraz wyświetlaczy statystycznych i okna informacji zwrotnej o statusie programu. Okno to wyświetla komunikaty o powodzeniu bądź błędach w działaniu funkcji programu. Moduł filtracji jest najbardziej zaawansowany i pozwala na dokładne dobranie parametrów tak, aby wyciągnąć z bazy jedynie interesujące nas dane. Po połączeniu z bazą, widżety opisane jako "data od", "data do" oraz "wiek od" i "wiek do" przyjmują wartości odpowiadające odpowiednio największym i najmniejszym występującym w bazie danych. Te wartości zostaną przypisane w programie do zmiennych bazowej filtracji. Okienka przy "płeć" oraz "modalność" są odblokowywane tylko jeśli w bazie występuje chociaż jeden rekord z określonym tagiem. Te tagi, które się pojawiają, są również automatycznie zaznaczane dla lepszego zrozumienia ich działania tj. gdy checkbox jest zaznaczony, dany tag jest dodany do warunków filtracji. Na końcu modułu filtracji znajdują się dwa przyciski, jeden do zatwierdzania i procesowania filtracji ("filtruj"), drugi do resetowania wszystkich powyższych pól do stanu bazowego ("resetuj filtry"). Aby wygenerować plik nawet przy korzystaniu z ustawień filtracji bazowej należy nacisnąć przycisk "filtruj" gdyż odblokowuje on przycisk eksportu.

Dwa wyświetlacze numeryczne pod modułem filtracji służą do poglądowego przedstawienia liczny indywidualnych pacjentów spełniających warunki filtracji oraz ilości badań im odpowiadających. Na samym dole okna programu znajduje się moduł eksportu na który składa się okno nazwy, w którym użytkownik musi wpisać żadaną nazwę dla pliku .xml. Przy próbie wykonania eksportu bez podania nazwy (lub użyciu niedopuszczalnych symboli lub zbyt dużej ich ilości) okno statusu zwróci odpowiedni komunikat. Po spełnieniu warunków, wciśnięcie przycisku "Eksportuj do XML" spowoduje wygenerowanie pliku .xml o podanej nazwie, a o sukcesie lub niepowodzeniu poinformuje w oknie statusu. Program można zakończyć w dowolnym momencie korzystając z przycisku "Zamknij".



Rysunek 2 - Interfejs programu STAT z dostępnymi funkcjami do filtracji, generowania i walidacji plików XML.

#### 4.4. Eksport i walidacja pliku XML

Po wygenerowaniu pliku XML przez aplikację, użytkownik wybiera nazwę pliku i uruchamia eksport. Aplikacja tworzy dokument XML z zachowaniem pełnej hierarchii danych oraz automatycznie dołącza sekcję zawierającą statystyki wyeksportowanych danych — liczbę pacjentów, badań, serii oraz obrazów. Gotowy plik XML jest zapisywany na dysku. Następnie, do walidacji poprawności struktury i zawartości wygenerowanego pliku XML względem schematu XSD, wykorzystywany jest osobny moduł napisany w C++ z użyciem biblioteki libxml2. Ten moduł uruchamia proces walidacji, tworząc kontekst parsera i kontekst walidacji na podstawie schematu XSD, a następnie sprawdza zgodność pliku XML z tym schematem. W przypadku wykrycia błędów walidacji użytkownik otrzymuje szczegółowe informacje o niezgodnościach, co pozwala na szybkie wykrycie i poprawę problemów. Dzięki temu cały proces — od importu danych, przez generowanie pliku XML wraz ze statystykami, aż po walidację — jest spójny i w pełni zautomatyzowany, a wynikowy plik jest zgodny z wymaganym schematem XSD.

## 5. Implementacja - wybrane pliki

### 5.1. Baza Danych w języku SQL

```
DROP TABLE IF EXISTS images;  
DROP TABLE IF EXISTS series;  
DROP TABLE IF EXISTS studies;  
DROP TABLE IF EXISTS patients;
```

```
CREATE TABLE patients (  
    id SERIAL PRIMARY KEY,  
    patient_id TEXT NOT NULL UNIQUE,  
    name TEXT,  
    age INTEGER,  
    sex CHAR(1)  
);
```

```
CREATE TABLE studies (  
    id SERIAL PRIMARY KEY,  
    patient_id INTEGER NOT NULL REFERENCES patients(id) ON DELETE CASCADE,  
    study_uid TEXT NOT NULL UNIQUE,  
    study_date DATE,  
    study_time TIME,  
    description TEXT,  
    referring_physician TEXT,  
    accession_number TEXT,  
    institution_name TEXT,  
    institution_address TEXT  
);
```

```
CREATE TABLE series (  
    id SERIAL PRIMARY KEY,  
    study_id INTEGER NOT NULL REFERENCES studies(id) ON DELETE CASCADE,  
    series_uid TEXT NOT NULL UNIQUE,  
    modality TEXT,  
    description TEXT,  
    series_number INTEGER  
);
```

```
CREATE TABLE images (  
    id SERIAL PRIMARY KEY,  
    series_id INTEGER NOT NULL REFERENCES series(id) ON DELETE CASCADE,  
    rows INTEGER,  
    columns INTEGER,  
    bits_allocated INTEGER,  
    instance_number INTEGER,  
    image_type TEXT  
);
```

```
CREATE INDEX idx_studies_patient_id ON studies(patient_id);  
CREATE INDEX idx_series_study_id ON series(study_id);  
CREATE INDEX idx_images_series_id ON images(series_id);
```

## 5.2. Eksport danych do pliku XML

Przy pomocy kodu poniżej generowany jest plik XML.

```
SELECT xmlelement(name export,
xmlelement(name statistics,
xmlforest(
  (SELECT count(*) FROM patients) AS patientCount,
  (SELECT count(*) FROM studies) AS studyCount,
  (SELECT count(*) FROM series) AS seriesCount,
  (SELECT count(*) FROM images) AS imageCount
)
),
xmlelement(name patients,
xmlagg(
  xmlelement(name patient,
    xmlattributes(p.id as id, p.patient_id as patient_id),
    xmlforest(p.name as name, p.age as age, p.sex as sex),
    (
      SELECT xmlagg(
        xmlelement(name study,
          xmlattributes(s.study_uid as study_uid),
          xmlforest(
            s.study_date as date,
            s.study_time as time,
            s.description as description,
            s.referring_physician as referring_physician,
            s.accession_number as accession_number,
            s.institution_name as institution_name,
            s.institution_address as institution_address
          ),
        (
          SELECT xmlagg(
            xmlelement(name series,
              xmlattributes(se.series_uid as series_uid),
              xmlforest(
                se.modality as modality,
                se.description as description,
                se.series_number as series_number
              ),
            (
              SELECT xmlagg(
                xmlelement(name image,
                  xmlforest(
                    i.rows,
                    i.columns,
                    i.bits_allocated,
                    i.instance_number,
                    i.image_type
                  )
                )
              )
            )
          FROM images i
          WHERE i.series_id = se.id
        )
      )
    )
  )
  FROM series se
```

```

        WHERE se.study_id = s.id /* FILTER_SERIES */
    )
)
)
FROM studies s
WHERE s.patient_id = p.id /* FILTER_STUDIES */
)
)
)
)
)
FROM patients p
WHERE 1=1 /* FILTER_PATIENTS */;

```

Poniżej przedstawiony został przykład pierwszego pacjenta z bazy danych, gdzie widać zachowaną strukturę pliku DICOM oraz pola z wyliczonymi danymi statystycznymi:

```

<export>
<statistics>
  <patientcount>623</patientcount>
  <studycount>469</studycount>
  <seriescount>769</seriescount>
  <imagecount>968</imagecount>
</statistics>
<patients>
  <patient id="1" patient_id="AUTO_0001">
    <name>Jablonski^Sandra</name>
    <age>0</age>
    <sex>F</sex>
    <study study_uid="1.2.826.0.1.3680043.10.1164.0.17.1.1.2.15">
      <date>1997-01-01</date>
      <time>02:15:00</time>
      <description>Bramka EKG</description>
      <referring_physician></referring_physician>
      <accession_number></accession_number>
      <institution_name>Warsaw University of Technology</institution_name>
      <institution_address>Plac Politechniki 1, 00-661 Warszawa, POLAND</institution_address>
      <series series_uid="1.2.826.0.1.3680043.10.1164.0.17.1.1.2.15.2.0">
        <modality>NM</modality>
        <description>APO Bramka EKG Serce</description>
        <series_number>0</series_number>
        <image>
          <rows>128</rows>
          <columns>128</columns>
          <bits_allocated>16</bits_allocated>
          <instance_number>0</instance_number>
          <image_type>DERIVED</image_type>
        </image>
        <image>
          <rows>128</rows>
          <columns>128</columns>
          <bits_allocated>16</bits_allocated>
          <instance_number>0</instance_number>
          <image_type>DERIVED</image_type>

```

```
</image>
<image>
  <rows>128</rows>
  <columns>128</columns>
  <bits_allocated>16</bits_allocated>
  <instance_number>0</instance_number>
  <image_type>DERIVED</image_type>
</image>
<image>
  <rows>128</rows>
  <columns>128</columns>
  <bits_allocated>16</bits_allocated>
  <instance_number>0</instance_number>
  <image_type>DERIVED</image_type>
</image>
</series>
</study>
</patient>
</patients>
</export>
```