POLITECHNIKA POZNAŃSKA

WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

Dokumentacja techniczna

HURTOWNIA DANYCH OBRAZOWYCH

Grupa
Aleksander Lisiecki, 136583
Jędrzej Smolarkiewicz, 136622
Grzegorz Otworowski, 136605
Szymon Pawlak, 136610

Prowadzący
dr hab. inż Robert Wrembel - Politechnika Poznańska
dr Michał Monkiewicz - Centrum Medyczne HCP

POZNAŃ 2021

Spis treści

1 Wstęp

Jednymi z największych problemów baz danych szpitali jest sposób i łatwość dostępu do plików. Prosty dostęp do plików jest niezbędny do szybkiego i sprawnego działania służby zdrowia. Szpitalne systemy są trudne w utrzymaniu ze względu na ich wielkość, poziom skomplikowania oraz niewyspecjalizowaną do ich obsługi kadrę. Rozwiązaniem tego problemu jest odpowiednie zaprojektowanie hurtowni danych oraz napisanie oprogramowania do jej obsługi.

Oprogramowanie, które jest wykorzystywane w szpitalach wiąże się z restrykcyjnymi umowami. Ograniczają one w znaczny sposób możliwość modyfikacji dostarczonej aplikacji. Z uwagi na takie hermetyczne środowisko problematyczna staje się próba zmiany sposobu działania hurtowni danych lub operatora takiej hurtowni. Wszelkie nieujęte w umowie funkcjonalności mogą zostać dodane jedynie przez twórcę oprogramowania, który zazwyczaj żąda za to wysokiej opłaty.

Celem projektu było zaproponowanie i opracowanie hurtowni danych obrazowych oraz aplikacji do zarządzania jej zawartością dla Centrum Medycznego HCP w Poznaniu. System powinien przechowywać obrazy w formacie DICOM pochodzące ze stacji radiologicznych szpitala oraz umożliwiać użytkownikowi ich przeglądanie i zarządzanie w przystępny sposób. Obrazy mogą zostać przypisane do danego pacjenta, badania oraz serii oraz można do nich dodawać opis medyczny. Podstawowym problemem, który należy rozwiązać, jest ujednolicenie systemu przechowywania danych obrazowych Centrum Medycznego HCP.

2 Wykorzystane technologie

W celu utworzenia prototypu aplikacji obsługującej hurtownię danych wybrana została aplikacja internetowa oparta na platformie Django w języku Python. Pozwala ona na dynamiczne generowanie bloków i proste przekazywanie wartości do wykorzystywanych szablonów.

Jedną z wykorzystanych funkcjonalności Django był domyślnie tworzony panel administracyjny służący do dodawania, edycji oraz usuwania rekordów z bazy danych. Platforma ta umożliwiła także na dodanie własnych formularzy zwanych modelami, które odpowiadają tabelom w bazie danych. Formularze te wykorzystywane są do zapisu w bazie danych odpowiednich obiektów.

Do odczytu danych z plików DICOM wykorzystana została biblioteka języka Python o nazwie *pydicom*. Poza tym, biblioteka ta umożliwia także modyfikację danych w plikach DICOM, wyświetlanie zawartych w nich obrazów, a także tworzenie własnych plików od podstaw.

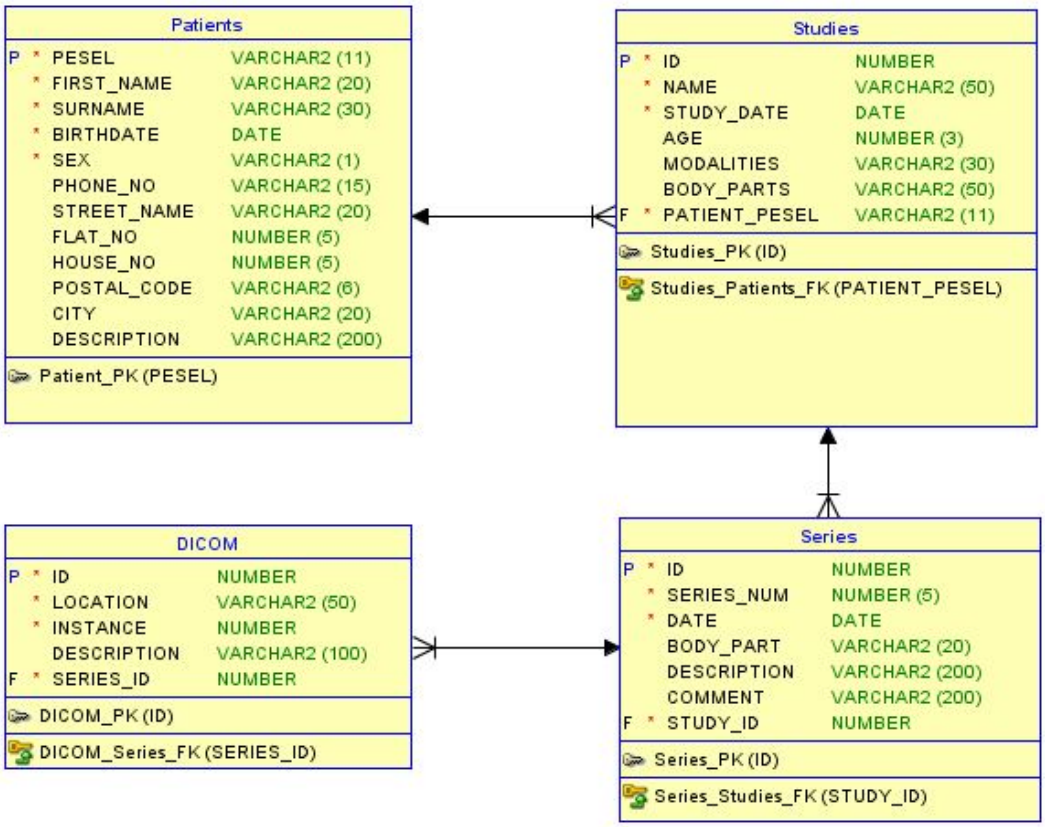
W celu utworzenia widoku strony wykorzystano bibliotekę Materialize języka CSS, której głównymi zaletami są łatwość implementacji oraz prostota elementów. Pozwala ona na wykorzystanie i modyfikację gotowych wzorców i widoków w celu nadania aplikacji przejrzystości oraz estetyki.

3 Implementacja rozwiązania

Cała implementacja aplikacji wraz z danymi testowymi znajduje się na platformie GitHub pod adresem `https://github.com/aleksanderLisiecki/projektSzpital`. Kompilacja projektu i uruchomienie serwera z aplikacją następuje poprzez wywołanie komendy *python manage.py runserver* w wierszu poleceń.

3.1 Baza danych

Działanie aplikacji opiera się na relacyjnej bazie danych *SQLite*. Zawartość bazy odzwierciedla najważniejsze informacje zawarte w plikach DICOM. Jej schemat został przedstawiony na Rysunku 3.1. Stworzone zostały następujące tabele (odpowiednikami których w Django są modele):



Rysunek 3.1. Schemat bazy danych.

- *Patients* - zawiera informacje o pacjencie:
  - *PESEL* - numer PESEL pacjenta,
  - *FIRST_NAME* - imię pacjenta,
  - *SURNAME* - nazwisko pacjenta,
  - *BIRTHDATE* - data urodzenia pacjenta,
  - *SEX* - płeć pacjenta,
  - *PHONE_NO* - numer telefonu pacjenta,
  - *STREET_NAME* - nazwa ulicy miejsca zamieszkania pacjenta,
  - *FLAT_NO* - numer domu miejsca zamieszkania pacjenta,
  - *HOUSE_NO* - numer mieszkania miejsca zamieszkania pacjenta,
  - *POSTAL_CODE* - kod pocztowy miejsca zamieszkania pacjenta,
  - *CITY* - miasto zamieszkania pacjenta,
  - *DESCRIPTION* - dodatkowy opis pacjenta.
- *Studies* - zawiera informacje o badaniu. Każde badanie przypisane jest do odpowiedniego pacjenta.
  - *ID* - numer ID badania,

  - *NAME* - nazwa badania,
  - *STUDY_DATE* - data przeprowadzenia badania,
  - *AGE* - wiek pacjenta w trakcie badania,
  - *MODALITIES* - zastosowane sposoby/procedury leczenia,
  - *BODY_PARTS* - badane części ciała,
  - *PATIENT_PESEL* - numer PESEL pacjenta.

- *Series* - zawiera informacje o serii badania. Każda seria przypisana jest do odpowiedniego badania.
  - *ID* - numer ID serii,
  - *SERIES_NUM* - numer serii w badaniu,
  - *DATE* - data przeprowadzenia serii badania,
  - *BODY_PART* - część ciała badana w serii,
  - *DESCRIPTION* - opis serii,
  - *COMMENT* - komentarz do serii,
  - *STUDY_ID* - numer ID badania.

- *DICOM* - zawiera informacje o pliku DICOM. Każdy plik DICOM przypisany jest do odpowiedniej serii badania.
  - *ID* - numer ID zdjęcia DICOM,
  - *LOCATION* - miejsce przechowywania pliku na dysku,
  - *INSTANCE* - instancja zdjęcia w serii,
  - *DESCRIPTION* - opis zdjęcia DICOM,
  - *SERIES_ID* - numer ID serii badania.

Implementacja każdego modelu to osobna klasa dziedzicząca po klasie *models.Model* dostarczonej wraz z platformą Django. Na Przykładzie 3.1 została pokazana implementacja jednego z modeli - *Studies*. Pozostałe modele zostały skonstruowane analogicznie.

Przykład 3.1

Poniższy kod implementuje model *Studies*. Wpierw definiowana jest liczba mnoga nazwy modelu, ponieważ Django automatycznie dodaje końcówkę 's' na końcu wyrazu. Następnie zdefiniowane zostały poszczególne pola modelu. Na końcu znajduje się definicja funkcji, która zwraca obiekt jako ciąg znaków. Wykorzystywane jest to m.in. w panelu administracyjnym.

```
1  class Studies(models.Model):
2      class Meta:
3          verbose_name_plural = "Studies"
4
5      name = models.CharField(max_length=50)
6      study_date = models.DateField()
7      age = models.IntegerField(blank=True)
8      modalities = models.CharField(max_length=30, blank=True)
9      body_parts = models.CharField(max_length=50, blank=True)
10     patient = models.ForeignKey(Patient, on_delete=models.CASCADE)
11
```

```
12      def __str__(self):
13          return self.patient.__str__() + '/' + self.name
```

Rekordy bazy danych mogą być dodawane, usuwane i modyfikowane za pomocą domyślnego panelu administracyjnego dostarczanego przez platformę Django.

Dane z modeli pobierane są przez pliki HTML za pomocą stosownej składni tak, aby w rezultacie zostać wyświetlonymi na właściwych stronach aplikacji internetowej.

3.2 Budowa strony internetowej

Strona internetowa składa się z części użytkowej służącej do przeglądania danych oraz części administracyjnej służącej do ręcznej obsługi danych z podłączonej bazy danych.

Wszystkie widoki stron były obsługiwane przez element *views* wchodzący w skład struktury Django, który dla odpowiedniej ścieżki URL zwracał widok wraz z danymi zaczerpniętymi z modeli. W razie potrzeby dane były dodatkowo modyfikowane na tym etapie. Każdy widok posiadał swój szablon zapisany w pliku HTML, który wedle potrzeby był importowany do widoku głównego z odpowiednimi danymi. Przykład 3.2 ukazuje ten mechanizm w praktyce.

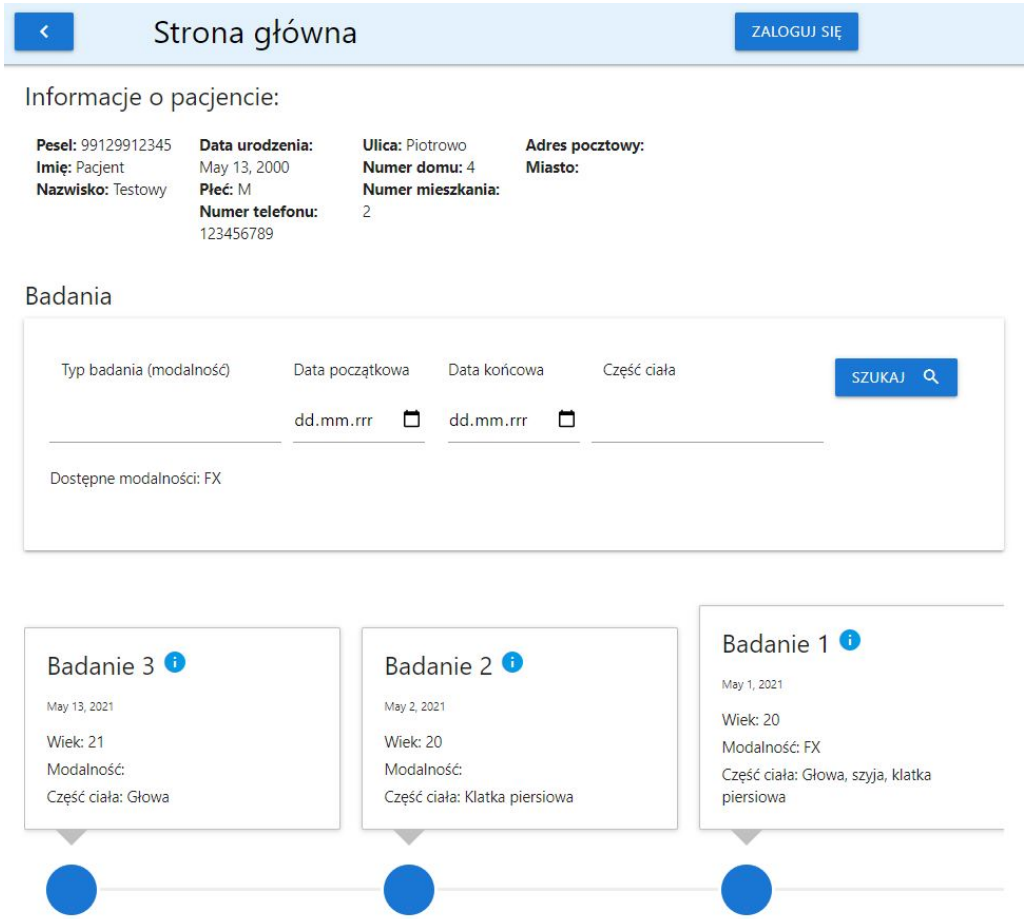
Układ strony tworzą: górny pasek nawigujący zawierający przycisk *Wstecz*, odnośnik do strony głównej (Rysunek 3.8) zawierającej listę pacjentów, oraz przycisk logowania do części administracyjnej. Dla każdego pacjenta wyświetlane są podstawowe informacje.

Rysunek 3.2. Widok startowy strony internetowej.

Kliknięcie w dany przycisk w kolumnie *Szczegóły* przenosi do kolejnego widoku (Rysunek 3.3) zawierającego szczegółowe informacje o pacjencie, a także zbiór badań przypisanych do tego pacjenta. Głównym założeniem strony jest umożliwienie użytkownikowi prostego dostępu do przeszłych badań pacjenta oraz ich wyboru. Z tego powodu zdecydowano się na wykorzystanie osi czasu, na której zostały umieszczone badania w kolejności od najnowszego do najstarszego. Dzięki temu użytkownik ma szybki dostęp do najnowszych badań, a w razie potrzeby może cofnąć się do starszych badań za pomocą suwaka.

Nad listą badań umieszczone zostały filtry pozwalające wyszukiwać badania o danym typie prześwietlenia, dacie wykonania i części ciała która została prześwietlona. Filtry działają na zasadzie formularza, którego wysłanie odbywa się poprzez naciśnięcie przycisku *Szukaj*.

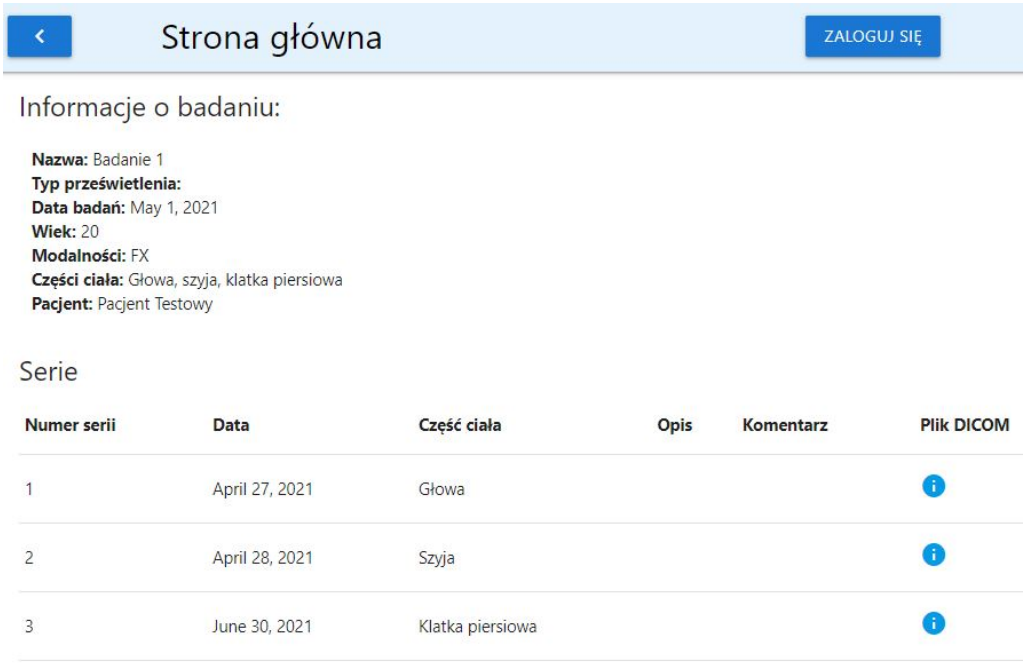
W widoku pacjenta wyświetlają się badania wraz z ogólnymi informacjami. Naciśnięcie odpowiedniej ikony przy nazwie badania pozwala na przejście do widoku szczegółów badania (Rysunek 3.4). W tym widoku, analogicznie do dwóch poprzednich, wyświetlone są szczegółowe informacje



Rysunek 3.3. Widok szczegółowy pacjenta.

dotyczące badania, a także zbiór serii wykonanych w jego obrębie. Podstawowe informacje dotyczące serii wyświetlone są poniżej informacji o badaniu w formie tabeli.

Dla każdej serii kliknięcie w odpowiednią ikonę w kolumnie *Szczegóły* otwiera widok pozwalający na wyświetlenie szczegółowych informacji o serii oraz listy przyporządkowanych do niej plików DICOM.



Rysunek 3.4. Widok badania.

Przykład 3.2

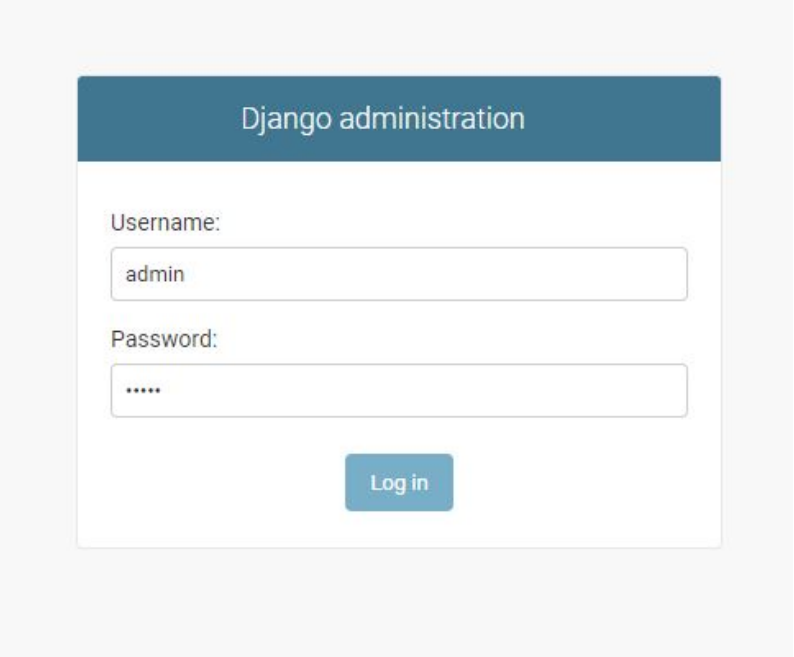
Przykład pokazuje funkcję odpowiadającą za wyświetlenie szablonu strony *studies_info* wraz z odpowiednio przefiltrowanymi danymi. Strona zawiera informacje o badaniu oraz zbiór przypisanych do niego serii.

```
def studies_info(request, studies_id):
    studies_data = Studies.objects.get(id=studies_id)
    studies_series = Series.objects.filter(study=studies_id)
    data = {
        'studies_data' : studies_data,
        'studies_series' : studies_series
    }
    return render(request, 'studies_info.html', data)
```

Dane przekazane do szablonu są w nim dostępne poprzez odwołanie się do zmiennej w nawiasach {{}}. Poniżej zawartość tabeli z informacjami o badaniu.

```
        ...
        <ul class="col s6">
            <li><b>Nazwa:</b>{{studies_data.name}}</li>
            <li><b>Typ prześwietlenia:</b>
                {{studies_data.xray_type}}</li>
            <li><b>Data badań:</b>{{studies_data.study_date}}</li>
            <li><b>Wiek:</b>{{studies_data.age}}</li>
            <li><b>Modalności:</b>{{studies_data.modalities}}</li>
            <li><b>Części ciała:</b>
                {{studies_data.body_parts}}</li>
            <li><b>Pacjent:</b>{{studies_data.patient}}</li>
        </ul>
        ...
```

Aby zalogować się do części administracyjnej, po kliknięciu przycisku *Zaloguj się*, należy wprowadzić odpowiednie dane użytkownika - nazwę oraz hasło.



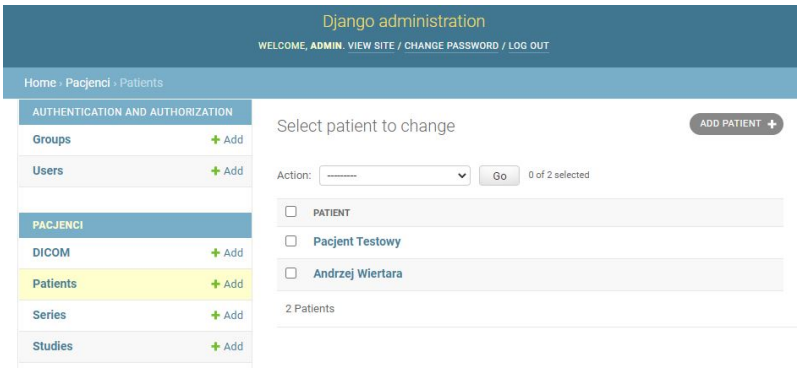
Rysunek 3.5. Panel administracyjny - ekran logowania.

Panel administracyjny umożliwia dodanie nowych użytkowników z dostępem do tegoż panelu a także informacji dotyczących pacjentów oraz badań, które zostaną zapisane w bazie danych a następnie wyświetlone w części użytkowej.

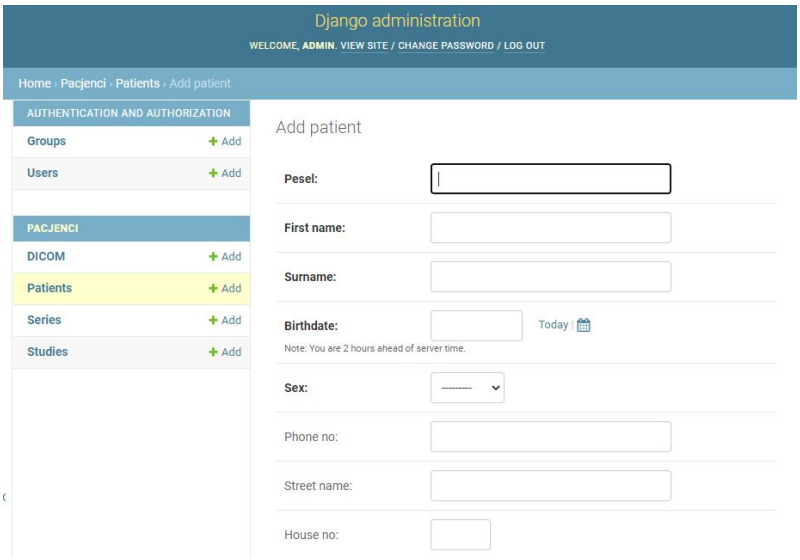
Rysunek 3.6. Panel administracyjny - strona główna.

Proces dodawania nowego rekordu do bazy danych wygląda następująco. Pierwszym krokiem jest wybór modelu oraz kliknięcie na jego nazwę - niebieski link - przykładowo *Patients*. Otwarta zostanie strona z listą aktualnie dodanych pacjentów. W celu dodania nowego pacjenta należy kliknąć przycisk *Add Patient*.

Po wypełnieniu wszystkich danych odnośnie nowego pacjenta należy kliknąć przycisk *Save* znajdujący się w dolnej części formularza.



Rysunek 3.7. Lista pacjentów.



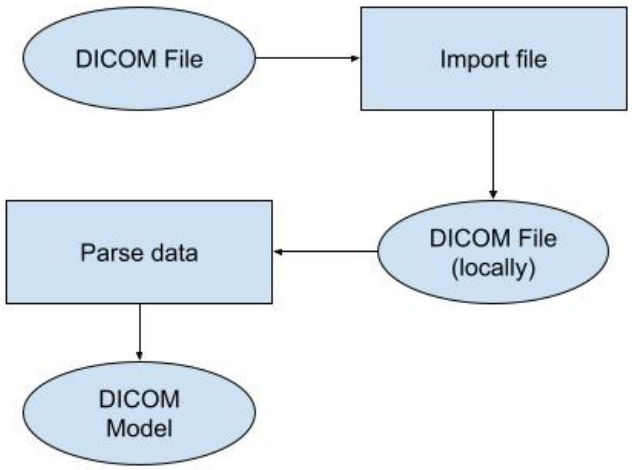
Rysunek 3.8. Formularz dodawania pacjenta.

3.3 Odczyt danych z plików DICOM

W aplikacji istnieje możliwość importu plików DICOM i ich zapisu w pamięci lokalnej. Odczytywane są z nich dane, które następnie zostają zapisane w polu *DESCRITPION* modelu DICOM. Proces ten został pokazany na Rysunku 3.9.

Biblioteka *pydicom* umożliwia odczyt danych począwszy od wieku pacjenta, płci, wzrostu, kończąc na opisie badania i szerokości źrenicy pacjenta. Oprócz wyżej wymienionych danych, w plikach znaleźć można także inne informacje, które opisane są w specyfikacji DICOM. Chociaż nie wszystkie dostępne pola są zawsze wykorzystywane.

Aby odczytać dany atrybut z pliku DICOM należy poznać jego *tag*. Spis wszystkich tagów można znaleźć w specyfikacji DICOM lub wykorzystać do tego funkcję *pydicom.filereader.dcmread* (Przykład 3.3). Są to dwie liczby heksadecymalne zapisane w formacie (XXXX,XXXX), gdzie X to dowolna cyfra lub litera z zakresu od A do F. W aplikacji za odczyt danych z pliku DICOM odpowiada klasa *DicomReader* przedstawiona w Przykładzie 3.4.



Rysunek 3.9. Diagram przepływu danych.

Przykład 3.3

W przykładzie pokazano wykorzystanie funkcji *pydicom.filereader.dcmread* do odczytania tagów przykładowego pliku DICOM, ich nazw oraz przechowywanych w nich danych. Fragment zwracanego wyniku pokazano na Rysunku 3.10.

```
filepath = 'F:/IM-0015-0001.dcm'
df = pydicom.filereader.dcmread(filepath)
print(df)
```

```
(0008, 0023) Content Date                       DA: '20180920'
(0008, 0030) Study Time                         TM: '113338'
(0008, 0031) Series Time                        TM: '113338'
(0008, 0032) Acquisition Time                   TM: '115152'
(0008, 0033) Content Time                       TM: '115306'
(0008, 0050) Accession Number                   SH: '429748'
(0008, 0060) Modality                           CS: 'MR'
(0008, 0070) Manufacturer                       LO: 'GE MEDICAL SYSTEMS'
(0008, 0080) Institution Name                   LO: 'HCP Poznan'
(0008, 0090) Referring Physician's Name         PN: 'NOWAK^VIOLETTA^^DR N. MED.^'
(0008, 1010) Station Name                       SH: 'GEMR'
(0008, 1030) Study Description                  LO: 'MR MOZGU I PNIA MOZGU BEZ'
(0008, 103e) Series Description                 LO: 'Apparent Diffusion Coefficient (mm2/s)'
(0008, 1090) Manufacturer's Model Name          LO: 'Optima MR450w'
(0008, 1111)  Referenced Performed Procedure Step Sequence  1 item(s) ----
   (0008, 0000) Group Length                        UL: 100
   (0008, 1150) Referenced SOP Class UID            UI: Modality Performed Procedure Step SOP Class
   (0008, 1155) Referenced SOP Instance UID         UI: 1.2.840.113619.6.410.1870738852839967512074
```

Rysunek 3.10. Fragment wyniku zwracanego po użyciu funkcji *pydicom.filereader.dcmread* na przykładowym pliku DICOM.

Przykład 3.4

W przykładzie pokazano klasę odczytującą niektóre atrybuty z pliku DICOM. Zmienne pisane wielkimi literami np. STUDY_DATE, PATIENT_AGE zawierają liczby odpowiadające tagom odpowiednich atrybutów w pliku. Dla przykładu STUDY_DATE = 0x00080020, gdyż atrybut z datą badania ma tag (0008,0020).

```
class DicomReader:
    study_date = ''
```

```
    patient_age = ''
    modality = ''
    body_part = ''
    series_num = ''
    series_date = ''
    series_description = ''
    instance_number = ''

    def __init__(self, dicom_file):
        self.study_date = self._getValue(dicom_file, STUDY_DATE)
        self.patient_age = self._getValue(dicom_file, PATIENT_AGE)
        self.modality = self._getValue(dicom_file, MODALITY)
        self.body_part = self._getValue(dicom_file, BODY_PART)
        self.series_num = self._getValue(dicom_file, SERIES_NUM)
        self.series_date = self._getValue(dicom_file, SERIES_DATE)
        self.series_description = self._getValue(dicom_file,
                                                 SERIES_DESCRIPTION
    )
        self.instance_number = self._getValue(dicom_file,
                                                 INSTANCE_NUMBER)

    def _getValue(self, dicom_file, attribute_val):
        return dicom_file[attribute_val].value
```

W przypadku, gdy do tworzonego rekordu modelu DICOM zaimportuje się plik DICOM, to po jego zapisie wywołana zostanie metoda *DICOM_handler* zaprezentowana w Przykładzie 3.5. Jest to metoda typu *post_save*. Oznacza to, że zostanie uruchomiona po zapisie zarówno samego rekordu, jak i importowanego pliku DICOM w lokalnym folderze serwera. Dzięki temu zabiegowi możliwe będzie następnie wykorzystanie funkcji *DicomReader* i zapisanie odczytanych danych w polu opisu danego rekordu modelu DICOM.

Przykład 3.5

W przykładzie pokazano klasę, która zapisuje odczytane wcześniej z zaimportowanego pliku DICOM informacje do pola *Description* rekordu modelu DICOM.

```
@receiver(post_save, sender=DICOM)
def DICOM_handler(sender, created, instance=False, **kwargs):
    if created:
        dicom_file = pydicom.filereader.dcmread(instance.
    dicom_file.path)
        dicom = DicomReader(dicom_file)
        study_date = dicom.study_date[:2] + '.' \
                     + dicom.study_date[4:6] \
                     + '.' + dicom.study_date[:4]
        series_date = dicom.study_date[:2] + '.' \
                      + dicom.study_date[4:6] \
                      + '.' + dicom.study_date[:4]
        description = "Data badania: " + study_date \
                      + '\nWiek: ' + dicom.patient_age \
                      + '\nModality: ' + dicom.modality \
                      + '\nCzesc ciala: ' + dicom.body_part \
                      + '\nNumer serii: ' + str(dicom.series_num)
    \
                      + '\nData wykoniania: ' + series_date \
                      + '\nOpis: ' + dicom.series_description \
                      + '\nNumer instancji: ' + str(dicom.
    instance_number)
        instance.description = description
```

```
21            instance.save()
```

4 Podsumowanie

W ramach projektu udało się stworzyć działający prototyp aplikacji oraz zaprojektować hurtownię danych. Przedstawiony problem jest bardzo obszerny. Ze względu na ograniczenia czasowe nie udało się go w pełni rozwiązać. Opracowany system nie jest w pełni funkcjonalnym rozwiązaniem, jednakże w sposób jasny nakreśla ideę rozwiązania problemu. Technologie użyte w projekcie dobrane zostały tak, aby umożliwić utworzenie systemu z docelową funkcjonalnością oraz zapewnić działanie niezależnie od ilości danych.

Przed oprogramowaniem postawione zostały wymagania, które ze względu na chaotyczność danych posiadanych przez Centrum Medyczne HCP nie udało się spełnić. Jednym z problemów znacznie utrudniających uporządkowanie danych jest niespójność opisów zdjęć radiologicznych sporządzanych przez techników wykonujących badania. Jednym z możliwych rozwiązań byłoby opracowanie modelu klasyfikacji obrazów z wykorzystaniem technik uczenia maszynowego. Obrazy wykonywane podczas badań u różnych pacjentów są do siebie podobne i można w nich zauważyć wzorce, które pozwalają na jednoznaczną identyfikację obrazów. Takie rozwiązanie umożliwiłoby wyeliminowanie błędu ludzkiego, przez co etykiety nadawane obrazom podczas badań byłyby nadawane w jednoznaczny sposób. Pozwoliłoby to lekarzom na wydajniejszą i mniej frustrującą pracę podczas analizy badań pacjentów na przestrzeni czasu. Co najważniejsze przejrzysty dostęp do całej historii badań zmniejszyłby szanse na przeoczenie istotnych informacji w przebiegu choroby pacjenta.

Wydział Informatyki i Telekomunikacji
Politechnika Poznańska

Skład przy użyciu systemu LaTeX na platformie Overleaf.