Elektroniczna Karta Pacjenta

Informatyka w Medycynie – projekt

Wykonanie: Paulina Pacura, 142179

1. Przedstawienie problemu oraz wybranych metod rozwiązania

Stworzony projekt to aplikacja pełniąca funkcję elektronicznej karty pacjenta. Do przetwarzania danych wykorzystany został **standard FHIR w wersji R4**. Z serwera Hapi-FHIR pobierane są następujące zasoby:

- Patient,
- Observation,
- MedicationRequest

Zasoby te zostają przetworzone na serwerze i następnie zwizualizowane z wykorzystaniem **aplikacji internetowej.** Stworzona aplikacja generuje również **wykresy** dla obserwacji zawierających dane liczbowe. Możliwe jest także wyszukiwanie pacjenta po nazwisku.

2. Opis głównych funkcji programu

Do wykonania projektu jako aplikację webową zdecydowałam się na stworzenie back-endu do pobrania i przetworzenia danych z bazy (lokalnego serwera hapi-FHIR) oraz front-endu, odpowiedzialnego za wyświetlenie danych w postaci strony internetowej.

2.1. Back-end: Spring Boot

Do stworzenia back-endu wykorzystałam framework Spring Boot (Java). Program zawiera dwie klasy typu *Rest Controller*, czyli obsługujące przychodzące z front-endu żądania HTTP. Obsługują one żądania dotyczące listy pacjentów, listy *Observation* dla konkretnego pacjenta oraz listy *MedicationRequest* dla konkretnego pacjenta. Po otrzymaniu danego żądania funkcja komunikuje się bezpośrednio z klasą typu *Repository* – pobiera z niej listę danych, a następnie przetwarza otrzymane dane na odpowiadających klas obiekty obiekty *DTO* (Data Transfer Object) (zawierające wyłącznie dane wybrane jako wystarczające do przedstawienia w karcie pacjenta). Lista otrzymanych tak obiektów przesyłana jest jako odpowiedź na dane żądanie.

Poniżej przedstawiam ważniejsze fragmenty kodu programu.

2.1.1. Pobieranie danych z serwera Hapi-Fhir

```
private <T> ArrayList<T> getResource(Class<T> resource, ICriterion condition, IParam param){
    try {
        Bundle queryResults = executeQuery(resource,condition,param);
        ArrayList<Resource> resourcesArray = queryResults.getEntry().stream()
                .map(Bundle.BundleEntryComponent::getResource)
                .collect(Collectors.toCollection(ArrayList::new));
        return (ArrayList<T>) resourcesArray;
    }catch (Exception e){
        System.out.println("Exception when getting Bundle");
        e.printStackTrace();
        return new ArrayList<>();
    }
}
```

```
private <T> Bundle executeQuery(Class<T> resource, ICriterion condition, IParam param){
    IQuery <IBaseBundle> query = client
            .search()
            .forResource((Class<? extends IBaseResource>) resource);
    if(condition!=null)
        query.where(condition);
    if(param!=null)
        query.sort().descending(param);
    return (Bundle)query.execute();
}
```

2.1.2. Tworzenie obiektów DTO

```
public PatientDto(Patient resource) {
    super(resource);
    name = resource
            .getNameFirstRep() HumanName
            .getGiven() List<StringType>
            .get(0) StringType
            .getValue();
    familyName = resource.getNameFirstRep().getFamily();
    gender = resource.getGender().getDisplay();
    birthDate = resource.getBirthDate();
}
```

```
public ObservationDto(Observation resource) {
    super(resource);
    setValue(String.valueOf(resource.getValueQuantity().getValue()));
    setValueCode(String.valueOf(resource.getValueQuantity().getCode()));
    setDescription(resource.getCode().getText());
    date = resource.getEffectiveDateTimeType().toHumanDisplay();

}
```

```
public MedicationRequestDto(MedicationRequest resource) {
    super(resource);
    setStatus(String.valueOf(resource.getStatus()));
    setDescription(resource.getMedicationCodeableConcept().getText());
    if (description==null) setDescription("No description yet");
    setDate(resource.getAuthoredOnElement().toHumanDisplay());
    setRequester(String.valueOf(resource.getRequester().getDisplay()));
}
```

2.1.3. Obsługa żądań z front-endu

```
@GetMapping
public ArrayList<PatientDto> patientsList() {
    try{
        ArrayList<Patient> patientsResources = dataServer.getPatientsData();
        ArrayList<PatientDto> patientsList = (ArrayList<PatientDto>) patientsResources.stream()
                .map(PatientDto::new)
                .collect(Collectors.toList());
        return patientsList;

    }catch (Exception e){
        System.out.println("Error getting Patients Data");
        return new ArrayList<>();
    }
}
```

```
@GetMapping("/search={familyname}")
@CrossOrigin("*")
public ArrayList<PatientDto>  patientsListFiltered(@PathVariable("familyname") String familyName) {
    try{
        ICriterion condition = Patient.FAMILY.matches().value(familyName);
        ArrayList<Patient> patientsFilteredResources = dataServer.getPatientsFilteredData(condition);
        ArrayList<PatientDto> patientsList = (ArrayList<PatientDto>) patientsFilteredResources.stream()
                .map(PatientDto::new)
                .collect(Collectors.toList());
        return patientsList;

    }catch (Exception e){
        System.out.println("Error getting Patients Data");
        return new ArrayList<>();
    }

}
```

2.2. Front-end: React

Do zrobienia front-endu wykorzystałam bibliotekę React (JavaScript + Html). Do estetycznej części aplikacji wykorzystałam także CSS. Aplikacja podzielona jest na kilka komponentów: komponent zarządzający listą pacjentów i komponent odpowiadający za pojedynczego pacjenta listy, komponent zarządzający listą obserwacji i komponent odpowiadający za pojedynczą obserwację na liście, komponent zarządzający listą leków i komponent odpowiadający za pojedynczy lek. W uproszczeniu, aplikacja umożliwia przegląd listy pacjentów, znalezienie po nazwisku lub wybranie pacjenta z listy i przejrzenie historii jego obserwacji oraz jego leków. Możliwe jest także sprawdzenie wykresów wszystkich obserwacji, dla których zanotowano wartości liczbowe.

Do pobrania danych aplikacja wykorzystuje moduł *Axios*. Asynchronicznie wysyła żądanie GET pod odpowiedni adres serwera back-end i pobiera otrzymaną odpowiedź. Tak pobrane dane zostają wyświetlone na stronie lub w razie konieczności przesłane na dalszą podstronę, aby uniknąć konieczności ponownego czekania na pobranie tych samych danych.

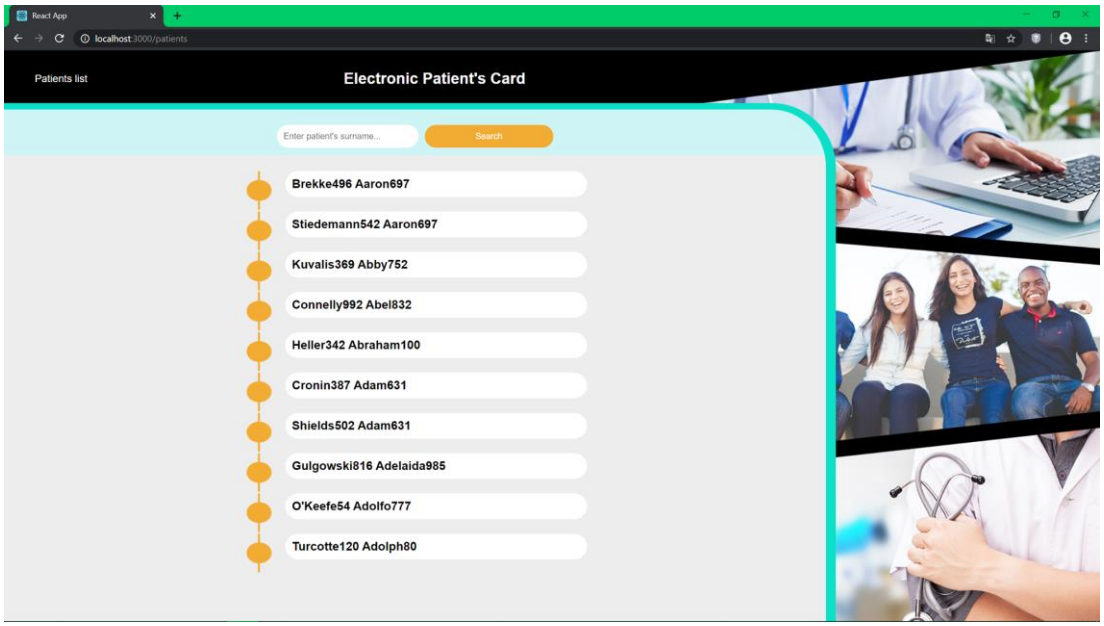
2.3. Wykresy przedstawiające zmiany parametrów w czasie

Program zaopatrzony jest także w funkcję generowania wykresów. Wykresy tworzone są na bieżąco i wyświetlone mogą zostać dla wszystkich tych obserwacji, w których dla danego pacjenta zapisano wartości liczbowe. Do tworzenia wykresów wykorzystano bibliotekę *Chart.js* . Zaimplementowałam także opcję wyboru zakresu dat, dla których wyświetlone mają być wyniki.

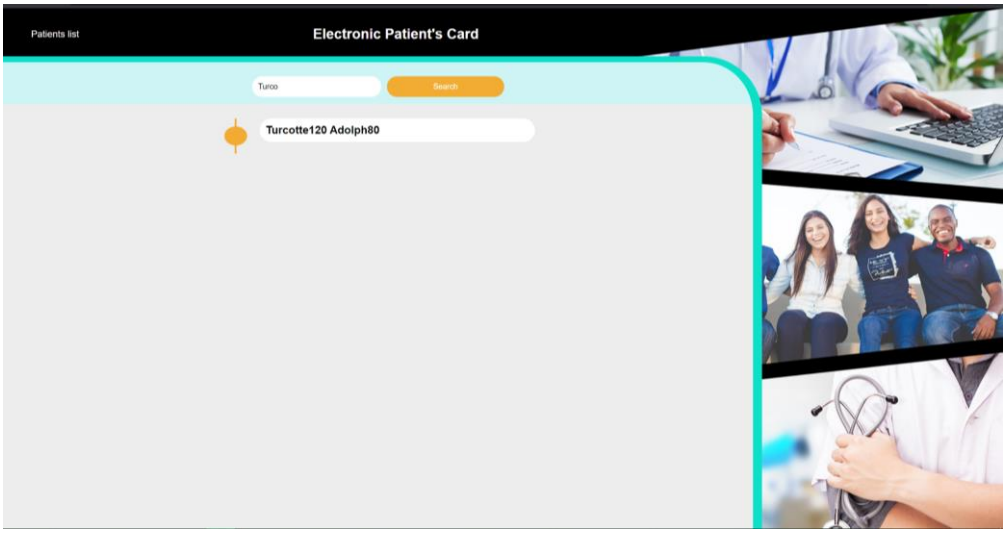
2.4. Przedstawienie wyglądu stworzonej aplikacji

2.4.1. Wersja desktopowa

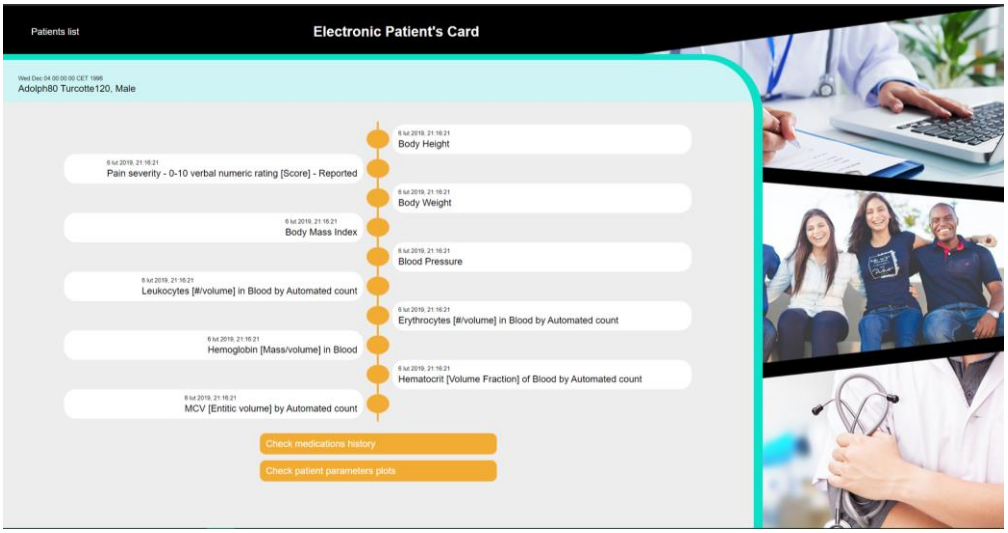
2.4.1.1. Lista pacjentów



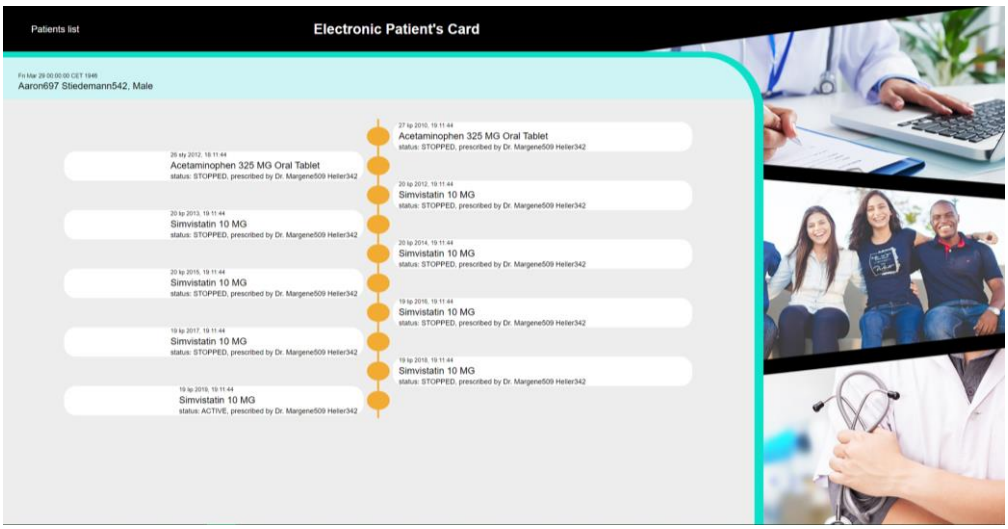
2.4.1.2. Wyszukiwanie pacjenta po nazwisku



2.4.1.3. Lista obserwacji konkretnego pacjenta

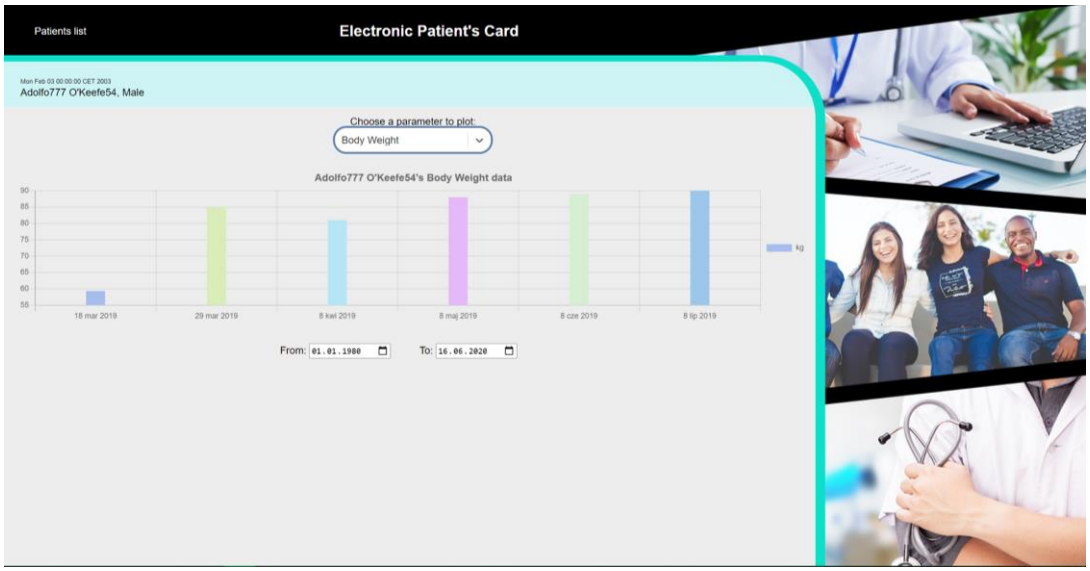


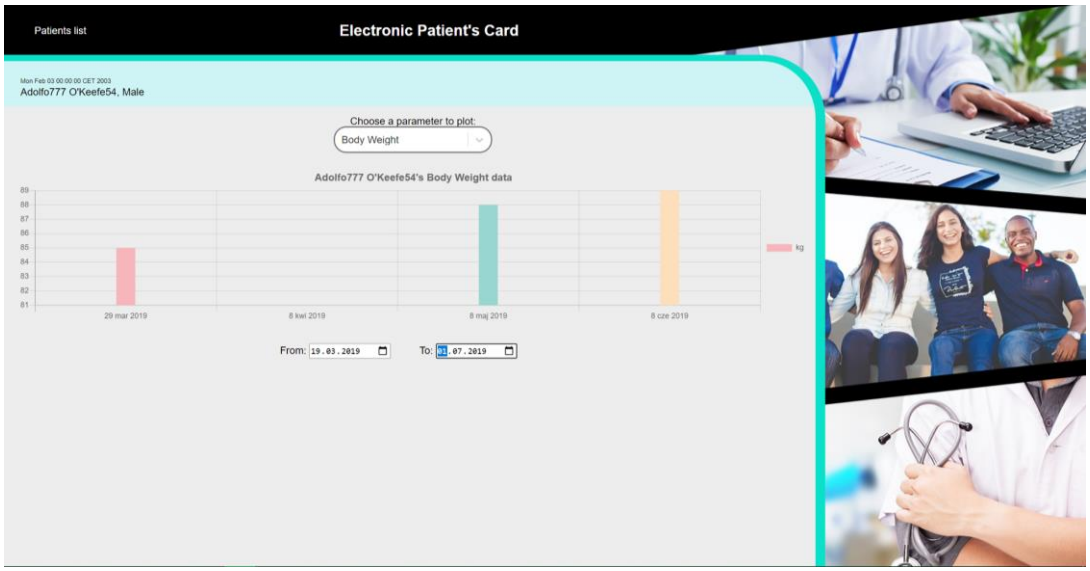
2.4.1.4. Lista leków konkretnego pacjenta



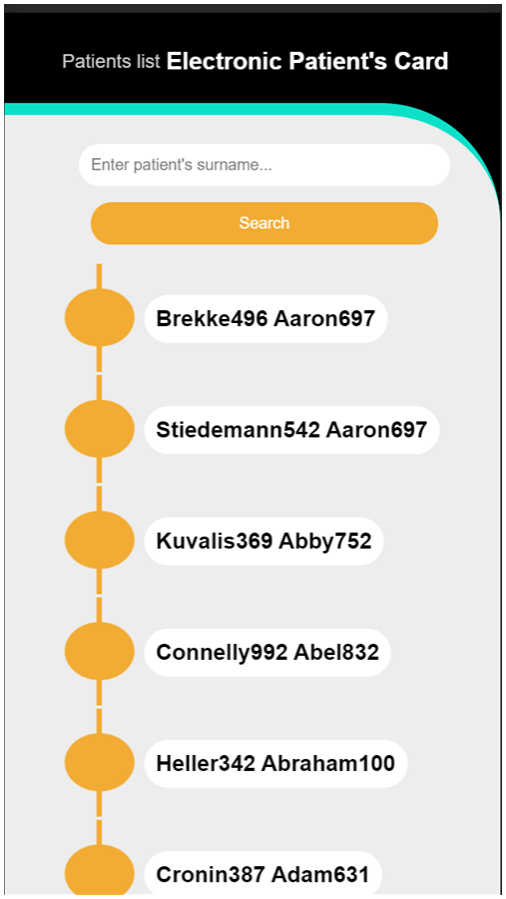
2.4.1.5. Wykresy danych konkretnego pacjenta

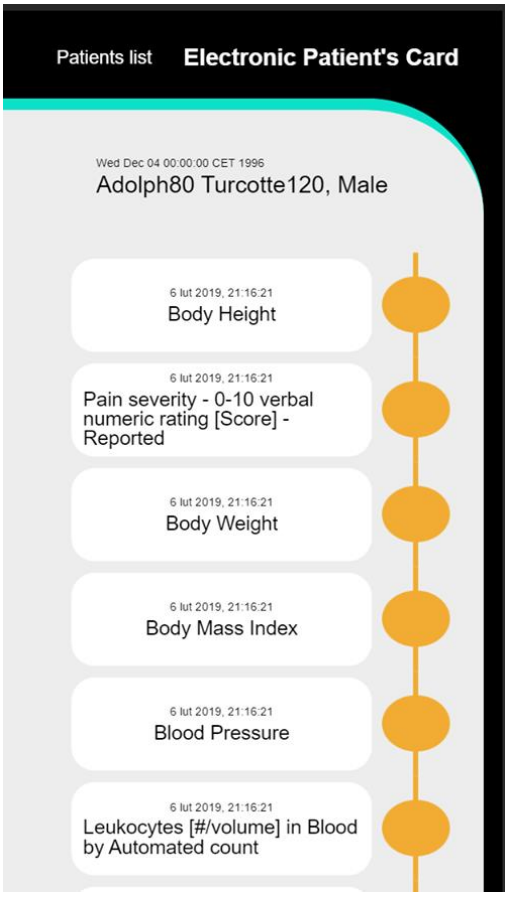
Jako, iż nie udało mi się znaleźć pacjenta, który miałby w swoich obserwacjach kilkukrotnie powtórzone to samo badanie, zdecydowałam się na wprowadzenie danych sztucznych w celu zademonstrowania działania wykresu i jego filtrowania.

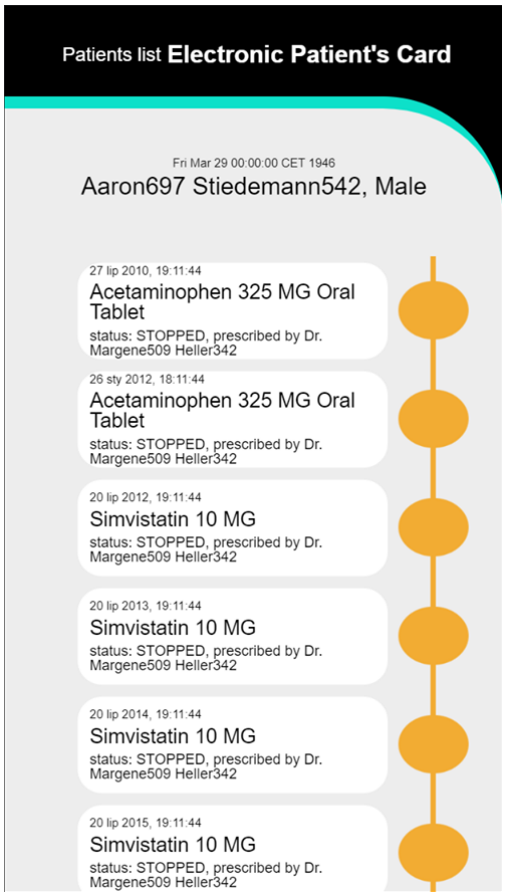


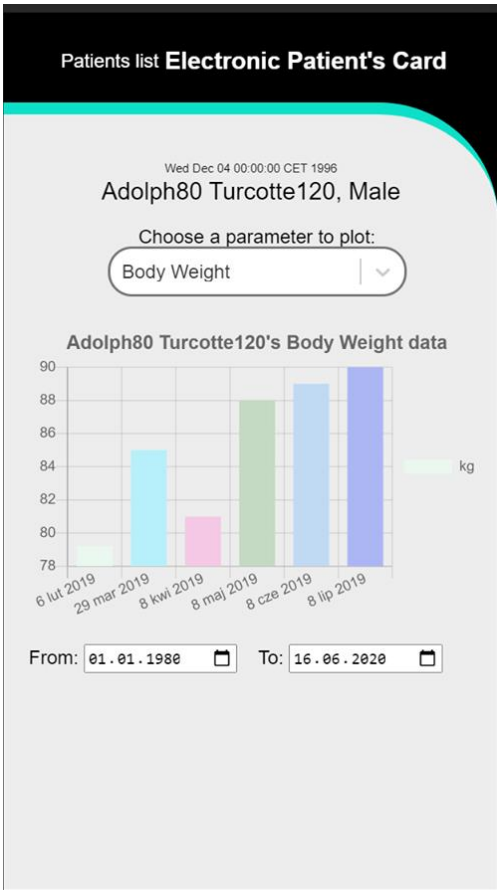


2.4.2. Wersja responsywna - mobilna









3. Wnioski

3.1. FHIR - Fast Healthcare Interoperability Resources – to standard opisujący formaty i elementy danych oraz interfejs programowania aplikacji do wymiany elektronicznych zapisów medycznych. Dzięki standaryzacji takich informacji możliwe jest łatwiejsze gromadzenie danych medycznych także na większą skalę, co może mieć duże znaczenie dla analityków danych medycznych, a także innych, globalnych inicjatyw medycznych. Utrzymywanie danych medycznych w jednym standardzie to także ułatwienie w przypadku konieczności integracji osobnych serwisów medycznych.

3.2. Zadanie pozwoliło na zapoznanie się z działaniem serwerów zarządzających danymi w standardzie FHIR oraz z praktycznym przetwarzaniem tych danych. Oprócz samego pobrania danych, należało także wybrać sposób ich prezentacji, co ma kluczowe znaczenie z punktu widzenia potencjalnego użytkownika i powinno być przemyślane, aby osiągnąć aplikację intuicyjną i przystępną, aby mogła stanowić ułatwienie pracy wspomnianemu potencjalnemu użytkownikowi. Tworzona aplikacja była uproszczona – w wersji podstawowej pozbawiona możliwości modyfikacji danych, z jedynie prostym filtrowaniem po nazwisku – dzięki czemu zadanie było ułatwione, gdyż liczba funkcjonalności była ograniczona. Mimo to sposób przedstawienia danych pacjenta w sposób czytelny okazał się pewnym wyzwaniem.