

# Bazy Danych

## Projekt Karta Pacjenta

12 stycznia 2020

### Spis treści

<b>1</b>	<b>Autorzy</b>	<b>2</b>
<b>2</b>	<b>Opis projektu „Karta Pacjenta”</b>	<b>2</b>
2.1	Przedstawienie aplikacji . . . . .	3
2.2	Testowanie aplikacji . . . . .	8
<b>3</b>	<b>Wykorzystane technologie</b>	<b>8</b>
<b>4</b>	<b>Implementacja</b>	<b>8</b>
4.1	Kontrola wersji . . . . .	8
4.2	Baza danych . . . . .	8
4.3	Logika aplikacji (ang. <i>backend</i> ) . . . . .	11
4.3.1	Punkty dostępowe (ang. <i>enpoint</i> ) . . . . .	11
4.3.2	Dlaczego REST? . . . . .	11
4.3.3	Zabezpieczenie danych - API . . . . .	12
4.3.4	Zabezpieczenie danych - baza danych . . . . .	12
4.3.5	Ograniczenia . . . . .	12
4.3.6	Przyspieszenie . . . . .	13
4.3.7	Możliwości dotyczące rozwoju - przyspieszenie . . . . .	13
4.4	Warstwa wizualna aplikacji „Karta Pacjenta” (ang. <i>frontend</i> ) . . . . .	13
4.4.1	Działanie warstwy wizualnej . . . . .	13
4.4.2	Prostota implementacji i wieloplatformowość . . . . .	14
4.4.3	Rola warstwy wizualnej aplikacji . . . . .	14
4.5	Testy obciążeniowe . . . . .	14
4.5.1	Testowane zagadnienia . . . . .	14
4.5.2	Wielowątkowość . . . . .	15
4.5.3	Rola testów w rozwoju aplikacji . . . . .	15

## 1 Autorzy

**Krzysztof Czarnecki** - implementacja niewidocznej dla użytkownika części programu, która posiada dostęp do wymaganych zasobów (ang. *backend*)

**Błażej Czekala** - implementacja testów obciążeniowych, z użyciem wielowątkowości

**Patryk Wenz** - implementacja wyglądu i zachowania strony (ang. *frontend*)

**Hubert Braun** - implementacja bazy danych i jej obsługi

## 2 Opis projektu „Karta Pacjenta”

Projekt „Karta Pacjenta” zakładał stworzenie aplikacji umożliwiającej przechowywanie danych pacjentów w serwisie bazodanowym. Aplikacja przechowuje informacje dotyczące chorób przebytych przez pacjenta, wystawionych przez lekarza recept. Umożliwia bezpieczne przechowywanie danych wrażliwych, takich jak pesel, numer telefonu itd. Umożliwia także prezentację tych danych oraz ich eksport (także w postaci anonimowej - bez danych osobowych - posiadających tylko informację o przebytej chorobie, nie o pacjencie).

Aplikacja powstała przy użyciu języków: Java (ang. *backend*) oraz Angular (TypeScript) (ang. *frontend*). Serwis postawiony jest na darmowej domenie dostępnej pod tym linkiem. Gorąco zachęcamy do zapoznania się z działaniem aplikacji.

Dzięki serwisowi Heroku możliwe było darmowe opublikowanie witryny w internecie. Nawet w darmowej wersji serwis ten zapewnia usługi związane z CI (ang. *Continuous Integration*). Efektem tego, jest fakt, że po każdej aktualizacji zdalnego repozytorium *Git* serwis automatycznie przebudowuje się.

## 2.1 Przedstawienie aplikacji

KartaPacjenta Login Register Open API

### Login

Username:

Title:

Log In

No rights reserved. All data in database are encrypted.

Rysunek 1: Ekran logowania.

KartaPacjenta Admin Site Patients Diseases Open API Log Out

Admin Site Users

Search users

User ID	Username	Email	Roles	Grant Admin	Grant Doctor	Grant Patient	Delete
1	admin	admin@email.ru	ROLE_PATIENT,ROLE_ADMIN,ROLE_DOCTOR	Grant Admin	Grant Doctor	Grant Patient	Delete
2	patient	admin@email.ru	ROLE_PATIENT	Grant Admin	Grant Doctor	Grant Patient	Delete
3	patient2	admin@email.ru	ROLE_PATIENT	Grant Admin	Grant Doctor	Grant Patient	Delete
4	doctor	admin@email.ru	ROLE_DOCTOR	Grant Admin	Grant Doctor	Grant Patient	Delete
14	bigbear612	aaron.freeman@example.com	ROLE_PATIENT	Grant Admin	Grant Doctor	Grant Patient	Delete
16	bigostrich383	kadir.durak@example.com	ROLE_PATIENT	Grant Admin	Grant Doctor	Grant Patient	Delete
18	greenpeacock680	sophie.chen@example.com	ROLE_PATIENT	Grant Admin	Grant Doctor	Grant Patient	Delete
20	beautifulsake763	adriana.crespo@example.com	ROLE_PATIENT	Grant Admin	Grant Doctor	Grant Patient	Delete
22	tinyzebra495	dragana.jemoine@example.com	ROLE_PATIENT	Grant Admin	Grant Doctor	Grant Patient	Delete

Rysunek 2: Panel administratora, umożliwiający nadawanie praw, oraz usuwanie użytkowników.

KartaPacjenta Admin Site Patients Diseases Open API Log Out			
Diseases All diseases Create disease			
Search diseases			
Disease ID	Name	Category	More info
11	Grypa	Zakaźne	<a href="#">info</a>
157	neighborly	stupid	<a href="#">info</a>
159	cooing	birth	<a href="#">info</a>
161	sight	digest	<a href="#">info</a>
162	horn	son	<a href="#">info</a>
164	longing	endanger	<a href="#">info</a>
167	learned	crowded	<a href="#">info</a>
170	frail	lively	<a href="#">info</a>
171	blue-eyed	home	<a href="#">info</a>

Rysunek 3: Zakładka zawierająca wypisane wszystkie zapisane w systemie choroby. Obok widoczna zakładka umożliwiająca dodanie nowej choroby. Przycisk „info” w tabeli „More info” umożliwia zapoznanie się z dostępnymi informacjami dotyczącymi danej choroby.

KartaPacjenta Admin Site Patients Diseases Open API Log Out				
Lew				
Patient ID	Name	Address	Personal Identity Number	Patient info
873	Lewis Anderson	New Zealand Northland Rotorua	11070191367004	<a href="#">info</a>
843	Theodore Lewis	New Zealand Marlborough Whangarei	12083769609540	<a href="#">info</a>
1184	Felicia Franklin	United States Maine Inglewood	14095271717632	<a href="#">info</a>

No rights reserved. All data in database are encrypted.

Rysunek 4: Zakładka ze wszystkimi dostępnymi pacjentami.

## Patient Info

First Name	Alyssa
Last Name	Garrett
Address	Australia Western Australia Coffs Harbour
Phone	02-8408-3009
Personal Identity Number	3399577699990
E-mail	alyssa.garrett@example.com

No rights reserved. All data in database are encrypted.

Rysunek 5: Zakładka zawierająca informacje o wybranym pacjencie.

Search in history				Generate Medical Documentation Generate Anonymize Medical Documentation	
ID	Disease Name	Visit Category	Doctor Name	Timestamp	More info
1299	Grypa	Wizyta kontrolna	Kris Admin	2020-01-08 21:21:56.76	Info
1300	Grypa	Wizyta po przebytej chorobie	Kris Admin	2020-01-08 21:23:56.529	Info

No rights reserved. All data in database are encrypted.

Rysunek 6: Historia wizyt pacjenta.

## Patient Info

Patient ID	682
Patient User ID	682
Username	purplepeacock811
Name	Alyssa Garrett
Address	Australia Western Australia Coffs Harbour
E-mail	alyssa.garrett@example.com
Phone Number	02-8408-3009
Personal Identity Number	3399577699990

## Visit from 2020-01-08 21:21:56.76

Visit ID	1299
Visit Category	Wizyta kontrolna
Disease Name	Grypa
Disease Category	Zakaźne
Disease Description	blablabla to jest jakiś opis
Doctor Name	Kris Admin
Doctor Phone Number	12312414
Doctor Email	admin@email.ru
Visit Time	2020-01-08 21:21:56.76
Patient Description	Boli mnie gardło
Doctor Description	Pacjent ma przewlekłe zapalenie strun głosowych oraz głęboki katar
Doctor Prescription	L4, antybiotyk, krople do nosa

## Visit from 2020-01-08 21:23:56.529

Visit ID	1300
Visit Category	Wizyta po przebytej chorobie
Disease Name	Grypa

Rysunek 7: Szczegółowa historia wizyt

Serwis umożliwia wygenerowanie danych o pacjencie i przebytych chorobach w zunifikowanym formacie *JSON*. Jest to możliwe na stronie związanej w historią choroby pacjenta, przedstawionej na rysunku ??.

```
// 20200108222414
// https://trunk-kartapacjentaservice.herokuapp.com/api/illnessCourse/682/full

{
  "patientId": 682,
  "patientUserId": 681,
  "patientUserName": "purplepeacock811",
  "patientFirstName": "Alyssa",
  "patientLastName": "Garrett",
  "patientAddress": "Australia Western Australia Coffs Harbour",
  "patientPhoneNumber": "02-8408-3009",
  "patientEmail": "alyssa.garrett@example.com",
  "patientPersonalIdentityNumber": "3399577699990",
  "medicalHistoryTOS": [
    {
      "courseOfIllnessId": 1299,
      "visitCategory": "Wizyta kontrolna",
      "diseaseName": "Grypa",
      "diseaseCategory": "Zakaźne",
      "diseaseDescription": "blablabla to jest jakiś opis",
      "patientId": 682,
      "doctorId": 12,
      "doctorFirstName": "Kris",
      "doctorLastName": "Admin",
      "doctorPhoneNumber": "12312414",
      "doctorEmail": "admin@email.ru",
      "visitTimeStamp": "2020-01-08 21:21:56.76",
      "patientDescription": "Boli mnie gardło",
      "doctorDescription": "Pacjent ma przewlekłe zapalenie strun głosowych oraz głęboki katar",
      "doctorPrescription": "L4,\nantybiotyky,\nknkrole do nosa\n"
    },
    {
      "courseOfIllnessId": 1300,
      "visitCategory": "Wizyta po przebytej chorobie",
      "diseaseName": "Grypa",
      "diseaseCategory": "Zakaźne",
      "diseaseDescription": "blablabla to jest jakiś opis",
      "patientId": 682,
      "doctorId": 12,
      "doctorFirstName": "Kris",
      "doctorLastName": "Admin",
      "doctorPhoneNumber": "12312414",
      "doctorEmail": "admin@email.ru",
      "visitTimeStamp": "2020-01-08 21:23:56.529",
      "patientDescription": "Już wyzdrowiałem",
      "doctorDescription": "Pacjent wyzdrowiał",
      "doctorPrescription": "Pacjent nie wymaga dalszego leczenia w kierunku grypy"
    }
  ]
}
```

Rysunek 8: Dane pacjenta i jego choroby w formacie JSON.

Serwis posiada także możliwość przetestowania punktów końcowych (ang. *endpoint*). Funkcjonalność dostępna jest na stronie.

## 2.2 Testowanie aplikacji

Do przetestowania wszystkich funkcji aplikacji „Karta Pacjenta” wymagane jest, aby skorzystać z konta administratora - konta różnych użytkowników posiadają różne uprawnienia. Administrator ma dostęp do wszystkich możliwych miejsc na stronie. Podczas testowania aplikacji proszę używać konta:

login: admin

hasło: admin

## 3 Wykorzystane technologie

- Język Programowania **Java** - (ang. *backend*),
- Szkielet aplikacyjny (ang. *framework*) **Spring Boot** - wykorzystywany do programowania logiki serwisu internetowego (ang. *backend*),
- Szkielet aplikacyjny (ang. *framework*) **Spring Security** - wykorzystywany do zabezpieczenia dostępu do punktów dostępowych serwisu internetowego (ang. *endpoint*),
- Szkielet aplikacyjny (ang. *framework*) **Angular CLI** - wykorzystywany do utworzenia warstwy wizualnej aplikacji (ang. *frontend*),
- System zarządzania relacyjnymi bazami danych **PostgreSQL**.

## 4 Implementacja

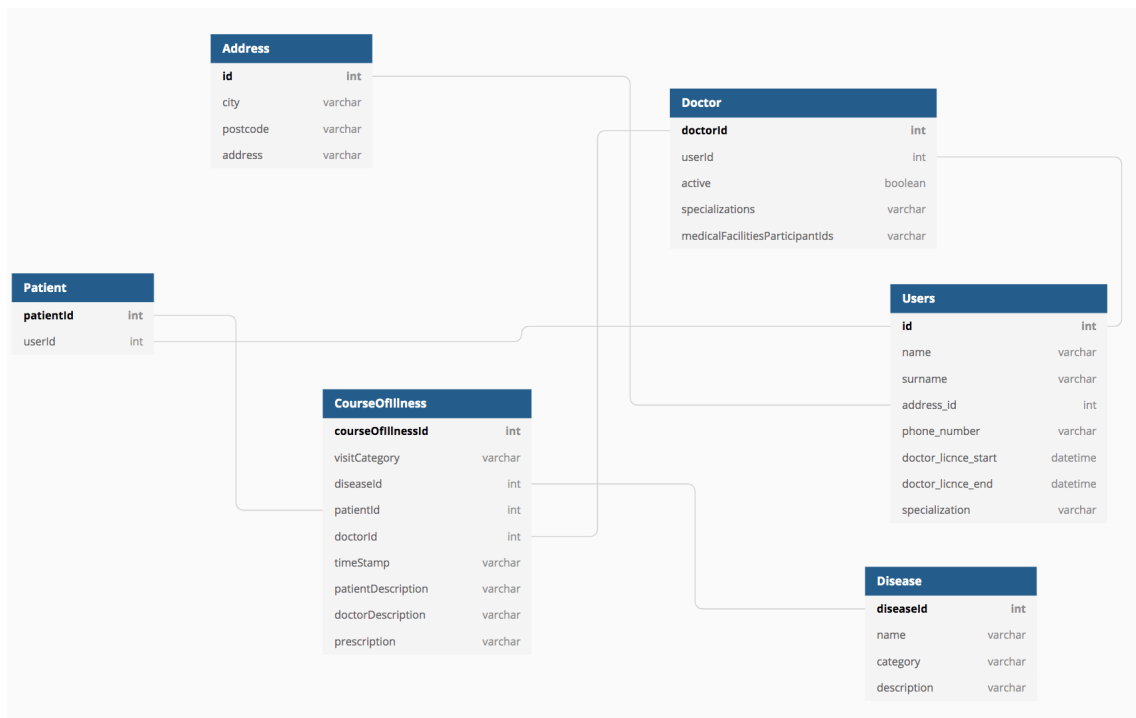
### 4.1 Kontrola wersji

Do pracy zespołowej wykorzystano narzędzie *Git*. Umożliwiło ono sprawne dzielenie się zmianami w kodzie, zarządzanie i wersjonowanie zmian.

### 4.2 Baza danych

Na rysunku 9 przedstawiony jest diagram ER schematu bazy danych.





Rysunek 9: Diagram ER bazy danych

Poniżej przedstawiono kod SQL wykorzystywany do inicjalizacji bazy danych:

```

1 CREATE TABLE "Address" (
2     "id" SERIAL PRIMARY KEY NOT NULL,
3     "city" varchar,
4     "postcode" varchar,
5     "address" varchar
6 );
7
8 CREATE TABLE "Users" (
9     "id" SERIAL PRIMARY KEY NOT NULL,
10    "name" varchar,
11    "surname" varchar,
12    "address_id" int,
13    "phone_number" varchar,
14    "doctor_licence_start" datetime,
15    "doctor_licence_end" datetime,
16    "specialization" varchar
17 );
18

```

```

19 CREATE TABLE "Disease" (
20     "diseaseId" SERIAL PRIMARY KEY NOT NULL,
21     "name" varchar,
22     "category" varchar,
23     "description" varchar
24 );
25
26 CREATE TABLE "Doctor" (
27     "doctorId" int PRIMARY KEY,
28     "userId" int,
29     "active" boolean,
30     "specializations" varchar,
31     "medicalFacilitiesParticipantIds" varchar
32 );
33
34 CREATE TABLE "Patient" (
35     "patientId" int PRIMARY KEY,
36     "userId" int
37 );
38
39 CREATE TABLE "CourseOfIllness" (
40     "courseOfIllnessId" int PRIMARY KEY,
41     "visitCategory" varchar,
42     "diseaseId" int,
43     "patientId" int,
44     "doctorId" int,
45     "timeStamp" varchar,
46     "patientDescription" varchar,
47     "doctorDescription" varchar,
48     "prescription" varchar
49 );
50
51 ALTER TABLE "Users"
52 ADD FOREIGN KEY ("address_id")
53 REFERENCES "Address" ("id");
54
55 ALTER TABLE "Doctor"

```

```

56 ADD FOREIGN KEY ("userId")
57 REFERENCES "Users" ("id");
58
59 ALTER TABLE "Patient"
60 ADD FOREIGN KEY ("userId")
61 REFERENCES "Users" ("id");
62
63 ALTER TABLE "CourseOfIllness"
64 ADD FOREIGN KEY ("diseaseId")
65 REFERENCES "Disease" ("diseaseId");
66
67 ALTER TABLE "CourseOfIllness"
68 ADD FOREIGN KEY ("patientId")
69 REFERENCES "Patient" ("patientId");
70
71 ALTER TABLE "CourseOfIllness"
72 ADD FOREIGN KEY ("doctorId")
73 REFERENCES "Doctor" ("doctorId");

```

## 4.3 Logika aplikacji (ang. *backend*)

### 4.3.1 Punkty dostępowe (ang. *endpoint*)

Dzięki ogromnej popularności aplikacji internetowych opartych na języku *Java* oraz *framework'a Spring* możliwe było szybkie wygenerowanie dokumentacji *Open API*. Pod linkiem dostępny jest spis wszystkich dostępnych w serwisie endpointów. Wejście w ten link będzie wymagało podania loginu i hasła (dostępnego tutaj: 2.2).

### 4.3.2 Dlaczego REST?

Zalety REST API:

- Bezstanowość klienta - serwer nie ma potrzeby zapamiętywania wcześniejszego stanu, ponieważ zapytania HTTP zawierają wszystkie potrzebne informacje,
- Łatwość manipulowania obiektami z poziomu URL - metodami HTTP,

- Czytelność wykonywanych działań ze względu na używanie metod HTTP zgodnych z ich przeznaczeniem (np, DELETE do usunięcia danych, GET do pobrania),
- Uniwersalność odpowiedzi serwisu - możliwe jest użycie tych samych danych wygenerowanych przez serwis do obsługi aplikacji klienckich na różnych urządzeniach (np. przeglądarka i aplikacja mobilna).

#### 4.3.3 Zabezpieczenie danych - API

Większość punktów dostępowych dostępnych w serwisie zabezpieczone jest przy użyciu metody *Basic Auth*. Bez podania loginu i hasła niemożliwy jest dostęp do serwisu. Jedyne dostępne bez konieczności autoryzacji punkty dostępowe to te dotyczące logowania i rejestracji.

#### 4.3.4 Zabezpieczenie danych - baza danych

Do zabezpieczenia danych skorzystaliśmy z symetrycznego szyfrowania. Informacje przechowywane w bazie są niemożliwe do odszyfrowania bez użycia klucza. Dane w punktach dostępowych są odszyfrowane. Odszyfrowywaniem zajmuje się aplikacja odpowiadająca za logikę serwisu.

disease_id	category	description	name
11	228Gvl1ayWi4tTsCOTC9FM/CjltG23rq	4244008	R9lUGdj105rKScEN3qlaGg==
157	Ahs9sivXTZiN8dLSv3E83A==	4244061	/kmRxUSltnHWP78i7+1H7RC8zyOBI8ey
159	dhuvYm7F+pTunxSWW3RLfg==	4244063	EzmYHvY8ean+1wfKmYLhAw==
161	m2OW4a7BKDwnG2SbAic4Mg==	4244065	N6R0HYaM7TbaYXePG12YIA==
162	7+SYxH+dngCVy2KaWrTLdg==	4244066	HRr/QNloPUXUUXphgSxTBw==
164	SMJaP6WUM/cFixoDcWVld1Z5mY0ZXM9i	4244068	Du/7tY/v8kgxb4xZUqWHrA==
167	c3VDamiKoVuPKycddQWj1w==	4244071	hoUoAdqklhwSdpg17tRbeg==
170	WTnmXt5+BVsjxZsyi2QkoQ==	4244074	n018onmCIN8m00rRNyk8rQ==
171	eJzoS/IV3OdnQ/JZuL+l3g==	4244082	LDkinmvzo4S/dppmIW486yxyKuslIRD
172	+sk8GFwMPwPcmEluY35YQ==	4244083	7PHknEwlfTp7oly4Ld8daQ==

See more...

Rysunek 10: Zaszyfrowane krotki bazy danych. Widoczne jest, że ciągi znaków zapisane w bazie danych nie są możliwe do odszyfrowania bez dekodowania.

#### 4.3.5 Ograniczenia

W trakcie implementacji kolejnych funkcjonalności musieliśmy zmierzyć się z ograniczeniami serwera Heroku. Obsługa zapytań (ang. *request*) jest wyko-

nywana na ograniczonym serwerze, stąd można zaobserwować wydłużony czas oczekiwania na duże zapytanie. Skorzystanie z darmowej domeny sprawia, że funkcjonowanie strony jest po prostu wolne.

#### 4.3.6 Przyspieszenie

W trakcie pierwszej wersji implementacji zastosowano wbudowany we framework Spring sterownik będący mostkiem pomiędzy obiektami programu a bazą danych. Jego zastosowanie umożliwiło stosowanie zapytań do bazy przy użyciu specjalnie spreparowanych nazw metod. Wykorzystując ten sposób i np. metodę

```
Optional<Patient> findByUserId(Long userId);
```

automatycznie generuje się kod SQL, który odpowiada za znalezienie użytkownika o zadanym ID.

Jest to bardzo wygodne, jednak korzystając z możliwości zbudowania własnego zapytania przy użyciu języka SQL udało się uzyskać **czterokrotne** przyspieszenie związane ze stosowaniem bardziej złożonych zapytań. Wstrzyknięcie zapytania SQL zaimplementowane jest w następujący sposób:

```
@Query(
value = "select distinct patients.patient_id ,
        my_app_users.* from my_app_users " +
        "join patients\n" +
        "on my_app_users.user_id=patients.user_id",
nativeQuery = true)
List<PatientInfoTO> findAllPatients();
```

#### 4.3.7 Możliwości dotyczące rozwoju - przyspieszenie

Przyspieszenie może zostać uzyskane np. poprzez zastosowanie architektury mikro-serwisowej, tak by każda atomowa operacja mogła zostać wykonywana niezależnie. Zapewniłoby to dużą skalowalność systemu i pod dużym obciążeniem przełożyłoby się to na przyspieszenie.

### 4.4 Warstwa wizualna aplikacji „Karta Pacjenta” (ang. *frontend*)

#### 4.4.1 Działanie warstwy wizualnej

Warstwa wizualna oparta jest na koncepcie reakcyjnego obsługiwanie zdarzeń użytkownika końcowego. Zamiast przeładowania strony po wykonaniu akcji

związanej z zdarzeniem wejścia aplikacji widoku preferowane jest odświeżenie jej części.

#### 4.4.2 Prostota implementacji i wieloplatformowość

Warstwa wizualna oraz warstwa serwisowa zostały oddzielone od siebie podczas implementacji - osobne repozytoria Git oraz osobne kontenery na serwerze. Rozdzielenie aplikacji w ten sposób umożliwia zapewnienie sposobu dostarczania aplikacji zgodnej z Ciągłą Integracją (ang. *Continuous integration*). Umożliwia to nieprzerwane działanie aplikacji oraz odizolowuje pracę nad częścią wizualną od części logiki aplikacji.

Podczas stylowania aplikacji wykorzystano framework Bootstrap. Obsługa akcji wykonywana jest przy użyciu frameworka Angular 7.

#### 4.4.3 Rola warstwy wizualnej aplikacji

Warstwa wizualna aplikacji jest kanałem komunikacji między serwerem a klientem. Taka zależność zapewnia szybkie wykonywanie akcji zadanych przez użytkownika bez znajomości wewnętrznej implementacji serwisu. Podczas tworzenia aplikacji „Karta Pacjenta” starano się, aby wystrój był maksymalnie przejrzysty, wszystkie funkcjonalności opisane i rozmieszczone w jednoznaczny sposób, a komunikacja między serwerem a klientem była jak najszybsza.

### 4.5 Testy obciążeniowe

#### 4.5.1 Testowane zagadnienia

Z uwagi na to, że dostęp do aplikacji powinno mieć w tym samym czasie dziesiątki tysięcy użytkowników (pacjenci i lekarze jednocześnie). Stąd bardzo ważną rzeczą jest obsługa endpointów, które dostarczają odpowiedzi zbiór danych.

Zaimplementowane zostały testy obciążeniowe mające na celu sprawdzenie wydajności naszego serwisu. Skorzystaliśmy z RestTemplate, czyli biblioteki dostępnej we frameworku Spring.

Badane zagadnienia:

1. Obsługa requestów http, metod GET, POST, PUT, DELETE,
2. Weryfikacja statusów HTTP,
3. Czas odpowiedzi serwera,

4. Odporność na żądania wielu (dziesiątki tysięcy) użytkowników.

#### 4.5.2 Wielowątkowość

Wielowątkowość została zaimplementowana po to, aby możliwie najlepiej oddać realny sposób użytkowania. W jednym momencie tysiące użytkowników może zażądać tego samego zbioru danych. Każde z urządzeń powinno zostać poprawnie obsłużone.

#### 4.5.3 Rola testów w rozwoju aplikacji

Testy pomagają zweryfikować, czy postawione endpointy spełniają swoją rolę.

## Literatura

[Walls(2015)] Craig Walls. *Spring in Action, Fourth Edition*. 2015. ISBN 9788328308497.

[Forta(2012)] Ben Forta. *Sams Teach Yourself*. 2012. ISBN 0672336073.

[Krystyna Balińska(2004)] Krzysztof T. Zwierzyński Krystyna Balińska. *Projektowanie algorytmów grafowych*. 2004. ISBN 8371435487.

[Bykowski(2019)] Przemysław Bykowski. Spring boot od podstaw. <https://www.youtube.com/playlist?list=PLUtcRmGoaP27ypMB5aokWbf9KWuWv3UDC/>, 2019.

[Brains(2019)] Java Brains. Spring security. <https://www.youtube.com/playlist?list=PLqq-6Pq4lTTYTEooakHchTGglSvkZAJnE/>, 2019.

[is A RESTful API?(2017)] What is A RESTful API? Traversy media. <https://www.youtube.com/watch?v=Q-BpqyOT3a8/>, 2017.

[in 8 Hours | Angular Tutorial For Beginners(2019)] Angular 8 Full Course Learn Angular in 8 Hours | Angular Tutorial For Beginners. edureka! <https://www.youtube.com/watch?v=oXr4lpXEg1o/>, 2019.