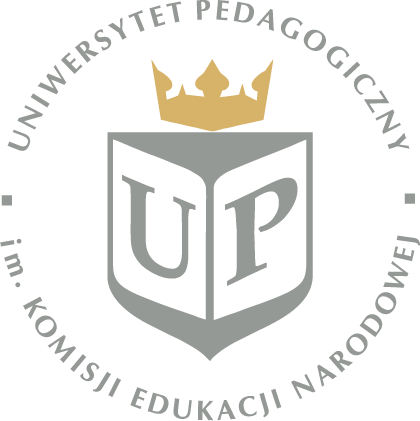
**UNIWERSYTET PEDAGOGICZNY**

im. Komisji Edukacji Narodowej w Krakowie

**INSTYTUT INFORMATYKI**

Kierunek: INFORMATYKA

specjalność: Administracja systemami informatycznymi

**Kacper Rzymkiewicz**

Nr albumu: 137971

**System obsługi praktyk studenckich – aplikacja webowa**

Praca inżynierska napisana pod kierunkiem

**Dr. Wojciech Gwizdała**

**Kraków 2021**

***Streszczenie***

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Fusce est. Vivamus a tellus. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin pharetra nonummy pede. Mauris et orci. Aenean nec lorem.

***Abstract (w języku angielskim)***

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Fusce est. Vivamus a tellus. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin pharetra nonummy pede. Mauris et orci. Aenean nec lorem.

Spis treści

[Wstęp 6](#_Toc55316616)

[Rozdział 1 Specyfikacja wymagań 8](#_Toc55316617)

[1.1 Cel projektu 8](#_Toc55316618)

[1.2 Wymagania funkcjonalne użytkownika 8](#_Toc55316619)

[1.3 Wymagania niefunkcjonalne użytkownika 8](#_Toc55316620)

[1.4 Wymagania niefunkcjonalne systemu 8](#_Toc55316621)

[Rozdział 2 ~~Projektowanie interfejsu aplikacji~~/ Analiza problemu 9](#_Toc55316622)

[2.1 Projekt interfejsu systemu 9](#_Toc55316623)

[2.1.1 Panel rejestracji nowej uczelni 9](#_Toc55316624)

[2.1.2 Panel rejestracji użytkownika 10](#_Toc55316625)

[2.1.3 Panel logowania 10](#_Toc55316626)

[2.2 Diagram tabel 11](#_Toc55316627)

[2.3 Diagram klas 11](#_Toc55316628)

[2.4 Model przypadków użycia 11](#_Toc55316629)

[2.4.1 Aktorzy 11](#_Toc55316630)

[2.4.2 Przypadki użycia 11](#_Toc55316631)

[2.4.2.1 Rejestracja do systemu obsługi praktyk 11](#_Toc55316632)

[2.4.2.2 Logowanie do systemu obsługi praktyk 11](#_Toc55316633)

[2.4.3 Opis scenariuszy 11](#_Toc55316634)

[Rozdział 3. Wykorzystane biblioteki i technologie 14](#_Toc55316635)

[3.1 Środowisko po stronie klienta 15](#_Toc55316636)

[3.1.1 Angular 8 15](#_Toc55316637)

[3.1.2 Node.js 16](#_Toc55316638)

[3.1.3 NPM 16](#_Toc55316639)

[3.1.4 Angular CLI 16](#_Toc55316640)

[3.1.5 Biblioteki zewnętrzne zapewniające wygląd aplikacji 17](#_Toc55316641)

[3.1.6 Opis działania aplikacji Angular 17](#_Toc55316642)

[3.2 Środowisko po stronie back-endu 18](#_Toc55316643)

[3.2.1 Framework Spring Boot 18](#_Toc55316644)

[3.2.2 Maven 19](#_Toc55316645)

[3.2.2.1 Plik POM 19](#_Toc55316646)

[3.2.2.2 Cykl życia projektu 21](#_Toc55316647)

[3.2.3 Spring Security 22](#_Toc55316648)

[3.2.4 JPA 23](#_Toc55316649)

[3.2.4.1 ORM 23](#_Toc55316650)

[3.2.5 Liquibase 24](#_Toc55316651)

[3.3 Relacyjna baza danych – PostgreSQL 25](#_Toc55316652)

[3.4 System kontroli wersji - Git 26](#_Toc55316653)

[3.5 Środowisko programistyczne – InteliJ IDEA 26](#_Toc55316654)

[3.6 Środowisko uruchomieniowe – Docker 27](#_Toc55316655)

[Rozdział 4 Architektura systemu 29](#_Toc55316656)

[4.1.2 Części składowe aplikacji Angular 29](#_Toc55316657)

[4.1.2.1 NgModule 29](#_Toc55316658)

[4.1.2.2 Component 29](#_Toc55316659)

[4.1.2.3 Template 29](#_Toc55316660)

[4.1.2.4 DOM Model 29](#_Toc55316661)

[4.1.2.5 Service 29](#_Toc55316662)

[4.1.2.6 Directive 29](#_Toc55316663)

[4.1.3 Przesyłanie zapytań HTTP wraz z tokenem JWT 30](#_Toc55316664)

[4.1.4 Warstwy aplikacji po stronie serwera 31](#_Toc55316665)

[4.1.5 DTO 31](#_Toc55316666)

[Rozdział 5 Proces powstawania aplikacji 32](#_Toc55316667)

[5.1 Instalacja Maven 32](#_Toc55316668)

[5.2 Konfiguracja bazy danych PostgreSQL 32](#_Toc55316669)

[5.3 Działania po stronie serwera 32](#_Toc55316670)

[5.3.1 Instalacja JVM 32](#_Toc55316671)

[5.3.2 Konfiguracja serwera aplikacji Spring Boot 32](#_Toc55316672)

[5.4 Konfiguracja po stronie klienta 33](#_Toc55316673)

[5.4.1 Instalacja Node.JS 33](#_Toc55316674)

[5.5 Funkcjonalności Angulara 33](#_Toc55316675)

[5.5.1 Routing 33](#_Toc55316676)

[5.5.2 Data binding 33](#_Toc55316677)

[5.5.3 Komunikacja międzykomponentowa 33](#_Toc55316678)

[5.5.4 HttpClient 33](#_Toc55316679)

[5.5.5 i18N – zapewnienie odpowiedniej translacji 33](#_Toc55316680)

[Rozdział 6 Testy funkcjonalne aplikacji 34](#_Toc55316681)

[Zakończenie 36](#_Toc55316682)

[Bibliografia 37](#_Toc55316683)

[Spis rysunków 38](#_Toc55316684)

[Spis wykresów 38](#_Toc55316685)

[Spis tabel 38](#_Toc55316686)

[Spis listingów 38](#_Toc55316687)

[Załączniki 39](#_Toc55316688)

[2.4.4 Formatowanie cytowanego kodu 40](#_Toc55316689)

# Wstęp

Ogólnie coś o studiach, że to etap, kiedy dana osoba zdobywa niezbędną do dalszego życia wiedzę, że przygotowuje się do dalszej pracy i że praktyki są jednym z elementów, które umożliwiają zdobycie tego niezbędnego doświadczenia

Praktyki studenckie to nieodłączny element studiów informatycznych. Stanowią kluczowy etap edukacji, w którym wiedza teoretyczna wykorzystywana jest w praktyce. W czasie ich trwania praktykanci zmagają się z realnymi problemami, nowymi technologiami, a także poznają pracę w zespole. Praktyki pozwalają pozyskać pierwsze doświadczenie komercyjne w branży, dzięki któremu pozycja studenta na rynku pracy staje się lepsza. Zanim jednak praktykant rozpocznie pracę, wymagane jest wypełnienie szeregu dokumentów, a w trakcie trwania praktyk prowadzenie dziennika praktyk. Celem projektu inżynierskiego jest powstanie interaktywnej aplikacji internetowej, która wyjdzie naprzeciw tym trudnościom i ułatwi zarządzanie praktykami studenckimi na wszystkich etapach.

Projekt aplikacji zakłada utworzenie 4 ról, takich jak: praktykant, opiekun praktyk, kierownik praktyk, administrator z ramienia uczelni. Każda z wyżej wymienionych ról posiada uprawnienia, które odpowiadają zadaniom jakie pełnią w trakcie trwania praktyk studenckich.

Aplikacja internetowa

Słownik pojęć:

Framework –

Front-end -

Back-end -

DOPISAĆ COŚ O DOKUMENTACH POTRZEBNYCH W TRAKCIE PRAKTYK, DZIENNIKU PRAKTYK

WSTĘP WSTĘP

WSTĘP

WSTĘP

WSTĘP

WSTĘP

WSTĘP

WSTĘP

WSTĘP

WSTĘP

WSTĘP

WSTĘP

WSTĘP

WSTĘP

WSTĘP

WSTĘP

Słownik pojęć

# Rozdział 1 Specyfikacja wymagań

## Cel projektu

Podstawowym celem pracy jest stworzenie w pełni funkcjonalnej aplikacji internetowej

## Wymagania funkcjonalne użytkownika

## Wymagania niefunkcjonalne użytkownika

## Wymagania niefunkcjonalne systemu

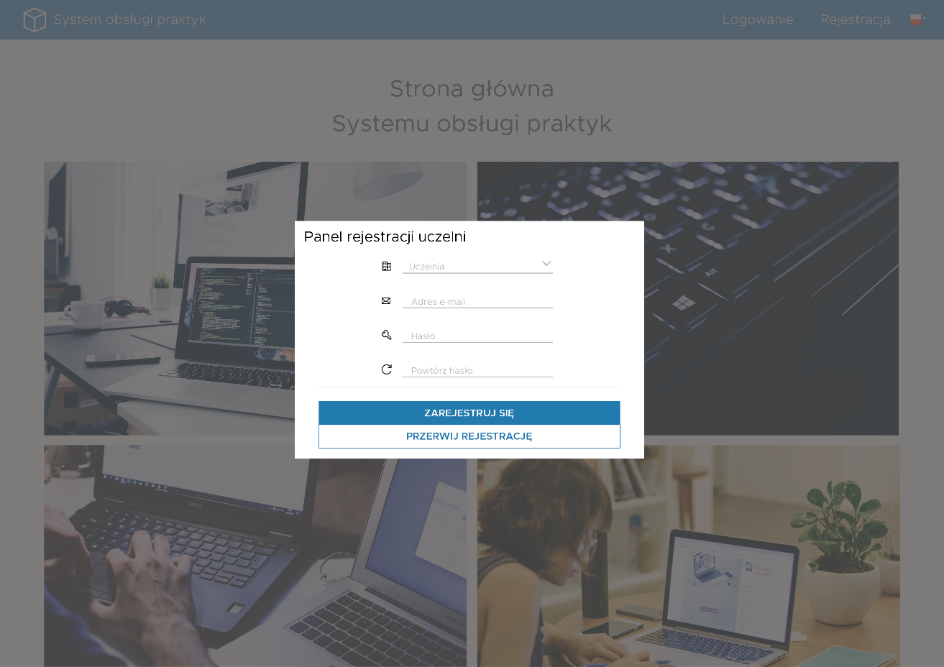
* 1. ~~Wymagania biznesowe~~
  2. ~~Reguły biznesowe~~

# Rozdział 2 ~~Projektowanie interfejsu aplikacji~~/ Analiza problemu

Pierwszym z etapów projektowania aplikacji jest utworzenie rysunków, odzwierciedlających wygląd stron, a także ich podstawowych funkcji. Projekt aplikacji został utworzony w aplikacji Adobe Ilustrator.

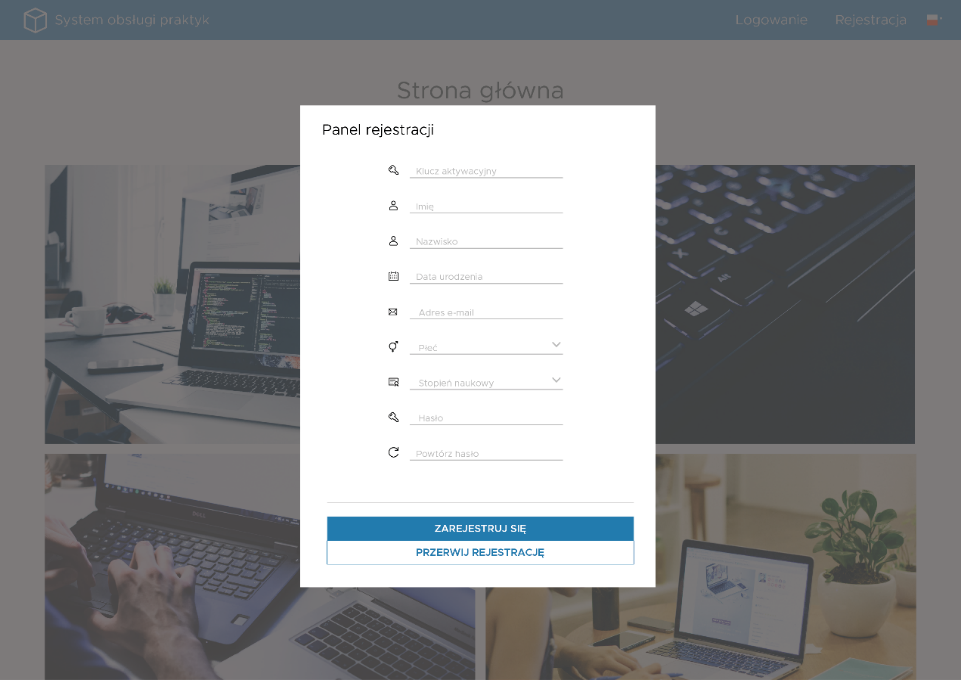
## Projekt interfejsu systemu

## Panel rejestracji nowej uczelni



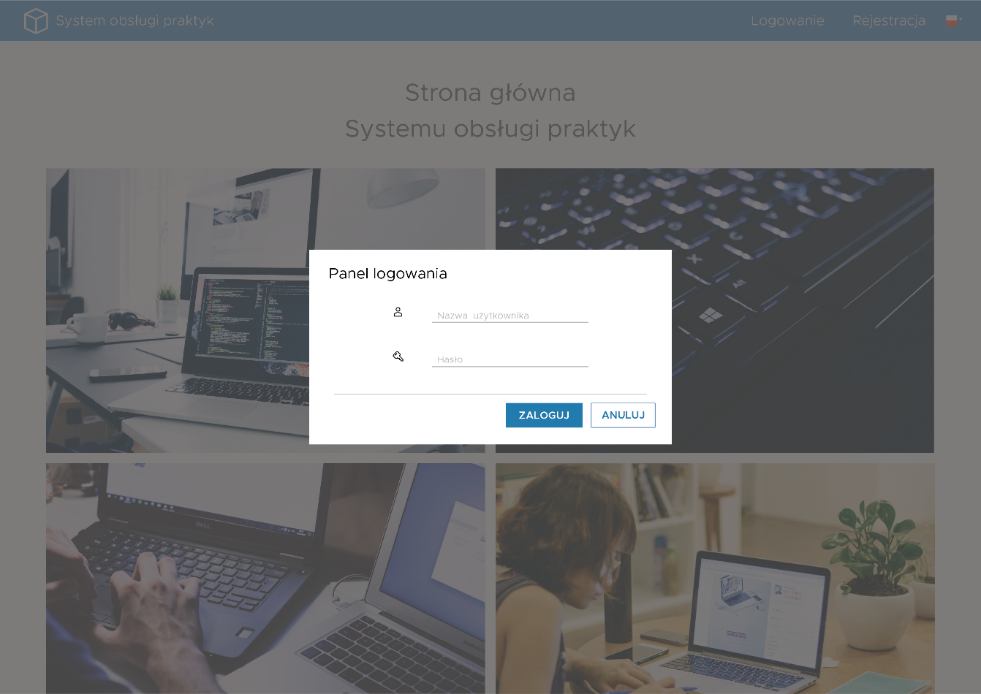
Rysunek 1 Projekt ekranu rejestracji uczelni

## Panel rejestracji użytkownika



Rysunek 2 Projekt panelu rejestracji użytkownika

## Panel logowania



Rysunek 3 Projekt ekranu logowania

## Diagram tabel

## Diagram klas

## Model przypadków użycia

## Aktorzy

## Przypadki użycia

## Rejestracja do systemu obsługi praktyk

## Logowanie do systemu obsługi praktyk

## Opis scenariuszy

# Rozdział 3. Wykorzystane biblioteki i technologie

Po ustaleniu wymagań względem programu i przeprowadzeniu analizy problemu, następnym etapem jest dokonanie wyboru technologii, w których zostanie stworzona aplikacja. Dobór odpowiednich narzędzi ma kluczowe znaczenie i wraz z nim powiązany jest czas potrzebny do utworzenia oprogramowania. Jedne potrafią znacząco przyśpieszyć rozwój, drugie przy braku ich znajomości i zrozumienia, mogą wydłużyć i znacząco utrudnić proces pisania oprogramowania. Z tego względu ważne jest skrupulatne zaplanowanie i przemyślenie tego co programista chce stworzyć. Spośród wielu różnych języków programowania i środowisk programistycznych do stworzenia Systemu Obsługi Praktyk Studenckich zostały wybrane takie, które są: bezpłatne, stale wspierane i rozwijane przez twórców, *open-source*[[1]](#footnote-1), a także wydajne i aktualnie szeroko stosowane. W związku z powyższymi założeniami został wyłoniony następujący *stack* *technologiczny*[[2]](#footnote-2):

* Po stronie klienta zostanie stworzona aplikacja, napisana w następujących językach:
  + HTML 5 – hipertekstowy język znaczników, tworzący szkielet aplikacji,
  + CSS 3 - język opisujący formę prezentacji,
  + TypeScript – język programowania definiujący logikę.

Całość działa w ramach frameworku Angular 8.

* Część serwerowa, będzie obsługiwana za pomocą aplikacji Spring Boot, wraz z technologią Hibernate, bibliotekami Liquibase, Swagger, a także zależnościami Spring m.in.:
  + Spring Security,
  + Spring JDBC,
  + Spring JPA.
* Zarządzanie relacyjną bazą danych, będzie zapewnione przez system PostgreSQL, wraz z programem służącym do zarządzania bazą – pgAdmin.



Rysunek 4 Diagram przedstawiający najważniejsze technologie użyte w projekcie

Źródło: Opracowanie własne

## Środowisko po stronie klienta

## Angular 8

Angular 8 jest to platforma programistyczna stworzony przez programistów Google, która służy do tworzenia aplikacji ładowanych dynamicznie tzw. SPA (Single Page Application). Kod aplikacji, powstał w języku TypeScript, który jest kompilowany do języka JavaScript. Główne zalety Angulara:

* jest to darmowe rozwiązanie, pozwalające tworzyć zaawansowane aplikacje,
* jest stale wspierany przez swoich twórców i nowe wersje zawierające różnego rodzaju poprawki wychodzą dwa razy w roku,
* duża społeczność deweloperów,
* zapewnia własny interfejs wiersza poleceń – Angular CLI, który przy użyciu komend pozwala na przyśpieszenie rozwoju aplikacji,
* zapewnia manager pakietów NPM – domyślny dla środowiska Node.js,
* możliwość wstrzykiwania zależności (ang. dependency injection),
* dwustronne bindowanie danych,
* wykrywanie zmian na stronie (ang. change detection),
* umożliwia implementację routingu aplikacji.

## Node.js

Środowisko Node.js nie jest wymagane w przypadku tworzenia aplikacji Angularowych, warto jednak dodatkowo je skonfigurować, ponieważ zapewnia wiele rowiązań out-of-the-box[[3]](#footnote-3) m.in.: manager pakietów (NPM) i Angular CLI. Dodatkowo umożliwia rozwój aplikacji przy użyciu języka TypeScript – pełni rolę transpilatora z języka JavaScript. Działania procesów transpilacji i kompilacji są podobne – oba polegają na przetworzeniu jednego języka programowania, w drugi. Różnica pomiędzy obiema operacjami polega na tym że transpilacja zamienia kod programu, na kod innego języku programowania, lecz na tym samym poziomie abstrakcji. Kompilacja doprowadza do przetworzenia programu, na język programowania o niższym poziomie abstrakcji.

## NPM

Npm (ang. Node Package Manager) – manager pakietów, który ułatwia zarządzanie wersjami, jak i samą instalacją zewnętrznych bibliotek. Obsługiwany za pomocą komend. Przykładowe polecenia:

* npm install @clr/icons @clr/ui @clr/angular – instalacja biblioteki Clarity Design System,
* npm update –g – aktualizacja wszystkich globalnych pakietów,
* npm ls – wypisuje w konsoli, wszystkie zainstalowane moduły.

## Angular CLI

Angular CLI - interfejs wiersza poleceń, którego głównym zadaniem jest ułatwienie konfiguracji nowego projektu Angular - za pomocą jednej komendy umożliwia utworzenie i konfigurację aplikacji. Dodatkową zaletą jest możliwość generowania szablonów składowych frameworku, takich jak:

* component,
* service,
* enum
* interface.

## Biblioteki zewnętrzne zapewniające wygląd aplikacji

Głównym celem stosowania bibliotek zewnętrznych, jest przyśpieszenie i ułatwienie procesu developmentu[[4]](#footnote-4) poprzez zastosowanie gotowych rozwiązań. W Systemie Obsługi Praktyk Studenckich zostały wykorzystane pakiety takie jak: *Clarity Desing System*[[5]](#footnote-5), a także *PrimeNg*[[6]](#footnote-6). Zapewniają wysokiej klasy komponenty, zaprojektowane przez ekspertów w dziedzinie tworzenia przejrzystych i poprawnych wizualnie stron internetowych.

## Opis działania aplikacji Angular

Utworzenie nowej aplikacji Angular, może być w znacznej części ułatwione po zainstalowaniu środowiska Node.js, które domyślnie zapewnia system nadzoru nad bibliotekami zewnętrznymi – NPM. Za pomocą komendy *npm install -g angular/cli*, instalujemy narzędzie Angular CLI, które dostarcza dostęp do konsoli wiersza poleceń Angular - zapewniającej instrukcje przydatne w pracy nad projektem. Po zainstalowaniu, dostępna jest komenda *ng new nazwa\_projektu,* która powoduje wygenerowanie struktury i domyślne konfigurację nowego projektu Angular. W momencie, w którym projekt zostanie pomyślnie utworzony, możliwe jest uruchomienie go przy pomocy polecenia *ng serve.* Po wykonaniu tej komendy, domyślna przeglądarka powinna wyświetlić przykładową stronę dla nowo wygenerowanego projektu na domenie: <http://localhost:4200>. Dodatkowo jeśli uruchomimy projekt za pomocą polecenia *ng serve –lr*, każda zmiana w kodzie aplikacji, spowoduje odświeżenie przeglądarki i wyświetlenie modyfikacji.

## Środowisko po stronie back-endu

## Framework Spring Boot



Rysunek 5 Logo frameworku Spring Boot

Źródło: <https://bgasparotto.com/change-spring-boot-embedded-container>

Spring Boot jest wolnodostępnym frameworkiem, bazujący na języku programowania Java. Klasyczny przykład rozwiązania convention-over-configuration[[7]](#footnote-7), które zapewnia szybki rozwój aplikacji i dostarczenie podstawowych funkcjonalności zdefiniowanych domyślnie. Koncepcja ta wpływa pozytywnie na szybkość powstawania aplikacji i zmniejsza ilość kodu, który musi zostać napisany, do absolutnego minimum, dzięki temu, że posiada zbiór podstawowych bibliotek, a także wstępną konfiguracje. Dodatkową zaletą frameworku Spring Boot jest wbudowany serwer aplikacyjny Tomcat. W związku z powyższym uruchomienie serwera aplikacji jest bardzo proste i nie jest wymagana od programisty zaawansowanej wiedza na temat działania serwera. Jest to dobre rozwiązanie zwłaszcza dla początkujących programistów. Framework Spring Boot wraz z dodatkowymi bibliotekami i narzędziami takimi jak: Hibernate, Liquibase, Maven, Tomcat, stanowi stabilną bazę do rozwoju aplikacji back-endowej, umożliwiającej m.in.: przetwarzanie, odczyt informacji z bazy danych, zapis informacji do bazy danych, autentykację, autoryzację, wysyłanie maili, a także obsługę zapytań HTTP. W związku z powyższym aplikacja back-endowa oparta na frameworku Spring Boot może stanowić wielozadaniowe narzędzie obsługujące przepływ danych z wielu źródeł.

### Maven

Jednym z podstawowych narzędzi, używanych w projekcie jest Maven, którego głównym zadaniem jest wsparcie w zarządzaniu projektem. Dodatkowym atutem jest standaryzacja i automatyzacja procesu budowania i kompilacji aplikacji, a także możliwość wygenerowania dokumentacji na podstawie kodu projektu i spakowania projektu w formacie .*jar*, bądź .*war*. Kolejna funkcja, którą zapewnia Maven to możliwość dodawania zależności do projektu.

### Plik POM

POM (ang. Project Object Model) to główny plik projektu zarządzanego przez Maven - zdefiniowany w formacie .*xml*.

1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5. https://maven.apache.org/xsd/maven-4.0.0.xsd">
6. <modelVersion>4.0.0</modelVersion>
7. <packaging>pom</packaging>
8. <modules>
9. <module>sop-web</module>
10. <module>sop-server</module>
11. </modules>
12. <parent>
13. <groupId>org.springframework.boot</groupId>
14. <artifactId>spring-boot-starter-parent</artifactId>
15. <version>2.2.2.RELEASE</version>
16. <relativePath/> <!-- lookup parent from repository -->
17. </parent>
18. <groupId>pl.sop</groupId>
19. <artifactId>sop</artifactId>
20. <version>0.0.1-SNAPSHOT</version>
21. <name>sop</name>
22. <description>System obsługi praktyk</description>
23. <properties>
24. <java.version>1.8</java.version>
25. </properties>
26. <dependencies>
27. <dependency>
28. <groupId>org.springframework.boot</groupId>
29. <artifactId>spring-boot-starter-actuator</artifactId>
30. </dependency>
31. <dependency>
32. <groupId>org.springframework.boot</groupId>
33. <artifactId>spring-boot-starter-data-jpa</artifactId>
34. </dependency>
35. </dependencies>
36. </project>

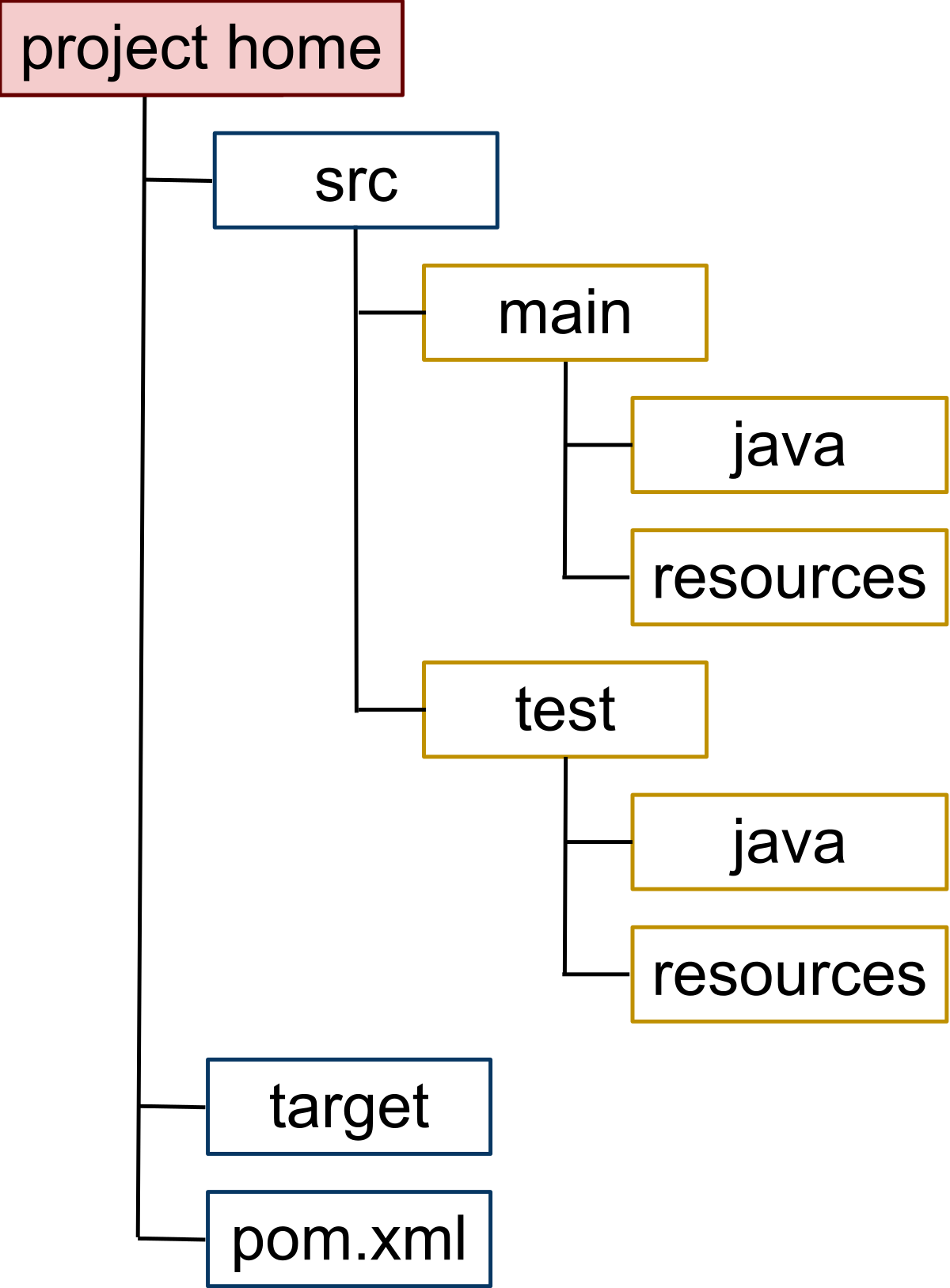
Listing 1 Plik POM.xml

Źródło: opracowanie własne

Plik *pom.xml* Przechowuje informacje o metadanych projektu, jest także miejscem gdzie znajdują się szczegóły dotyczące:

* zastosowanych zależności,
* użytych wtyczek,
* programistów i twórców projektu,
* zasobów projektu,
* modułów wchodzących w skład aplikacji.

Dodatkowo Maven podczas inicjalizacji generuje podstawową strukturę projektu:



Rysunek 6 Diagram przedstawiający podstawową strukturę projektu Maven

Źródło: <https://commons.wikimedia.org/wiki/File:Maven_CoC.svg>

### Cykl życia projektu

W trakcie rozwoju oprogramowania można wyróżnić poszczególne cykle życia. Każdy z nich jest odpowiedzialny za inną operację i są wykonywane kaskadowo. Poniżej najważniejsze pozycje:

* Validate – zadaniem operacji *validate* [[8]](#footnote-8)jest sprawdzenie czy są dostępne wszystkie potrzebne informacje do zbudowania aplikacji,
* Compile – kompilacja kodu źródłowego projektu,
* Test-compile – kompilacja kodu źródłowego testów,
* Test – uruchomienie testów jednostkowych,
* Package – pakowanie skompilowanego kodu źródłowego do łatwego w dystrybucji formatu np. .*jar*, .*war*.
* Integration-test – przetwarzanie i wdrażanie spakowanego projektu, w celu uruchomienia testów integracyjnych.
* Verify – weryfikacja testów integracyjnych,
* Install – umieszczenie spakowanego projektu w lokalnym repozytorium,
* Deploy – umieszczenie spakowanego projektu w zdalnym repozytorium.

Jak wynika z wyżej wymienionych punktów, w pracy programisty języka Java, technologia Maven jest przydatnym narzędziem automatyzującym wszelkie fazy cyklu rozwoju projektu.

### Spring Security

Spring Security – narzędzie wspierające obsługę autoryzacji i uwierzytelniania aplikacji Spring Boot. Zapewnia gotowe mechanizmy, dzięki czemu ilość kodu jest minimalna i może zawierać się w kilku metodach.

1. protected void configure(HttpSecurity http) throws Exception {
2. http.cors().and().csrf().disabled()
3. .exceptionHandling()
4. .authenticationEntryPoint(unauthorizedHandler).and()
5. .sessionManagement()
6. .sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
7. .authorizeRequests()
8. .antMatchers("/api/auth/\*\*").permitAll()
9. .antMatchers("/api/available-colleges/\*\*").permitAll()
10. .anyRequest().authenticated();
11. }

Listing 2 Metoda zawierająca konfigurację Spring Security

Źródło: opracowanie własne

Powyżej przedstawiona metoda, zapewnia w Systemie Obsługi Praktyk Studenckich ochronę dostępnych endpointów[[9]](#footnote-9), przed próbą pobrania, bądź przesłania danych przez niezautoryzowanego użytkownika (linia nr.10). Wyjątek stanowią dwa adresy URL – „api/auth/\*” i „api/available-colleges/\*”, które odpowiednio zapewniają obsługę logowania, rejestracji i zwracają listę dostępnych uczelni, które nie zostały jeszcze zarejestrowane w systemie. Dodatkowo możliwe jest zabezpieczenie, każdego z punktów dostępu danych z osobna. Przykładowe rozwiązanie poniżej:

1. @GetMapping(value = "/api/college-structure")
2. @PreAuthorize("hasRole('ROLE\_ADMIN')")
3. public ResponseEntity<CollegeStructureDTO> getCollegeStructure(Authentication authentication) {
4. // ciało metody
5. }

Listing 3 Zabezpieczenie metody przyjmującej zapytania http

Źródło: opracowanie własne

W momencie wysłania zapytania GET pod endpoint /api/college-structure, sprawdzane są role jakie posiada użytkownik próbujący pozyskać dane. Jeżeli brak wymaganych uprawnień, zostanie zwrócony kod http 401 - Unauthoried[[10]](#footnote-10) i nie zostaną zwrócone żądane dane. W przypadku jeśli użytkownik ma przyznaną wymaganą rolę, system przepuści żądanie i rozpocznie wykonywanie metody przypisanej do wykonania pod danym adresem.

### JPA

JPA (Java Persistence API) jest specyfikacją zawierającą zbiór interfejsów i definicji dotyczących mapowania obiektowo-relacyjnego.

### ORM

ORM (ang. Object-Relational Mapping) – odwzorowanie obiektowo-relacyjne zdefiniowane przez JPA jest techniką przekształcającą obiekty znajdujące się w bazie danych w encje zdefiniowane w kodzie aplikacji. Wraz z kolumnami odwzorowane mogą zostać również relacje bazy danych. Dzięki takiemu podejściu możliwe jest zmniejszenie nakładu kodu potrzebnego do zdefiniowania. Kod encji wykorzystywanej w Systemie Obsługi Praktyk Studenckich poniżej:

1. @Entity
2. @Table(name = "Colleges")
3. public class College extends BasicEntity implements Serializable {
4. // Ciało klasy
5. }

Źródło: opracowanie własne

W przypadku odwzorowywania obiektów bazy danych, na encje napisane w języku programowania, bardzo ważnym elementem są adnotacje @Entity i @Table. Obie definiowane przez specyfikację JPA. Pierwsza informuje jaką rolę pełni dana klasa – encja. Druga natomiast dostarcza wiadomości na temat tabeli jaką dana encja odzwierciedla.

* + - 1. Adnotacje dostarczane przez JPA

JPA zapewnia o wiele większy wachlarz adnotacji. Najczęściej używane pozycje, wraz z definicją:

* @Id – określa, które pole w klasie przechowuje identyfikator. Adnotacja wymagana w każdej encji,
* @Column – definiuje relację pomiędzy polem encji i kolumną znajdującą się w bazie danych,
* @OneToOne – używane do odwzorowania relacji jeden-do-jednego z bazy danych
* @OneToMany – zastosowanie znajduje w przypadku relacji jeden-do-wielu
* @ManyToMany – odwzorowuje relację wiele-do-wielu występującą w bazie danych
* @ManyToOne – określa występowanie relacji wiele-do-jednego.
  + - 1. JPQL

JPQL (Java Persistence Query Language) – obiektowo zorientowany język zapytań, mocno zbliżony do SQL. Różnica polega na operowaniu na obiektach, w przeciwieństwie do działania na tabelach. Zdefiniowany w specyfikacji JPA. Kod zapytania JQPL:

1. @Query(value = "select u from User u "
2. + " left join fetch u.colleges colleges"
3. + " left join fetch u.faculties faculties"
4. + " left join fetch u.institutes institutes "
5. + " left join fetch u.departments departments "
6. + " where u.username = :username and u.active = true")
7. User findByUsername(@Param("username") String username);

Listing 4 Kod zapytania JPQL

Źródło: opracowanie własne

### Liquibase

Liquibase jest to biblioteka, która wspiera dokumentowanie rozwoju bazy danych. Jest ona dostępna jako zależność, którą można dodać do pliku pom.xml. Jej koncepcja polega na zapisywaniu w pliku, bądź w wielu plikach wszystkich zmian, które zostały wprowadzone w bazie danych. Mogą one mieć rozszerzenie: .*xml*, .*yaml*, .*json*, bądź .*sql.* Dzięki temu zabiegowi struktura bazy danych jest jednoznacznie zdefiniowana i bez problemu może zostać odtworzona na innym urządzeniu. W tym celu wystarczy posiadać skonfigurowane połączenie z bazą danych i uruchomić aplikację Spring Boot. Na podstawie wpisów w plikach, baza zaktualizuje się i utworzy wszelkie zdefiniowane struktury i rekordy. Każdy nowy wpis w pliku zarządzającym strukturą bazy definiujemy za pomocą słowa kluczowego „changeSet”, wraz z informacją o autorze i numerem identyfikacyjnym wpisu. Przykładowo wpis, którego zadaniem będzie utworzenie nowej tabeli, w ciele znacznika „changeSet” zostanie rozpoczęty przez kolejny tag - „<createTable name=”users”>”, gdzie właściwość „name” to nazwa tabeli. Następnie wewnątrz poprzedniego znacznika, zdefiniowane zostaną kolumny, wchodzące w skład danej tabeli. Definiowanie nowej kolumny rozpoczyna słowo kluczowe „column”, wraz z nazwą i typem danych przechowywanych w kolumnie. Składnia biblioteki jest przejrzysta i umożliwia łatwe definiowanie struktury bazy danych. Rozwiązanie zapewnia przenoszalność kodu i w przypadku pracy kilku osób nad kodem aplikacji ułatwia synchronizację bazy danych. Przykładowy wpis definiujący tabelę w bazie danych poniżej:

1. <createTable tableName="users">
2. <column name="id" type="bigint" autoIncrement="true">
3. <constraints primaryKey="true" nullable="false"/>
4. </column>
5. <column name="login" type="varchar(50)">
6. <constraints unique="true" nullable="false"></constraints>
7. </column>
8. <column name="password" type="varchar(60)"></column>
9. <column name="first\_name" type="varchar(50)"></column>
10. <column name="last\_name" type="varchar(50)"></column>
11. <column name="birth\_date" type="date"></column>
12. <column name="email" type="varchar(50)">
13. <constraints unique="true" nullable="false"></constraints>
14. </column>
15. <column name="phone " type="varchar(15)"></column>
16. <column name="academic\_title" type="varchar(50)"></column>
17. <column name="activation\_key" type="varchar(50)"></column>
18. <column name="active" type="boolean"></column>
19. <column name="deleted" type="boolean"></column>
20. </createTable>

Listing 5 Plik definiujący nową tabelę w formacie .xml

Źródło: Opracowanie własne

Praca z biblioteką Liquibase jest prosta i w przypadku Systemu Obsługi Praktyk Studenckich jest oparta na plikach o rozszerzeniu .*xml*. Składnia zależności jest czytelna, używane są jednoznaczne nazwy, które w przejrzysty sposób definiują dane operacje wykonywane w bazie danych. Narzędzie jest kompatybilne z najpopularniejszymi systemami bazodanowymi takimi jak MySQL, Oracle, SQL Server, a także PostgreSQL, na którym oparty jest bieżący projekt.

## Relacyjna baza danych – PostgreSQL

Baza danych Systemu Obsługi Praktyk Studenckich obsługiwana jest za pomocą systemu PostgreSQL, którego największa zaletą jest stabilność i wydajność. Dodatkowo platforma będąc obiektowo-relacyjnym systemem pozwala na tworzenie własnych typów danych.

* + 1. pgAdmin

Razem z systemem zarządzania relacyjnymi bazami danych zapewniony jest program przeznaczony do zarządzania i pracy z bazą danych.

## System kontroli wersji - Git

System Obsługi Praktyk Studenckich korzysta z systemu kontroli wersji Git, wraz z serwisem internetowym GitHub. Zapewnia graficzny interfejs, a także wiele funkcji, które ułatwiają pracę nad projektem, m.in.:

* Śledzenie zmian dokonanych w kodzie,
* Możliwość tworzenia dokumentacji,
* Wsparcie dla rozgałęzień projektu (ang. branching),
* Obsługa graficzna łączenia gałęzi projektu (ang. merging).

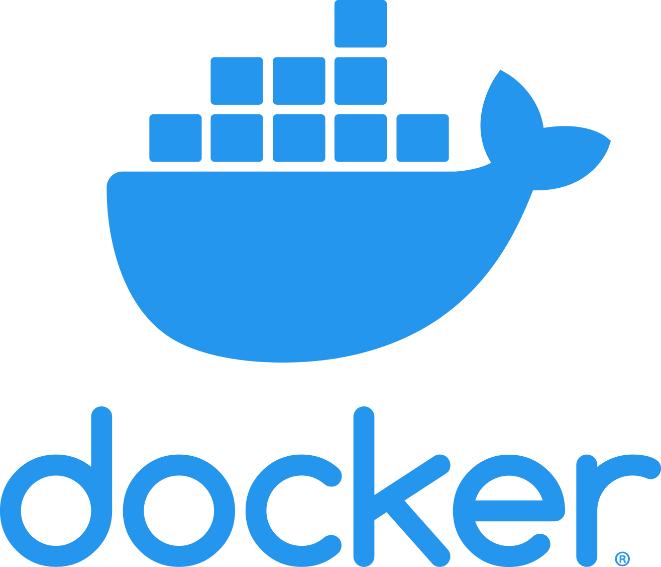
## Środowisko programistyczne – InteliJ IDEA

InteliJ IDEA to zintegrowane środowisko programistyczne (IDE[[11]](#footnote-11)). Producentem oprogramowania jest firma JetBrains, posiadająca główną siedzibę w Czechach, a dokładniej w stolicy – Pradze. Pierwsza wersja oprogramowania wyszła w styczniu 2001 roku. Główne zalety środowiska to:

* narzędzie do refaktoringu kodu [[12]](#footnote-12),
* obsługa narzędzi automatyzujących budowanie aplikacji np. Maven,
* graficzna obsługa systemu kontroli wersji Git, SVN, Mercurial,
* obsługa głównych serwerów aplikacji tj: Tomcat, JBoss, Glassfish,
* wsparcie popularnych frameworków,
* szeroka gama dostępnych języków programowania,
* wbudowany terminal,
* przejrzystość,
* stabilność.

W związku z powyższymi zaletami InteliJ IDEA jest obecnie najlepszym IDE na rynku i oferuje najnowocześniejsze rozwiązania ułatwiające i wspomagające rozwój oprgramowania.

## Środowisko uruchomieniowe – Docker



Rysunek 7 Logo narzędzia Docker

Źródło:https://www.docker.com/sites/default/files/d8/2019-07/vertical-logo-monochromatic.png

System Obsługi Praktyk Studenckich stanowią dwie aplikacje. Pierwsza odpowiedzialna za front-end, oparta o framework Angular, druga odpowiedzialna za back-end stworzona w ramach oprogramowania Spring Boot. Wdrożenie systemu powinno być w największym stopniu zautomatyzowane. Naprzeciw temu wymaganiu staje narzędzie Docker, które zapewnia przenoszalność aplikacji na inne urządzenia i systemy operacyjne.



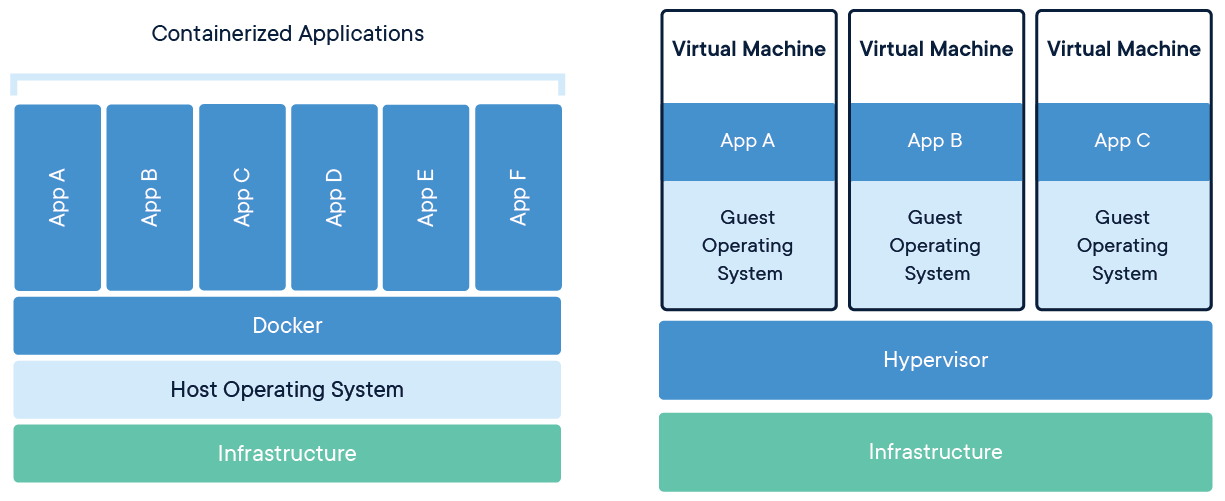
Rysunek 8 Schemat przedstawiający wieloplatformowość Dockera

Źródło:https://www.docker.com/sites/default/files/d8/styles/large/public/2018-11/Docker-Website-2018-Diagrams-071918-V5\_26\_Docker-today.png?itok=-N\_HRaz2

Pozostałe zalety rozwiązania to wydajność, która wynika z zasady działania kontenerów, w których uruchamiana jest aplikacja. Warto również wspomnieć o skalowalności, która w tym przypadku jest zapewniona i w razie potrzeby rozbudowania aplikacji, dodanie i uruchomienie nowego komponentu nie stanowi problemu i polega na dodaniu nowego kontenera.

* + 1. Konteneryzacja, a wirtualizacja

Pojęcie konteneryzacji w kontekście informatyki występuje stosunkowo od niedawna i oznacza umieszczenie aplikacji, wraz z procesami, zależnościami i konfiguracją, w wirtualnym „opakowaniu”, zwanym kontenerami. Każdy z nich zawiera osobno wydzielony obszar pamięci RAM, pamięci na dysku i prywatny adres IP. Różnica pomiędzy konteneryzacją, a wirtualizacją polega na sposobie uruchomienia i zawartości poszczególnych instancji. Maszyny wirtualne zawierają kopię systemu operacyjnego, pliki binarne, biblioteki, które z racji swojego rozmiaru mogą uruchamiać się długo. Podstawowe zalety stosowania konteneryzacji to lekkość obrazu kontenera i szybkość uruchomienia, które wynikają z jego struktury. W ten sposób wykorzystują mniejszą ilość zasobów, przez co możliwe jest uruchomienie większej ilości aplikacji przy posiadaniu pewnych ograniczeń sprzętowych, niż w przypadku maszyn wirtualnych.



Rysunek 9 Porównanie działania konteneryzacji i wirtualizacji

Źródło: https://www.docker.com/sites/default/files/d8/2018-11/docker-containerized-and-vm-transparent-bg.png

# Rozdział 4 Architektura systemu

Komunikacja pomiędzy aplikacjami wchodzącymi w skład Systemu Obsługi Praktyk opiera się na przesyle zapytań HTTP z klienta – przeglądarki internetowej, do serwera – aplikacji Spring Boot, wystawiającej interfejs programistyczny aplikacji (ang. Application Programming Interface) – API.



Rysunek 10 Schemat przepływu danych

*Źródło: https://phpenthusiast.com/theme/assets/images/blog/what\_is\_rest\_api.png?021019a*

* + 1. Wzorzec MVC po stronie klienta

## Części składowe aplikacji Angular

## NgModule

## Component

## Template

## DOM Model

## Service

## Directive

## Przesyłanie zapytań HTTP wraz z tokenem JWT

## Warstwy aplikacji po stronie serwera

## DTO

# Rozdział 5 Proces powstawania aplikacji

## Instalacja Maven

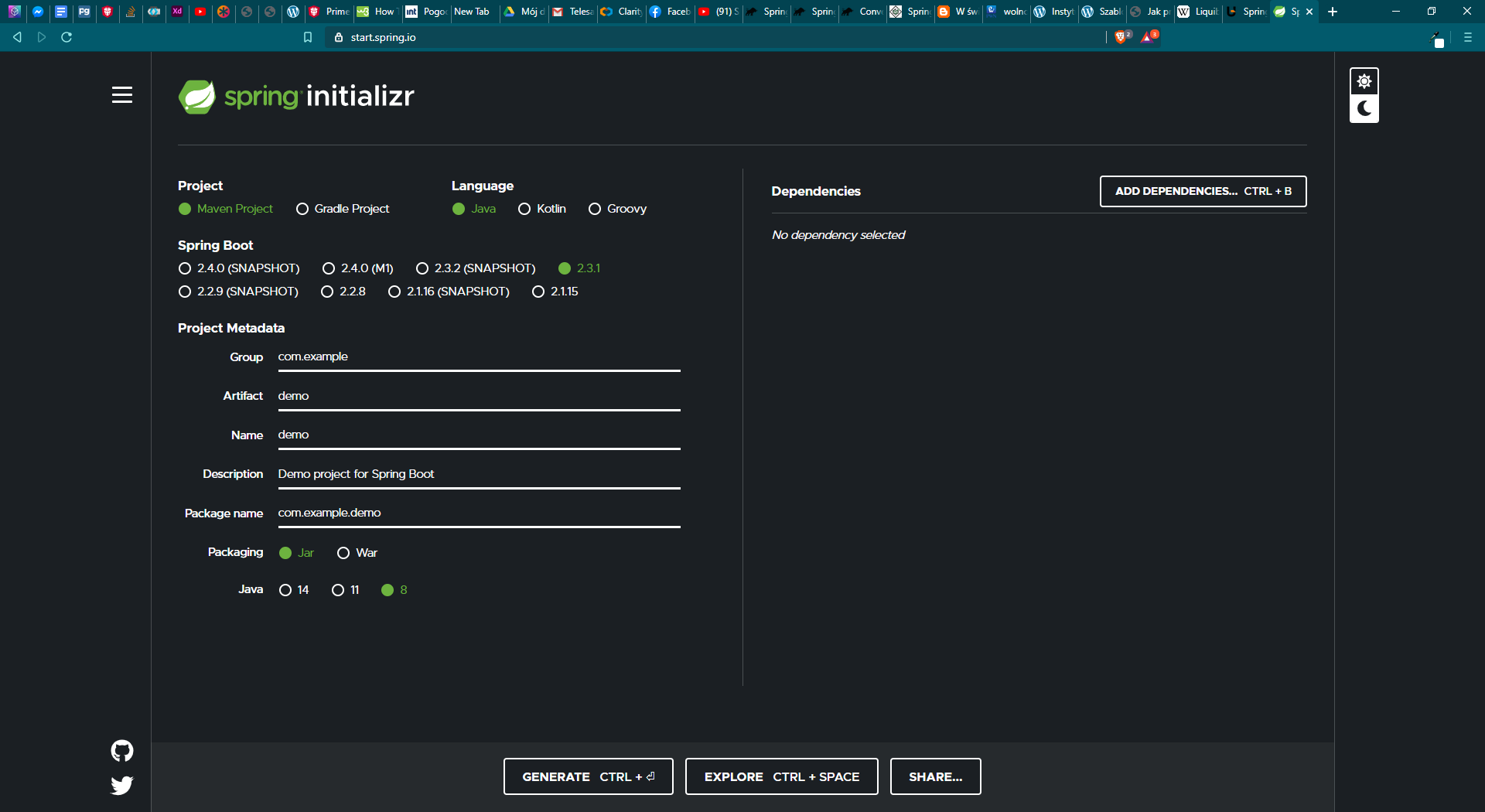
## Konfiguracja bazy danych PostgreSQL

## Działania po stronie serwera

## Instalacja JVM

## Konfiguracja serwera aplikacji Spring Boot

Aby stworzyć własny lokalny serwer aplikacji Spring Boot należy udać się pod adres WWW*:* [*https://start.spring.io/*](https://start.spring.io/), na którym znajduje się generator aplikacji Spring Boot. Na stronie istnieje możliwość wyboru wersji aplikacji Spring Boot, języka programowania, dopisania metadanych aplikacji, a także dobrania właściwych zależności potrzebnych w projekcie, które automatycznie zostaną dowiązane do aplikacji.



Rysunek 2 Strona z generatorem aplikacji Spring Boot

*Źródło:* [*https://start.spring.io/*](https://start.spring.io/)

W celu zakończenia generowania nowej aplikacji, należy wybrać przycisk *Generate.* Po tej czynności pojawi się okno dotyczące miejsca zapisu skompresowanej, nowo powstałej aplikacji. Należy wybrać miejsce docelowe dla pliku, następnie go rozpakować i zaimportować w dostosowanym do języka Java IDE[[13]](#footnote-13). Wyżej opisany proces zapewnia wygenerowanie podstawowej aplikacji Spring Boot gotowej do dalszej implementacji.

## Konfiguracja po stronie klienta

## Instalacja Node.JS

## Funkcjonalności Angulara

## Routing

## Data binding

## Komunikacja międzykomponentowa

## HttpClient

## i18N – zapewnienie odpowiedniej translacji

# Rozdział 6 Testy funkcjonalne aplikacji

# Zakończenie

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urna. Nunc viverra imperdiet enim. Fusce est.

Vivamus a tellus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin pharetra nonummy pede. Mauris et orci. Aenean nec lorem.

In porttitor. Donec laoreet nonummy augue. Suspendisse dui purus, scelerisque at, vulputate vitae, pretium mattis, nunc. Mauris eget neque at sem venenatis eleifend. Ut nonummy.

Fusce aliquet pede non pede. Suspendisse dapibus lorem pellentesque magna. Integer nulla. Donec blandit feugiat ligula. Donec hendrerit, felis et imperdiet euismod, purus ipsum pretium metus, in lacinia nulla nisl eget sapien.

Donec ut est in lectus consequat consequat. Etiam eget dui. Aliquam erat volutpat. Sed at lorem in nunc porta tristique. Proin nec augue.

# Bibliografia

|  |  |
| --- | --- |
| [1] | F. Chollet, Deep Learning. Praca z językiem Python i biblioteką Keras, Gliwice: Helion, 2019. |
| [2] | P. Drygaś, S. Gluzman, V. Mityushev i P. Nawalaniec, „Effective elastic constants of hexagonal array of soft fibers,” *Computational Materials Science,* nr 139, pp. 395-405., 2017. |

Można cytować strony internetowe – podając pełną ścieżkę i datę uzyskania informacji

[3] Zagadnienia egzaminacyjne, <http://www.ii.up.krakow.pl/egzamin_dypl.html> [dostęp:18.05.2020]

Bibliografia spis własny:

https://www.docker.com/resources/what-container

https://angular.io/guide/architecture

Spis rysunków, tabel, czy listingów powinien być również generowany automatycznie.

# Spis rysunków

[Rysunek 1 Projekt ekranu rejestracji uczelni 9](#_Toc54562934)

[Rysunek 2 Projekt panelu rejestracji użytkownika 10](#_Toc54562935)

[Rysunek 3 Projekt ekranu logowania 11](#_Toc54562936)

[Rysunek 4 Logo frameworku Spring Boot 19](#_Toc54562937)

[Rysunek 5 Diagram przedstawiający podstawową strukturę projektu Maven 21](#_Toc54562938)

# Spis wykresów

[Wykres 1 Przykładowy wykres 6](#_Toc41414970)

# Spis tabel

[Tabela 1 Tytuł przykładowej tabeli 6](#_Toc41420738)

# Spis listingów

[Listing 1 Plik POM.XML 20](#_Toc54642684)

[Listing 2 Metoda, zawierająca konfigurację Spring Security 23](#_Toc54642685)

[Listing 3 Zabezpieczenie metody przyjmującej zapytania http 23](#_Toc54642686)

[Listing 3 Zabezpieczenie metody przyjmującej zapytania http 24](#_Toc54642687)

[Listing 4 Listing w formie bezpośredniego wpisu w pracy 38](#_Toc54642688)

[Listing 5 Listing w formie „zrzut z ekranu” 38](#_Toc54642689)

W niektórych przypadkach, dla zachowania czytelności pracy, pewne jej elementy, jak na przykład obszerniejsze części kodu, dodatkowe obliczenia lub inne materiały badawcze, warto umieścić w formie załączników.

# Załączniki

1. Tytuł pierwszego załącznika, np.: Kod biblioteki obsługującej mechanizm komunikacji między użytkownikami.

2. Tytuł drugiego załącznika, np.: Opis metodyki wyznaczania mglistości języka pracy dyplomowej.

### Formatowanie cytowanego kodu

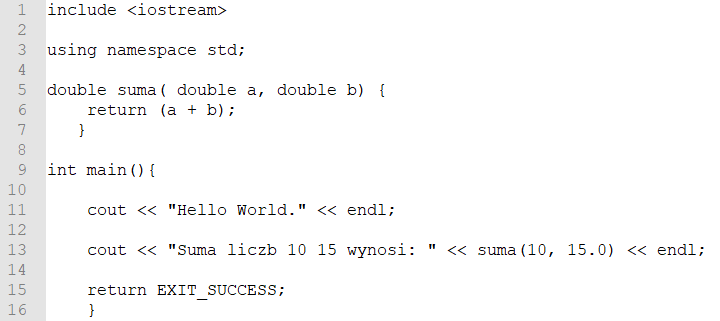
Fragmenty kodu programu muszą mieć numerowane linie. Kod należy pisać bezpośrednio w pracy, czcionką „Courier New” o rozmiarze 10, obramowując cały fragment kodu pojedynczą ramką.

1. include <iostream>
2. using namespace std;  
   f
3. double suma( double a, double b){
4. return (a + b);
5. }
6. int main(){
7. cout << "Hello World." << endl;
8. cout << "Suma liczb 10 i 15 wynosi: " << suma(10.0, 15.0) << endl;
9. return EXIT\_SUCCESS;
10. }

Listing 6 Listing w formie bezpośredniego wpisu w pracy

Źródło: opracowanie własne lub odwołanie do pozycji bibliograficznej

Uwaga: kody źródłowe w postaci zrzutu ekranu, czyli w formie obrazu rastrowego nie są zalecane. **W przypadku kilku takich obiektów nie da się dobrać rozdzielczości**, tak aby były zachowane parametry tekstu. Fragment kodu można zamieścić jako zrzut z ekranu tylko w wyjątkowych sytuacjach, gdy istnieje pewność, iż będą przedstawione czytelnie i este­tycznie.



Listing 7 Listing w formie „zrzut z ekranu”

Źródło: opracowanie własne lub odwołanie do pozycji bibliograficznej

Niezależnie od formy wstawienia fragmentów kodu źródłowego, jeżeli nie jest to pojedyncze wystąpienie w całej pracy, należy je podpisać zgodnie z formatem podpisu rysunku, jednakże wykorzystując etykietę Listing. Dobrą praktyką jest używanie numeracji linii jak pokazano w Listing 1 i Listing 2, ułatwia to późniejsze odwoływanie się do użytych fragmentów kodu.

1. Open-source - z angielskiego: wolnodostępne – nie są pobierane żadne opłaty, a kod jest ogólnodostępny [↑](#footnote-ref-1)
2. Stack technologiczny – z angielskiego: stos. Jest to zbiór technologii wykorzystywanych, do tworzenia aplikacji [↑](#footnote-ref-2)
3. Out-of-the-box – (z ang.) – możliwość uzyskania pewnych funkcji, możliwości programu komputerowego bez konieczności dodatkowej instalacji. [↑](#footnote-ref-3)
4. Development(ang.) - rozwój [↑](#footnote-ref-4)
5. Clarity design system - [↑](#footnote-ref-5)
6. PrimeNg - [↑](#footnote-ref-6)
7. Convention-over-configuration – koncepcja, która zakłada użycie gotowych rozwiązań, niewymagających zbytecznej pracy nad kwestiami, które w większości przypadków pojawiają się jako domyślne. [↑](#footnote-ref-7)
8. Validate – (z ang. sprawdzać poprawność) [↑](#footnote-ref-8)
9. Endpoint – (ang. [↑](#footnote-ref-9)
10. Unauthoried – (z ang.) nieautoryzowany dostęp do zasobów [↑](#footnote-ref-10)
11. IDE – od ang. integrated development environment [↑](#footnote-ref-11)
12. Refaktoring kodu - [↑](#footnote-ref-12)
13. IDE (od ang. Integrated development enviroment) – jest to program, który umożliwia proces tworzenia oprogramowania np. InteliJ, Eclipse, NetBeans. [↑](#footnote-ref-13)