Politechnika Świętokrzyska Wydział Elektrotechniki, Automatyki i Informatyki	
Technologie obiektowe - projekt	Autor: Jarosław Kot Data: 29.06.2023
Temat: **Platform as a service (PaaS)**	

Spis treści:

1.Wstęp

W ramach części praktycznej należało opracować aplikacje wykorzystującą jedno z rozwiązań w tym wypadku jest to Google App Engine. Aplikacja, która została stworzona w tym celu to System Antyplagiatowy wraz z przechowywaniem plików. Jest to program działający jako strona internetowa, gdzie użytkownik ma możliwość sprawdzenia czy plik, który chce wgrać do bazy, nie jest przypadkiem taki sam jak reszta plików w bazie danych, dodatkowo może wszystkie pliki wyświetlić, pobrać oraz usunąć.

2. Technologie

Do stworzenia tego projektu wykorzystane były narzędzia, frameworki itd. takie jak:

*Języki oraz frameworki:

- Java,

- SpringBoot,

- Thymeleaf,

- MariaDB,

- Maven,

- Google Cloud: Cloud Code.

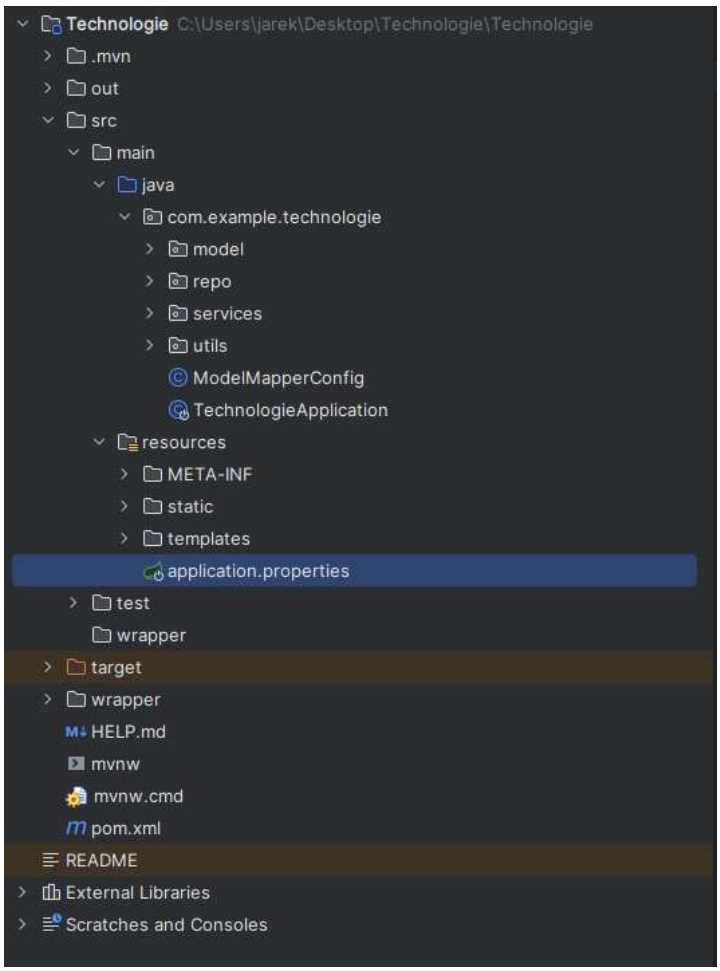
*Narzędzia:

- Intellij Idea 2023.1,

- Docker.

3. Struktura Projektu

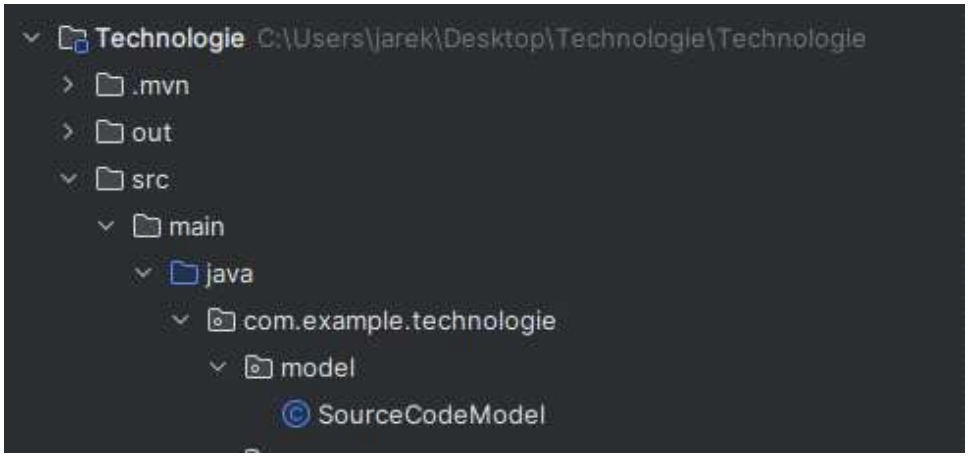
Projekt Aplikacji “System Antyplagiatowy” posiada strukturę z podziałem na foldery oraz klasy tak jak można zobaczyć na rysunku 3.1 poniżej.



Rys. 3.1.

Jak można zauważyć, w strukturze mamy główny folder src/main/java.com.example.technologie. Folder ten podzielony jest na 4 “funkcjonalne” foldery, a oprócz tego posiada dwie klasy -“ModelMapperConfig” służącą jako bibliotekę wykorzystywaną do mapowania obiektów między różnymi modelami lub klasami, a także “TechnologieApplication”, która służy jako kontroler oraz starter aplikacji.

3.1. Model



Rys. 3.2.

Klasa SourceCodeModel służy do reprezentowania danych kodu źródłowego w aplikacji. Może być używana wraz z frameworkiem Spring Data JPA do operacji na bazie danych, takich jak zapisywanie, odczytywanie, aktualizowanie i usuwanie rekordów związanych z kodem źródłowym.

Listing 3.1.

```
package com.example.technologie.model;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.*;

@Entity
@Table(name = "SourceCodeData")
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class SourceCodeModel {
 @Id
 @GeneratedValue(strategy = GenerationType.IDENTITY)
 private Long id;

 private String name;
 private String type;
 @Lob
```

```
 @Column(name = "sourcecode", length = 100000)
 private byte[] sourceCode;
}
```

Opis:

@Entity: Adnotacja oznaczająca, że SourceCodeModel jest encją, czyli klasą reprezentującą encję w bazie danych.

@Table(name = "SourceCodeData"): Adnotacja określająca nazwę tabeli w bazie danych, do której będzie mapowana encja.

@Data: Adnotacja z biblioteki Lombok generująca za nas standardowe metody toString(), equals(), hashCode() i getters/setters.

@AllArgsConstructor: Adnotacja z biblioteki Lombok generująca konstruktor zawierający wszystkie pola klasy.

@NoArgsConstructor: Adnotacja z biblioteki Lombok generująca konstruktor bezargumentowy.

@Builder: Adnotacja z biblioteki Lombok generująca wzorzec projektowy Builder, umożliwiający łatwe tworzenie instancji klasy.

Pola:

private Long id: Pole reprezentujące unikalny identyfikator (klucz główny) dla rekordu w bazie danych. Jest oznaczone adnotacją @Id, co wskazuje, że jest to pole identyfikujące.

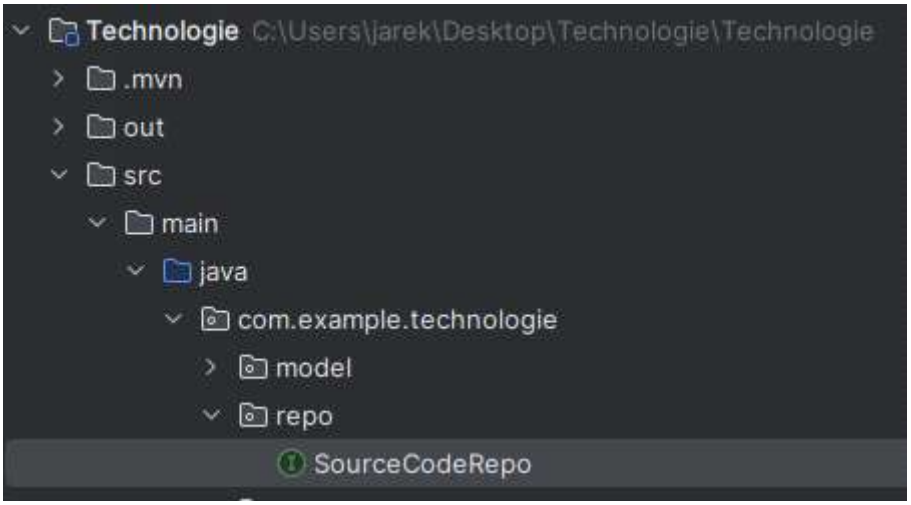
private String name: Pole przechowujące nazwę kodu źródłowego.

private String type: Pole przechowujące typ kodu źródłowego.

@Lob: Adnotacja wskazująca, że pole sourceCode jest typu LOB (Large Object), co umożliwia przechowywanie większych danych.

@Column(name = "sourcecode", length = 100000): Adnotacja określająca nazwę kolumny w bazie danych, do której będzie mapowane pole sourceCode. Dodatkowo, określono maksymalną długość kolumny jako 100000 bajtów.

3.2. Repository



Rys. 3.3.

Interfejs SourceCodeRepo definiuje metody dostępne do operacji na encji SourceCodeModel w bazie danych. Dzięki wykorzystaniu Spring Data JPA, nie trzeba implementować tych metod ręcznie. Spring automatycznie generuje implementację tych metod na podstawie nazw i typów zdefiniowanych w interfejsie. Można używać tych metod do wykonywania różnych operacji na danych kodu źródłowego, takich jak wyszukiwanie, sprawdzanie istnienia i inne.

Listing 3.2.

```
package com.example.technologie.repo;

import com.example.technologie.model.SourceCodeModel;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;
import java.util.Optional;

public interface SourceCodeRepo extends JpaRepository<SourceCodeModel, Long> {

 Optional<SourceCodeModel> findByName(String fileName);
 boolean existsByName(String name);

}
```

Opis:

public interface SourceCodeRepo extends JpaRepository<SourceCodeModel, Long>: Interfejs SourceCodeRepo rozszerza interfejs JpaRepository dostarczany przez Spring Data JPA. Interfejs JpaRepository zapewnia gotowe metody do wykonywania operacji CRUD (Create, Read, Update, Delete) na encji SourceCodeModel.

Optional<SourceCodeModel> findByName(String fileName): Metoda abstrakcyjna, która pozwala wyszukać rekord w bazie danych na podstawie nazwy pliku (fileName). Zwraca wynik w postaci Optional, co oznacza, że może zwrócić wartość null, jeśli rekord nie zostanie znaleziony.

boolean existsByName(String name): Metoda abstrakcyjna, która sprawdza, czy istnieje rekord w bazie danych o podanej nazwie (name). Zwraca true, jeśli rekord istnieje lub false, jeśli nie istnieje.

3.3. Service

Rys. 3.4.

Klasa SourceCodeService pełni rolę serwisu, który dostarcza logikę biznesową związaną z zarządzaniem kodem źródłowym w tym wypadku plikami oraz sprawdzaniem plagiatu. Jest ona wykorzystywana w klasie TechnologiApplication, w której wykorzystujemy metody z tej klasy, aby móc wykonywać operacje na plikach w naszej aplikacji.

Listing 3.3.

```
package com.example.technologie.services;

import com.example.technologie.model.SourceCodeModel;
```

```
import com.example.technologie.repo.SourceCodeRepo;
import org.springframework.mock.web.MockMultipartFile;

import org.modelmapper.ModelMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;
import org.apache.commons.compress.archivers.ArchiveEntry;
import org.apache.commons.compress.archivers.ArchiveException;
import org.apache.commons.compress.archivers.ArchiveInputStream;
import org.apache.commons.compress.archivers.ArchiveStreamFactory;
import org.apache.commons.compress.utils.IOUtils;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

@Service
public class SourceCodeService {
 private final SourceCodeRepo repository;
 private final ModelMapper modelMapper;

 @Autowired
 public SourceCodeService(SourceCodeRepo repository, ModelMapper modelMapper) {
 this.repository = repository;
 this.modelMapper = modelMapper;
 }

 public List<SourceCodeModel> getSourceCodeList() {
 List<SourceCodeModel> sourceCodeList = repository.findAll();
 return sourceCodeList.stream()
 .map(sourceCodeModel -> modelMapper.map(sourceCodeModel,
SourceCodeModel.class))
 .collect(Collectors.toList());
 }

 public String uploadSourceCode(MultipartFile file) throws IOException {
 String fileName = file.getOriginalFilename();

Optional<SourceCodeModel> existingFile = repository.findByName(fileName);
 if (existingFile.isPresent()) {
 return "Już istnieje taki plik w bazie: " + fileName;
 }
```

```
SourceCodeModel sourceCodeModel = SourceCodeModel.builder()
.name(fileName)
.type(file.getContentType())
.sourceCode(file.getBytes())
.build();
SourceCodeModel savedFile = repository.save(sourceCodeModel);

if (savedFile != null) {
return "Plik wgrano pomyślnie: " + fileName;
}

return null;
}

public byte[] downloadSourceCode(String fileName) {
Optional<SourceCodeModel> dbSourceCodeData = repository.findByName(fileName);
return dbSourceCodeData.get().getSourceCode();
}

public static class PlagiarismResult {
private final boolean isPlagiarized;
private final double similarityPercentage;

public PlagiarismResult(boolean isPlagiarized, double similarityPercentage) {
this.isPlagiarized = isPlagiarized;
this.similarityPercentage = similarityPercentage;
}

public boolean isPlagiarized() {
return isPlagiarized;
}

public double getSimilarityPercentage() {
return similarityPercentage;
}
}

public PlagiarismResult checkPlagiarismByLineContent(MultipartFile sourceCodeFile)
throws IOException {
List<SourceCodeModel> storedSourceCodes = repository.findAll();
String sourceCode = new String(sourceCodeFile.getBytes());

for (SourceCodeModel storedSourceCode : storedSourceCodes) {
String code = new String(storedSourceCode.getSourceCode());

String[] sourceCodeLines = sourceCode.split("\\r?\\n");
```

```
String[] codeLines = code.split("\\r?\\n");

if (sourceCodeLines.length == codeLines.length) {
int totalLines = sourceCodeLines.length;
int matchingLines = 0;

for (int i = 0; i < totalLines; i++) {
if (compareLineContent(sourceCodeLines[i], codeLines[i])) {
matchingLines++;
}
}

double similarityPercentage = (double) matchingLines / totalLines * 100.0;
if (similarityPercentage >= 80.0) {
return new PlagiarismResult(true, similarityPercentage);
}
}
}

return new PlagiarismResult(false, 0.0);
}

private boolean compareLineContent(String line1, String line2) {
String trimmedLine1 = line1.trim();
String trimmedLine2 = line2.trim();

return trimmedLine1.equalsIgnoreCase(trimmedLine2);
}


public PlagiarismResult checkPlagiarism(MultipartFile sourceCodeFile, String method)
throws IOException {
if (method.equals("lineContent")) {
PlagiarismResult lineContentPlagiarismResult =
checkPlagiarismByLineContent(sourceCodeFile);
return lineContentPlagiarismResult;
} else {
List<SourceCodeModel> storedSourceCodes = repository.findAll();
byte[] sourceCode = sourceCodeFile.getBytes();
double maxSimilarityPercentage = 0.0;

for (SourceCodeModel storedSourceCode : storedSourceCodes) {
byte[] storedCode = storedSourceCode.getSourceCode();
double similarityPercentage = calculateSimilarityPercentage(sourceCode, storedCode);
maxSimilarityPercentage = Math.max(maxSimilarityPercentage, similarityPercentage);
}
```

```
return new PlagiarismResult(maxSimilarityPercentage >= 80.0, maxSimilarityPercentage);
}
}

private double calculateSimilarityPercentage(byte[] code1, byte[] code2) {
int matchCount = 0;
int totalComparisonCount = Math.min(code1.length, code2.length);

for (int i = 0; i < totalComparisonCount; i++) {
if (code1[i] == code2[i]) {
matchCount++;
}
}

return (double) matchCount / totalComparisonCount * 100.0;
}


public PlagiarismResult checkPlagiarismForRAR(MultipartFile rarFile, String method) throws
IOException, ArchiveException {
File tempDir = new File(System.getProperty("java.io.tmpdir"));
String tempDirPath = tempDir.getAbsolutePath();
String tempFilePath = tempDirPath + File.separator + rarFile.getOriginalFilename();

File tempFile = new File(tempFilePath);
rarFile.transferTo(tempFile);

double maxSimilarityPercentage = 0.0;
boolean isPlagiarized = false;

try (ArchiveInputStream archiveInputStream = new
ArchiveStreamFactory().createArchiveInputStream(new FileInputStream(tempFile))) {
List<SourceCodeModel> storedSourceCodes = repository.findAll();
List<byte[]> storedCodes = new ArrayList<>();

for (SourceCodeModel storedSourceCode : storedSourceCodes) {
storedCodes.add(storedSourceCode.getSourceCode());
}

ArchiveEntry entry;
while ((entry = archiveInputStream.getNextEntry()) != null) {
if (!entry.isDirectory()) {
byte[] sourceCode = IOUtils.toByteArray(archiveInputStream);

if (isTextFile(entry.getName())) {
MockMultipartFile mockMultipartFile = new MockMultipartFile(
entry.getName(),
```

```
sourceCode
);

PlagiarismResult plagiarismResult = checkPlagiarismByLineContent(mockMultipartFile);

if (plagiarismResult.isPlagiarized()) {
return plagiarismResult;
} else {
maxSimilarityPercentage = Math.max(maxSimilarityPercentage,
plagiarismResult.getSimilarityPercentage());
isPlagiarized = true;
}
} else {
for (byte[] storedCode : storedCodes) {
double similarityPercentage = calculateSimilarityPercentage(sourceCode, storedCode);
maxSimilarityPercentage = Math.max(maxSimilarityPercentage, similarityPercentage);
if (similarityPercentage >= 80.0) {
isPlagiarized = true;
break;
}
}
}
}
} finally {
tempFile.delete();
}

if (isPlagiarized || maxSimilarityPercentage >= 80.0) {
return new PlagiarismResult(true, maxSimilarityPercentage);
} else {
return new PlagiarismResult(false, maxSimilarityPercentage);
}
}


private boolean isTextFile(String fileName) {

String[] textFileExtensions = { ".txt", ".java", ".cpp" };
for (String extension : textFileExtensions) {
if (fileName.toLowerCase().endsWith(extension)) {
return true;
}
}
return false;
}
```

```
public void deleteSourceCodeFile(Long id) {
repository.deleteById(id);
}
}
```

Opis:

- `public class SourceCodeService`: Klasa `SourceCodeService` jest serwisem (komponentem) aplikacji, który dostarcza metody do manipulacji danymi kodu źródłowego.

- `@Service`: Adnotacja `@Service` wskazuje, że klasa jest serwisem i jest zarządzana przez kontener Spring.

- `private final SourceCodeRepo repository`: Pole `repository` jest typu `SourceCodeRepo` i jest wykorzystywane do wykonywania operacji na bazie danych (CRUD) dla encji `SourceCodeModel`.

- `private final ModelMapper modelMapper`: Pole `modelMapper` jest typu `ModelMapper` i jest wykorzystywane do mapowania obiektów między różnymi strukturami danych.

Metody:

- `public List<SourceCodeModel> getSourceCodeList()`: Metoda zwraca listę wszystkich obiektów `SourceCodeModel` pobranych z bazy danych. Wykorzystuje `repository.findAll()` do pobrania wszystkich rekordów, a następnie mapuje je na listę obiektów `SourceCodeModel`.

- `public String uploadSourceCode(MultipartFile file) throws IOException`: Metoda umożliwia przesłanie pliku kodu źródłowego i zapisanie go w bazie danych. Sprawdza, czy plik o takiej samej nazwie już istnieje w bazie danych. Tworzy obiekt `SourceCodeModel` na podstawie przesłanego pliku i zapisuje go w bazie danych za pomocą `repository.save()`.

- `public byte[] downloadSourceCode(String fileName)`: Metoda pobiera kod źródłowy o podanej nazwie pliku z bazy danych i zwraca go w postaci tablicy bajtów (`byte[]`).

- `public static class PlagiarismResult`: Wewnętrzna klasa `PlagiarismResult` reprezentuje wynik sprawdzania podobieństwa. Zawiera informację, czy kod został uznany za plagiat (`isPlagiarized`) oraz procentowe podobieństwo (`similarityPercentage`).

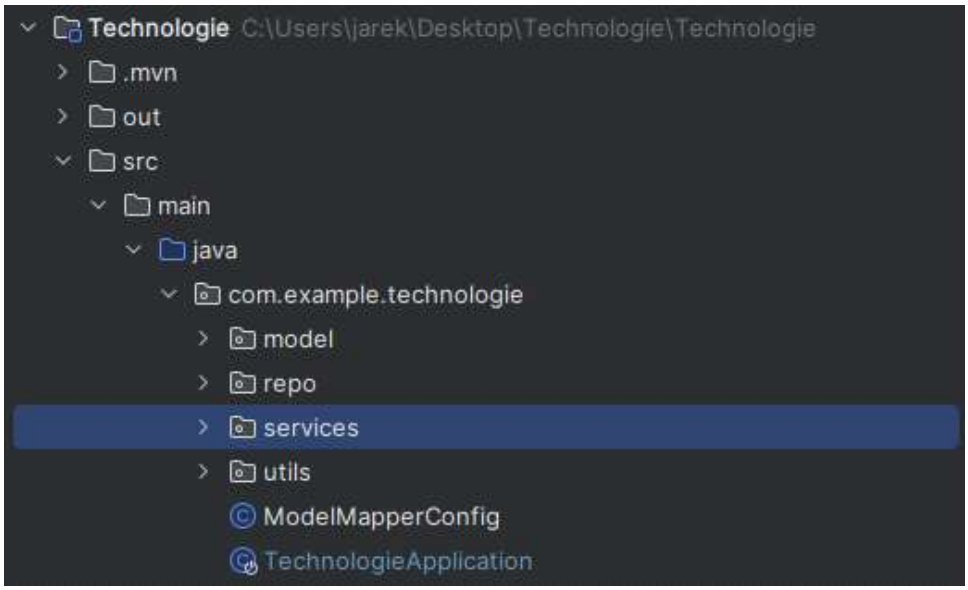
- `public PlagiarismResult checkPlagiarismByLineContent(MultipartFile sourceCodeFile) throws IOException`: Metoda sprawdza podobieństwo kodu źródłowego w pliku do kodu

źródłowego przechowywanego w bazie danych na podstawie zawartości linii. Wykorzystuje metodę `compareLineContent` do porównywania zawartości linii kodu.

- `public PlagiarismResult checkPlagiarism(MultipartFile sourceCodeFile, String method) throws IOException`: Metoda sprawdza podobieństwo kodu źródłowego w pliku do kodu źródłowego przechowywanego w bazie danych. Możliwe jest wybranie metody porównywania (np. porównywanie zawartości linii). Wykorzystuje metodę `calculateSimilarityPercentage` do obliczania procentowego podobieństwa między dwoma kodami.

- `public PlagiarismResult checkPlagiarismForRAR(MultipartFile rarFile, String method) throws IOException, ArchiveException`: Metoda sprawdza podobieństwo kodu źródłowego w pliku RAR do kodu źródłowego przechowywanego w bazie danych. Rozpakowuje plik RAR, a następnie sprawdza każdy plik w archiwum pod kątem podobieństwa. Wykorzystuje metodę `isTextFile` do sprawdzenia, czy plik jest plikiem tekstowym. - `public void deleteSourceCodeFile(Long id)`: Metoda usuwa plik kodu źródłowego z bazy danych na podstawie jego identyfikatora (`id`).

3.4 Controller



Rys. 3.5.

Klasa TechnologieApplication jest główną klasą aplikacji Spring Boot. Jest oznaczona adnotacją @SpringBootApplication, która integruje różne adnotacje konfiguracyjne i umożliwia uruchomienie aplikacji Spring Boot. Dodatkowo, jest oznaczona adnotacją @Controller, co oznacza, że pełni rolę kontrolera.

Listing 3.4.

```
package com.example.technologie;
import com.example.technologie.model.SourceCodeModel;
import com.example.technologie.repo.SourceCodeRepo;
```

```
import com.example.technologie.services.SourceCodeService;
import org.apache.commons.compress.archivers.ArchiveException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;
import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;
import javax.sql.DataSource;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

@Controller
@RequestMapping("/source-code")
@SpringBootApplication
public class TechnologieApplication {

 @Autowired
 private SourceCodeService service;
 @Autowired
 private final SourceCodeRepo sourceCodeRepo;
 @Autowired
 public TechnologieApplication(SourceCodeRepo sourceCodeRepo) {
 this.sourceCodeRepo = sourceCodeRepo;
 }
 @PostMapping("/upload")
 public String uploadSourceCode(@RequestParam("file") MultipartFile file,
@RequestParam(value = "method", defaultValue = "similarity") String method, Model
model) throws IOException, ArchiveException {
 SourceCodeService.PlagiarismResult plagiarismResult;

 if (file.getOriginalFilename().endsWith(".rar")) {
 plagiarismResult = service.checkPlagiarismForRAR(file, method);
 } else {
 if (method.equals("lineContent")) {
 SourceCodeService.PlagiarismResult lineContentPlagiarismResult =
service.checkPlagiarismByLineContent(file);
```

```
plagiarismResult = lineContentPlagiarismResult;
} else {
plagiarismResult = service.checkPlagiarism(file, method);
}
}

if (plagiarismResult.isPlagiarized()) {
model.addAttribute("message", "Wykryto plagiat. Plik odrzucono. Podobieństwo: " +
plagiarismResult.getSimilarityPercentage() + "%");
} else {
String uploadResult = service.uploadSourceCode(file);
model.addAttribute("message", uploadResult);
}

List<SourceCodeModel> sourceCodeList = service.getSourceCodeList();
model.addAttribute("sourceCodeList", sourceCodeList);

return "filess.html";
}

@GetMapping("/files")
public String getFiles(Model model) {
List<SourceCodeModel> sourceCodeList = service.getSourceCodeList();
model.addAttribute("sourceCodeList", sourceCodeList);
return "filess.html";
}
@GetMapping("/files/delete/{id}")
public String deleteFile(@PathVariable("id") Long id) {
sourceCodeRepo.deleteById(id);
return "redirect:/source-code/files";
}
@GetMapping("/home")
public String home() {
return "index.html";
}

@PostMapping("/check-plagiarism")
public String checkPlagiarism(@RequestParam("file") MultipartFile file,
@RequestParam(value = "method", defaultValue = "similarity") String method,
RedirectAttributes redirectAttributes) throws IOException {
SourceCodeService.PlagiarismResult plagiarismResult = service.checkPlagiarism(file,
method);

if (plagiarismResult.isPlagiarized()) {
redirectAttributes.addFlashAttribute("message", "Wykryto plagiat. Plik odrzucono.
Podobieństwo: " + plagiarismResult.getSimilarityPercentage() + "%");
} else {
```

```
String uploadResult = service.uploadSourceCode(file);
redirectAttributes.addFlashAttribute("message", "Plik wgrano pomyślnie: " +
file.getOriginalFilename());
}

return "redirect:/source-code/home";
}


@GetMapping("/{fileName}")
public ResponseEntity<byte[]> downloadSourceCode(@PathVariable String fileName) {
byte[] sourceCode = service.downloadSourceCode(fileName);
HttpHeaders headers = new HttpHeaders();
headers.setContentDispositionFormData("attachment", fileName);
headers.setContentType(MediaType.APPLICATION_OCTET_STREAM);

return ResponseEntity.ok()
.headers(headers)
.body(sourceCode);
}


public static void main(String[] args) {
SpringApplication.run(TechnologieApplication.class, args);

}
}
```

Metody:

- public static void main: Metoda statyczna main jest punktem wejścia do aplikacji. Uruchamia aplikację Spring Boot poprzez wywołanie metody run klasy SpringApplication i przekazanie klasy TechnologieApplication oraz argumentów wiersza poleceń.

- public String uploadSourceCode: Obsługuje żądanie przesłania pliku kodu źródłowego. Przyjmuje parametry:

-file - przesłany plik kodu źródłowego,

-method - metoda sprawdzania plagiatu (domyślnie "similarity"),

model - obiekt Model używany do przekazania danych do widoku. Metoda wywołuje odpowiednie metody serwisu SourceCodeService w zależności od rozszerzenia pliku i metody sprawdzania plagiatu. Następnie aktualizuje model z wynikami przetwarzania i zwraca nazwę widoku "filess.html".

- public String getFiles: Obsługuje żądanie pobrania listy wszystkich plików kodu źródłowego przechowywanych w bazie danych. Przyjmuje parametr model, który jest obiektem Model używanym do przekazania danych do widoku. Metoda pobiera listę plików kodu źródłowego za pomocą serwisu -SourceCodeService, aktualizuje model i zwraca nazwę widoku

"filess.html".

- public String deleteFile: Obsługuje żądanie usunięcia pliku kodu źródłowego o określonym identyfikatorze. Przyjmuje parametr id, który jest identyfikatorem pliku do usunięcia. Metoda wywołuje odpowiednią metodę w repozytorium SourceCodeRepo w celu usunięcia pliku. Następnie przekierowuje na adres "/source-code/files".

- public String home: Obsługuje żądanie wyświetlenia strony głównej. Metoda zwraca nazwę widoku "index.html".

- public String checkPlagiarism: Obsługuje żądanie sprawdzenia plagiatu przesłanego pliku kodu źródłowego. Przyjmuje parametry:

-file - przesłany plik kodu źródłowego,

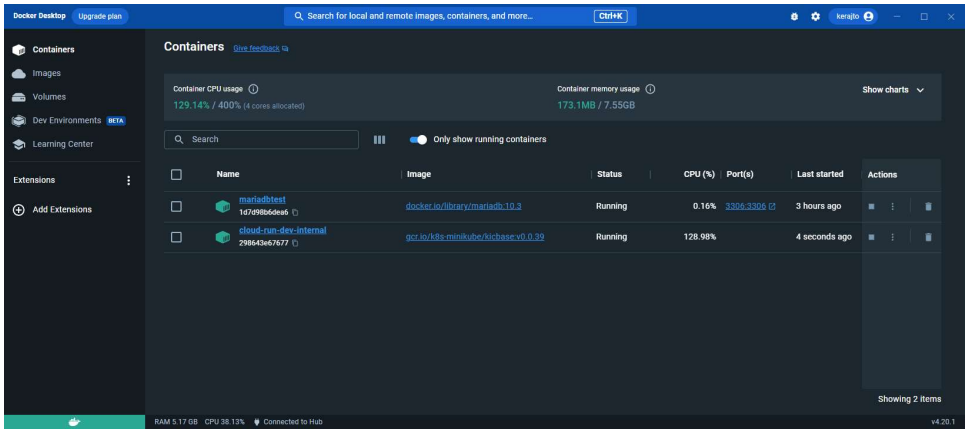
-method - metoda sprawdzania plagiatu (domyślnie "similarity"),

-redirectAttributes - obiekt RedirectAttributes używany do przekazania komunikatu do widoku. Metoda wywołuje odpowiednią metodę serwisu SourceCodeService w celu sprawdzenia plagiatu. Następnie aktualizuje redirectAttributes z wynikami przetwarzania i przekierowuje na adres "/source-code/home".

- public ResponseEntity<byte[]> downloadSourceCode: Obsługuje żądanie pobrania pliku kodu źródłowego o określonej nazwie. Przyjmuje parametr fileName, który jest nazwą pliku do pobrania. Metoda wywołuje odpowiednią metodę serwisu SourceCodeService w celu pobrania pliku kodu źródłowego. Następnie tworzy odpowiedź ResponseEntity zawierającą dane pliku i nagłówki odpowiednie dla pobrania pliku.

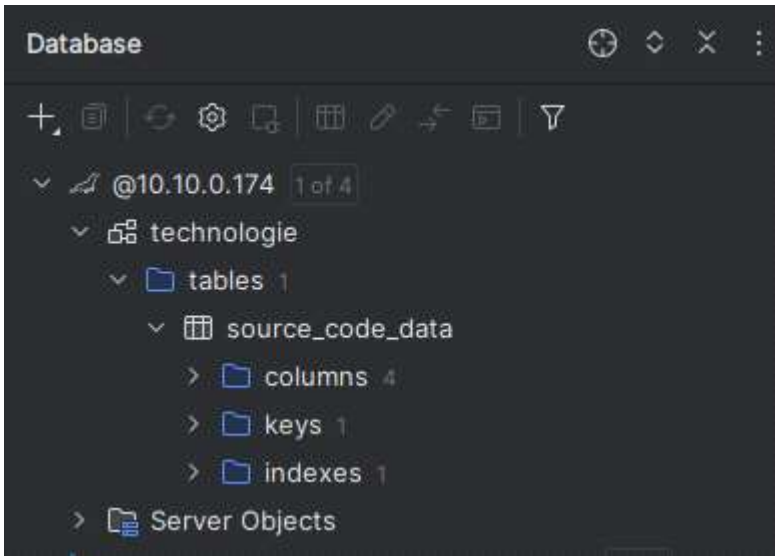
4. Uruchomienie aplikacji

Aby uruchomić aplikację webową należy posiadać narzędzia takie jak Docker, Intellij oraz dowolna przeglądarka internetowa, dodatkowo do środowiska Intellij musimy pobrać plugin o nazwe “Google Cloud: Cloud Code”. Po pobraniu pluginu oraz zainstalowaniu Dockera, który jest wymagany, aby “Cloude Code” mógł uruchomić się lokalnie i stworzył kontener, oprócz tego w Dockerze należy zainstalować także relacyjną bazę danych MariaDB oraz stworzyć sobie kontener wraz z nazwą użytkownika oraz hasłem i nazwą kontenera. Wygląda to jak na rysunku poniżej.



Rys. 4.1.

Aby korzystać z bazy danych w MariaDB możemy połączyć się z nią za pomocą Intellij i stworzyć schema.



Rys. 4.2.

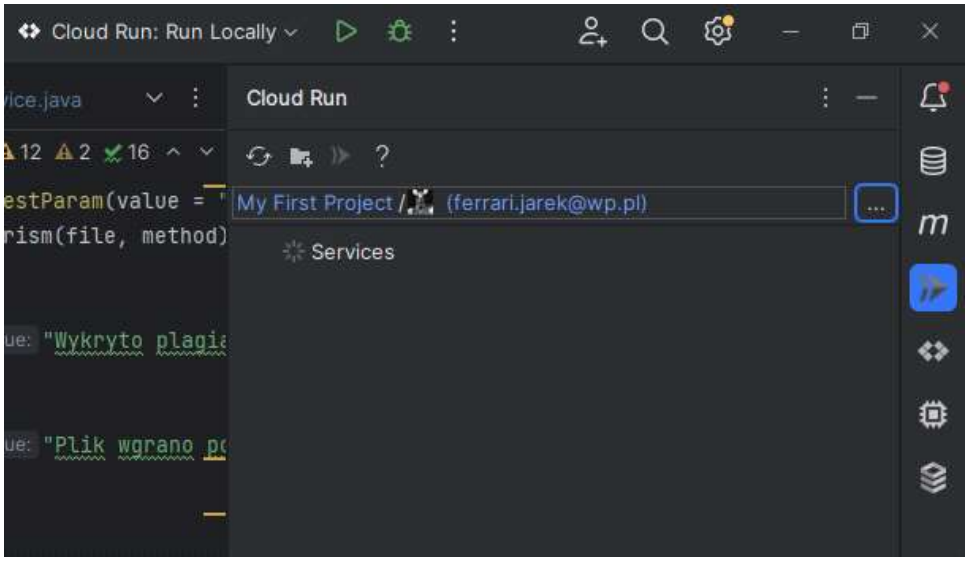
Następnie trzeba skonfigurować plik application.properties w następujący sposób:

```
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.datasource.url = jdbc:mariadb://10.10.0.174:3306/technologie
spring.datasource.username = root
spring.datasource.password = mypass
spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto = update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MariaDB103Dialect
spring.thymeleaf.enabled=true
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html
```

Rys. 4.3.

Po zainstalowaniu i uruchomieniu wszystkich potrzebnych narzędzi możemy przejść do

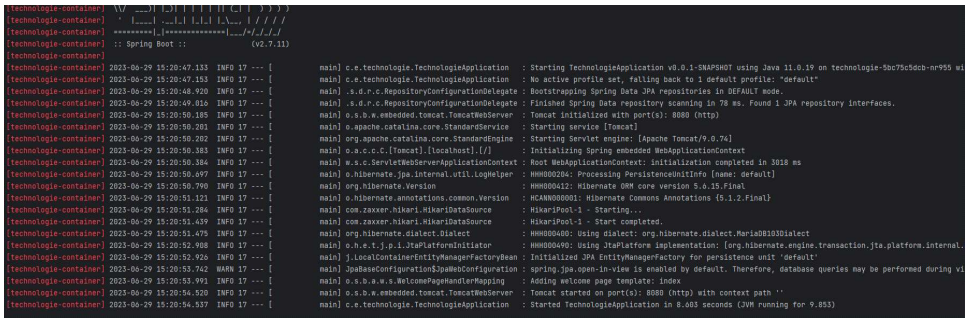
sekcji Cloud Run.



Rys. 4.4.

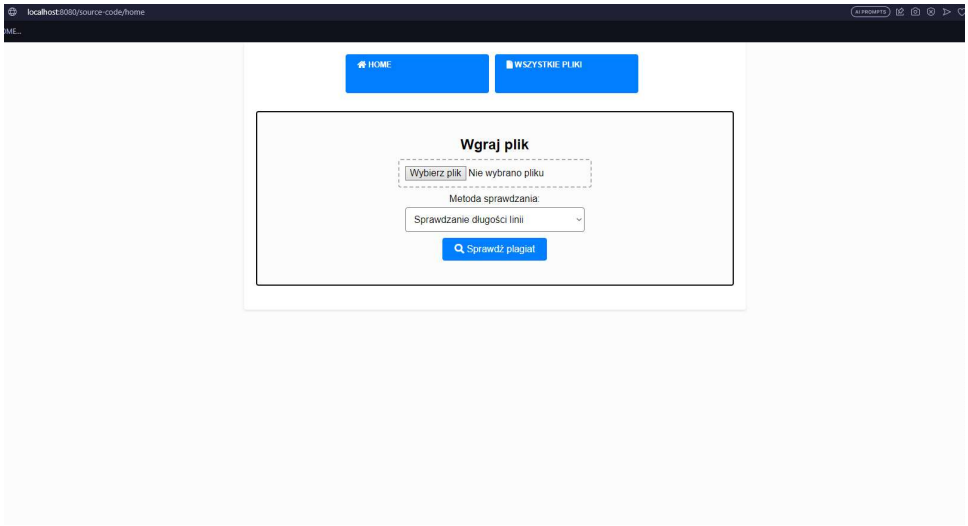
Uruchamiamy aplikacje po wciśnięciu zielonego przycisku play, tak jak na rysunku powyżej. Po uruchomieniu powinien nam się stworzyć w Dockerze automatycznie kontener, który był widoczny na rysunku wcześniej oraz sama aplikacja powinna się uruchomić i działać pod adresem url localhost:3306/source-code/home.

Widok działającej aplikacji uruchomionej za pomocą Google Cloud:



Rys. 4.5.

Gdy wszystko się uruchomiło bez problemów możemy przejść na stronę aplikacji.

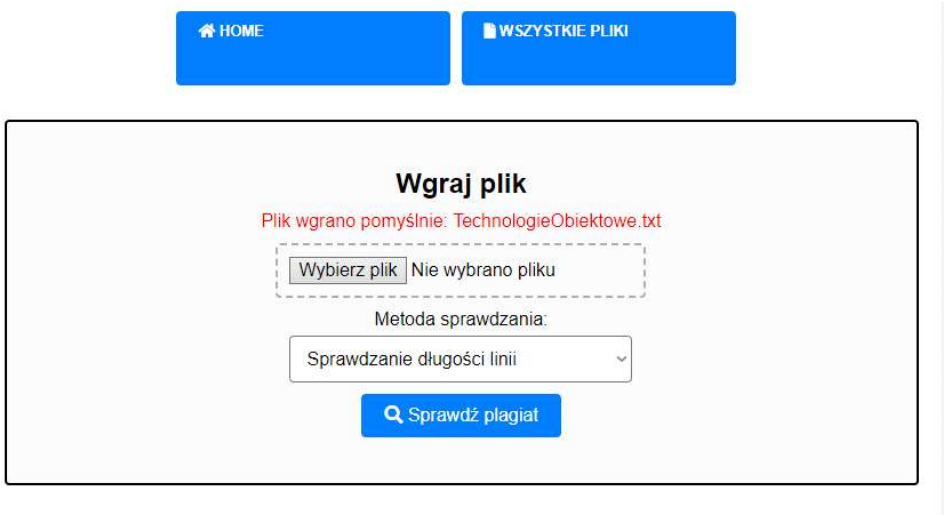


Rys. 4.6.

Teraz możemy wgrać pierwszy plik do bazy danych. Aby to zrobić należy wcisnąć “Wybierz plik” a następnie nacisnąć na “Sprawdź plagiat”. W tym wypadku będzie to plik TechnologieObiektowe.

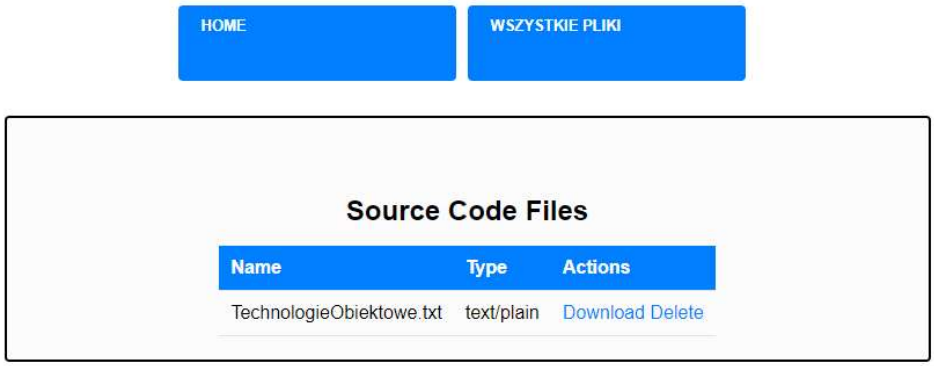
Rys. 4.7.

Po wybraniu pliku i przejściu przez plagiat dostajemy informacje o pomyślnym przejściu procesu.



Rys. 4.8.

Teraz możemy przejść na stronę z wyświetlanie plików poprzez wciśnięcie “WSZYSTKIE PLIKI” i możemy zobaczyć, że faktycznie plik został dodany.



Rys. 4.9.

Natomiast gdy dodamy drugi plik o nazwie TechnologieObiektowe2 tylko z usuniętym słowem tak jak poniżej na rysunku to dostaniemy informacje o plagiacie.

Rys. 4.10.

Po wgraniu pliku i sprawdzeniu plagiatu otrzymujemy taki wynik:

Rys. 4.11.

5. Podsumowanie

Temat na projekt, czyli zbudowanie aplikacji, którą będzie można uruchomić za pomocą Google App Engine został pomyślnie zrealizowany. Aplikacja posiada funkcjonalności takie jak usuwanie, pobieranie oraz dodawanie plików do bazy wraz z poprzednich ich sprawdzaniem w systemie antyplagiatowym. Projekt ten można by było rozwinąć o bardziej zaawansowane metody sprawdzania antyplagiatu oraz dostosować go do korzystania z bazy danych z Google Cloud a także by odpalać go z wirtualnej maszyny Google Cloud: Compute Engine co polepszyłoby oraz przyśpieszyło działanie aplikacji.