

Podstawy Springa



Toolkit:

```
java.version 17  
spring 2.6.5  
maven 3.8.4  
Git  
Intelij Idea Community
```

Autor
Jan Górkiewicz

Framework

- Framework – Rozszerzenie języka o gotowe moduły

Zalety frameworków:

- Szybkie dodanie typowej funkcjonalności (np. Łączenie z bazą danych, Testowanie, logowanie do systemu)
- Nie ma potrzeby pisania kodu jeszcze raz
- Skupienie się na rzeczywistych potrzebach projektu

ZAKRES WARSZTATÓW

Podstawy modułów



Spring Framework

Provides core support for dependency injection, transaction management, web apps, data access, messaging, and more.



Spring Data

Provides a consistent approach to data access – relational, non-relational, map-reduce, and beyond.



Spring Boot

Takes an opinionated view of building Spring applications and gets you up and running as quickly as possible.



Spring Cloud

Provides a set of tools for common patterns in distributed systems. Useful for building and deploying microservices.



Spring Security

Protects your application with comprehensive and extensible authentication and authorization support.



Spring Integration

Supports the well-known Enterprise Integration Patterns through lightweight messaging and declarative adapters.



Spring for GraphQL

Spring for GraphQL provides support for Spring applications built on GraphQL Java.



Spring Session

Provides an API and implementations for managing a user's session information.

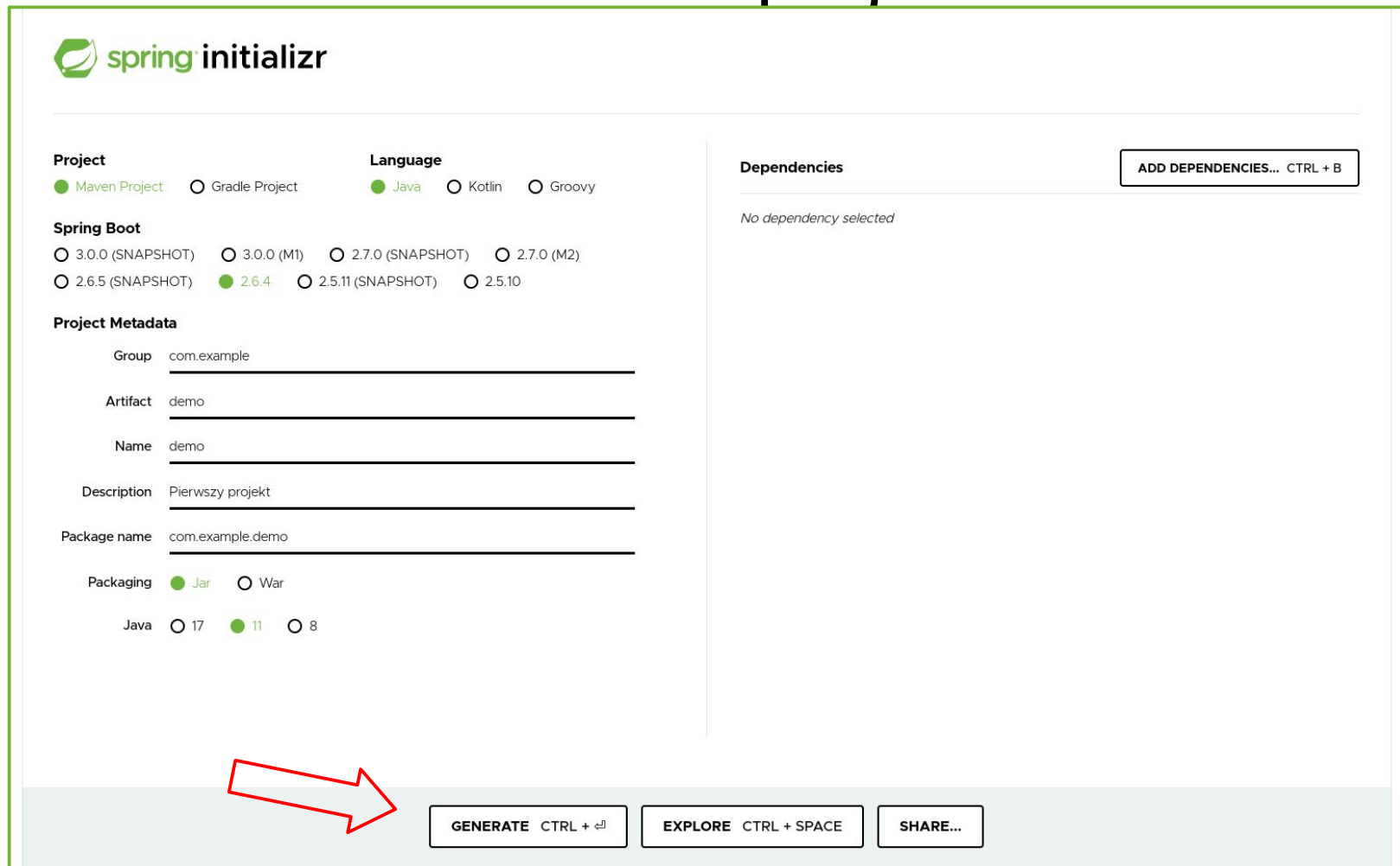
Spring Boot

- W Spring-u każdy moduł musiał być skonfigurowany ręcznie
- Często ta konfiguracja była niemalże identyczna dla większości projektów, wyłącznie z drobnymi zmianami np. inne dane logowania do bazy danych
- Skutkowało to kopiowaniem kodu pomiędzy projektami

Rozwiązaniem tego problemu jest Spring Boot:

- Zawiera on w sobie zbiór domyślnych konfiguracji, które można łatwo modyfikować

Tworzenie projektu



The image shows the Spring Initializr web interface for creating a new project. The form is divided into several sections: Project, Language, Spring Boot, Project Metadata, and Dependencies. The 'Project' section has 'Maven Project' selected. The 'Language' section has 'Java' selected. The 'Spring Boot' section has '2.6.4' selected. The 'Project Metadata' section contains text input fields for Group, Artifact, Name, Description, and Package name, with 'demo' entered in the last three. The 'Packaging' section has 'Jar' selected. The 'Dependencies' section is empty. At the bottom, there are three buttons: 'GENERATE', 'EXPLORE', and 'SHARE...'. A red arrow points to the 'GENERATE' button.

Project

☒ Maven Project ☐ Gradle Project

Language

☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (M1) ☐ 2.7.0 (SNAPSHOT) ☐ 2.7.0 (M2)
☐ 2.6.5 (SNAPSHOT) ☒ 2.6.4 ☐ 2.5.11 (SNAPSHOT) ☐ 2.5.10

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging

☒ Jar ☐ War

Java ☐ 17 ☒ 11 ☐ 8

Dependencies

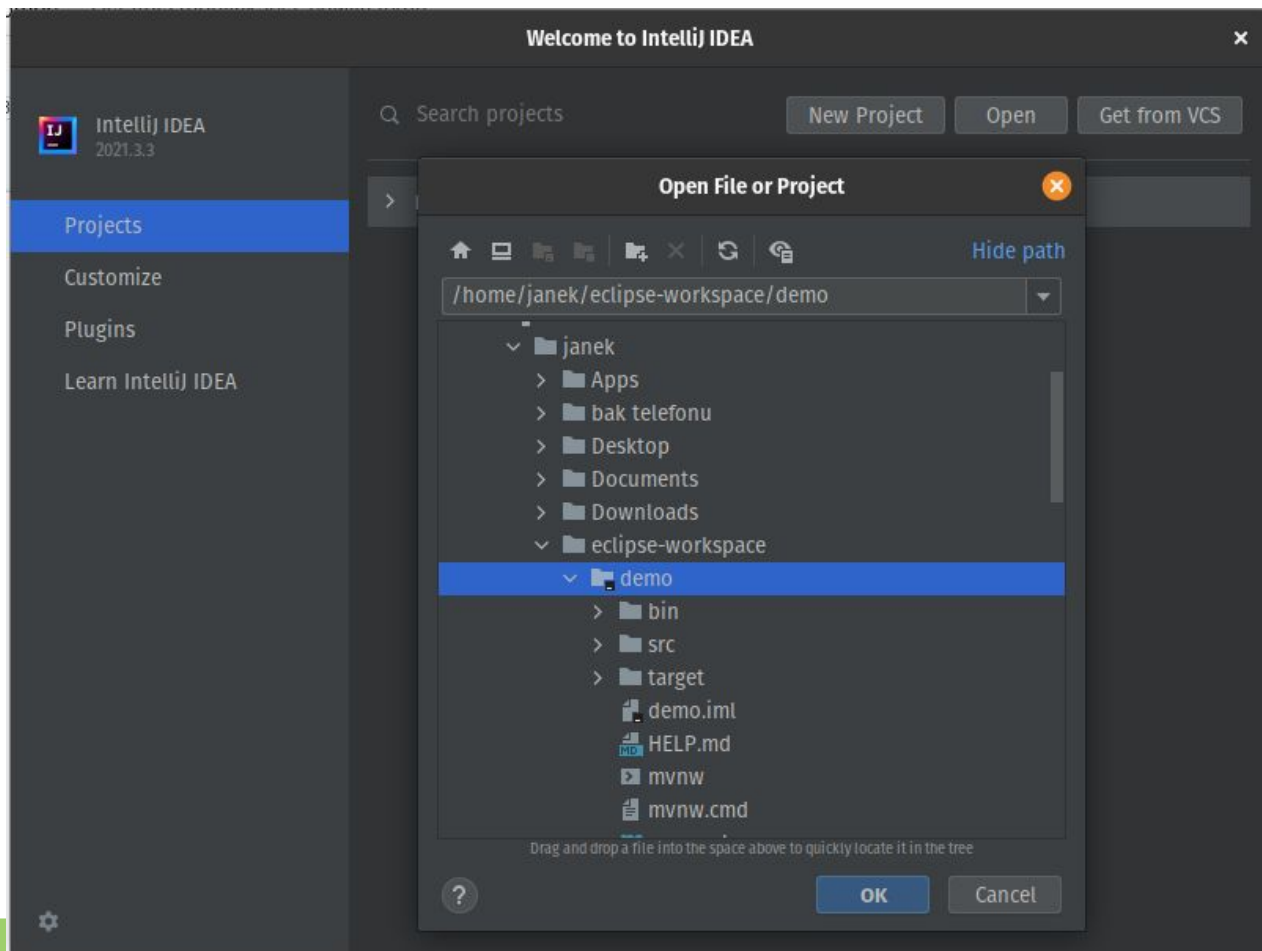
[ADD DEPENDENCIES... CTRL + B](#)

No dependency selected

[GENERATE CTRL + G](#) [EXPLORE CTRL + SPACE](#) [SHARE...](#)

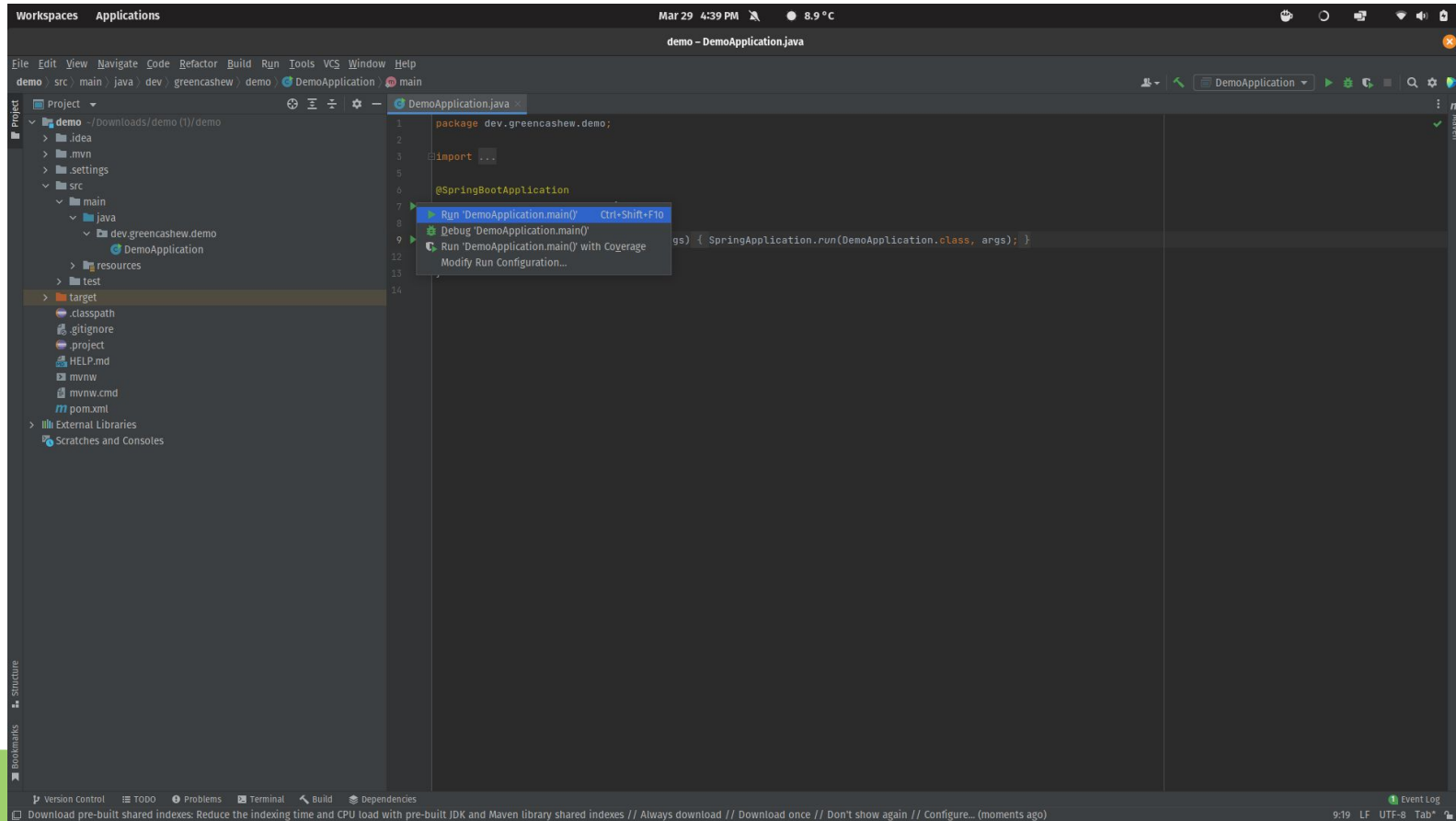
Importowanie Projektu - IntelliJ Idea

Open -> Open File or Project -> Trust Project



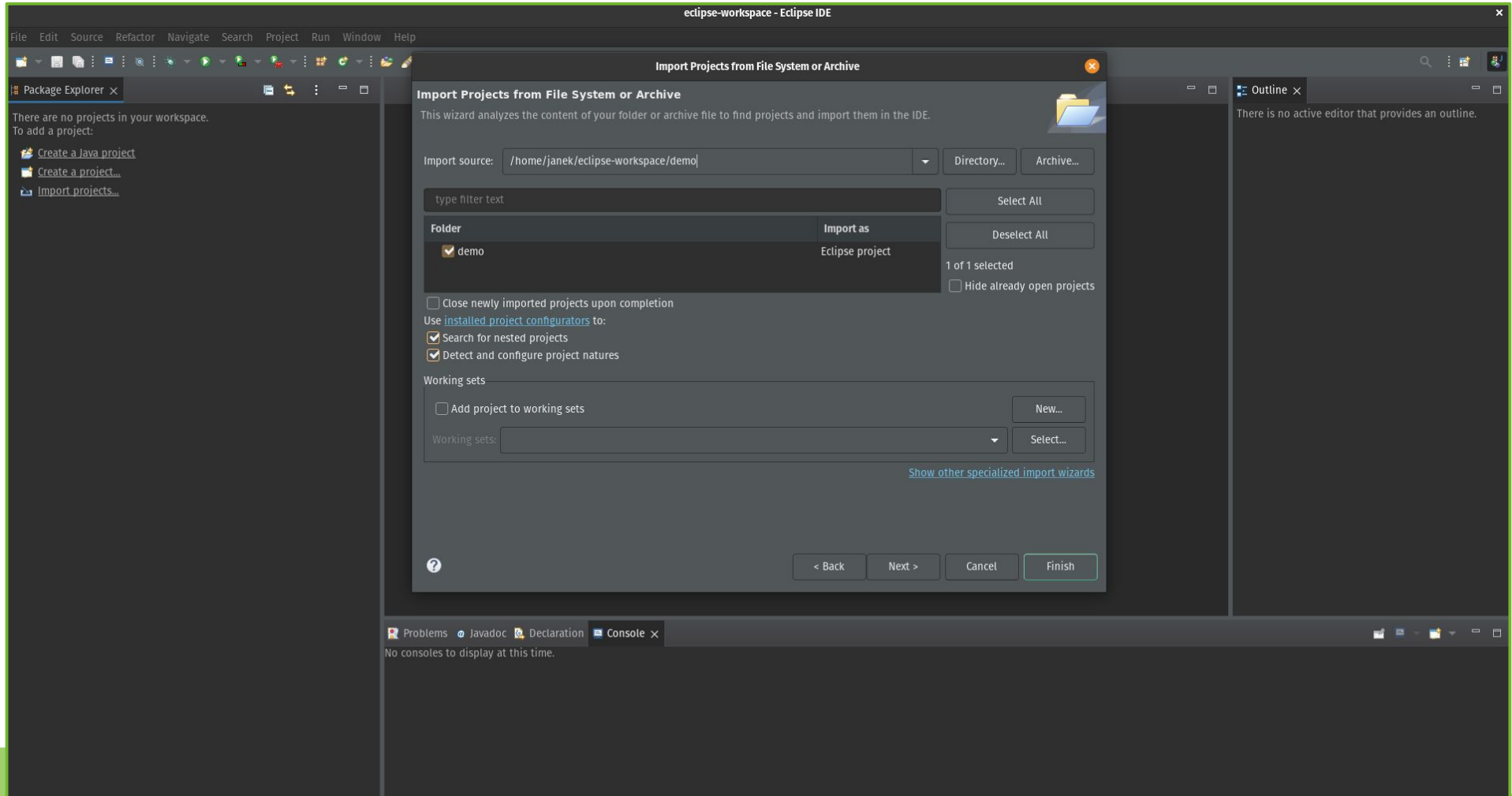
Uruchamianie Projektu - IntelliJ Idea

1. Czekamy aż projekt zostanie zaimportowany
2. Otwieramy klasę src/main/java/nazwa-pakietu/DemoApplication.java
3. Klikamy zielony przycisk i wybieramy Run 'DemoApplication.main()'



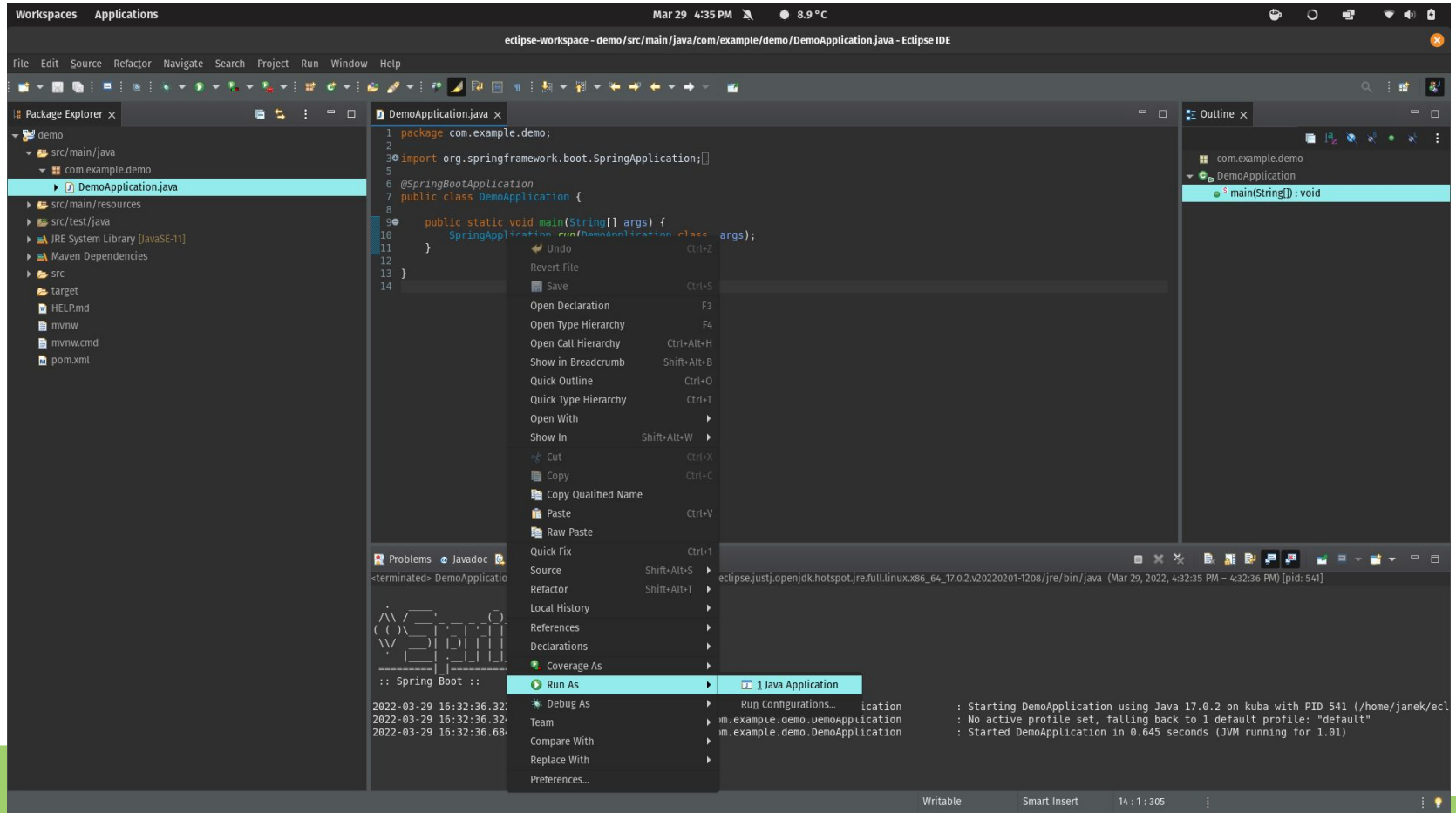
Importowanie Projektu - Eclipse

File -> Open Projects from File System... -> Directory... or Archive... -> Finish



Uruchamianie Projektu - Eclipse

1. Czekamy aż projekt zostanie zaimportowany
2. Otwieramy klasę src/main/java/nazwa-pakietu/DemoApplication.java
3. Kliknij prawy przycisk myszy wewnątrz okna edytora -> Run As -> Java Application



Uruchamianie Projektu - Maven

1. Upewnij się że JAVA_HOME (ścieżka do naszej javy) jest dodana do zmiennych środowiskowych
2. Otwórz terminal/konsolę w katalogu
3. Uruchom za pomocą komendy:
 - **MacOS/Linux:** `./mvnw spring-boot:run`
 - **Windows:** `mvnw spring-boot:run`

```
* demo ./mvnw spring-boot-run
[INFO] Scanning for projects...
[INFO] -----< dev.greencashew.demo >-----
[INFO] Building demo 0.0.1-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
>>> spring-boot-maven-plugin:2.6.5:run (default-cli) > test-compile @ demo >>>
[INFO]
--- maven-resources-plugin:3.2.0:resources (default-resources) @ demo ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO]
--- maven-compiler-plugin:3.8.1:compile (default-compile) @ demo ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
--- maven-resources-plugin:3.2.0:testResources (default-testResources) @ demo ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] skip non existing resourceDirectory /home/janek/Downloads/demo (1)/demo/src/test/resources
[INFO]
--- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ demo ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
<<< spring-boot-maven-plugin:2.6.5:run (default-cli) < test-compile @ demo <<<
[INFO]
[INFO]
[INFO] --- spring-boot-maven-plugin:2.6.5:run (default-cli) @ demo ---
[INFO] Attaching agents: []
```

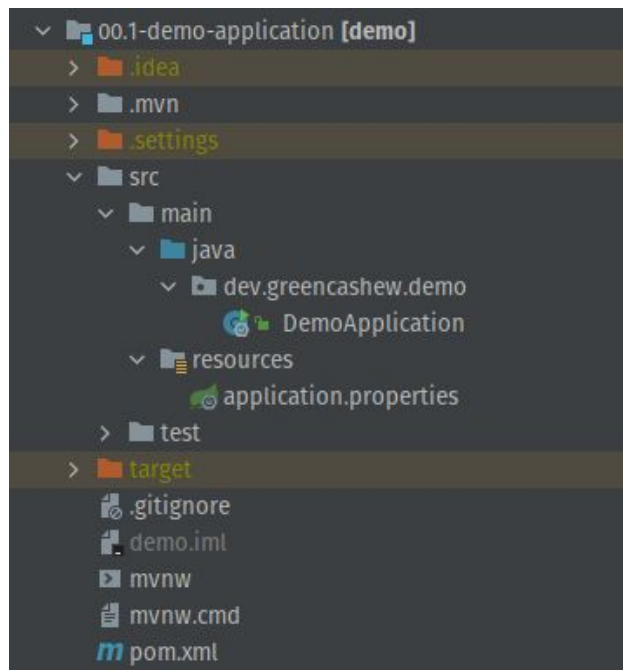
:: Spring Boot ::
(v2.6.5)

2022-03-29 16:59:58.437 INFO 39720 -- [main]	dev.greencashew.demo.DemoApplication : Starting DemoApplication using Java 17.0.2 on kuba with PID 39720 (/home/janek/Downloads/d
demo (1)/demo/target/classes started by janek in /home/janek/Downloads/demo (1)/demo)	
2022-03-29 16:59:58.438 INFO 39720 -- [main]	dev.greencashew.demo.DemoApplication : No active profile set, falling back to 1 default profile: "default"
2022-03-29 16:59:58.633 INFO 39720 -- [main]	dev.greencashew.demo.DemoApplication : Started DemoApplication in 0.347 seconds (JVM running for 0.512)

```
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 1.172 s
[INFO] Finished at: 2022-03-29T16:59:58+02:00
```

Wygenerowane pliki springa

- **application.properties** - Główny plik konfiguracyjny **spring boot**.
- **pom.xml** - Maven-owa konfiguracja projektu, w niej znajdują się kluczowe zależności springa
- **DemoApplication** - Klasa inicjująca uruchomienie aplikacji



```
m pom.xml (demo)
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>2.6.5</version>
9     <relativePath/> <!-- lookup parent from repository -->
10  </parent>
11  <groupId>dev.greencashew</groupId>
12  <artifactId>demo</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>demo</name>
15  <description>Demo project for Spring Boot</description>
16  <properties>
17    <java.version>17</java.version>
18  </properties>
19  <dependencies>
20    <dependency>
21      <groupId>org.springframework.boot</groupId>
22      <artifactId>spring-boot-starter</artifactId>
23    </dependency>
24
25    <dependency>
26      <groupId>org.springframework.boot</groupId>
27      <artifactId>spring-boot-starter-test</artifactId>
28      <scope>test</scope>
29    </dependency>
30  </dependencies>
31
32  <build>
33    <plugins>
34      <plugin>
35        <groupId>org.springframework.boot</groupId>
36        <artifactId>spring-boot-maven-plugin</artifactId>
37      </plugin>
38    </plugins>
39  </build>
40</project>
```

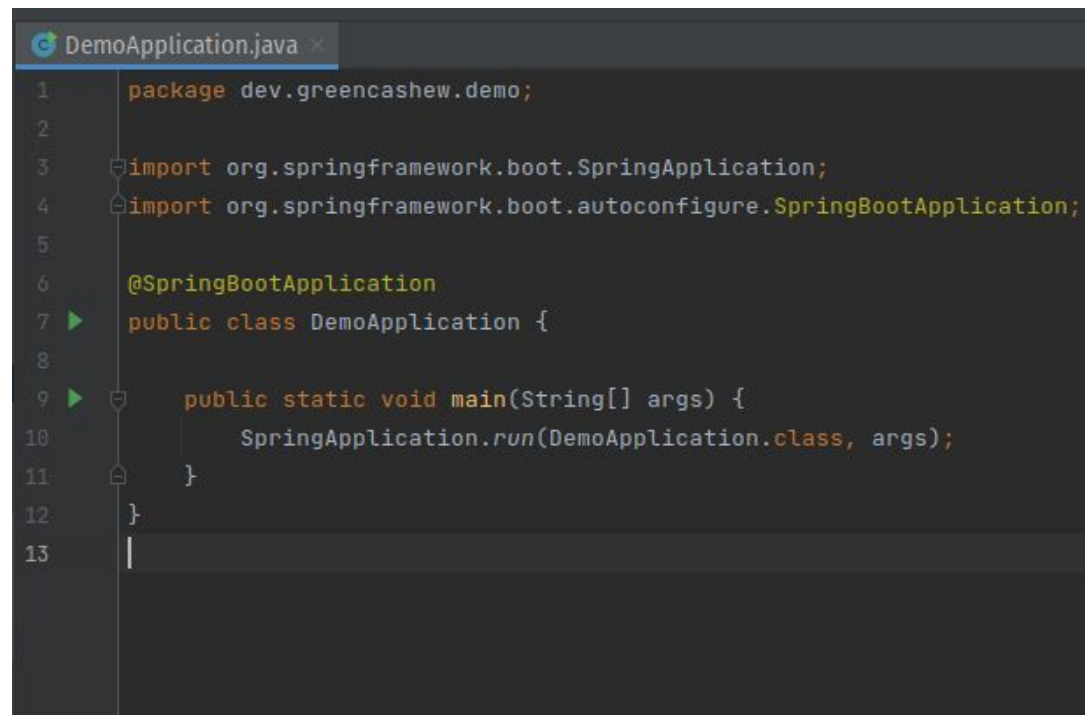
Wyjaśnienie Klasy DemoApplication

Metoda ***SpringApplication.run()*** jest odpowiedzialna za uruchomienie aplikacji

Tworzy ona **Inversion of Control (IoC) Container**, który jest odpowiedzialny za zarządzanie cyklem życia obiektów podanych w konfiguracjach aplikacji.

Adnotacja **@SpringBootApplication** składa się z:

- **@SpringBootConfiguration** - Główna klasa konfiguracji spring boot-a
- **@EnableAutoConfiguration** - Pozwala Spring Boot-owi na dodawanie domyślnych konfiguracji, na podstawie zależności springa dodanych w **pom.xml**
- **@ComponentScan** - informuje Springa, aby przeszukał projekt w poszukiwaniu klas konfiguracji lub komponentów, które mają być dodane do kontenera IoC.



```
1 package dev.greencashew.demo;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class DemoApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(DemoApplication.class, args);
11     }
12 }
13
```

- Występuje silny coupling logiczny pomiędzy klasami
- Przypus testowania całego modułu

Dependency Injection / Inversion of Control

```
public class NoDependencyInjection {
    public static void main(String[] args) {
        GoldenTransaction goldenTransaction = new GoldenTransaction();
        goldenTransaction.complete();

        CryptoCurrencyTransaction cryptoCurrencyTransaction = new CryptoCurrencyTransaction();
        cryptoCurrencyTransaction.complete();
    }
}

record Seller(String name) {}

record Client(String name) {}

class GoldenTransaction {
    private Seller seller = new Seller( name: "Maciej");
    private Client client = new Client( name: "Wojciech");

    void complete() {
        System.out.println("GoldenTransaction between " + client.name() + " and " + seller.name() + " completed");
    }
}

class CryptoCurrencyTransaction {
    private Seller seller = new Seller( name: "Maciej");
    private Client client = new Client( name: "Wojciech");

    void complete() {
        System.out.println("CryptoCurrencyTransaction between " + client.name() + " and " + seller.name() + " completed");
    }
}
```

```
public class DependencyInjection {
    public static void main(String[] args) {
        Seller seller = new Seller( name: "Maciej");
        Client client = new Client( name: "Wojciech");

        GoldenTransaction goldenTransaction = new GoldenTransaction(seller, client);
        goldenTransaction.complete();

        CryptoCurrencyTransaction cryptoCurrencyTransaction = new CryptoCurrencyTransaction(seller, client);
        cryptoCurrencyTransaction.complete();
    }
}

record Seller(String name) {}

record Client(String name) {}

class GoldenTransaction {
    private final Seller seller;
    private final Client client;

    public GoldenTransaction(Seller seller, Client client) {
        this.seller = seller;
        this.client = client;
    }

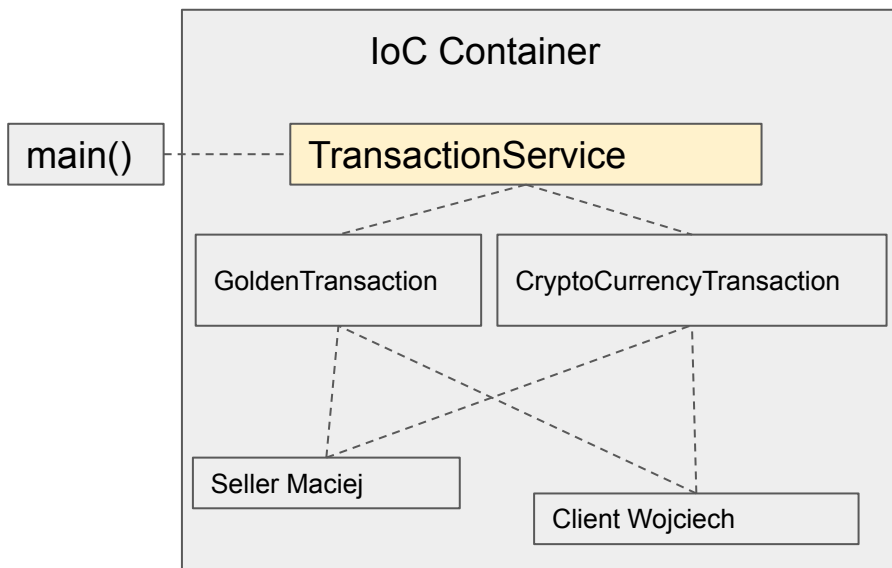
    void complete() {
        System.out.println("GoldenTransaction between " + client.name() + " and " + seller.name() + " completed");
    }
}

class CryptoCurrencyTransaction {
    private final Seller seller;
    private final Client client;

    public CryptoCurrencyTransaction(Seller seller, Client client) {
        this.seller = seller;
        this.client = client;
    }

    void complete() {
        System.out.println("CryptoCurrencyTransaction between " + client.name() + " and " + seller.name() + " completed");
    }
}
```

Dependency Injection / Inversion of Control - Spring



- Tylko IoC container wie jaki obiekt zostanie wstrzyknięty

```
@SpringBootApplication
public class SpringDependencyInjectionApplication {

    public static void main(String[] args) {
        final ConfigurableApplicationContext applicationContext = SpringApplication.run(SpringDependencyInjectionApplication.class, args);
        var transactionService : SpringDependencyInjectionApplication.TransactionService = applicationContext.getBean(TransactionService.class);
        transactionService.run();
    }

    @Service // Creating bean with annotation
    record TransactionService(GoldenTransaction goldenTransaction, CryptoCurrencyTransaction cryptoCurrencyTransaction) {
        public void run() {
            goldenTransaction.complete();
            cryptoCurrencyTransaction.complete();
        }
    }

    record Client(String name) { }
    record Seller(String name) { }

    @Component // Creating bean with annotation
    record GoldenTransaction(Seller seller, Client client) {
        public void complete() {
            System.out.println("GoldenTransaction between " + client.name() + " and " + seller.name() + " completed!");
        }
    }

    @Component // Creating bean with annotation
    record CryptoCurrencyTransaction(Seller seller, Client client) {
        void complete() {
            System.out.println("CryptoCurrencyTransaction between " + client.name() + " and " + seller.name() + " completed!");
        }
    }

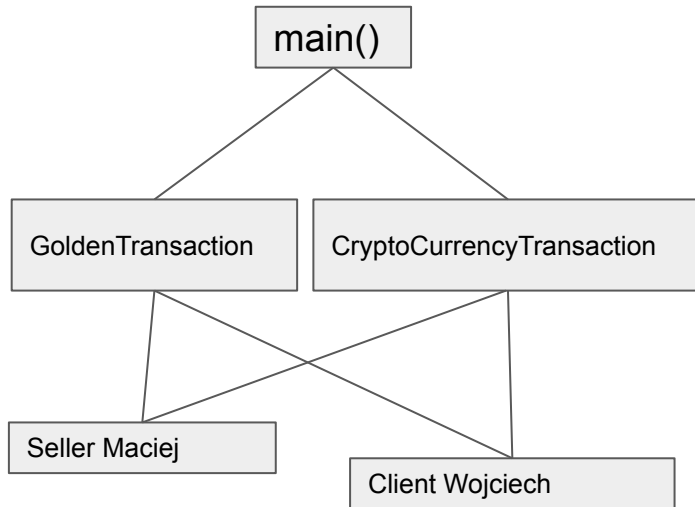
    @Configuration // Creating beans with configuration class
    static class DependencyInjectionConfiguration {
        @Bean
        Seller getSeller() {
            return new Seller( name: "Maciej");
        }

        @Bean
        Client getClient() {
            return new Client( name: "Wojciech");
        }
    }
}
```


Dependency Injection / Inversion of Control

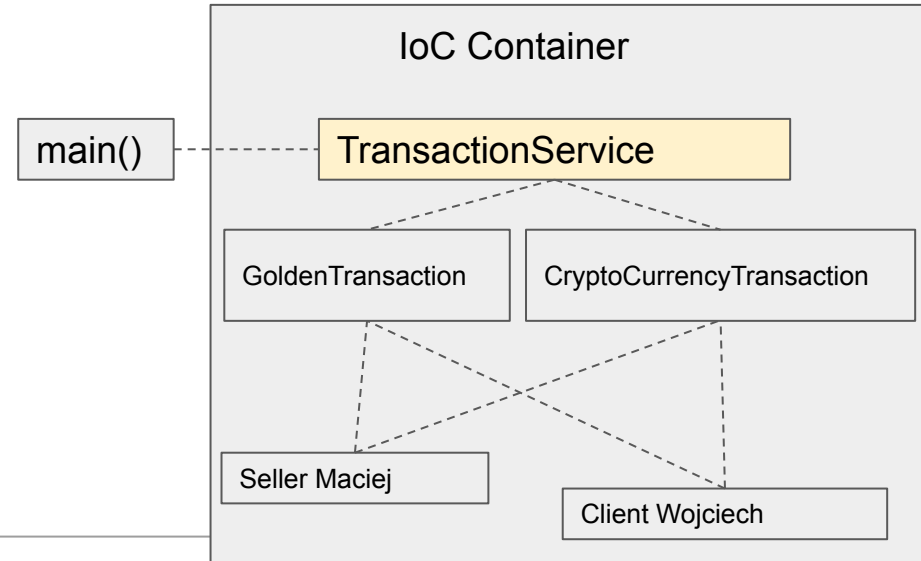
Klasyczne podejście

- Zależności są tworzone od klasy **main()** w dół
- Przymus modyfikacji klas zależnych w przypadku zmiany konstruktora



Spring

- Zależności są wstrzykiwane z zewnętrznego kontenera IoC
- Niski logiczny coupling
- Większa kontrola nad obiektami (np. mogą być singletonem)



Dependency Inversion Principle

Zależności pomiędzy dwiema klasami są separowane interfejsem

Klasa A

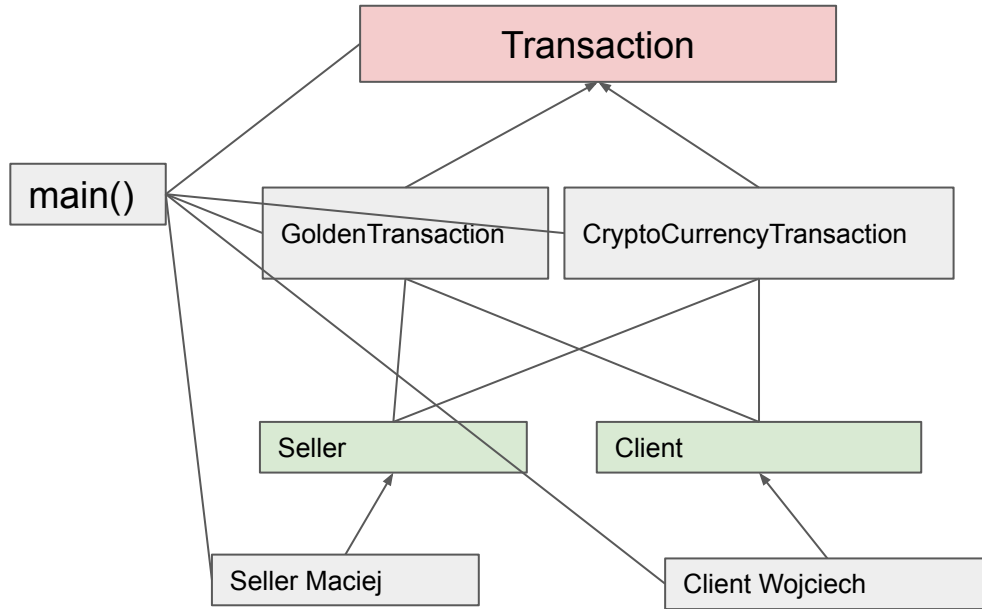
Interfejs

Klasa B

```
NoInversionPrinciple.java
1 class NoInversionPrinciple {
2
3     public static void main(String[] args) {
4         final Driver msciwojDriver = new Driver();
5         msciwojDriver.drive(new Tico()); // Can't drive Matiz
6     }
7 }
8
9 class Driver {
10     void drive(Tico car) {
11         System.out.println("Driving " + car.getClass().getSimpleName());
12     }
13 }
14
15 class Tico {
16 }
17
18 class Matiz {
19 }
20
```

```
DependencyInversionPrinciple.java
1 class DependencyInversionPrinciple {
2
3     public static void main(String[] args) {
4         final Driver driver = new Driver();
5         driver.drive(new Tico());
6         driver.drive(new Matiz());
7     }
8 }
9
10 interface Car {
11 }
12
13 class Driver {
14     void drive(Car car) { System.out.println("Driving " + car.getClass().getSimpleName()); }
15 }
16
17 class Tico implements Car {
18 }
19
20 class Matiz implements Car {
21 }
22
23
```


Dependency Inversion Principle and Dependency Injection bez Spring



```
class DependencyInjectionInversion {
    public static void main(String[] args) {
        Seller seller = new Seller( name: "Maciej");
        Client client = new Client( name: "Wojciech");

        var transactions :List<Record & DependencyInjectionInversion.Transaction> = List.of(
            new GoldenTransaction(seller, client),
            new CryptoCurrencyTransaction(seller, client)
        );

        transactions.forEach(Transaction::complete);
    }

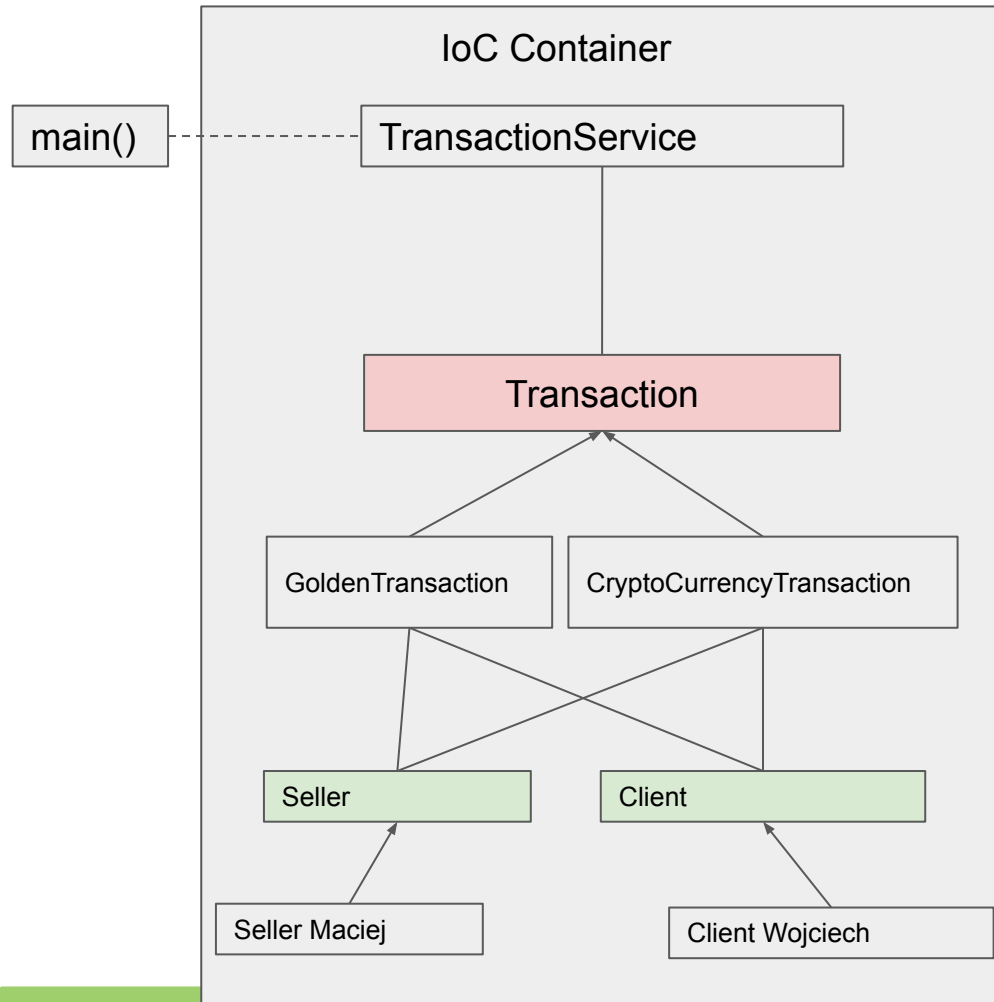
    interface Transaction {
        void complete();
    }

    record GoldenTransaction(Seller seller, Client client) implements Transaction {
        public void complete() {
            System.out.println("GoldenTransaction between " + client.name() + " and " + sel
        }
    }

    record CryptoCurrencyTransaction(Seller seller, Client client) implements Transaction {
        public void complete() {
            System.out.println("CryptoCurrencyTransaction between " + client.name() + " and
        }
    }

    record Seller(String name) { }
    record Client(String name) { }
}
```

Dependency Inversion Principle and Dependency Injection ze Spring



```
@SpringBootApplication
public class SpringDependencyInjectionInversionApplication {

    public static void main(String[] args) {
        final ConfigurableApplicationContext applicationContext = SpringApplication.run(SpringDependencyInjectionInversionApplication.class,
        var transactionService :SpringDependencyInjectionInversionApplication.TransactionService = applicationContext.getBean(TransactionService.class);
        transactionService.run();
    }

    @Service // Creating bean with annotation
    record TransactionService(List<Transaction> transactions) {
        public void run() {
            transactions.forEach(Transaction::complete);
        }
    }

    interface Transaction {
        void complete();
    }

    @Component // Creating bean with annotation
    record GoldenTransaction(Seller seller, Client client) implements Transaction {
        public void complete() {
            System.out.println("GoldenTransaction between " + client.name() + " and " + seller.name() + " completed!");
        }
    }

    @Component // Creating bean with annotation
    record CryptoCurrencyTransaction(Seller seller, Client client) implements Transaction {
        public void complete() {
            System.out.println("CryptoCurrencyTransaction between " + client.name() + " and " + seller.name() + " completed!");
        }
    }

    @Configuration // Creating beans with configuration class
    static class DependencyInjectionConfiguration {
        @Bean
        Seller getSeller() { return new Seller( name: "Maciej"); }

        @Bean
        Client getClient() { return new Client( name: "Wojciech"); }
    }

    record Client(String name) { }
    record Seller(String name) { }
}
```

Zarządzanie właściwościami (properties)

Właściwości do naszej aplikacji możemy ustawiać na różne sposoby:

- Za pomocą argumentów w commandline
- Za pomocą zmiennych środowiskowych
- Za pomocą adnotacji **@PropertySource**
- Za pomocą **application.properties**, **application.yml**
- Oraz innych metod

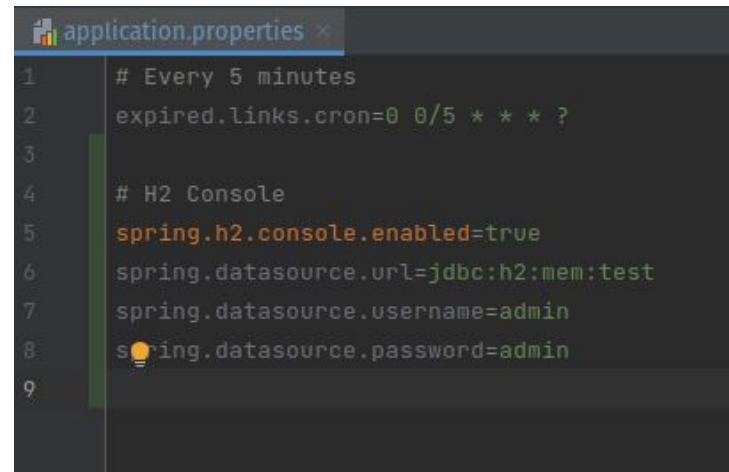
Nie powinno się trzymać właściwości w kodzie!

Właściwości możemy wstrzyknąć za pomocą adnotacji **@Value** np.

```
@Value("${spring.application.name}")
```

Kiedy najczęściej używa się properties?

- W celu przekazywania connection string do bazy danych (host, login, hasło) [za pomocą environment variables],
- W celu szybkiego zmianu stanu aplikacji, podczas jej działania
- W celu szybkiej edycji zmieniających się często rzeczy bez potrzeby budowania projektu
- W celu sterowania różnymi zależnościami dla różnych środowisk

A screenshot of an IDE window titled 'application.properties'. The window shows a list of configuration properties for a Spring application. Line 1: '# Every 5 minutes'. Line 2: 'expired.links.cron=0 0/5 * * * ?'. Line 3: (empty). Line 4: '# H2 Console'. Line 5: 'spring.h2.console.enabled=true'. Line 6: 'spring.datasource.url=jdbc:h2:mem:test'. Line 7: 'spring.datasource.username=admin'. Line 8: 'spring.datasource.password=admin'. Line 9: (empty). The text is in a dark-themed font with syntax highlighting.

```
1 # Every 5 minutes
2 expired.links.cron=0 0/5 * * * ?
3
4 # H2 Console
5 spring.h2.console.enabled=true
6 spring.datasource.url=jdbc:h2:mem:test
7 spring.datasource.username=admin
8 spring.datasource.password=admin
9
```

Zarządzanie właściwościami (properties) - Zadanie

1. Zaimplementuj interfejs **CommandLineRunner** w głównej klasie aplikacji.
2. Utwórz springowy component o nazwie **Metadata**, zawierający metodę wyświetlającą w konsoli metadane projektu **printMetadata()**
3. Dodaj właściwość:
 - a. **applicationName** - pobraną z **application.properties**
 - b. **developerSentence** - Wstrzykniętą bezpośrednio przez **@Value**
 - c. Zmienną środowiskową
 - d. Domyślną wartość `${spring.application.name:Unknown Name}`
 - e. * **authors** - listę autorów z pliku **application.properties**
 - f. * która Wyświetli aktywne **profile** `spring.profiles.active`, a w przypadku braku, zwróci informację o braku aktywnych profili.



The image shows a snippet of the application.properties file. The first line is `spring.application.name=Aplikacja testowa`. The text 'Aplikacja testowa' is underlined in the original image.

```
@SpringBootApplication
public class ApplicationPropertiesExampleApplication implements CommandLineRunner {

    private final Metadata metadata;

    public ApplicationPropertiesExampleApplication(Metadata metadata) {
        this.metadata = metadata;
    }

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

    @Override
    public void run(String... args) throws Exception {
        metadata.printMetadata();
    }
}
```

```
@Component
record Metadata(String applicationName) {

    Metadata(
        @Value("${spring.application.name}") String applicationName
    ) {
        this.applicationName = applicationName;
    }

    void printMetadata() {
        System.out.println("----- METADATA -----");
        System.out.println("APPLICATION NAME: " + applicationName);
        System.out.println("----- END METADATA -----");
    }
}
```

Używanie Loggera zamiast System.out.println()

Dlaczego Logger?

- Separacja pod względem ważności np. w **java.util.logging**
 - SEVERE (największy)
 - WARNING
 - INFO
 - CONFIG
 - FINE
 - FINER
 - FINEST
- Separacja pod względem plików
- Możliwość logowania do pliku
- Łatwa modyfikacja formatu logów

```
@Component
record Metadata(String developerSentence
                String applicationName
                List<String> authors
                String unknownSystemProperty
                String javaHome
                String activeProfiles) {

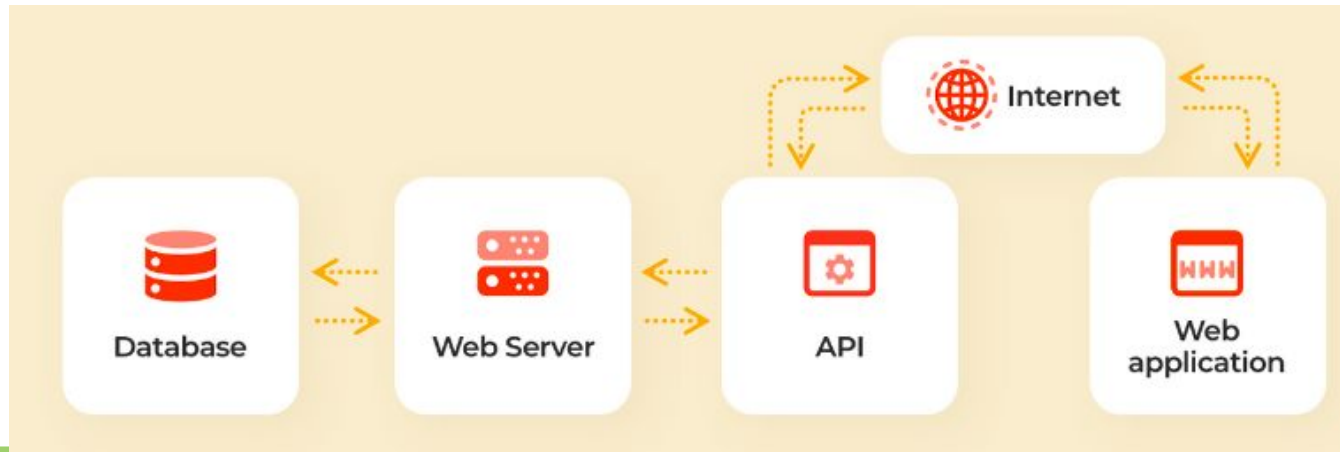
    private final static Logger LOGGER = Logger.getLogger(Metadata.class.getName());

    Metadata(
        @Value("Tygodniem programowania można zaoszczędzić 2 godziny projektowania
        :)") String developerSentence
        @Value("${spring.application.name}") String applicationName
        @Value("${application.authors}") List<String> authors
        @Value("${application.unknown.property:Unknown property}") String
        unknownSystemProperty
        @Value("${JAVA_HOME}") String javaHome
        @Value("${spring.profiles.active:No Active Profiles!}") String
        activeProfiles
    ) {
        this.developerSentence = developerSentence;
        this.applicationName = applicationName;
        this.authors = authors;
        this.unknownSystemProperty = unknownSystemProperty;
        this.javaHome = javaHome;
        this.activeProfiles = activeProfiles;
    }

    void printMetadata() {
        LOGGER.info("----- METADATA -----");
        LOGGER.info("DEVELOPER SENTENCE: " + developerSentence);
        LOGGER.info("APPLICATION NAME: " + applicationName);
        LOGGER.info("AUTHORS: " + String.join(" <=> ", authors));
        LOGGER.info("UNKNOWN PROPERTY: " + unknownSystemProperty);
        LOGGER.info("JAVA HOME: " + javaHome);
        LOGGER.info("ACTIVE PROFILES: " + activeProfiles);
        LOGGER.info("----- END METADATA -----");
    }
}
```

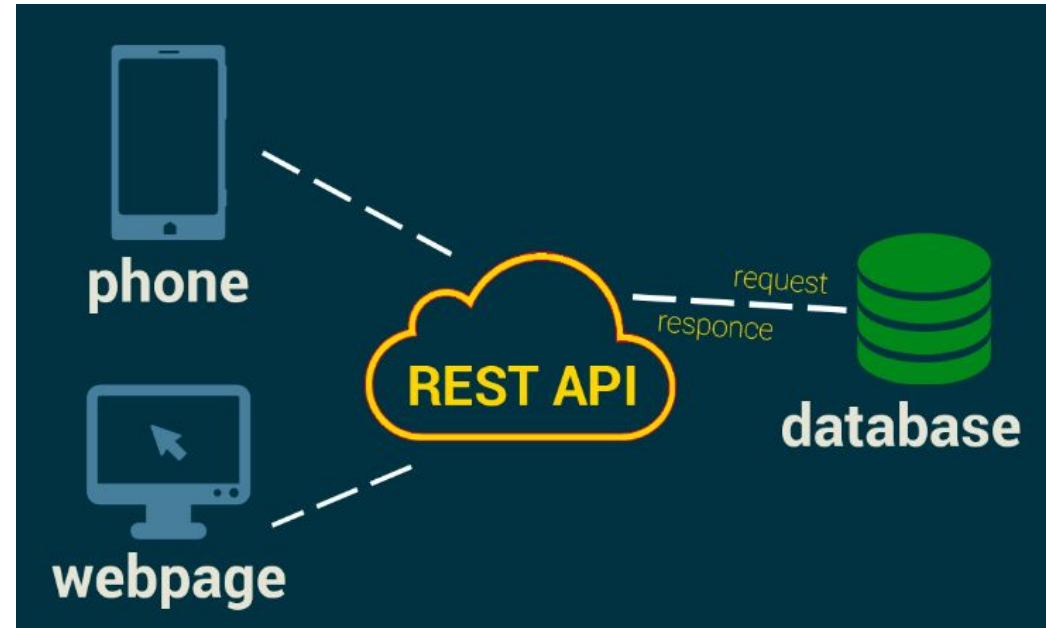
API

- Współczesne aplikacje serwerowe ze względu na wieloplatformowość oraz złożoność rozwiązań nie generują gotowych widoków
- Komunikacja między frontendem (aplikacją webową, aplikacją na telefon, bezpośrednio aplikacją klienta), a backendem (serwerem) odbywa się poprzez wymianę konkretnych zasobów, poprzez API (Application Programming Interface)
 - Może być to biblioteka z kodem źródłowym (np. zależność dodana mavenem)
 - Usługa sieciowa (interakcja za pomocą HTTP)



REST API

- **Architektura klient serwer (Client Server)** - Klient wysyła żądanie http, a serwer na nie odpowiada
- **Bezstanowość (Stateless)** - żądanie musi zawierać wszystkie niezbędne informacje do jego przetworzenia
- **Cache** - jeżeli serwer przekazuje iż odpowiedź jest cache-owalna, klient może ją wykorzystać ponownie bez ponownego odpytywania serwera
- **Jednolity interfejs (Uniform Interface)** - Serwer z danymi powinien również informować o dostępnych akcjach i zasobach
- **Wielowarstwowość (Layered System)** - Pomiędzy serwerem a klientem mogą występować warstwy (np. load balancing, cache)



Żądania i odpowiedzi HTTP

Przy aktualizacji:

→ **PUT** - Podmiana zasobu na nowy

← **200 OK, 204 No Content** - zasób udało się poprawnie podmienić

← **201 Created** - zasób udało się utworzyć

← **404 Not Found** - jeżeli zasób nie istnieje

→ **PATCH** - Częściowa aktualizacja istniejącego obiektu

← **204 No Content** - jeżeli zasób udało się zaktualizować

← **404 Not Found** - jeżeli zasób nie istnieje

Przy usuwaniu:

→ **DELETE**

← **202 Accepted** - żądanie zostało poprawnie odebrane, nie wiadomo kiedy nastąpi usunięcie

← **204 No Content** - jeżeli zasób został usunięty lub nie został znaleziony

← **200 OK** - jeżeli operacja usunięcia została wykonana i przesyłamy informacje zwrotne

← **404 Not Found** - jeżeli zasób nie został znaleziony

Przy odczycie:

→ **GET**

← **200 OK** - Zasób znaleziony i przesyłamy informacje zwrotne

← **404 Not Found** - jeżeli zasób nie istnieje

Przy zapisie danych:

→ **POST**

← **201 Created** + adres nowego zasobu w nagłówku **Location**

Listę wszystkich zapytań HTTP można znaleźć tutaj: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods

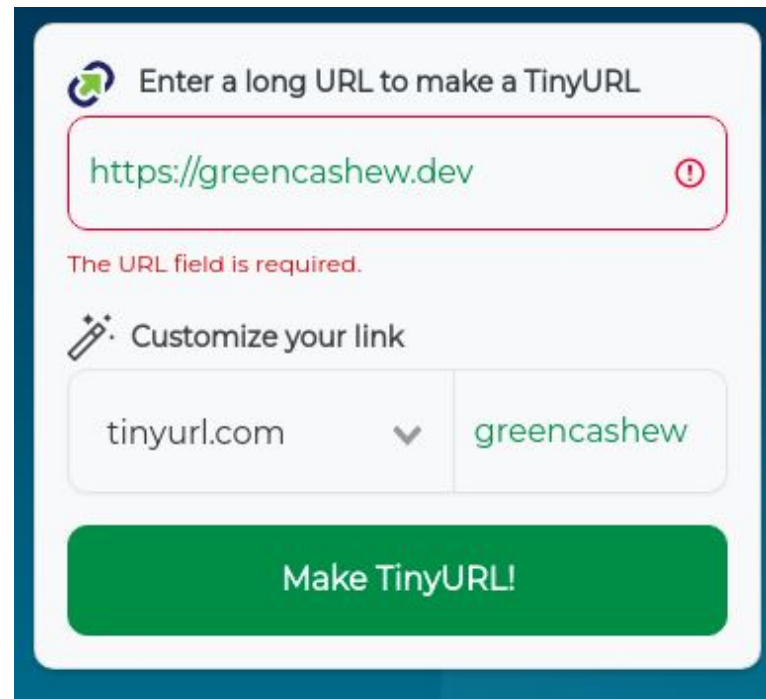
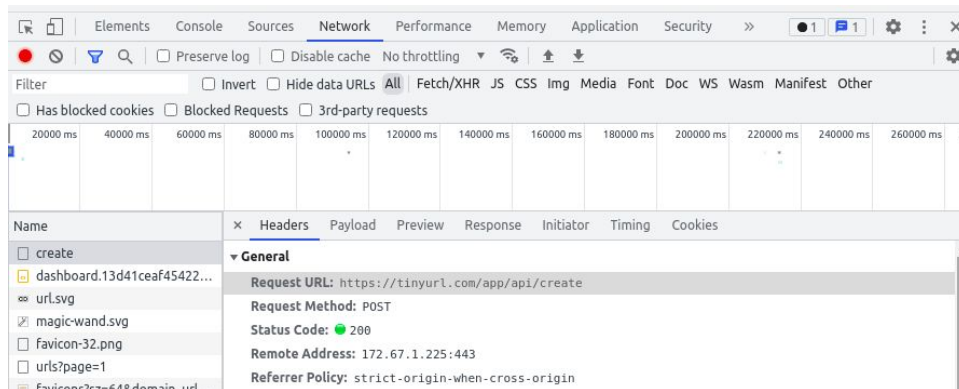
Listę wszystkich odpowiedzi HTTP można znaleźć tutaj: https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Projekt - Skracacz Linków - Komunikacja z API

Podobna aplikacja: <https://tinyurl.com>

Sposób działania:

1. W aplikacji frontend-owej wpisujemy wymagane dane
2. Klikamy Make TinyURL!
3. Frontend wysyła http request do API, z payloadem zawierającym, dane do skrócenia linku
4. Serwer przekazuje odpowiedź, która jest obsługana przez frontend.



Na stronie klikamy prawy przycisk myszy i wybieramy opcję: **Zbadaj Element / Inspect**

Projekt - Skracacz Linków - Komunikacja z API

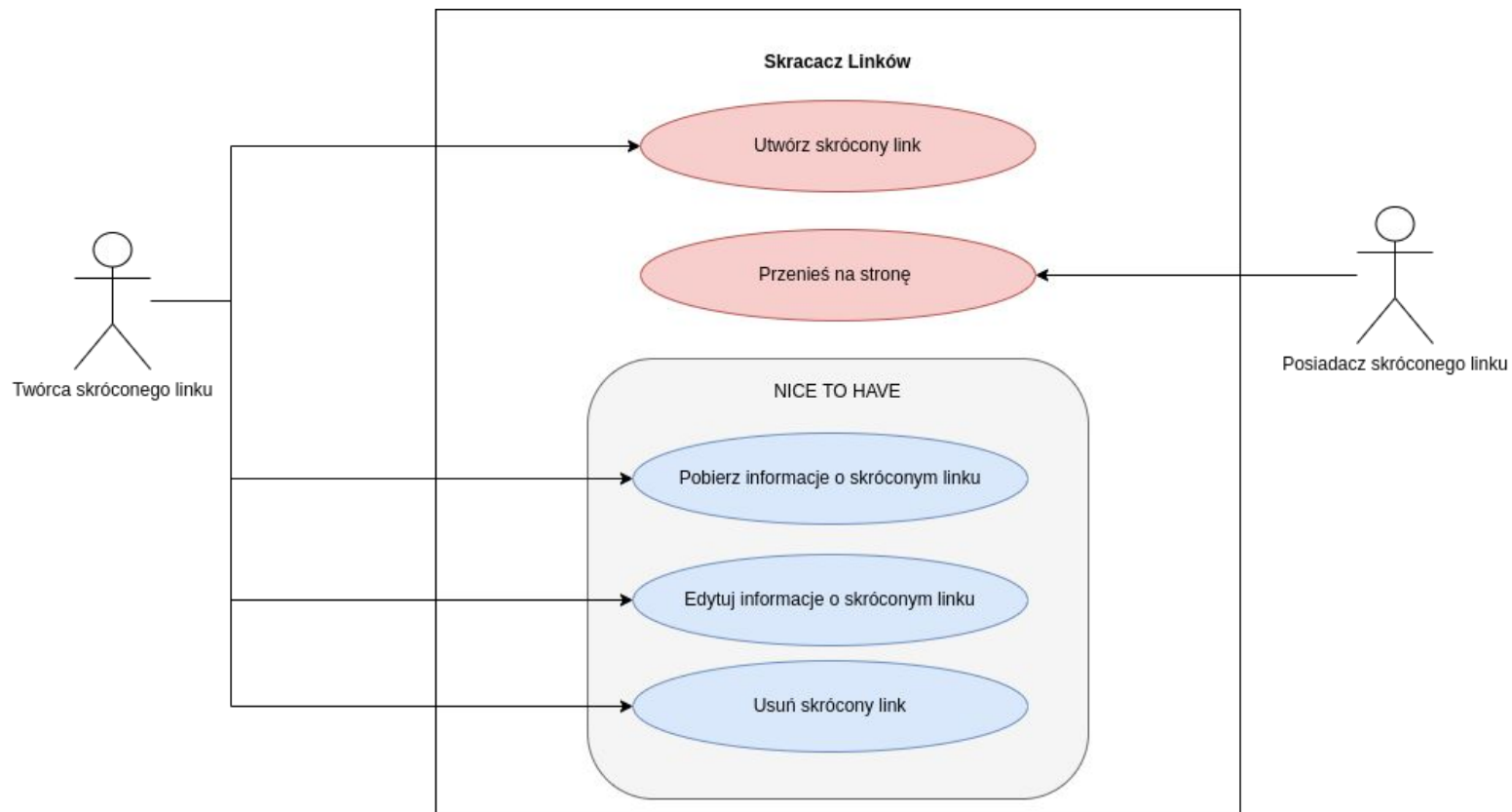
Wysyłany Payload

```
× Headers Payload Preview Respo
▼ Request Payload view source
  ▼ {url: "greencashew.dev", domain: "t
    alias: "greencashew"
    busy: true
    domain: "tinyurl.com"
    ▼ errors: {errors: {}}
      errors: {}
    successful: false
    tags: []
    url: "greencashew.dev"
```

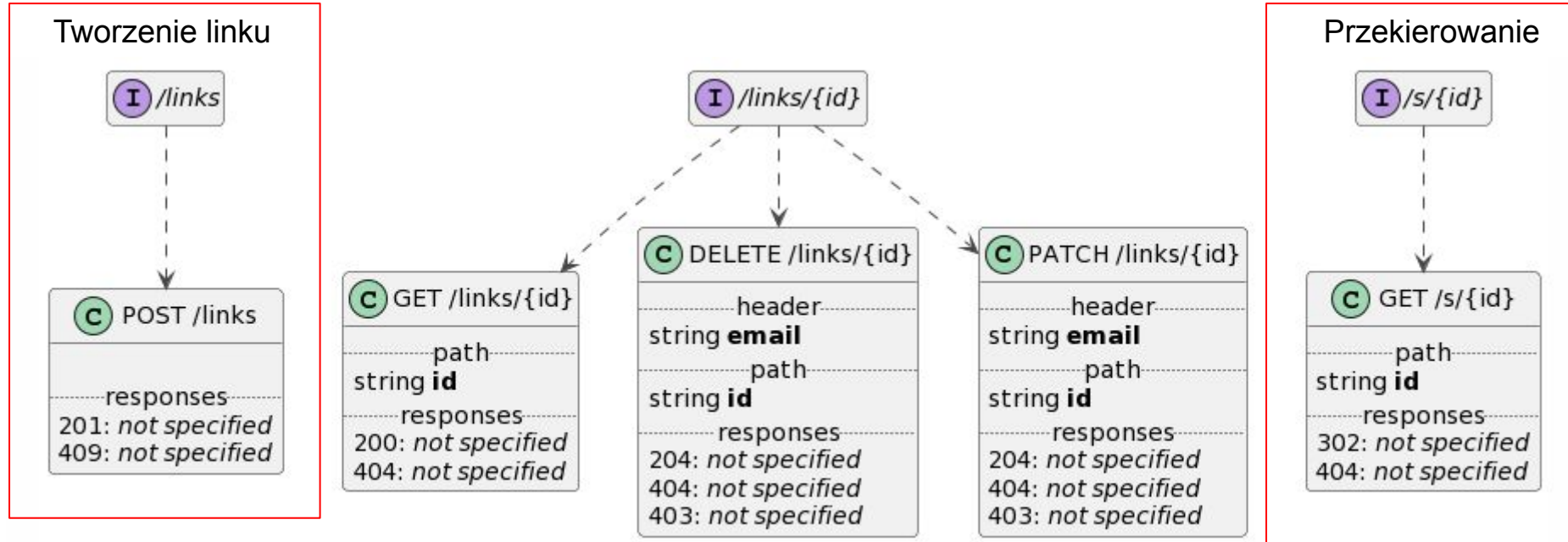
Otrzymana odpowiedź

```
Headers Payload Preview Response Initiator Timing Cookies
1 {
  "data": [
    {
      "url": "http:\\\\greencashew.dev",
      "active": true,
      "hash": "ybbgwk3h",
      "no_affiliate": false,
      "aliases": [
        {
          "domain": "tinyurl.com",
          "alias": "greencashew",
          "is_main": true,
          "is_archived": false,
          "is_terminated": false,
          "is_deleted": false,
          "no_affiliate": false,
          "stats": {
            "enabled": false,
            "public": false
          },
          "tags": [],
          "created_at": "2022-04-01T08:08:16.000000Z",
          "expires_at": null,
          "tiny_url": "https:\\\\tinyurl.com\\greencashew",
          "affiliates": false,
          "read_only": true
        }
      ]
    }
  ]
}
```

Projekt - Skracacz Linków - Use Case



Projekt - Skracacz Linków - Gotowy Projekt



Dokumentacja API:

<https://spring-link-shortener.herokuapp.com/swagger-ui.html>

Skracacz Linków

◀ ▶ ↻

🔒 spring-link-shortener.herokuapp.com

Link identifier:

Email:

Target URL:

Expiration Date (optional):
 📅

Link shortened: <https://spring-link-shortener.herokuapp.com/s/testowy-alias>

Skracacz Linków - Dokumentacja

redirect-controller

GET /s/{id}

Retrieve link by it's identifier

Parameters

Name	Description
id <small>required</small> string (path)	<input type="text" value="link-alias"/>

Responses

Code	Description
302	User is redirected to expected location.
404	Shortened link not found.

Media type:

Example Value

```
{
  "errorMessage": "Shortened link link-alias not found."
}
```

link-manage-controller

POST /links

Create shortened link

Parameters

No parameters

Request body required

Example Value | Schema

```
{
  "id": "link-alias",
  "email": "test@greencashew.dev",
  "targetUrl": "https://github.com/greencashew/warsztaty-podstawy-springa",
  "expirationDate": "2054-06-23"
}
```

Responses

Code	Description
201	Shortened link.
409	Link with same identifier already exists.

Media type:

Controls Accept header.

Example Value

```
{
  "id": "link-alias",
  "targetUrl": "https://github.com/greencashew/warsztaty-podstawy-springa",
  "expirationDate": "2054-06-23",
  "visits": 0,
  "shortenedLink": "http://localhost:8080/s/link-alias"
}
```

Media type:

Example Value

```
{
  "errorMessage": "Link with id: link-alias already exists."
}
```

CRUD

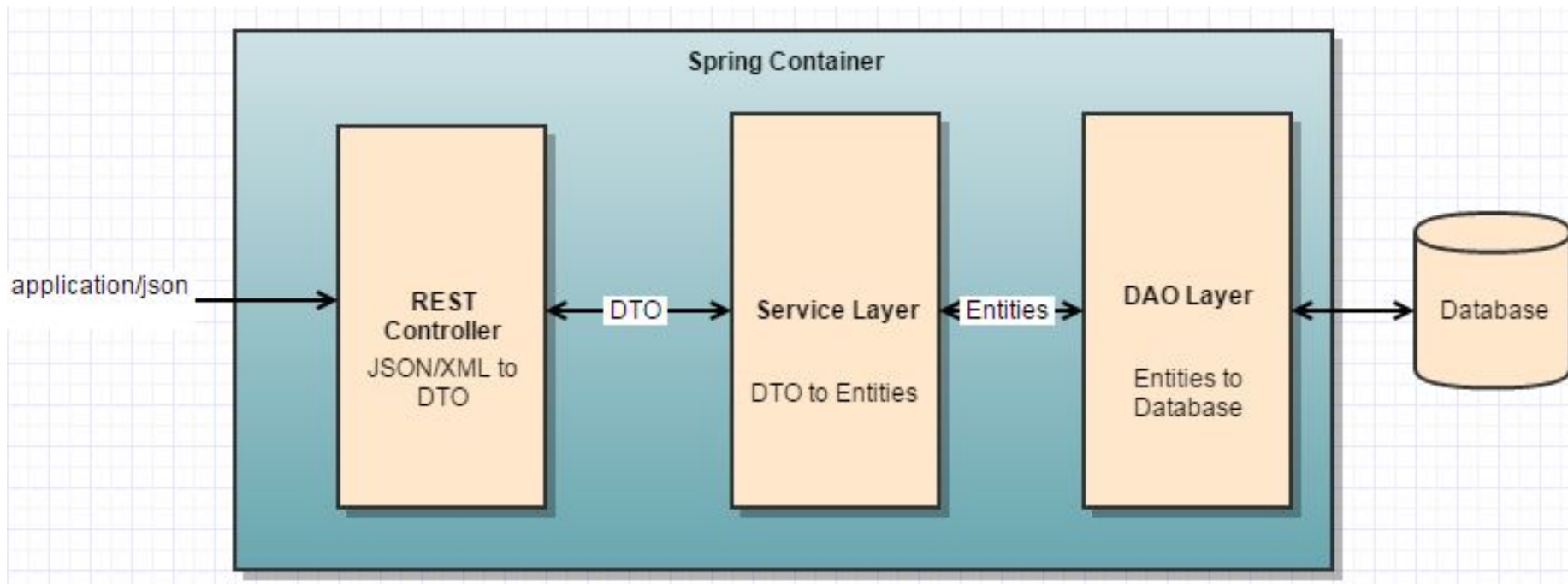
CRUD oznacza podstawowe operacje implementowane w aplikacjach bazodanowych.

Create - dodanie nowych zasobów

Read - odczytanie istniejących zasobów

Update - modyfikacje istniejących zasobów

Delete - usunięcie zasobów



Tworzenie Projektu - start.spiring.io



Project

☒ Maven Project ☐ Gradle Project

Language

☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (M2) ☐ 2.7.0 (SNAPSHOT) ☐ 2.7.0 (M3)
☐ 2.6.7 (SNAPSHOT) ☒ 2.6.6 ☐ 2.5.13 (SNAPSHOT) ☐ 2.5.12

Project Metadata

Group dev.greencashew

Artifact link-shortener

Name link-shortener

Description Link Shrotener App

Package name dev.greencashew.link-shortener

Packaging ☒ Jar ☐ War

Java ☐ 18 ☒ 17 ☐ 11 ☐ 8

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

H2 Database SQL

Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

Lombok DEVELOPER TOOLS

Java annotation library which helps to reduce boilerplate code.

Spring Boot DevTools DEVELOPER TOOLS

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

GENERATE CTRL + G

EXPLORE CTRL + SPACE

SHARE...

Tworzenie Projektu - Github

- Ustawiamy naszą nazwę i email w gicie, w konsoli wpisujemy:

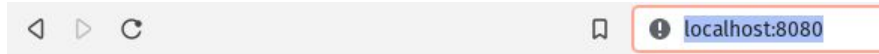
```
git config --global user.name "Nazwa Użytkownika"  
git config --global user.email "nazwa_uzytkownika@users.noreply.github.com"
```

- Tworzymy nowe repozytorium na githubie <https://github.com/new>, o nazwie **skracacz-linkow**
- Inicjalizujemy repozytorium w naszym projekcie, w konsoli wpisujemy:

```
echo "# skracacz-linkow" >> README.md  
git init  
git add README.md  
git commit -m "first commit"  
git branch -M main  
git remote add origin https://github.com/TWOJA_NAZWA_UZYTEKOWNIKA/skracacz-linkow.git  
git push -u origin main
```

Uruchamianie aplikacji lokalnie

1. Czekamy aż IDE zaimportuje mavenowe zależności
2. Uruchamiamy aplikację
3. Wchodzimy pod adres `http://localhost:8080/`



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Fri Apr 01 18:32:45 CEST 2022

There was an unexpected error (type=Not Found, status=404).

No message available

```
2022-04-01 18:34:17.501 INFO 14629 --- [ restartedMain] d.g.l.LinkShortenerApplication : Starting LinkShortenerApplication using Java 17.0.2 on kuba with PID 14629 (/home/janek/proj/spr...)
2022-04-01 18:34:17.503 INFO 14629 --- [ restartedMain] d.g.l.LinkShortenerApplication : No active profile set, falling back to 1 default profile: "default"
2022-04-01 18:34:17.548 INFO 14629 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-properties' to 'false' to disable
2022-04-01 18:34:17.548 INFO 14629 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'
2022-04-01 18:34:17.997 INFO 14629 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2022-04-01 18:34:18.005 INFO 14629 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 3 ms. Found 0 JPA repository interfaces.
2022-04-01 18:34:18.422 INFO 14629 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-04-01 18:34:18.429 INFO 14629 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-04-01 18:34:18.429 INFO 14629 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.60]
2022-04-01 18:34:18.462 INFO 14629 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-04-01 18:34:18.463 INFO 14629 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 914 ms
2022-04-01 18:34:18.483 INFO 14629 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2022-04-01 18:34:18.571 INFO 14629 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2022-04-01 18:34:18.578 INFO 14629 --- [ restartedMain] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:8c0b2b4e-c316-4bc1-8ca9-9a9a9a9a9a9a'
2022-04-01 18:34:18.672 INFO 14629 --- [ restartedMain] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
2022-04-01 18:34:18.698 INFO 14629 --- [ restartedMain] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.6.7.Final
2022-04-01 18:34:18.776 INFO 14629 --- [ restartedMain] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2022-04-01 18:34:18.840 INFO 14629 --- [ restartedMain] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
2022-04-01 18:34:18.951 INFO 14629 --- [ restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2022-04-01 18:34:18.955 INFO 14629 --- [ restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-04-01 18:34:18.987 WARN 14629 --- [ restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during
2022-04-01 18:34:19.191 INFO 14629 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2022-04-01 18:34:19.216 INFO 14629 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2022-04-01 18:34:19.224 INFO 14629 --- [ restartedMain] d.g.l.LinkShortenerApplication : Started LinkShortenerApplication in 1.983 seconds (JVM running for 2.289)
```

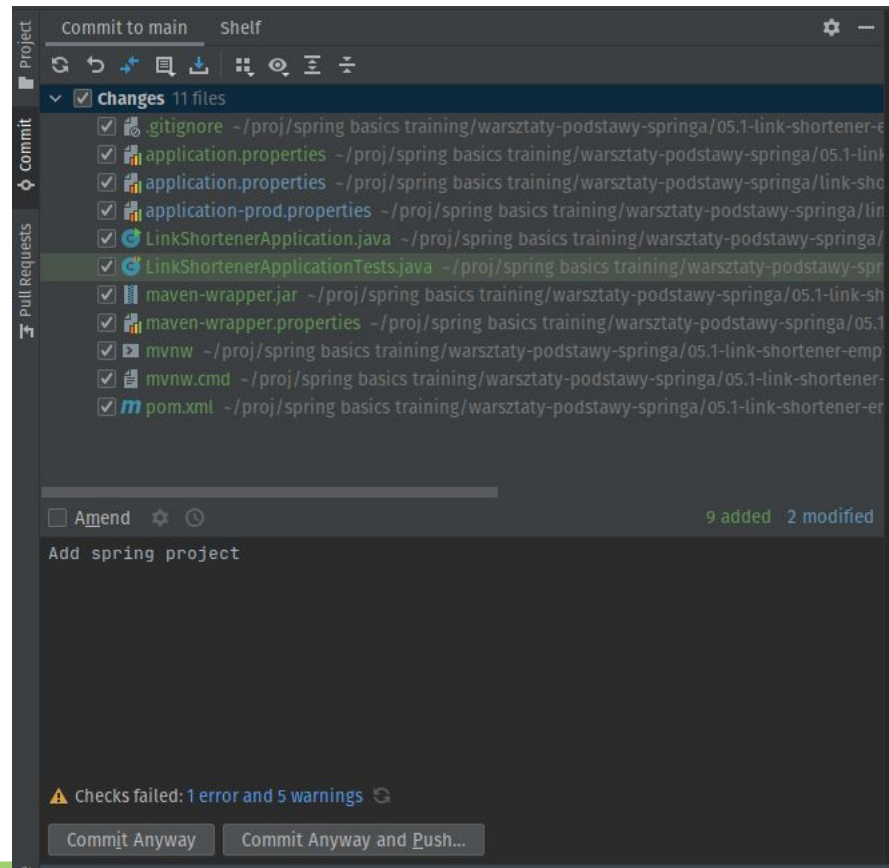
Dodanie projektu do gita

W konsoli:

- `git add --all`
- `git status`
- `git commit -m "Add spring project"`
- `git push`

```
➔ 05.1-link-shortener-empty-project git:(main) ✕ git commit -m "Add spring project"
[main de763b7] Add spring project
9 files changed, 634 insertions(+)
create mode 100644 .gitignore
create mode 100644 .mvn/wrapper/maven-wrapper.jar
create mode 100644 .mvn/wrapper/maven-wrapper.properties
create mode 100755 mvnw
create mode 100644 mvnw.cmd
create mode 100644 pom.xml
create mode 100644 src/main/java/dev/greencashew/linkshortener/LinkShortenerApplication.java
create mode 100644 src/main/resources/application.properties
create mode 100644 src/test/java/dev/greencashew/linkshortener/LinkShortenerApplicationTests.java
➔ 05.1-link-shortener-empty-project git:(main) ✕ git push
Enumerating objects: 26, done.
Counting objects: 100% (26/26), done.
Delta compression using up to 8 threads
Compressing objects: 100% (16/16), done.
Writing objects: 100% (25/25), 59.02 KiB | 29.51 MiB/s, done.
Total 25 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:greencashew/skracacz-linkow.git
 f30c791..de763b7  main -> main
```

W IntelliJ:



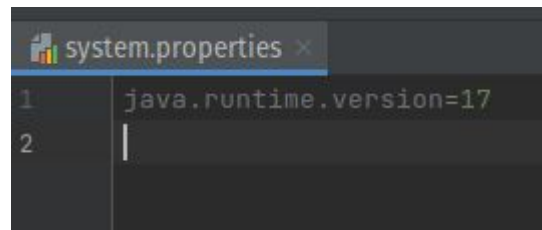
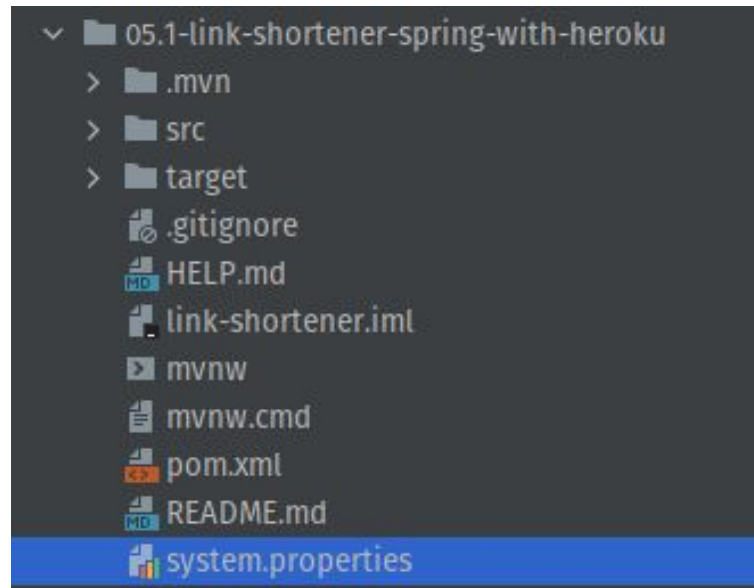
Integracja repozytorium z Heroku

- W głównym katalogu projektu dodajemy plik **system.properties** z wartością `java.runtime.version=17`
- Dodajemy plik do remote git repository
 - git add --all
 - git status
 - git commit -m "Add spring project"
 - git push

main 1 branch 0 tags

Go to file Add file Code

	greencashew Set heroku runtime environment	49a7352 1 minute ago 3 commits
.mvn/wrapper	Add spring project	17 minutes ago
src	Set heroku runtime environment	1 minute ago
.gitignore	Add spring project	17 minutes ago
README.md	first commit	1 hour ago
mvnw	Add spring project	17 minutes ago
mvnw.cmd	Add spring project	17 minutes ago
pom.xml	Set heroku runtime environment	1 minute ago
system.properties	Set heroku runtime environment	1 minute ago



Integracja repozytorium z Heroku

- <https://dashboard.heroku.com/new-app>
- W zakładce **Deploy**, w Deployment Method wybieramy **GitHub**.
- Integrujemy się z GitHubem.

The screenshot shows the Heroku dashboard for a personal account. The app name is 'greencashew-link-shortener'. The 'Deploy' tab is selected, showing options to 'Add this app to a pipeline' or 'Add this app to a stage in a pipeline to enable additional features'. A 'Choose a pipeline' dropdown is visible.

Deployment method



The 'Create New App' form shows the app name 'greencashew-link-shortener' with a green checkmark indicating it is available. The region is set to 'Europe'. There is an 'Add to pipeline...' button and a 'Create app' button.

Integracja repozytorium z Heroku

- Po zintegrowaniu z GitHubem, powinna pojawić się sekcja o nazwie **Connect to GitHub**
- Klikamy przycisk **Connect**

The screenshot shows the Heroku dashboard's 'Deploy' tab. At the top, there are navigation links: Overview, Resources, Deploy (active), Metrics, Activity, Access, and Settings. Below these, there are two main sections. The first section, 'Add this app to a pipeline', contains instructions on creating a new pipeline or adding the app to an existing one, and a 'Choose a pipeline' dropdown menu. The second section, 'Deployment method', shows three options: Heroku Git (Use Heroku CLI), GitHub (Connect to GitHub), and Container Registry (Use Heroku CLI). The 'Connect to GitHub' option is highlighted. Below this, the 'Connect to GitHub' section is visible, with instructions to connect the app to GitHub to enable code diffs and deploys. It includes a search bar with 'greencashew' and 'skracacz-linkow' entered, a 'Search' button, and a list of results showing 'greencashew/skracacz-linkow' with a 'Connect' button.

Overview Resources **Deploy** Metrics Activity Access Settings

Add this app to a pipeline

Create a new pipeline or choose an existing one and add this app to a stage in it.

Add this app to a stage in a pipeline to enable additional features

Pipelines let you connect multiple apps together and **promote code** between them. [Learn more](#)

Pipelines connected to GitHub can enable **review apps**, and create apps for new pull requests. [Learn more](#)

Choose a pipeline

Deployment method

Heroku Git Use Heroku CLI

GitHub Connect to GitHub

Container Registry Use Heroku CLI

Connect to GitHub

Connect this app to GitHub to enable code diffs and deploys.

Search for a repository to connect to

greencashew skracacz-linkow **Search**

Missing a GitHub organization? [Ensure Heroku Dashboard has team access](#)

greencashew/skracacz-linkow **Connect**



Integracja repozytorium z Heroku

- W sekcji **Automatic deploys** wybieramy przycisk **Enable Automatic Deploys**
- W sekcji **Manual deploy** klikamy **Deploy Branch**
- Po zbudowaniu i zdeployowaniu aplikacji możemy otworzyć stronę klikając **View**


Deploy a GitHub branch


This will deploy the current state of the branch you specify below. [Learn more.](#)


Choose a branch to deploy


 main 

Deploy Branch


Receive code from GitHub 

Build **main** 49a7352a 

Release phase 



Deploy to Heroku 

Your app was successfully deployed.

 **View**

App connected to GitHub

Code diffs, manual and auto deploys are available for this app.

Connected to  [greencashew/skracz-linkow](#) by  [greencashew](#) **Disconnect...**

Releases in the [activity feed](#) link to GitHub to view commit diffs

Automatic deploys



Enables a chosen branch to be automatically deployed to this app.

You can now change your main deploy branch from "master" to "main" for both manual and automatic deploys, please follow the instructions [here](#).

Enable automatic deploys from GitHub

Every push to the branch you specify here will deploy a new version of this app. **Deploys happen automatically:** be sure that this branch is always in a deployable state and any tests have passed before you push. [Learn more.](#)

Choose a branch to deploy

 main 

☐ Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.



Enable Automatic Deploys

Manual deploy

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. [Learn more.](#)

Choose a branch to deploy

 main 

Deploy Branch

greencashew-link-shortener.herokuapp.com

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Fri Apr 01 17:25:55 UTC 2022

There was an unexpected error (type=Not Found, status=404).

Tworzenie Restowego Kontrolera - Spring MVC

LinkManageController.java

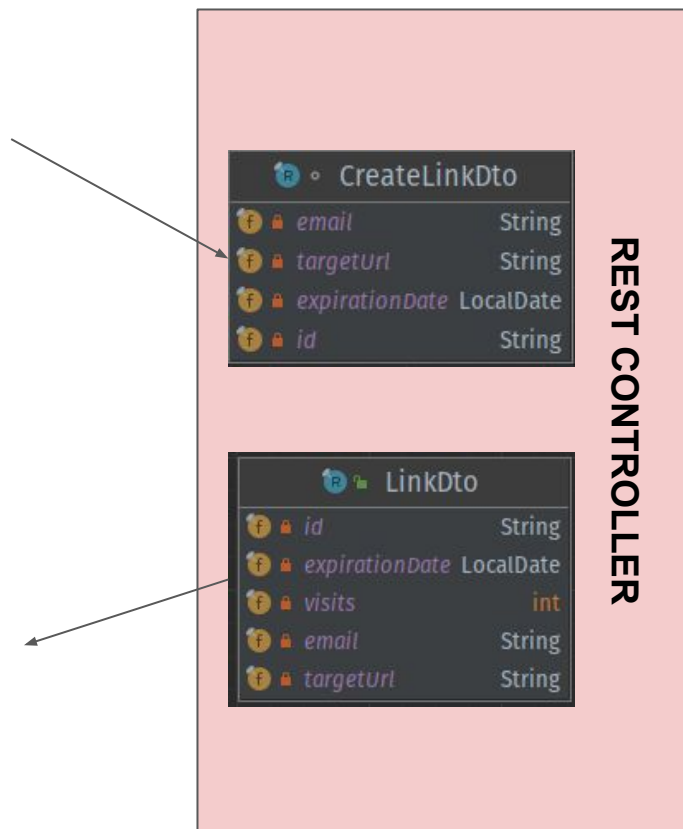
```
@RestController
@RequestMapping ("/links")
class LinkManageController {

    @PostMapping
    @ResponseBody
    @ResponseStatus (HttpStatus.CREATED)
    LinkDto createLink (@RequestBody CreateLinkDto link) {
        return link.toDto() ;
    }
}
```

CreateLinkDto.java

```
record CreateLinkDto(
    String id,
    String email,
    String targetUrl,
    LocalDate expirationDate) {

    LinkDto toDto() {
        return new LinkDto(
            id,
            email,
            targetUrl,
            expirationDate,
            0
        );
    }
}
```



Tworzenie linku

I /links

C POST /links

responses

201: not specified
409: not specified

Redirect Controller

Redirect musi przekazać odpowiednie http status kod **302 Found**, wraz z nową lokacją, który informuje przeglądarkę o przekierowaniu.

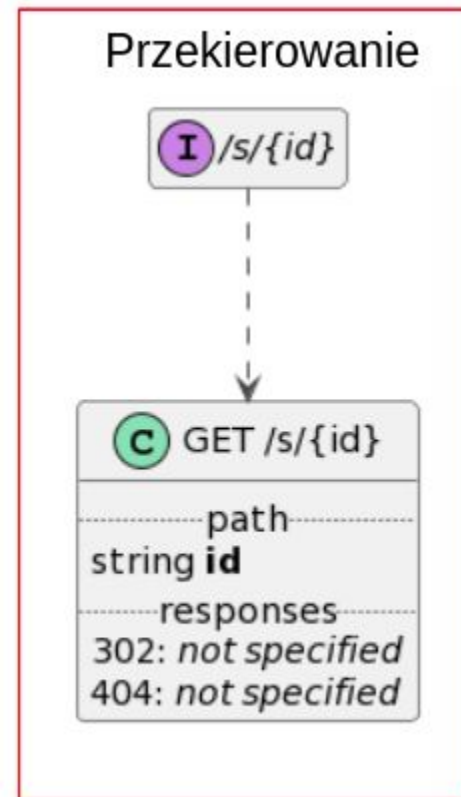
Adnotacje, które powinny być użyte w implementacji:

- `@RequestMapping`
- `@GetMapping`
- `@PathVariable`

Zaimplementowana metoda redirecta:

```
RedirectController.java
```

```
public void redirectLink (  
    @PathVariable String id, HttpServletResponse httpServletResponse) throws IOException {  
    httpServletResponse.sendRedirect( "https://github.com/greencashew/warsztaty-podstawy-springa" );  
}
```



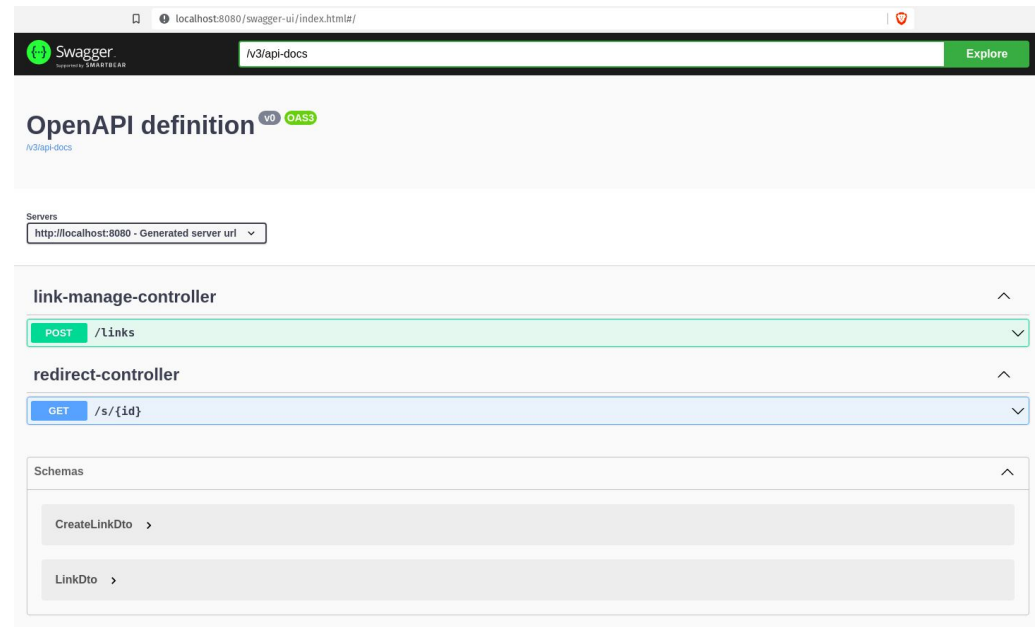
Dodawanie dokumentacji OpenAPI - Swagger UI

- Nie mamy frontendu
- Testowanie zapytań nie jest możliwe bezpośrednio z poziomu przeglądarki
- Testowanie API jest możliwe przez np:
 - curl - bezpośrednie zapytania z poziomu konsoli
 - Postman - zaawansowane narzędzie do testowania api
- Swagger UI - mały komponent który umożliwia udostępnienie dokumentacji oraz testowanie endpointów, bezpośrednio z poziomu aplikacji

Wystarczy dodać następującą zależność do **pom.xml**:

```
pom.xml

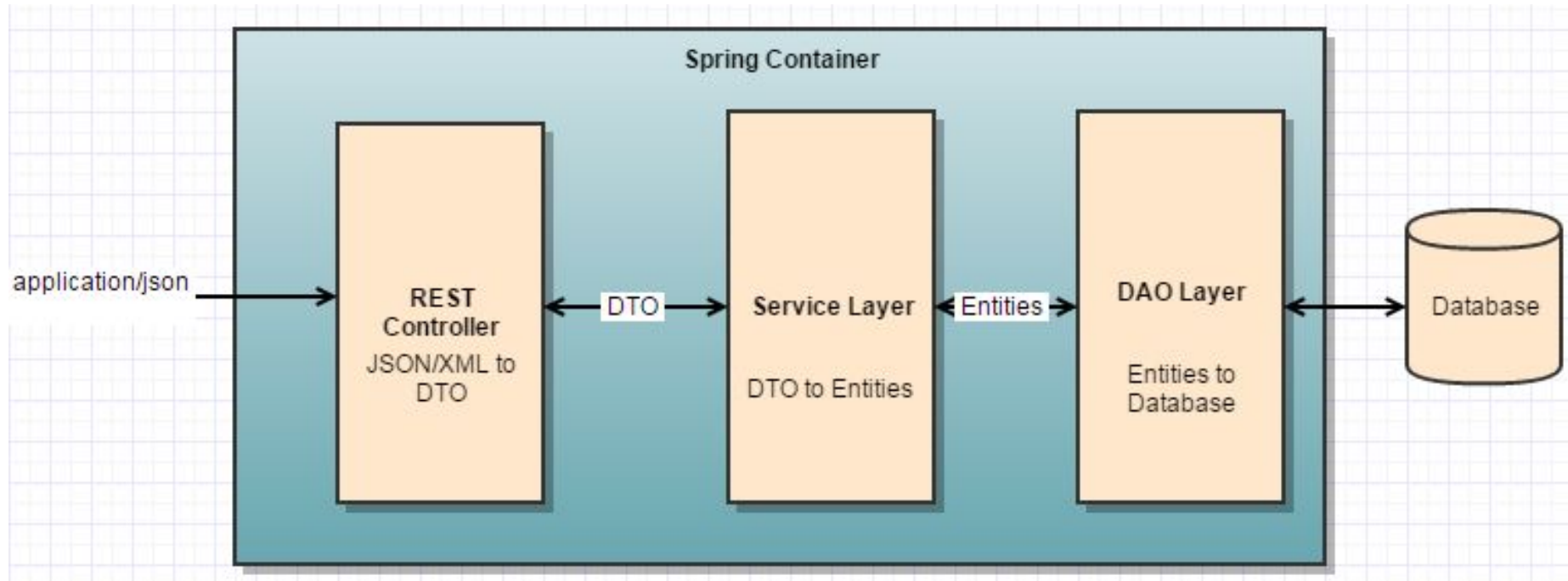
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-ui</artifactId>
  <version>1.6.6</version>
</dependency>
```



CreateLinkDto.java

```
record CreateLinkDto(
    @Schema(description = "Identifier/alias to link. Used for redirection.", example = "link-alias", required = true)
    String id,
```

CRUD



Spring Data - JPA - Hibernate

Technologia	Jak działa?	Plusy	Minusy
DAO (Data Access Object)	Wywoływanie bezpośrednio zapytań SQL do bazy danych	<ul style="list-style-type: none">- Duża kontrola i modyfikowalność	<ul style="list-style-type: none">- Ogromna ilość zbędnego powtarzalnego kodu- Łatwość w popełnianiu błędów- Trudna utrzymywalność
Hibernate (implementacja JPA)	ORM (Object Relational Mapping) - przekształcanie obiektów javy na relacyjne i odwrotnie.	<ul style="list-style-type: none">- Zarządza połączeniem z bazą danych.- Nie ma konieczności ręcznego konstruowania zapytań do bazy.- Mapuje wynik zapytania SQL na obiekt Javy i odwrotnie.	<ul style="list-style-type: none">- Mniejsza kontrola- Trudniejsze wdrożenie przy bardzo złożonych zadaniach
Spring Data (warstwa abstrakcji na JPA)	Spring Data opakowuje i redukuje ilość powtarzającego się kodu z JPA	<ul style="list-style-type: none">- Świetne do prostszych CRUD-ów- Można odpytywać bazę danych za pomocą "jednej linijki" kodu	<ul style="list-style-type: none">- Trudniejsze do zrozumienia ze względu na wysoki poziom abstrakcji- Dużo trudniejsze wdrożenie przy bardzo złożonych zadaniach

Encje

- Klasy odpowiadające tabelą w bazie danych
- Wymagania dotyczące encji w JPA:
 - Adnotacja **@Entity**
 - Klasy nie mogą być finalne
 - bezparametrowy konstruktor (public/protected)
 - Klasa najwyższego rzędu
 - Musi mieć kolumnę oznaczoną **@Id**, która jest odpowiednikiem **primary_key** w tabeli
- Spring poszukuje adnotacji **@Entity** na podobnej zasadzie jak **@Component**.

```
LinkEntity.java

import dev.greencashew.linkshortener.link.api.LinkDto ;
import lombok.*;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import java.time.LocalDate ;

@Entity
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
class LinkEntity {
    @Id
    @Column(name = "id", nullable = false, updatable = false)
    private String id;

    @Column(nullable = false)
    private String email;

    @Column(nullable = false)
    private String targetUrl;

    private LocalDate expirationDate;

    private int visits;
}
```

H2 in memory database



- Sql-owa baza danych
- Uruchamiana w pamięci (po każdym restarcie aplikacji jest przywracana do poprzedniego stanu)
- Świetna do celów deweloperskich
- Udostępnia webowego klienta pod adresem:

<http://localhost:8080/h2-console>

pom.xml

```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
```

application.properties

```
# H2 Console
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:test
spring.datasource.username=admin
spring.datasource.password=admin
```

Run | Run Selected | Auto complete | Clear | SQL statement:

SELECT * FROM LINK_ENTITY

ID	EMAIL	EXPIRATION_DATE	TARGET_URL	VISITS
link-alias	test@greencashew.dev	2054-06-23	https://github.com/greencashew/warsztaty-podstawy-springa	0
link-alias1	test@greencashew.dev	2054-06-23	https://github.com/greencashew/warsztaty-podstawy-springa	0
link-alias2	test@greencashew.dev	2054-06-23	https://github.com/greencashew/warsztaty-podstawy-springa	0
link-alias3	test@greencashew.dev	2054-06-23	https://github.com/greencashew/warsztaty-podstawy-springa	0
link-alias4	test@greencashew.dev	2054-06-23	https://github.com/greencashew/warsztaty-podstawy-springa	0
link-alias5	test@greencashew.dev	2054-06-23	https://github.com/greencashew/warsztaty-podstawy-springa	0

(6 rows, 3 ms)

Edit

Lombok

- Generator kodu:

- `@AllArgsConstructor` - Dodaje konstruktor wraz z wszystkimi polami zdefiniowanymi w klasie
- `@RequiredArgsConstructor` - Tworzy konstruktor wyłącznie z pól wymaganych (oznaczonych `final-em`)
- `@NoArgsConstructor` - Dodaje domyślny bezargumentowy konstruktor
- `@Getter` - Dodaje gettery (dla wszystkich pól w klasie lub dla konkretnego argumentu)
- `@Setter` - Dodaje settery (dla wszystkich pól w klasie lub dla konkretnego argumentu)
- `@ToString` - Dodaje implementacje interfejsu `toString()`
- `@Builder` - Generuje builder, szczególnie użyteczny w przypadku dużej ilości pól
- Pełna lista funkcjonalności:

<https://projectlombok.org/features/all>

```
pom.xml:dependencies
```

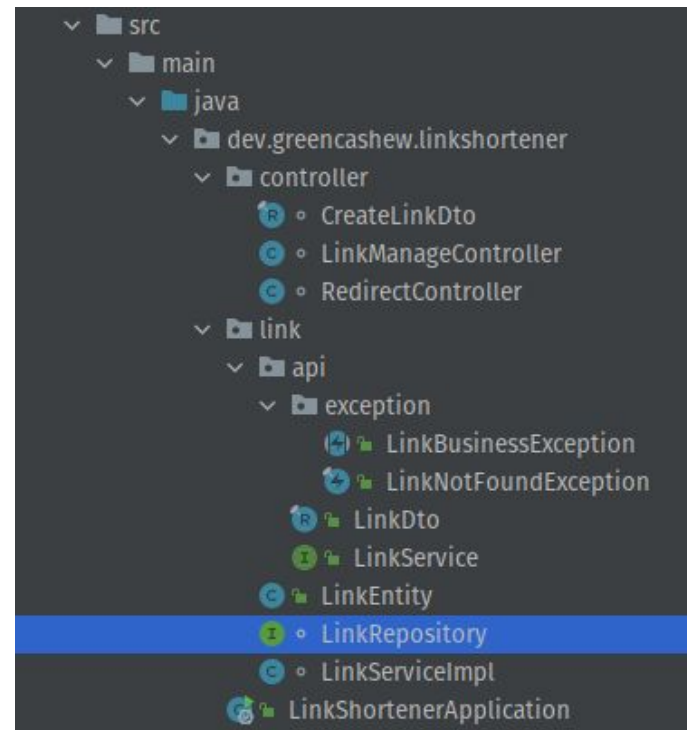
```
<dependency>
<groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
```

```
pom.xml
```

```
<build>
  <plugins>
    <plugin>
      <groupId> org.springframework.boot </groupId>
      <artifactId> spring-boot-maven-plugin </artifactId>
      <configuration>
        <excludes>
          <exclude>
            <groupId> org.projectlombok </groupId>
            <artifactId> lombok</artifactId>
          </exclude>
        </excludes>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Dodanie warstwy serwisu oraz bazy danych

- Utworzony nowy pakiet **link**
- **LinkDto** przeniesiony do **link.api**
- Dodano odnośnik do serwisów
- Zostały dodane nowe klasy:
 - **LinkServiceImpl** - znajduje się tam logika biznesowa aplikacji
 - **LinkEntity** - Klasa encji
 - **LinkRepository** - Repozytorium rozszerzające **CrudRepository** z projektu Spring Data
 - **LinkNotFoundException** - w przypadku nie znalezienia linku



LinkRepository - CrudRepository - Spring Data

- Spring Data dostarcza różne implementacje repozytoriów
 - **CrudRepository** - Dostarcza podstawowe metody takie jak zapis, odczyt, liczenie.
 - **PagingAndSortingRepository** - Rozszerza CrudRepository o paginację, sortowanie stron
 - **JpaRepository** - Rozszerza **PagingAndSortingRepository** o możliwości dotyczące Jpa
- Spring automatycznie wygeneruje implementację dla interfejsu repozytorium
- Spring data również oferuje dodawanie własnych metod w interfejsie, z których na podstawie konwencji zostanie wygenerowana implementacja
 - **Temat** - cel metody
 - find, delete, count
 - **Predykat** - umieszczany po słowie **by**
- `List<LinkEntity> findAllByVisitsGreaterThan(int minimumVisits);`

LinkRepository.java

```
interface LinkRepository extends CrudRepository<LinkEntity, String> {  
}
```

LinkService

- Domyślnie czas transakcji równy czasowi kontekstu z nią związanego
- Repozytorium nie posiada metody update()
- Aktualizacja musi być zrobiona z poziomu service layer
- **@Transactional**
 - Pozwala zachować wiązanie encji z operacjami na bazie

LinkServiceImpl.java

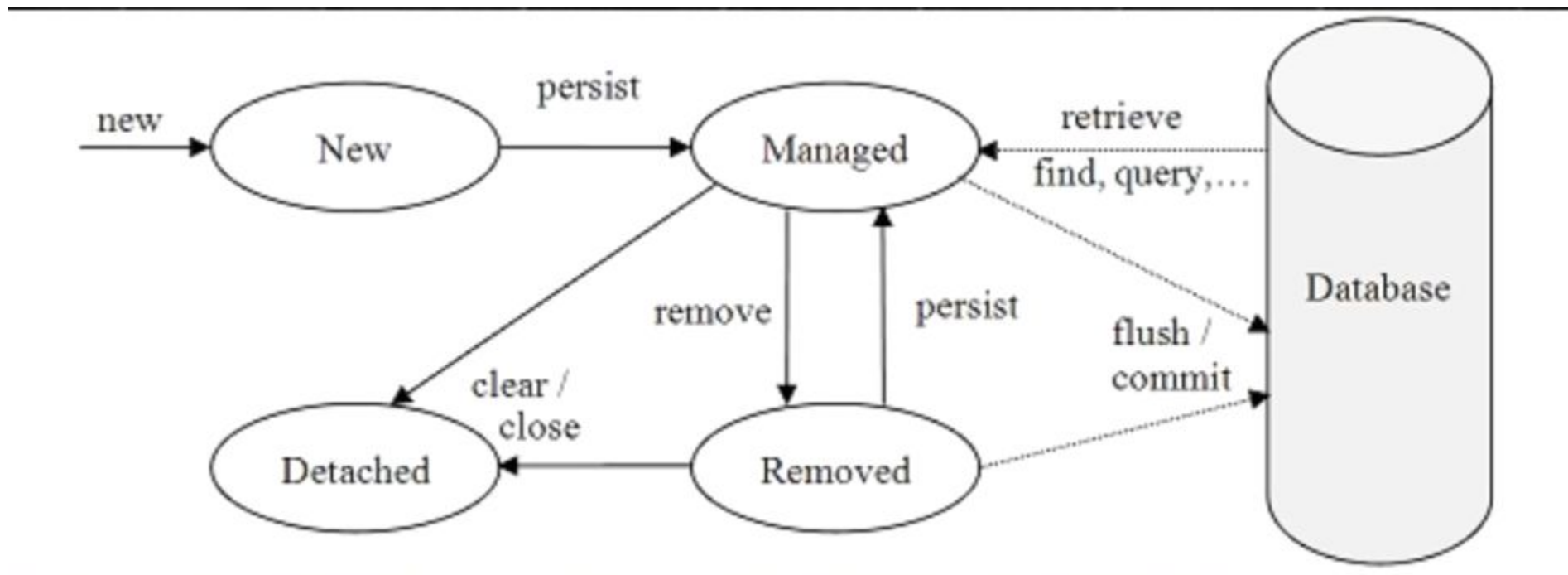
```
@Service
@AllArgsConstructor
class LinkServiceImpl implements LinkService {

    private final LinkRepository repository;

    @Override
    @Transactional
    public LinkDto createLink(final LinkDto createLink) {
        final LinkEntity linkEntity = LinkEntity.fromDto(createLink);
        repository.save(linkEntity);
        return createLink;
    }

    @Override
    @Transactional
    public LinkDto gatherLinkAndIncrementVisits(final String id) {
        final LinkEntity linkEntity = repository.findById(id)
            .orElseThrow(() -> new LinkNotFoundException(id));
        linkEntity.setVisits(linkEntity.getVisits() + 1);
        return linkEntity.toDto();
    }
}
```

Cykl życia encji



Implementacja kolejnych zasobów

