Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

Informatyka, Aplikacje internetowe i mobilne, semestr 5

Programowanie aplikacji webowych

Laboratorium nr 8

Tworzenie REST API z wykorzystaniem frameworka Spring.

Celem zajęć jest stworzenie aplikacji backendowej umożliwiającej zarządzanie bazą kotów i pobieranie ciekawostek o nich, czerpanych z zewnętrznego serwisu.

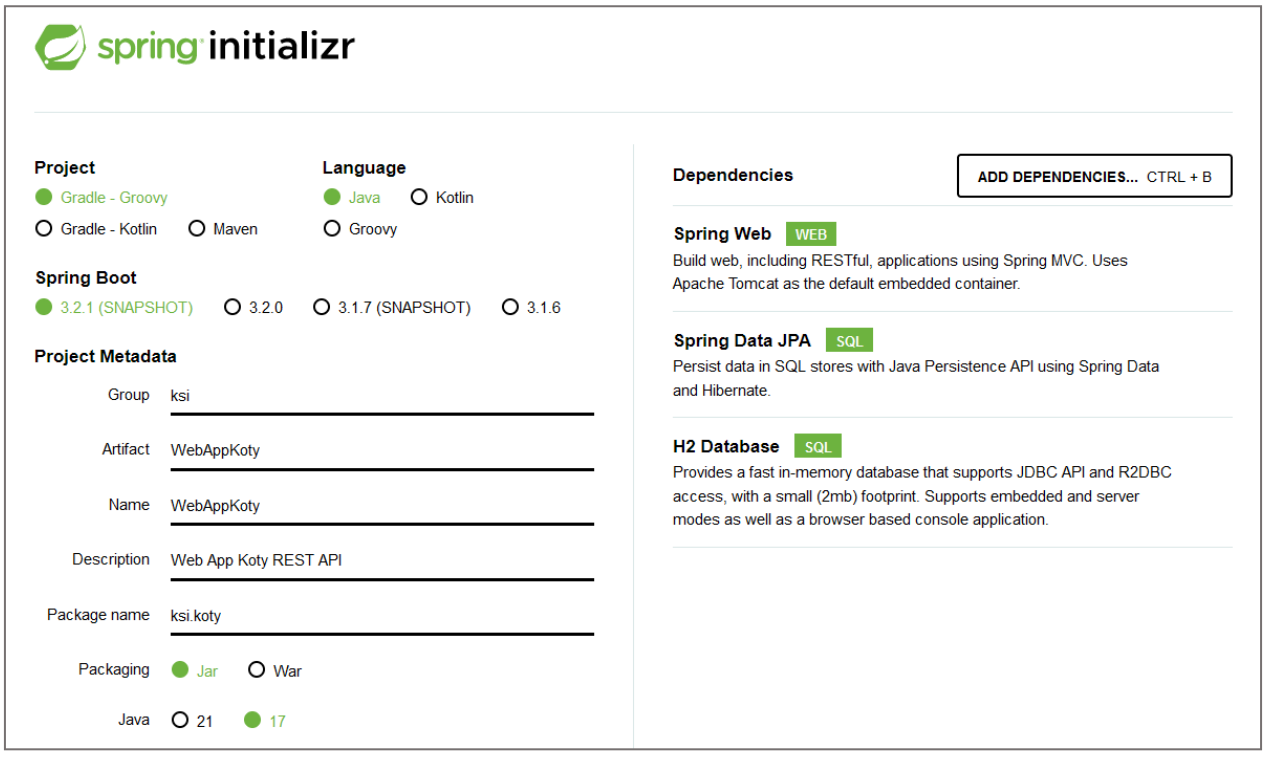
W tym projekcie wykorzystamy:

- webową wersję Spring Initializr (zamiast pluginu w Eclipse),
- system baz danych H2,

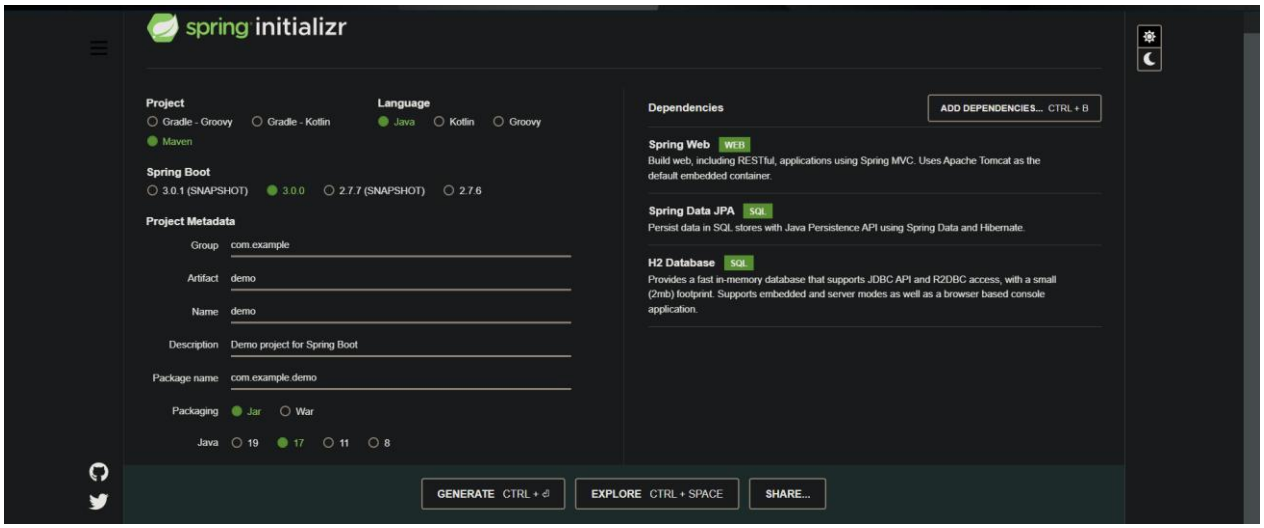
System bazodanowy **H2** został napisany w Javie i może działać jako oprogramowanie klient serwer lub być osadzony w aplikacji Java. W tym projekcie skorzystamy z osadzonej bazy danych.

1. Web Spring Initializr i importowanie projektu

Nowe projekty Spring Boot można utworzyć również w Webowej wersji Spring Initializr. Skonfiguruj nowy projekt o nazwie WebAppKoty korzystając z https://start.spring.io/. Dodaj zależności Spring Web, Spring Data JPA H2 Database (baza danych będzie tu osadzona w aplikacji), kliknij Generate, zapisz plik .zip, a następnie rozpakuj projekt w swoim katalogu Eclipse Workspace. Przejdź do Eclipse i zaimportuj utworzony projekt jako projekt Gradle.



Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni



2. Model

Na początku utworzymy model reprezentujący kota poprzez jego id, nazwę i rasę. Wykorzystamy *@Entity*, czyli znacznik informujący o reprezentacji tabeli w bazie danych oraz *@Id i @GeneratedValue*, czyli znaczniki informujące Springa, aby tworzyć automatycznie generowane id.

```
@Entity class Cat {
  private @Id @GeneratedValue Long id;
  private String name;
  private String breed;
```

Przydadzą się także konstruktory:

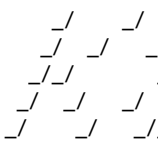
```
  Cat() {}

  Cat(String name, String breed) {
    this.name = name;
    this.breed = breed;
  }
```

Zmienne mają modyfikatory private więc potrzebne będą również settery (metody ustawiające wartości zmiennych) i gettery (metody pobierające wartości zmiennych):

```
  public Long getId() {
    return this.id;
  }

  public String getName() {
```


Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

```
    return this.name;
  }

  public String getBreed() {
    return this.breed;
  }

  public void setId(Long id) {
    this.id = id;
  }

  public void setName(String name) {
    this.name = name;
  }

  public void setBreed(String breed) {
    this.breed = breed;
  }
```

Dodamy jeszcze metody: equals, hashCode i toString (Source | Generate Delegate Methods…). Każdą z adnotacją @Override.

Metoda do porównywania obiektów equals powinna mieć kod:

```
  @Override
  public boolean equals(Object object) {
    if (this == object)
      return true;
    if (!(object instanceof Cat))
      return false;
    Cat cat = (Cat) object;
    return Objects.equals(this.id, cat.id) && Objects.equals(this.name, cat.name)
        && Objects.equals(this.breed, cat.breed);
  }
```

Kod metody do pobierania hash code, czyli unikatowego identyfikatora obiektu to:

```
  @Override
  public int hashCode() {
    return Objects.hash(this.id, this.name, this.breed);
  }
```

Kod metody toString generującej tekstową informację o obiekcie to:

```
  @Override
  public String toString() {
    return "Cat{" + "id=" + this.id + ", name='" + this.name + '\'' + ", breed='" +
this.breed + '\'' + '}';
```

```
  }
}
```

3. Repository

W celu przeprowadzenia operacji na zmapowanych do obiektów relacjach kotów potrzebny będzie nowy interfejs, rozszerzający JpaRepository, dla nowej klasy Cat:

```
@Repository
interface CatRepository extends JpaRepository<Cat, Long> {
}
```

Dostarcza on podstawowych funkcji CRUD, można w nim również zaimplementować inne, bardziej rozbudowane metody.

4. Zasiewanie bazy danych

Dzięki zastosowaniu osadzonej bazy danych H2, można skupić się na programowaniu, bez konieczności manualnego korzystania z zewnętrznego systemu baz danych np. PostgreSQL, a także bez konfiguracji połączenia, co może być przydatne np. w fazie testowania lub budowania prototypu.

Utworzymy plik dodający pierwsze koty do bazy H2, w tym celu utworzymy klasę LoadDatabase z adnotacją `@Configuration`. Adnotacja ta informuje o tym, że klasa zawiera metody do tworzenia obiektów typu bean. Dzięki dodaniu adnotacji `@Bean` przed metodą `initDatabase`, która zwraca w wyniku obiekt klasy `CommandLineRunner`, nie musimy sami tworzyć tego obiektu, zajmie się tym Spring.

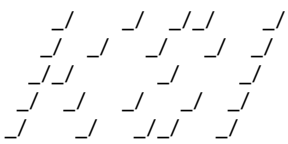
W metodzie wykorzystamy log z pakietu `org.slf4j` do prezentowania informacji, a na końcu zapiszemy nowe koty do bazy oraz wyświetlimy o nich informacje w logu.

```
@Configuration
class LoadDatabase {

  private static final Logger log = LoggerFactory.getLogger(LoadDatabase.class);

  @Bean
  CommandLineRunner initDatabase(CatRepository repository) {
    return args -> {
      log.info("Preloading " + repository.save(new Cat("Felix", "Mieszaniec")));
      log.info("Preloading " + repository.save(new Cat("Filemon", "Maine Coon")));
    };
  }
}
```

Ruchom aplikację i sprawdź informacje o kotach w logu.


Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

5. Kontroler

Ostatnim krokiem do uruchomienia kociej aplikacji jest stworzenie kontrolera.

Stosując znacznik `@RestController` informujemy Springa, że dana klasa powinna być interpretowana jako kontroler RESTowy, a używając mapowania, np. `@GetMapping("/cats")`, deklarujemy ścieżki oraz metody żądań. Kod powinien mieć postać:

```
@RestController
class CatController {

  private final CatRepository repository;

  CatController(CatRepository repository) {
    this.repository = repository;
  }
```

Potrzebnych będzie jeszcze kilka metod.

Metoda zwracająca wszystkie koty:

```
  @GetMapping("/cats")
  List<Cat> getCats() {
    return repository.findAll();
  }
```

Metoda zwracająca konkretnego kota:

```
  @GetMapping("/cats/{id}")
  Optional<Cat> getCat(@PathVariable("id") Long id) {
    return repository.findById(id);
  }
```

Metoda wysyłająca zapytanie do serwisu catfact.ninja i zwracająca z niej ciekawostkę, wykorzystująca obiekt klasy `RestTemplate`, umożliwiający wysyłanie żądań HTTP.

```
  @GetMapping("/fun-fact")
  String getFunFact() {
  final String uri = "https://catfact.ninja/fact";
  RestTemplate restTemplate = new RestTemplate();
  String result = restTemplate.getForObject(uri, String.class);
  return result;
  }
```

Metoda odbierająca i zapisująca w bazie nowego kota:

```
  @PostMapping("/cats")
  Cat newCat(@RequestBody Cat newCat) {
    return repository.save(newCat);
  }
```

Metoda aktualizująca istniejącego kota:

```
  @PutMapping("/cats/{id}")
  Cat replaceCat(@RequestBody Cat newCat, @PathVariable("id") Long id) {
    return repository.findById(id)
      .map(cat -> {
        cat.setName(newCat.getName());
        cat.setBreed(newCat.getBreed());
        return repository.save(cat);
      })
      .orElseGet(() -> {
        newCat.setId(id);
        return repository.save(newCat);
      });
  }
```

I ostatnia metoda, usuwająca kota:

```
  @DeleteMapping("/cats/{id}")
  void deleteCat(@PathVariable("id") Long id) {
    repository.deleteById(id);
  }
}
```

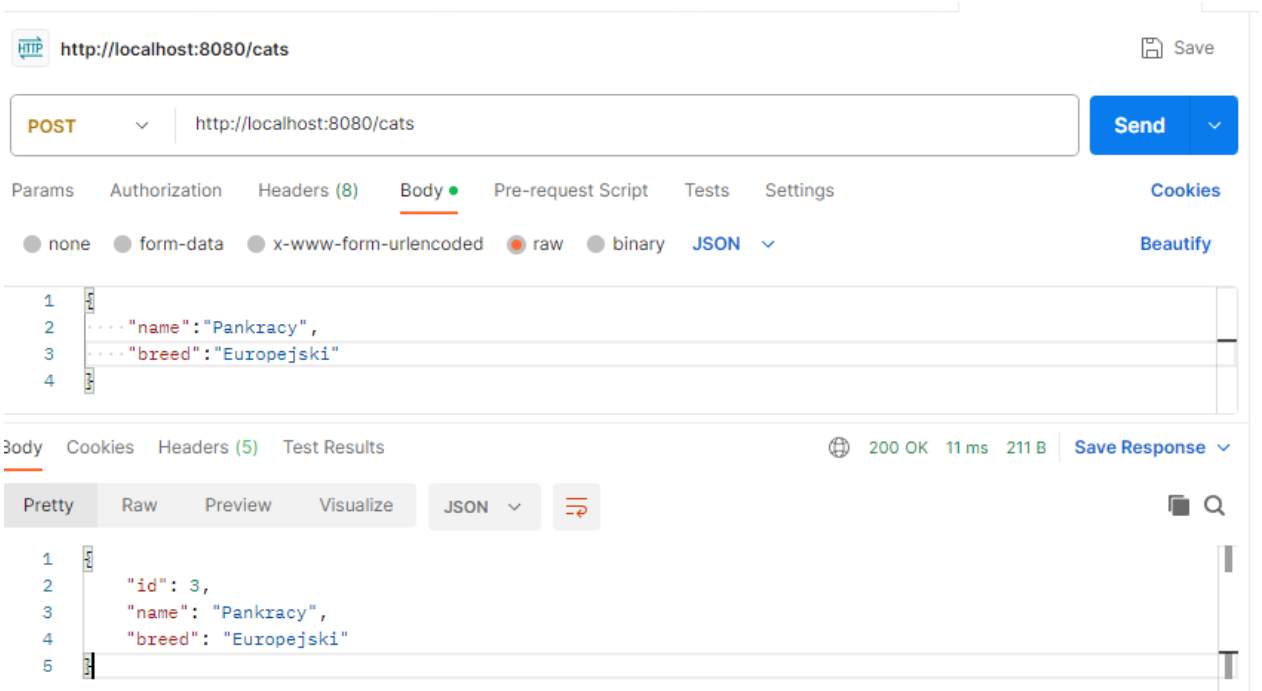
6. Testowanie aplikacji z użyciem Postmana

Dzięki zależności Spring Web, nasza aplikacja ma już wbudowany serwer Tomcat. Uruchom aplikację.

Działanie metody GET można przetestować w przeglądarce np: http://localhost:8080/cats/2. Do przetestowania pozostałych metod wykorzystaj narzędzie Postman.

Sprawdź działanie metody `newCat` dodając kota Pankracego, jak na obrazku poniżej:

Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni



Wykonaj zrzuty ekranu do sprawozdania sprawdzając każdą metodę kontrolera na własnym przykładzie.

7. Modyfikacje

Oprogramuj 3 wybrane błędy, które mogą pojawić się w odpowiedziach HTTP aplikacji korzystając z klasy ResponseEntity. Informacje na ten temat można znaleźć m.in. pod adresem:

https://spring.io/guides/tutorials/rest/
https://www.baeldung.com/spring-response-entity

Dodaj do pliku .pdf zrzuty ekranu z Postmana ilustrujące wykonane modyfikacje.

8. Frontend

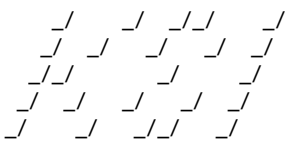
Utwórz prostą stronę HTML + JS, z przyciskiem i polem tekstowym pobierającą informacje o kocie.

Podpowiedź:

```
fetch('http://localhost:8080/cats/2')
  .then((response) => response.json())
  .then((data) => console.log(data));
```

Sprawozdanie

W sprawozdaniu w systemie Sprawer wyślij plik .pdf ze zrzutami ekranu z Postmana (i przeglądarki) oraz spakowane pliki aplikacji.



Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni