Northwind

Użyte technologie:
- Java 1.8
- Spring Boot 2.2.0
- Bazy danych MySQL
- Gatling
- Hibernate
- srodowisko IntelliJ
- JDK

1. Tworzenie szablonu projektu:
- Wejdź na stronę "https://start.spring.io"
- uzupełnij formularz:
Project - Maven
Language - Java
Spring Boot - 2.2.0
Project Metadate - pl.agh
Artifact - db2
options -> name - Northwind
options -> descriptions - Data base Northwind
options -> packaging - Jar
options -> Java - 8
options -> search Dependencies to add:
- spring-boot-starter-web
- spring-boot-starter-data-jpa
- spring-boot-starter-jdbc
- spring-boot-starter-validation
- mysql-connector-java
- po zakończeniu wypełniania formularza kreatora projektu naciśnij "Generate", aby wygenerować projekt.
- projekt został pobrany do katalogu pobrane na dysku lokalnym w formacie .zip
- należy rozpakować archwium w dowolnym miejscu na dysku (np: c:/projekt/..)

2. Środowisko IntelliJ - import projektu
- otwórz środowisko IntelliJ
- zaimportu projekt
- wybierz ustawiania odpowiednie dla Twojego projektu

3. Dodawanie dependencies:
- w przeglądarce internetowej wpisz adres "https://mvnrepository.com"
- w polu wyszukiwania wpisz "Lombok"
- Z listy wybierz "Project Lombok"
- Z listy wersji wybierz dowolną wersje i skopiuj odpowiednią dependencje

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.2</version>
  <scope>provided</scope>
</dependency>
```

- Należy usunąć wersje i wkleić do pliku pom.xml między znacznikami
<dependencies></dependencies>

```
...
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <scope>provided</scope>
</dependency>
...
```

4. Podstawowa konfiguracja projektu
Dla potrzeb naszego projektu należy skonfigurować nasz projekt w pliku
"application.properties" należy dodać:

```
#DARABASE CONFIGURATION
spring.jpa.hibernate.ddl-auto=create
spring.jpa.database=mysql
spring.datasource.username=root
spring.datasource.password=password
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/northwind?
useSSL=false&createDatabaseIfNotExist=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL57InnoDBDialect
```

- W pliku tym zostało skonfigurowane połączenie z bazą danych MySQL,
- dane dostępowe do bazy username oraz password
- określenie lokalizacji bazy danych
- ustawineie połączenia z bazy danych

5. Konfiguracja Swagger
- Tworzymy podkatalog "Config" w katalogu głównym
- w stworzonym podkatalogu tworzymy klasę konfiguracyjną CoreConfig.java"
- w stworzonej klasie pnad klasą dodajemy adnotacje @EnableScheduling,
@EnableSwagger2, @Configuration
- adnotacja ustawiają klasę na klasę konfiguracyjną Swagger
- W klasie tworzymy metody oznaczone adnotacją @Bean
- klasa ta konfiguruje Swaggera, który jest odpowiedzialny za tworzenie dokumentacji do
zapytań restowych

```
 @Bean
 public RestTemplate restTemplate() {
   return new RestTemplate();
 }

 @Bean
 public Docket api() {
   return new Docket(DocumentationType.SWAGGER_2).select()
       .apis(RequestHandlerSelectors.basePackage("pl.agh.db2.northwind"))
       .paths(PathSelectors.any())
       .build();
 }
```

```
  @Override
  public void addResourceHandlers(ResourceHandlerRegistry registry) {

    registry.addResourceHandler("/lib/**").addResourceLocations("/lib/").setCachePeriod(0);

registry.addResourceHandler("/images/**").addResourceLocations("/images/").setCacheP
eriod(0);

registry.addResourceHandler("/css/**").addResourceLocations("/css/").setCachePeriod(0);
    registry.addResourceHandler("swagger-
ui.html").addResourceLocations("classpath:/META-INF/resources/");
    registry.addResourceHandler("/webjars/**").addResourceLocations("classpath:/META-
INF/resources/webjars/");
  }
```

6. Mapowanie struktury bazy danych na hibernate
- W głównym katalogu tworzymy nowy katalog o nazwie model, który będzie przechowywał klasy modelu
- Stwórz klasę o nazwie "Category.java"
- dodaj do niej adnotacje wpisując je ponad nazwą klasy

```
@AllArgsConstructor
@NoArgsConstructor
@Entity(name = "CATEGORIES")
@Getter
@Setter
@ToString
@EqualsAndHashCode
@Table(name = "CATEGORIES")
public class Category {
```

Powyższe adnotacje dodają konstruktory do wszystkich pół, dodają metody get oraz set, które są niezbędne do działania na polach w bazie danych. Nadawana jest nazwa tabeli (Jeżeli jest inna niż nazwa klasy)

7. Tworzenie modelu "Category"

Należy stworzyć model dla wszystkich tabel w bazie "Northwind"
- np: tworzenie modelu dla tabeli "Category"

```
  Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @Column(name = "categoryId", unique = true)
  private int categoryId;

  @Column(name = "category_name")
  private String categoryName;

  @Column(name = "description")
  private String description;
```

```
  @Column(name = "picture")
  private String picture;
```

@Id /*Pole oznaczone tą adnotacją jest identyfikatorem danej tabeli*/
@ GeneratedValue /*Oznacza, że wartość tego pola jest generowana automatycznie*/
@Column /*Nazwa poszczególnych pól (można podać w nawiasach nazwę kolumny w tabeli jeżeli jest inna niż nazwa pola w klasie */

- czynności jak w przykładzie należy powtórzyć dla wszystkich tabel w bazie Northwind

8. Tworzenie repository
- W głównym katalogu tworzymy katalog repository, który będzie przechowywał repozytoria wszystkich klas modelu
- w katalogu "repository" należy stworzyć interfejs categoryDeo.java
- należy określić jakie metody mają być w każdym repozytorium
- Interfejs powinien rozszerzać interfejs "Transactional
@Repository
public interface CategoryDao extends JpaRepository<>"
- ponad nazwą interfejsu należy doać adonotacje

Transactional
@Repository
public interface CategoryDao extends JpaRepository<Category,Integer>

Dzięki rozszerzeniu interfejsu można wykonywać podstawowe operacje na obiektach Category (CRUD) takie jak: tworzenie, odczyt, znajdowanie, aktualizowanie, usuwanie... Adnotacje nad interfejsem zaznaczają, że podany interfejs jest repozytorium oraz jest transakcyjny.

9. Tworzenie mapper'ów

- W katalogu głównym stwórz podkatalog mapper
- w podkatalogu mapper dla wszystkich tabeli stwórz klasy mapper jak w przykładzie
- ponad klasą umieść adnotacje @Component
- Stwórz metodę mapperCategory przyjmującą jako parametr CategoryDeo i zwracającą Category (Metoda tworzy nowy obiekt Category i zwraca go)
- Kolejną metodą jaką należy stworzyć jest mapToCategoryDto przyjmującą jako parametr Category i zwracającą CategoryDeo. Metoda tworzy nowy CategoryDeo i zwraca go.
- Następną metodą jaką należy stworzyć jest mapToCategoryDtoList przyjmującą jako parametr listę Category i zwracającą listę Category.
Klasa ta jest niesbędna do mapowania Category do formatu JSON. Powyższe czynności nalzey wykonać dla wszystkich tabel w bazie

```
@Component
public class CategoryMapper {

  public Category mapToCategory(final CategoryDto categoryDto) {
    return new Category(
        categoryDto.getCategoryId(),
        categoryDto.getCategoryName(),
        categoryDto.getDescription(),
        categoryDto.getPicture()
```

```
    );
  }

  public CategoryDto mapToCategoryDto(final Category category) {
    return new CategoryDto(
        category.getCategoryId(),
        category.getCategoryName(),
        category.getDescription(),
        category.getPicture()
    );
  }

  public List<CategoryDto> mapToCategoryDtoList(final List<Category> categoryList) {
    return categoryList.stream()
        .map(t -> new CategoryDto(t.getCategoryId(), t.getCategoryName(),
t.getDescription(), t.getPicture()))
        .collect(Collectors.toList());
  }
}
```

10. Tworzenie services
- W głównym katalogu projektu należy stworzyć podkatalog services
- w stworzonym podkatalogu tworzymy services dla wszystkich tabel np.
- Tworzy klasę CategoryService
- w stworzonej klasie wstrzykujemy CategoryDao za pomocą adnotacji @Autowired
- tworzymy kolejno metody zapisu i wyszukiwania do obsługi danej tabeli
- powyższe czynności wykonujemy dla wszystkich klas

```
public class CategoryService {

  @Autowired
  private CategoryDao categoryDao;

  public Category save(Category c) {

    categoryDao.save(c);

    return c;
  }

  public void delete(Category c){

    categoryDao.delete(c);
  }

}
```

11. Tworzenie klas kontrolerów
- W katalogu głównym należy utworzyć podkatalog "controllers"
- w podkatalogu controllers należy utworzyć klasę CategoryController.class
- należy dodać adnotacje określające Spring Boot'owi, że jest to klasa kontrolera

- W stworzonej klasie wstrzykujemy DbService, OrderDetailsMapper, OrderMapper, CustomerMapper, ShipperMapper oznaczając adnotacjami @Autowired
- Tworzymy metodę kontrolera i oznaczamy ją adnotacją @RequestMapping
- Metoda tworzy zamówienie, następnie dodaje do niego szczegóły

```
@CrossOrigin(origins = "*")
@RestController
@RequestMapping("/v1")
public class CartController {

  @Autowired
  private DbService service;

  @Autowired
  private OrderMapper orderMapper;

  @Autowired
  private OrderDetailsMapper orderDetailsMapper;

  @Autowired
  private CustomerMapper customerMapper;

  @Autowired
  private ShipperMapper shipperMapper;

  @RequestMapping(method = RequestMethod.POST, value = "/cart", consumes =
APPLICATION_JSON_VALUE)
  public void addToCart(@RequestParam Integer customerId, @RequestParam Integer
employeeId, @RequestParam Integer shipperId,
          @RequestBody ArrayList<OrderDetailsHelper> detailsDto) {
    CustomerDto customer =
customerMapper.mapToCustomerDto(service.getCustomerDao().getOne(customerId));
    ShipperDto shipper =
shipperMapper.mapToShipperDto(service.getShipperDao().getOne(shipperId));
    Order order = service.getOrderDao().save(orderMapper.mapToOrder(new OrderDto(0 ,
       service.getCustomerDao().getOne(customerId),
       service.getEmployeeDao().getOne(employeeId),
       new Date(),
       new Date(),
       new Date(),
       service.getShipperDao().getOne(shipperId),
       shipper.getCompanyName(),
       customer.getCompanyName(),
       customer.getAddress(),
       customer.getCity(),
       customer.getRegion(),
       customer.getPostalCode(),
       customer.getCountry())));

    detailsDto.stream().map(t ->
       new OrderDetailsDto(
          order.getOrderID(),
```

```
        t.getProductId(),

service.getProductDao().getOne(t.getProductId()).getUnitPrice(),t.getQuantity(),t.getDisco
unt())
    ).forEach(x ->
service.getDetailsDao().save(orderDetailsMapper.mapToOrderDetails(x)));

  }
}
```

- tworzymy kontroler Category controller
- dodajemy adnotacje @CrosOrigin, @RestController, @RequestMapping
- wstrzykujemy CategoryMapper, CategoryMapper i oznaczamy adnotacją @Autowired
- tworzymy metody CategoryDto, getGategories, deleteCategory, updateCategory, createCategory i oznaczamy je adnotacją @RequestMapping
- tworzymy metodę zwracającą kategorię po podaniu id w adresie z adnotacją @RequestParam
- tworzymy metodę zwracającą liste kategori
- tworzymy metodę usuwającą kategorie
- tworzymy metodę aktualizującą kategorie
- tworzymy metodę tworzącą kategorię

```
@CrossOrigin(origins = "*")
@RestController
@RequestMapping("/v1")
public class CategoryController {

  @Autowired
  private CategoryMapper dbService;

  @Autowired
  private CategoryMapper categoryMapper;

  @RequestMapping(method = RequestMethod.GET, value = "/category/{id}")
  public CategoryDto getCategory(@RequestParam Integer id) {
    return categoryMapper.mapToCategoryDto(dbService.getCategoryDao().getOne(id));
  }

  @RequestMapping(method = RequestMethod.GET, value = "/category")
  public List<CategoryDto> getGategories() {
<<<<<<< HEAD
    return
categoryMapper.mapToCategoryDtoList().ToCategoryDyoList(dbService.getCategoryDao()
.findAll());
=======
    return categoryMapper.mapToCategoryDtoList(dbService.getCategoryDao().findAll());
>>>>>>> 02f6d5b130a6bbab0d7058e4d09068891493dceb
  }

  @RequestMapping(method = RequestMethod.DELETE, value = "/category/{id}")
  public void deleteCategory(@RequestParam Integer id) {
```

```
    dbService.getCategoryDao().deleteById(id);
  }

  @RequestMapping(method = RequestMethod.PUT, value = "/category")
  public CategoryDto updateCategory(@RequestBody CategoryDto categoryDto) {
    return
categoryMapper.mapToCategoryDto(dbService.getCategoryDao().save(categoryMapper.m
apToCategory(categoryDto)));
  }

  @RequestMapping(method = RequestMethod.POST, value = "/category", consumes =
APPLICATION_JSON_VALUE)
  public void createCategory(@RequestBody CategoryDto categoryDto) {
    dbService.getCategoryDao().save(categoryMapper.mapToCategory(categoryDto));
  }
}
```

- tworzymy kontroler CustomerController.java
- Stworzoną klasę oznaczamy adnotacjami @CrosOrigin, @RestController, @RequestMapping
- wstrzykujemy DbService, CustomerMapper i oznaczamy je adnotacją @Autowired
- tworzymy metodę getCustomers zwracającą listę klientów i oznaczamy ją adnotacją @RequestMapping
- tworzymy metodę getCustomer zwracającą customerDeo przyjmującą jako parametr @RequestParam i oznaczamy ją adnotacją @RequestMapping
- tworzymy metodę usuwającą deleteCategory która nic nie zwraca, przyjmuje jako parametr @RequestParam i oznaczamy ją adnotacją @RequestMapping
- tworzymy metodę aktualizującą updateCustomer przyjmującą jako parametr @RequestBody CustomerDeo i zwracającą CustomerDeo
- tworzymy metodę createCustomer która jako parametr przyjmuje @RequestBody, metoda tworzy nowego Klienta i zwraca obiekt Customer

```
@CrossOrigin(origins = "*")
@RestController
@RequestMapping("/v1")
public class CustomerController {

  @Autowired
  private DbService service;

  @Autowired
  private CustomerMapper customerMapper;

  @RequestMapping(method = RequestMethod.GET, value = "/customer")
  List<CustomerDto> getCustomers() {
    return customerMapper.mapToCustomerDtoList(service.getCustomerDao().findAll());
  }


  @RequestMapping(method = RequestMethod.GET, value = "/customer/{id}")
  public CustomerDto getCustomer(@RequestParam  Integer id){
    return customerMapper.mapToCustomerDto(service.getCustomerDao().getOne(id));
```

```
  }

  @RequestMapping(method = RequestMethod.DELETE, value = "/customer/{id}")
  public void deleteCategory(@RequestParam Integer id) {
    service.getCustomerDao().deleteById(id);
  }

  @RequestMapping(method = RequestMethod.PUT, value = "/customer")
  private CustomerDto updateCustomer(@RequestBody CustomerDto customerDto) {
    return
customerMapper.mapToCustomerDto(service.getCustomerDao().save(customerMapper.m
apToCustomer(customerDto)));
  }

  @RequestMapping(method = RequestMethod.POST, value = "/customer", consumes =
APPLICATION_JSON_VALUE)
  public void createCustomer(@RequestBody CustomerDto customerDto) {
    service.getCustomerDao().save(customerMapper.mapToCustomer(customerDto));
  }
}
```

- tworzymy kontroler EmployeeController.java
- Stworzoną klasę oznaczamy adnotacjami @CrosOrigin, @RestController, @RequestMapping
- wstrzykujemy DbService, EmployeeMapper i oznaczamy je adnotacją @Autowired
- tworzymy metodę getEmployee zwracającą listę pracowników Employee i oznaczamy ją adnotacją @ RequestMapping
- tworzymy metodę usuwającą deleteEmployee która nic nie zwraca, przyjmuje jako parametr @RequestParam i oznaczamy ją adnotacją @RequestMapping
- tworzymy metodę aktualizującą updateEmployee przyjmującą jako parametr @RequestBody EmployeeDeo i zwracającą EmployeeDeo
- tworzymy metodę createEmployee która jako parametr przyjmuje @RequestBody, metoda tworzy nowego pracownika i zwraca obiekt Employee

```
@CrossOrigin(origins = "*")
@RestController
@RequestMapping("/v1")
public class EmployeeController {

  @Autowired
  private DbService service;

  @Autowired
  private EmployeeMapper employeeMapper;

  @RequestMapping(method = RequestMethod.GET, value = "/employee")
  public List<EmployeeDto> getEmployees() {
    return employeeMapper.mapToEmployeeDtoList(service.getEmployeeDao().findAll());
  }
```

```
  @RequestMapping(method = RequestMethod.GET, value = "/employee/{id}")
  public EmployeeDto getEmployee(@RequestParam Integer id) {
    return employeeMapper.mapToEmployeeDto(service.getEmployeeDao().getOne(id));
  }

  @RequestMapping(method = RequestMethod.DELETE, value = "/employee/{id}")
  public void deleteEmployee(@RequestParam Integer id) {
    service.getEmployeeDao().deleteById(id);
  }

  @RequestMapping(method = RequestMethod.PUT, value = "/employee")
  public EmployeeDto updateEmployee(@RequestBody EmployeeDto employeeDto) {
    return
employeeMapper.mapToEmployeeDto(service.getEmployeeDao().save(employeeMapper.
mapToEmployee(employeeDto)));
  }

  @RequestMapping(method = RequestMethod.POST, value = "/employee", consumes =
APPLICATION_JSON_VALUE)
  public void createEmployee(@RequestBody EmployeeDto employeeDto) {
    service.getEmployeeDao().save(employeeMapper.mapToEmployee(employeeDto));
  }
}
```

- tworzymy kontroller zamówień OrderController.java
- Stworzoną klasę oznaczamy adnotacjami @CrosOrigin, @RestController, @RequestMapping
- wstrzykujemy DbService, OrderMapper i oznaczamy je adnotacją @Autowired
- tworzymy metodę getOrder przyjmującą jako parametr @RequestParam integer id i zwracającą zamówienie Order oznaczaną adnotacją @RequestMapping
- tworzymy metodę getOrders zwracającą listę produktów oznaczoną adnotacją @RequestMapping
- tworzymy metodę usuwającą zamówienie Order przyjmującą jako parametr @RequestParam oznaczoną adnotacją @RequestMapping
- tworzymy metodę updateOrder przyjmującą jako parametr @RequestBody OrderDeo aktualizującą OrderDeo i zwracającą OrderDeo
- tworzymy metodę createOrder tworzącą Order metoda przyjmuje @RequestBody OrderDeo i inic nie zwraca

```
@CrossOrigin(origins = "*")
@RestController
@RequestMapping("/v1")
public class OrderController {

  @Autowired
  private DbService service;

  @Autowired
  private OrderMapper orderMapper;

  @RequestMapping(method = RequestMethod.GET, value = "/order/{id}")
```

```
  public OrderDto getOrder(@PathVariable Integer id){
    return orderMapper.mapToOrderDto(service.getOrderDao().getOne(id));

  }

  @RequestMapping(method = RequestMethod.GET, value = "/order")
  public List<OrderDto> getOrders() {
    return orderMapper.mapToOrderDtoList(service.getOrderDao().findAll());
  }

  @RequestMapping(method = RequestMethod.DELETE, value = "/order/{id}")
  public void deleteOrder(@RequestParam Integer id) {
    service.getOrderDao().deleteById(id);
  }

  @RequestMapping(method = RequestMethod.PUT, value = "/order")
  public OrderDto updateOrder(@RequestBody OrderDto orderDto) {
    return
orderMapper.mapToOrderDto(service.getOrderDao().save(orderMapper.mapToOrder(orde
rDto)));
  }

  @RequestMapping(method = RequestMethod.POST, value = "/order", consumes =
APPLICATION_JSON_VALUE)
  public void createOrder(@RequestBody OrderDto orderDto) {
    service.getOrderDao().save(orderMapper.mapToOrder(orderDto));
  }
}
```