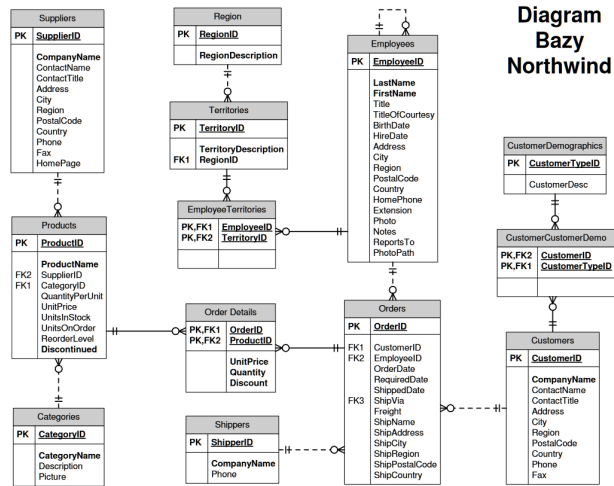Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Wydział Informatyki, Elektroniki i Telekomunikacji

Bazy danych – Northwind

Diagram
Bazy
Northwind

System do składania zamówień

Autorzy

- Kamil Gliński
- Mateusz Popielarz
- Michał Flak

Spis treści

Wstęp

Przebieg prac:

Do synchronizowania efektów pracy używamy oprogramowania GIT i serwisu GitHub

Adres do repozytorium:

https://github.com/pixellos/agh.6.bd

Użyte technologie:

- PostgreSql,
- Hibernate,
- Java,
- Spring boot
- Swagger
- SwaggerUI

Uruchomienie dla developera:

W celu uruchomienia aplikacji należy:

- Sklonować repozytorium,
- Zainstalować na lokalnym komputerze bazę danych PostgreSql
- Wykonać na bazie danych skrypty które znajdują się w repozytorium w lokalizacji /resources/db-schema
- Uruchomić aplikacje backendową przez klasę NorthwindApplication.java

Uruchomienie:

W celu uruchomienia aplikacji należy zainstalować:

- Docker for Windows
- WSL2

Wykonujemy `initialize.ps1` i aplikacja działa na

http://localhost:5000

jest też hostowana

https://northwind-java-pixellos.cloud.okteto.net/swagger-ui/

Odnośniki w aplikacji

- pobranie produktow po kategorii

http://localhost:8080/products/category/Beverages

- pobranie produktow po kraju zapewniajacego

http://localhost:8080/products/supplier/country/USA

http://localhost:8080/products/supplier/country/Japan

- pobranie produktow po zapewniajacym

http://localhost:8080/products/supplierId/1

- pobranie produktow po id klienta

http://localhost:8080/orders/customer/SUPRD

- poranie zamowien po id klienta

http://localhost:8080/orders/customer/VINET

- pobranie zamowien po id klienta

http://localhost:8080/orders/employee/2

- pobranie pracownikow po id

http://localhost:8080/employees/2

- pobranie detale zamowien po id zamowienia

http://localhost:8080/orderDetails/order/10248

- pobranie detali zamowien po id produku

http://localhost:8080/orderDetails/product/11

- pobranie detali zamowien w kategorii produktow

http://localhost:8080/orderDetails/product/category/Beverages

- pobranie detali zamowien po id zapewniajacego produkt

http://localhost:8080/orderDetails/product/supplier/1

- pobranie zamowien po id spedytora

http://localhost:8080/orders/shipper/1

Dokumentacja funkcjonalna

Docker

Co chcemy osiągnąć w tej sekcji?

Zbudować aplikację Java korzystającą z bazy danych

Prerekwizyty

https://docs.microsoft.com/en-us/windows/wsl/install-win10

https://docs.docker.com/docker-for-windows/install/

Java

Będziemy korzystać z Maven

W katalogu src\ znajdują się pliki projektu Java

Mając plik pom.xml w root solucji i korzystając z odpowiedzi https://stackoverflow.com/a/27768965/5381370 możemy w root solucji stworzyć Dockerfile, który będzie służył za postawę do postawienia naszej aplikacji

Dockerfile

./Dockerfile

```
FROM maven:3.6.0-jdk-11-slim AS build
COPY src /home/app/src
COPY pom.xml /home/app
RUN mvn -f /home/app/pom.xml clean package

FROM openjdk:11-jre-slim
COPY --from=build /home/app/target/*.jar /usr/local/lib/app.jar
EXPOSE 8080
ENTRYPOINT ["java","-jar","/usr/local/lib/app.jar"]
```

Budowanie

Możemy sprawdzić, czy obraz się poprawnie buduje

Switch t służy do nazwania obrazu

```
docker build . -t java-service:latest
```

Po kilku minutach budowanie powinno zakończyć się bez błędu

Uruchamianie

Wtedy możemy uruchomić nasz obraz

```
docker run --publish 8080:8080 --detach --name java java-service:latest
```

i w przeglądarce powinniśmy dać rady połączyć się z aplikacją
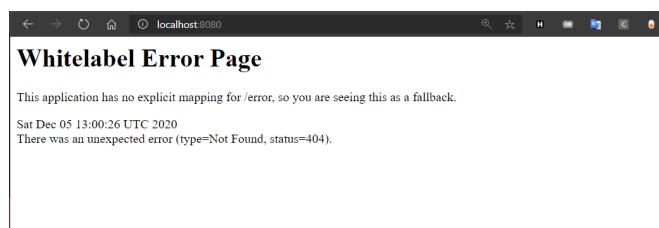
możemy go zatrzymać wywołując `docker stop java`

Ale w dalszym ciągu nie mamy bazy

PostGres

Do PostGres istnieje gotowy obraz, rozszerzmy go

Dockerfile

./Dockerfile

```
### Jako base używamy oficjalnego obrazu postgres
FROM postgres:latest

### Wszystkie pliki skopiowane do `/docker-entrypoint-initdb.d/` są wywoływane gdy
nie ma bazy
COPY src/main/resources/db-schema/db-schema.sql /docker-entrypoint-initdb.d/2_db-
schema.sql
COPY src/main/resources/db-schema/data.sql /docker-entrypoint-initdb.d/3_data.sql

ENV POSTGRES_HOST_AUTH_METHOD=trust
ENV POSTGRES_PASSWORD=postgres
ENV POSTGRES_DB=northwind
ENV POSTGRES_USER=postgres

### Komendy, które udają orginalny obraz
```

```
ENTRYPOINT ["docker-entrypoint.sh"]
EXPOSE 5432
CMD ["postgres"]
```

Budowanie

```
docker build . -f Dockerfile-northwind -t pg-service:latest
```


Uruchamianie

```
docker run --publish 5432:5432 --detach --name pg pg-service:latest
```


Komunikacja pomiędzy kontenerami `docker-compose`

Jako, że Docker nie ma domyślnie żadnego wbudowanego sposobu na łączność pomiędzy kontenerami użyjemy `docker-compose`

docker-compose.yaml

```yaml
version: '3.1'
services:
  northwind:
    build:
      context: .
      dockerfile: Dockerfile-northwind
    ports:
      - 5432:5432
    networks:
      - postgres
    volumes:
      - database-data2:/var/lib/postgresql/data/ ### persist data even if container
shuts down

  northwind-java:
    depends_on:
        - northwind
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - 8080:8080
```

```
    networks:
      - postgres

  volumes:
    database-data2:

  networks:
    postgres:
      driver: bridge
```

Ustawiliśmy sobie most sieciowy przez który połączymy sobie aplikacje

```
  networks:
    postgres:
      driver: bridge
```

w każdym Dockerfile

```
   networks:
        - postgres
```

oraz wykorzystujemy poprzednio stworzone `dockerfile`

```
  build:
    context: .
    dockerfile: Dockerfile
```

po wywołaniu

```
  & docker-compose build
  & docker-compose up
```

Nasza baza i aplikacja powinny się uruchomić, i powinniśmy być w stanie otworzyć
http://localhost:8080
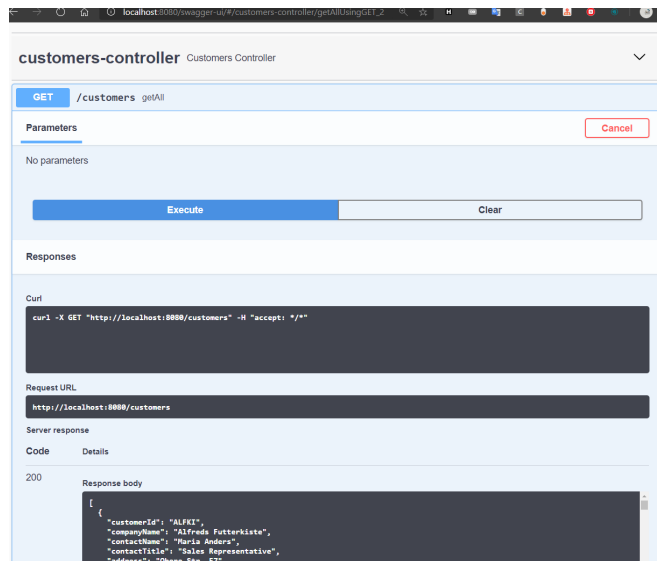
swagger ui z danymi

Deploy używając Okteto-stacks

Naszą konstelacje aplikacji możemy zdeployować za darmo używając Okteto

https://okteto.com/
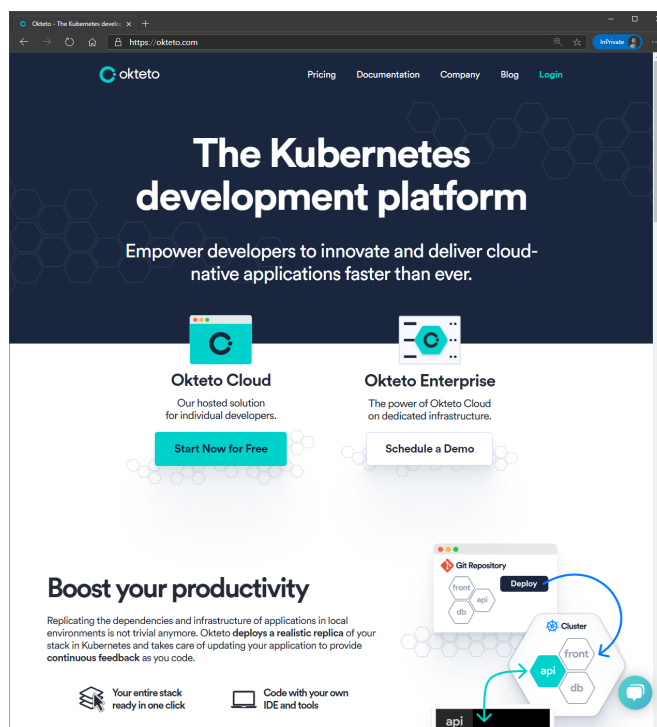
Robimy konto, polecam przez githuba

Gdy potwierdzimy mail dodajemy nasze repo
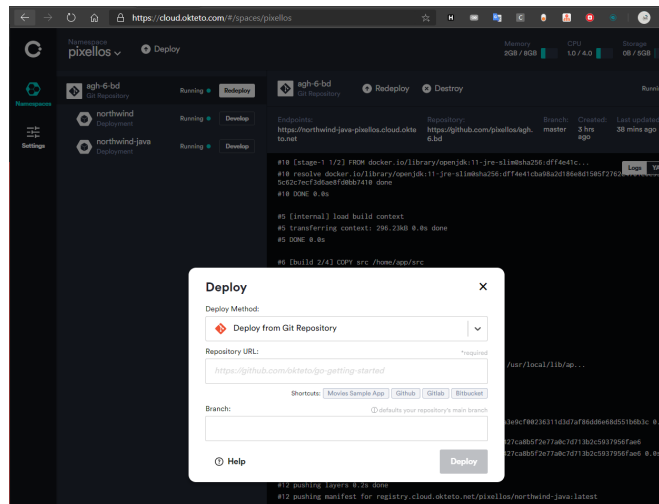
I teraz możemy przejść do konfiguracji

W repo solucji tworzymy plik `okteto-stack.yml` i odwzorowujemy naszego `docker-compse`

```
name: myapp
services:
  northwind:
    environment:
      - POSTGRES_HOST_AUTH_METHOD=trust
    image: okteto.dev/northwind
    build:
      context: .
      dockerfile: Dockerfile-northwind
    ports:
      - 5432

  northwind-java:
    public: true
    image: okteto.dev/northwind-java
    build: .
    ports:
      - 8080
```

Klikamy `redeploy`
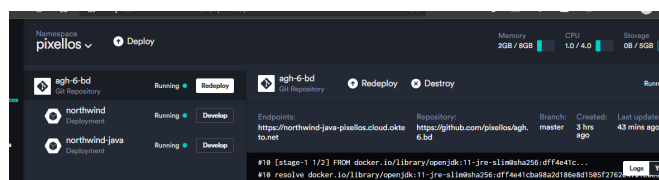
I możemy używać naszej aplikacji

https://northwind-java-pixellos.cloud.okteto.net/swagger-ui/

Interfejs użytkownika

Podstawowym interfejsem użytkownika jest Swagger UI, który pozwala na łatwy dostęp do endpointów aplikacji z poziomu przeglądarki
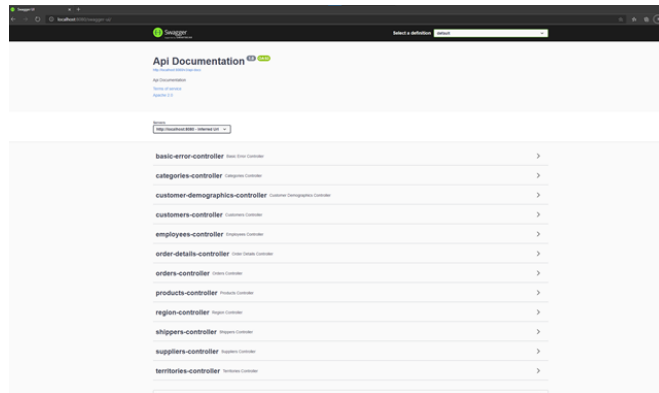
Rysunek X. Swagger UI W aplikacji

Konfiguracja

Aby go skonfigurować trzeba:

Dodać paczki

Rysunek X. Zrzut ekranu z paczkami

Skonfigurować middleware

```
package com.agh;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import
springfox.documentation.spring.data.rest.configuration.SpringDataRestConfiguration;
import springfox.documentation.swagger2.annotations.EnableSwagger2;
```

```
@SpringBootApplication
@EnableSwagger2
@Configuration
@Import(SpringDataRestConfiguration.class)
public class NorthwindApplication {

  @Bean
  public InternalResourceViewResolver defaultViewResolver() {
    return new InternalResourceViewResolver();
  }

  public static void main(String[] args) {
    SpringApplication._run_(NorthwindApplication.class, args);
  }
}
```

Rysunek X. Konfiguracja middleware

Trzeba zwrócić uwagę na linię

```
@Bean
public InternalResourceViewResolver defaultViewResolver() {
  return new InternalResourceViewResolver();
}
```

Rysunek 4. Konfiguracja ViewResolvera

W obecnej wersji w swaggerUI występuje błąd, przez który ViewResolver działa niepoprawnie z najnowszym springiem. Rozwiązaniem jest ustawienie defaultViewResolvera na właściwy typ.

Część backendowa - bottom up

Odtąd przedstawiona będzie konstrukcja backendu, zaczynając od warstwy dostępu do danych, kończąc na warstwie interfejsu ze światem oraz na przykładowych wywołaniach wraz z generowanym SQL-em.

Mapowanie obiektowo - relacyjne

W tym celu użyliśmy Hibernate'a. Konfiguracja w pliku `DatabaseConfig`:

```
@Configuration
@ComponentScan(basePackages = "com.agh")
public class DatabaseConfig {

    @Bean
    public LocalSessionFactoryBean hibernateSessionFactory() {
        LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();
        sessionFactory.setDataSource(dataSource());
        sessionFactory.setPackagesToScan("com.agh");
```

```
        sessionFactory.setHibernateProperties(hibernateProperties());
        return sessionFactory;
    }

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("org.postgresql.Driver");
        dataSource.setUrl("jdbc:postgresql://northwind:5432/northwind");
        dataSource.setUsername("postgres");
        dataSource.setPassword("postgres");
        return dataSource;
    }

    private Properties hibernateProperties() {
        Properties properties = new Properties();
        properties.put("hibernate.dialect",
"org.hibernate.dialect.PostgreSQLDialect");
        properties.put("hibernate.show_sql", "true");
        properties.put("hibernate.format_sql", "true");
        return properties;
    }
}
```

Encje zostały zamodelowane jako klasy Java odpowiadające tabelom bazy Northwind, wraz ze związkami między nimi. Przykładowo Products:

```
@Entity
@Table(name = "products")
public class Products {

    @Id
    @Column(name = "product_id")
    @SequenceGenerator(name = "productSEQ", sequenceName = "product_id_seq",
allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "productSEQ")
    private short productId;
    @Column(name = "product_name")
    private String productName;
    @Column(name = "quantity_per_unit")
    private String quantityPerUnit;
    @Column(name = "unit_price")
    private BigDecimal unitPrice;
    @Column(name = "units_in_stock")
    private Short unitsInStock;
    @Column(name = "units_on_order")
    private Short unitsOnOrder;
    @Column(name = "reorder_level")
    private Short reorderLevel;
    @Column(name = "discontinued")
    private int discontinued;
```

```java
@ManyToOne
@JoinColumn(name = "supplier_id", referencedColumnName = "supplier_id")
private Suppliers suppliers;

@ManyToOne
@JoinColumn(name = "category_id", referencedColumnName = "category_id")
private Categories categories;

public Products() {
}


public short getProductId() {
    return productId;
}

public void setProductId(short productId) {
    this.productId = productId;
}

public String getProductName() {
    return productName;
}

public void setProductName(String productName) {
    this.productName = productName;
}

public String getQuantityPerUnit() {
    return quantityPerUnit;
}

public void setQuantityPerUnit(String quantityPerUnit) {
    this.quantityPerUnit = quantityPerUnit;
}

public BigDecimal getUnitPrice() {
    return unitPrice;
}

public void setUnitPrice(BigDecimal unitPrice) {
    this.unitPrice = unitPrice;
}

public Short getUnitsInStock() {
    return unitsInStock;
}

public void setUnitsInStock(Short unitsInStock) {
    this.unitsInStock = unitsInStock;
}
```

```java
    public Short getUnitsOnOrder() {
        return unitsOnOrder;
    }

    public void setUnitsOnOrder(Short unitsOnOrder) {
        this.unitsOnOrder = unitsOnOrder;
    }

    public Short getReorderLevel() {
        return reorderLevel;
    }

    public void setReorderLevel(Short reorderLevel) {
        this.reorderLevel = reorderLevel;
    }

    public int getDiscontinued() {
        return discontinued;
    }

    public void setDiscontinued(int discontinued) {
        this.discontinued = discontinued;
    }

    public Suppliers getSuppliers() {
        return suppliers;
    }

    public void setSuppliers(Suppliers suppliers) {
        this.suppliers = suppliers;
    }

    public Categories getCategories() {
        return categories;
    }

    public void setCategories(Categories categories) {
        this.categories = categories;
    }
}
```

Do zapisywania / odczytywania danych użyliśmy wzorca Repository, żeby zenkapsulować szczegóły implementacji. `AbstractRepository` prezentuje się następująco:

```java
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;

public abstract class AbstractRepository {

    @Autowired
```

```java
    private SessionFactory sessionFactory;

    protected Session getSession() {
        return sessionFactory.getCurrentSession();
    }

    protected Session getOpenSession() {
        return sessionFactory.openSession();
    }

}
```

A przykładowe repozytorium dla produktów:

```java
@Repository
public class ProductsRepository extends AbstractRepository {

    public List<Products> getAll() {
        Session session = getOpenSession();
        Transaction transaction = session.beginTransaction();
        List<Products> products = session
                .createQuery("SELECT p FROM Products p" +
                        " INNER JOIN FETCH p.suppliers s" +
                        " INNER JOIN FETCH p.categories c", Products.class)
                .list();
        transaction.commit();
        session.close();
        return products;
    }

    public Optional<Products> getById(Short productId) {
        Session session = getOpenSession();
        Transaction transaction = session.beginTransaction();
        Optional<Products> product = session
                .createQuery("SELECT p FROM Products p WHERE p.productId=:productId",
Products.class)
                .setParameter("productId", productId)
                .uniqueResultOptional();
        transaction.commit();
        session.close();
        return product;
    }

    public List<Products> getAllByCategory(String categoryName) {
        Session session = getOpenSession();
        Transaction transaction = session.beginTransaction();

        List<Products> products = session.createQuery(
                "SELECT p FROM Products p" +
                        " INNER JOIN FETCH p.suppliers s" +
                        " INNER JOIN FETCH p.categories c" +
```

```java
                    " WHERE c.categoryName=:categoryName", Products.class)
                .setParameter("categoryName", categoryName)
                .list();

        transaction.commit();
        session.close();
        return products;
    }

    public List<Products> getAllBySupplierId(Short supplierId) {
        Session session = getOpenSession();
        Transaction transaction = session.beginTransaction();

        List<Products> products = session.createQuery(
                "SELECT p FROM Products p" +
                        " INNER JOIN FETCH p.suppliers s" +
                        " INNER JOIN FETCH p.categories c" +
                        " WHERE s.supplierId=:supplierId", Products.class)
                .setParameter("supplierId", supplierId)
                .list();

        transaction.commit();
        session.close();
        return products;
    }

    public List<Products> getAllBySuppliersCountry(String suppliersCountry) {
        Session session = getOpenSession();
        Transaction transaction = session.beginTransaction();

        List<Products> products = session.createQuery(
                "SELECT p FROM Products p" +
                        " INNER JOIN FETCH p.suppliers s " +
                        " INNER JOIN FETCH p.categories c" +
                        " WHERE s.country=:suppliersCountry", Products.class)
                .setParameter("suppliersCountry", suppliersCountry)
                .list();

        transaction.commit();
        session.close();
        return products;
    }

    public void persist(Products product) {
        Session session = getOpenSession();
        Transaction transaction = session.beginTransaction();
        session.persist(product);
        transaction.commit();
        session.close();
    }
}
```

Można zauważyć tu kilka rzeczy:

- Metody `GET...` korzystają z napisanych ręcznie kwerend. Encje, które zawierają obiekty powiązane w relacji pobieramy za pomoca komendy JOIN FETCH w jednym zapytaniu, dzięki temu unikamy problemu n+1 zapytań
- Zgodnie z sugestią prowadzącego, zamieniono `left joiny` na `inner join`-y, których działanie jest bardziej optymalne
- Korzystamy z `join fetch` dla eager loading
- Odczyt / zapis odbywa się w transakcjach, zgodnie z zasadmi ACID
- Zapisywanie odbywa sie dzieki metodzie `persist` , po wczesniejszym przejsciu przez walidacje.

Część biznesowo - aplikacyjna

Z repozytoriów korzystają następnie klasy `service`, zawierające odrobinę więcej logiki biznesowej - na przykład konstrukcja odpowiedniego obiektu do zapisana na podstawie requestu, jak również walidacja tego obiektu.

Przykładowo - `ProductsService`:

```java
package com.agh.service;

import com.agh.model.Categories;
import com.agh.model.Products;
import com.agh.model.Suppliers;
import com.agh.repository.ProductsRepository;
import com.agh.request.CreateProductRequest;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class ProductsService {

    private final ProductsRepository productsRepository;
    private final SuppliersService suppliersService;
    private final CategoriesService categoriesService;
    private final ValidationService validationService;

    @Autowired
    public ProductsService(ProductsRepository productsRepository, SuppliersService
suppliersService, CategoriesService categoriesService,
                           ValidationService validationService) {
        this.productsRepository = productsRepository;
        this.suppliersService = suppliersService;
        this.categoriesService = categoriesService;
        this.validationService = validationService;
    }

    public List<Products> getAll() {
        return productsRepository.getAll();
    }
```

```java
    public Products getById(Short productId) {
        return
productsRepository.getById(productId).orElseThrow(IllegalArgumentException::new);
    }

    public List<Products> getAllByCategory(String categoryName) {
        return productsRepository.getAllByCategory(categoryName);
    }

    public List<Products> getAllBySupplierId(Short supplierId) {
        return productsRepository.getAllBySupplierId(supplierId);
    }

    public List<Products> getAllBySuppliersCountry(String countryName) {
        return productsRepository.getAllBySuppliersCountry(countryName);
    }

    public void create(CreateProductRequest request) {
        Products product = new Products();
        Suppliers supplier = suppliersService.getById(request.getSupplierId());
        Categories category = categoriesService.getById(request.getCategoryId());
        product.setSuppliers(supplier);
        product.setCategories(category);

        product.setProductName(request.getProductName());
        product.setQuantityPerUnit(request.getQuantityPerUnit());
        product.setUnitPrice(request.getUnitPrice());
        product.setUnitsInStock(request.getUnitsInStock());
        product.setUnitsOnOrder(request.getUnitsOnOrder());
        product.setReorderLevel(request.getReorderLevel());
        product.setDiscontinued(request.getDiscontinued());

        validationService.validate(product);
        productsRepository.persist(product);
    }
}
```

ValidationService sprawdza, czy obiekt który chcemy zapisać spełnia nasze założenia:

```java
@Service
public class ValidationService {

    public void validate(Products product) {
        if (product.getProductName() == null ||
product.getProductName().trim().isEmpty()) {
            throw new IllegalArgumentException("Product name cannot be empty");
        }

        if (product.getQuantityPerUnit() == null ||
```

```
product.getQuantityPerUnit().trim().isEmpty()) {
            throw new IllegalArgumentException("Quantity per unit cannot be empty");
        }

        if (product.getUnitPrice().compareTo(BigDecimal.ZERO) <= 0) {
            throw new IllegalArgumentException("Unit price has to be greater than
0");
        }

        if (product.getUnitsInStock() <= 0 || product.getUnitsOnOrder() <= 0) {
            throw new IllegalArgumentException("Units in stock and Units on order has
to be greater than 0");
        }
    }

    public void validate(Orders order) {
        if (order.getOrderDate().isAfter(order.getRequiredDate())) {
            throw new IllegalArgumentException("Order date has to be before required
date");
        }

        if (order.getShippedDate().isAfter(order.getRequiredDate())) {
            throw new IllegalArgumentException("Shipped date has to be before
required date");
        }

        if (order.getFreight() <= 0) {
            throw new IllegalArgumentException("Freight has to be greater than 0");
        }

        if (order.getShipName() == null || order.getShipName().trim().isEmpty()) {
            throw new IllegalArgumentException("Ship name cannot be empty");
        }

        if (order.getShipAddress() == null ||
order.getShipAddress().trim().isEmpty()) {
            throw new IllegalArgumentException("Ship address cannot be empty");
        }

        if (order.getShipCity() == null || order.getShipCity().trim().isEmpty()) {
            throw new IllegalArgumentException("Ship city cannot be empty");
        }

        if (order.getShipRegion() == null || order.getShipRegion().trim().isEmpty())
{
            throw new IllegalArgumentException("Ship region cannot be empty");
        }

        if (order.getShipPostalCode() == null ||
order.getShipPostalCode().trim().isEmpty()) {
            throw new IllegalArgumentException("Ship postal code cannot be empty");
        }
```

```java
        if (order.getShipCountry() == null ||
order.getShipCountry().trim().isEmpty()) {
            throw new IllegalArgumentException("Ship country cannot be empty");
        }
    }

    public void validate(OrderDetails orderDetails) {
        if (orderDetails.getQuantity() <= 0) {
            throw new IllegalArgumentException("Quantity has to be greater than 0");
        }

        if (orderDetails.getUnitPrice() <= 0) {
            throw new IllegalArgumentException("Unit price has to be greater than
0");
        }

        if (orderDetails.getOrderDetailsId().getOrders() == null) {
            throw new IllegalArgumentException("OrderDetails has to be in
relationship with Order");
        }

        if (orderDetails.getOrderDetailsId().getProducts() == null) {
            throw new IllegalArgumentException("OrderDetails has to be in
relationship with Product");
        }
    }
}
```

Serwisy następnie używane są przez kontrolery frameworka Spring, który obsługuje zapytania po HTTP. Przykładowo produkt:

```java
@RestController
public class ProductsController {

    private final ProductsService productsService;

    @Autowired
    public ProductsController(ProductsService productsService) {
        this.productsService = productsService;
    }

    @GetMapping("products")
    public ResponseEntity<List<Products>> getAll() {
        return new ResponseEntity<>(productsService.getAll(), HttpStatus.OK);
    }

    @GetMapping("products/category/{categoryName}")
    public ResponseEntity<List<Products>> getAllByCategory(@PathVariable String
categoryName) {
        return new ResponseEntity<>(productsService.getAllByCategory(categoryName),
HttpStatus.OK);
```

```java
    }

    @GetMapping("products/supplierId/{supplierId}")
    public ResponseEntity<List<Products>> getAllBySupplierId(@PathVariable Short
supplierId) {
        return new ResponseEntity<>(productsService.getAllBySupplierId(supplierId),
HttpStatus.OK);
    }

    @GetMapping("products/supplier/country/{countryName}")
    public ResponseEntity<List<Products>> getAllBySuppliersCountry(@PathVariable
String countryName) {
        return new ResponseEntity<>
(productsService.getAllBySuppliersCountry(countryName), HttpStatus.OK);
    }

    @PostMapping("products")
    public ResponseEntity<Void> create(@RequestBody CreateProductRequest request) {
        productsService.create(request);
        return new ResponseEntity<>(HttpStatus.OK);
    }
}
```

Tutaj też pojawiają się klasy będące ciałem requestów - jak `CreateProductRequest`:

```java
public class CreateProductRequest {
    private Short supplierId;
    private Short categoryId;
    // ***
    private String productName;
    private String quantityPerUnit;
    private BigDecimal unitPrice;
    private Short unitsInStock;
    private Short unitsOnOrder;
    private Short reorderLevel;
    private Integer discontinued;
        // [...getters, setters...]
}
```

Przykładowe wywołania na produkcie:

1. GetAll

```
GET http://localhost:8080/products
```

Kod z ProductRepository:

```
        List<Products> products = session
                .createQuery("SELECT p FROM Products p" +
                        " INNER JOIN FETCH p.suppliers s" +
                        " INNER JOIN FETCH p.categories c", Products.class)
                .list();
```

Generowany SQL:

```
Hibernate:
    select
        products0_.product_id as product_1_15_0_,
        suppliers1_.supplier_id as supplier1_20_1_,
        categories2_.category_id as category1_0_2_,
        products0_.category_id as category9_15_0_,
        products0_.discontinued as disconti2_15_0_,
        products0_.product_name as product_3_15_0_,
        products0_.quantity_per_unit as quantity4_15_0_,
        products0_.reorder_level as reorder_5_15_0_,
        products0_.supplier_id as supplie10_15_0_,
        products0_.unit_price as unit_pri6_15_0_,
        products0_.units_in_stock as units_in7_15_0_,
        products0_.units_on_order as units_on8_15_0_,
        suppliers1_.address as address2_20_1_,
        suppliers1_.city as city3_20_1_,
        suppliers1_.company_name as company_4_20_1_,
        suppliers1_.contact_name as contact_5_20_1_,
        suppliers1_.contact_title as contact_6_20_1_,
        suppliers1_.country as country7_20_1_,
        suppliers1_.fax as fax8_20_1_,
        suppliers1_.homepage as homepage9_20_1_,
        suppliers1_.phone as phone10_20_1_,
        suppliers1_.postal_code as postal_11_20_1_,
        suppliers1_.region as region12_20_1_,
        categories2_.category_name as category2_0_2_,
        categories2_.description as descript3_0_2_,
        categories2_.picture as picture4_0_2_
    from
        products products0_
    inner join
        suppliers suppliers1_
            on products0_.supplier_id=suppliers1_.supplier_id
    inner join
        categories categories2_
            on products0_.category_id=categories2_.category_id
```

2. GetAllByCategory

```
GET http://localhost:8080/products/category/Beverages
```

Kod z ProductRepository:

```
        List<Products> products = session.createQuery(
                "SELECT p FROM Products p" +
                        " INNER JOIN FETCH p.suppliers s" +
                        " INNER JOIN FETCH p.categories c" +
                        " WHERE c.categoryName=:categoryName", Products.class)
                .setParameter("categoryName", categoryName)
                .list();
```

Generowany SQL:

```
  Hibernate:
      select
          products0_.product_id as product_1_15_0_,
          suppliers1_.supplier_id as supplier1_20_1_,
          categories2_.category_id as category1_0_2_,
          products0_.category_id as category9_15_0_,
          products0_.discontinued as disconti2_15_0_,
          products0_.product_name as product_3_15_0_,
          products0_.quantity_per_unit as quantity4_15_0_,
          products0_.reorder_level as reorder_5_15_0_,
          products0_.supplier_id as supplie10_15_0_,
          products0_.unit_price as unit_pri6_15_0_,
          products0_.units_in_stock as units_in7_15_0_,
          products0_.units_on_order as units_on8_15_0_,
          suppliers1_.address as address2_20_1_,
          suppliers1_.city as city3_20_1_,
          suppliers1_.company_name as company_4_20_1_,
          suppliers1_.contact_name as contact_5_20_1_,
          suppliers1_.contact_title as contact_6_20_1_,
          suppliers1_.country as country7_20_1_,
          suppliers1_.fax as fax8_20_1_,
          suppliers1_.homepage as homepage9_20_1_,
          suppliers1_.phone as phone10_20_1_,
          suppliers1_.postal_code as postal_11_20_1_,
          suppliers1_.region as region12_20_1_,
          categories2_.category_name as category2_0_2_,
          categories2_.description as descript3_0_2_,
          categories2_.picture as picture4_0_2_
      from
          products products0_
      inner join
          suppliers suppliers1_
              on products0_.supplier_id=suppliers1_.supplier_id
      inner join
          categories categories2_
              on products0_.category_id=categories2_.category_id
```

```
    where
        categories2_.category_name=?
```

3. Create

```
POST http://localhost:8080/products/

{
  "categoryId": 1,
  "discontinued": 0,
  "productName": "test",
  "quantityPerUnit": "5",
  "reorderLevel": 1,
  "supplierId": 1,
  "unitPrice": 1,
  "unitsInStock": 1,
  "unitsOnOrder": 1
}
```

Kod z ProductRepository:

```
        session.persist(product);
```

Generowany SQL:

```
Hibernate:
    select
        suppliers0_.supplier_id as supplier1_20_,
        suppliers0_.address as address2_20_,
        suppliers0_.city as city3_20_,
        suppliers0_.company_name as company_4_20_,
        suppliers0_.contact_name as contact_5_20_,
        suppliers0_.contact_title as contact_6_20_,
        suppliers0_.country as country7_20_,
        suppliers0_.fax as fax8_20_,
        suppliers0_.homepage as homepage9_20_,
        suppliers0_.phone as phone10_20_,
        suppliers0_.postal_code as postal_11_20_,
        suppliers0_.region as region12_20_
    from
        suppliers suppliers0_
    where
        suppliers0_.supplier_id=?
Hibernate:
    select
        categories0_.category_id as category1_0_,
        categories0_.category_name as category2_0_,
```

```
        categories0_.description as descript3_0_,
        categories0_.picture as picture4_0_
    from
        categories categories0_
    where
        categories0_.category_id=?
Hibernate:
    select
        nextval ('product_id_seq')
Hibernate:
    select
        categories_.category_id,
        categories_.category_name as category2_0_,
        categories_.description as descript3_0_,
        categories_.picture as picture4_0_
    from
        categories categories_
    where
        categories_.category_id=?
Hibernate:
    select
        suppliers_.supplier_id,
        suppliers_.address as address2_20_,
        suppliers_.city as city3_20_,
        suppliers_.company_name as company_4_20_,
        suppliers_.contact_name as contact_5_20_,
        suppliers_.contact_title as contact_6_20_,
        suppliers_.country as country7_20_,
        suppliers_.fax as fax8_20_,
        suppliers_.homepage as homepage9_20_,
        suppliers_.phone as phone10_20_,
        suppliers_.postal_code as postal_11_20_,
        suppliers_.region as region12_20_
    from
        suppliers suppliers_
    where
        suppliers_.supplier_id=?
Hibernate:
    insert
    into
        products
        (category_id, discontinued, product_name, quantity_per_unit, reorder_level,
supplier_id, unit_price, units_in_stock, units_on_order, product_id)
    values
        (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

Przykładowe wywołania na Orderze:

1. GetAll

```
GET http://localhost:8080/orders
```

Kod z OrderRepository:

```
        List<Orders> orders = session
                .createQuery("SELECT o FROM Orders o" +
                        " INNER JOIN FETCH o.customers c" +
                        " INNER JOIN FETCH o.employees e" +
                        " INNER JOIN FETCH o.shippers s ", Orders.class)
```

Generowany SQL:

```
  Hibernate:
      select
          orders0_.order_id as order_id1_13_0_,
          customers1_.customer_id as customer1_5_1_,
          employees2_.employee_id as employee1_9_2_,
          shippers3_.shipper_id as shipper_1_18_3_,
          orders0_.customer_id as custome12_13_0_,
          orders0_.employee_id as employe13_13_0_,
          orders0_.freight as freight2_13_0_,
          orders0_.order_date as order_da3_13_0_,
          orders0_.required_date as required4_13_0_,
          orders0_.ship_address as ship_add5_13_0_,
          orders0_.ship_city as ship_cit6_13_0_,
          orders0_.ship_country as ship_cou7_13_0_,
          orders0_.ship_name as ship_nam8_13_0_,
          orders0_.ship_postal_code as ship_pos9_13_0_,
          orders0_.ship_region as ship_re10_13_0_,
          orders0_.shipped_date as shipped11_13_0_,
          orders0_.ship_via as ship_vi14_13_0_,
          customers1_.address as address2_5_1_,
          customers1_.city as city3_5_1_,
          customers1_.company_name as company_4_5_1_,
          customers1_.contact_name as contact_5_5_1_,
          customers1_.contact_title as contact_6_5_1_,
          customers1_.country as country7_5_1_,
          customers1_.fax as fax8_5_1_,
          customers1_.phone as phone9_5_1_,
          customers1_.postal_code as postal_10_5_1_,
          customers1_.region as region11_5_1_,
          employees2_.address as address2_9_2_,
          employees2_.birth_date as birth_da3_9_2_,
          employees2_.city as city4_9_2_,
          employees2_.country as country5_9_2_,
          employees2_.extension as extensio6_9_2_,
          employees2_.first_name as first_na7_9_2_,
          employees2_.hire_date as hire_dat8_9_2_,
          employees2_.home_phone as home_pho9_9_2_,
          employees2_.last_name as last_na10_9_2_,
```

```
        employees2_.notes as notes11_9_2_,
        employees2_.photo as photo12_9_2_,
        employees2_.photo_path as photo_p13_9_2_,
        employees2_.postal_code as postal_14_9_2_,
        employees2_.region as region15_9_2_,
        employees2_.title as title16_9_2_,
        employees2_.title_of_courtesy as title_o17_9_2_,
        shippers3_.company_name as company_2_18_3_,
        shippers3_.phone as phone3_18_3_
    from
        Orders orders0_
    inner join
        customers customers1_
            on orders0_.customer_id=customers1_.customer_id
    inner join
        employees employees2_
            on orders0_.employee_id=employees2_.employee_id
    inner join
        shippers shippers3_
            on orders0_.ship_via=shippers3_.shipper_id
```

2. GetAllByShipperId

```
GET http://localhost:8080/orders/shipper/1
```

Kod z OrderRepository:

```
        List<Orders> orders = session
                .createQuery("SELECT o FROM Orders o" +
                        " INNER JOIN FETCH o.customers c" +
                        " INNER JOIN FETCH o.employees e" +
                        " INNER JOIN FETCH o.shippers s " +
                        " WHERE s.shipperId=:shipperId", Orders.class)
                .setParameter("shipperId", shipperId)
                .list();
```

Generowany SQL:

```
Hibernate:
    select
        orders0_.order_id as order_id1_13_0_,
        customers1_.customer_id as customer1_5_1_,
        employees2_.employee_id as employee1_9_2_,
        shippers3_.shipper_id as shipper_1_18_3_,
        orders0_.customer_id as custome12_13_0_,
        orders0_.employee_id as employe13_13_0_,
        orders0_.freight as freight2_13_0_,
```

```
        orders0_.order_date as order_da3_13_0_,
        orders0_.required_date as required4_13_0_,
        orders0_.ship_address as ship_add5_13_0_,
        orders0_.ship_city as ship_cit6_13_0_,
        orders0_.ship_country as ship_cou7_13_0_,
        orders0_.ship_name as ship_nam8_13_0_,
        orders0_.ship_postal_code as ship_pos9_13_0_,
        orders0_.ship_region as ship_re10_13_0_,
        orders0_.shipped_date as shipped11_13_0_,
        orders0_.ship_via as ship_vi14_13_0_,
        customers1_.address as address2_5_1_,
        customers1_.city as city3_5_1_,
        customers1_.company_name as company_4_5_1_,
        customers1_.contact_name as contact_5_5_1_,
        customers1_.contact_title as contact_6_5_1_,
        customers1_.country as country7_5_1_,
        customers1_.fax as fax8_5_1_,
        customers1_.phone as phone9_5_1_,
        customers1_.postal_code as postal_10_5_1_,
        customers1_.region as region11_5_1_,
        employees2_.address as address2_9_2_,
        employees2_.birth_date as birth_da3_9_2_,
        employees2_.city as city4_9_2_,
        employees2_.country as country5_9_2_,
        employees2_.extension as extensio6_9_2_,
        employees2_.first_name as first_na7_9_2_,
        employees2_.hire_date as hire_dat8_9_2_,
        employees2_.home_phone as home_pho9_9_2_,
        employees2_.last_name as last_na10_9_2_,
        employees2_.notes as notes11_9_2_,
        employees2_.photo as photo12_9_2_,
        employees2_.photo_path as photo_p13_9_2_,
        employees2_.postal_code as postal_14_9_2_,
        employees2_.region as region15_9_2_,
        employees2_.title as title16_9_2_,
        employees2_.title_of_courtesy as title_o17_9_2_,
        shippers3_.company_name as company_2_18_3_,
        shippers3_.phone as phone3_18_3_
    from
        Orders orders0_
    inner join
        customers customers1_
            on orders0_.customer_id=customers1_.customer_id
    inner join
        employees employees2_
            on orders0_.employee_id=employees2_.employee_id
    inner join
        shippers shippers3_
            on orders0_.ship_via=shippers3_.shipper_id
    where
        shippers3_.shipper_id=?
```

3. Create

```
POST http://localhost:8080/orders/

{
  "customerId": "ALFKI",
  "employeeId": 1,
  "freight": 1,
  "requiredDate": "01-01-2022",
  "shipAddress": "string",
  "shipCity": "string",
  "shipCountry": "string",
  "shipName": "string",
  "shipPostalCode": "string",
  "shipRegion": "string",
  "shippedDate": "01-01-2022",
  "shipperId": 1
}
```

Kod z OrderRepository:

```
        session.persist(order);
```

Generowany SQL:

```
Hibernate:
    select
        shippers0_.shipper_id as shipper_1_18_,
        shippers0_.company_name as company_2_18_,
        shippers0_.phone as phone3_18_
    from
        shippers shippers0_
    where
        shippers0_.shipper_id=?
Hibernate:
    select
        employees0_.employee_id as employee1_9_,
        employees0_.address as address2_9_,
        employees0_.birth_date as birth_da3_9_,
        employees0_.city as city4_9_,
        employees0_.country as country5_9_,
        employees0_.extension as extensio6_9_,
        employees0_.first_name as first_na7_9_,
        employees0_.hire_date as hire_dat8_9_,
        employees0_.home_phone as home_pho9_9_,
        employees0_.last_name as last_na10_9_,
        employees0_.notes as notes11_9_,
        employees0_.photo as photo12_9_,
```

```
        employees0_.photo_path as photo_p13_9_,
        employees0_.postal_code as postal_14_9_,
        employees0_.region as region15_9_,
        employees0_.title as title16_9_,
        employees0_.title_of_courtesy as title_o17_9_
    from
        employees employees0_
    where
        employees0_.employee_id=?
Hibernate:
    select
        customers0_.customer_id as customer1_5_,
        customers0_.address as address2_5_,
        customers0_.city as city3_5_,
        customers0_.company_name as company_4_5_,
        customers0_.contact_name as contact_5_5_,
        customers0_.contact_title as contact_6_5_,
        customers0_.country as country7_5_,
        customers0_.fax as fax8_5_,
        customers0_.phone as phone9_5_,
        customers0_.postal_code as postal_10_5_,
        customers0_.region as region11_5_
    from
        customers customers0_
    where
        customers0_.customer_id=?
Hibernate:
    select
        nextval ('order_id_seq')
Hibernate:
    select
        customers_.customer_id,
        customers_.address as address2_5_,
        customers_.city as city3_5_,
        customers_.company_name as company_4_5_,
        customers_.contact_name as contact_5_5_,
        customers_.contact_title as contact_6_5_,
        customers_.country as country7_5_,
        customers_.fax as fax8_5_,
        customers_.phone as phone9_5_,
        customers_.postal_code as postal_10_5_,
        customers_.region as region11_5_
    from
        customers customers_
    where
        customers_.customer_id=?
Hibernate:
    select
        employees_.employee_id,
        employees_.address as address2_9_,
        employees_.birth_date as birth_da3_9_,
        employees_.city as city4_9_,
        employees_.country as country5_9_,
```

```
        employees_.extension as extensio6_9_,
        employees_.first_name as first_na7_9_,
        employees_.hire_date as hire_dat8_9_,
        employees_.home_phone as home_pho9_9_,
        employees_.last_name as last_na10_9_,
        employees_.notes as notes11_9_,
        employees_.photo as photo12_9_,
        employees_.photo_path as photo_p13_9_,
        employees_.postal_code as postal_14_9_,
        employees_.region as region15_9_,
        employees_.title as title16_9_,
        employees_.title_of_courtesy as title_o17_9_
    from
        employees employees_
    where
        employees_.employee_id=?
Hibernate:
    select
        shippers_.shipper_id,
        shippers_.company_name as company_2_18_,
        shippers_.phone as phone3_18_
    from
        shippers shippers_
    where
        shippers_.shipper_id=?
Hibernate:
    insert
    into
        Orders
        (customer_id, employee_id, freight, order_date, required_date, ship_address,
ship_city, ship_country, ship_name, ship_postal_code, ship_region, shipped_date,
ship_via, order_id)
    values
        (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

Encje z rozbudowanym CRUD-em

- Products
  - getAll
  - getById
  - getAllByCategory
  - getAllBySupplierId
  - getAllBySuppliersCountry
  - persist
- Orders
  - getAll
  - getById
  - getAllByCustomerId
  - getAllByEmployeeId

- - getAllByShipperId
    - persist
- OrderDetails
  - getAll
  - getByOrderId
  - getByProductId
  - getByProductsCategory
  - getBySupplierId
  - persist

Powyższe zrealizowane są analogicznie w stosunku do tego co przedstawiono wyżej dla Produktu.

Pozostałe mają podstawowe `getAll` oraz `getById`.