

Dawid Majchrowski

Hibernate, JPA

Sprawozdanie - 24.11.2019

Sprawozdanie kontynuujemy od miejsca zakończenia ćwiczeń (zad 6):

6. Nowa klasa Category

Dodajemy klasę Category z relacją 1 do wielu po obu stronach.

```
5  @Entity
6  public class Category {
7      @Id
8      @GeneratedValue(strategy = GenerationType.AUTO)
9      private int CategoryID;
10     String name;
11
12     @OneToMany(mappedBy = "category")
13     private List<Product> products;
14
15     public Category(){
16         this.products = new ArrayList<Product>();
17     }
18
19     public Category(String name) {
20         this.name = name;
21         this.products = new ArrayList<Product>();
22     }
23
24     public List<Product> getProducts() {
25         return products;
26     }
27
28     public String getName() {
29         return name;
30     }
31
32     public void addProduct(Product product){
33         this.products.add(product);
34         product.addCategory(this);
35     }
```

```
<mapping class="Category"></mapping>
```

```
@ManyToOne
@JoinColumn(name = "CategoryFK")
private Category category;
```

```
public void addCategory(Category category){
    category.getProducts().add(this);
    this.category = category;
}
```

W Mainie dodajemy kilka produktów oraz kategori.

```
Product product3 = new Product( productName: "Prod3", unitsonstock: 5);  
Supplier supplier = new Supplier( companyName: "AGH2", street: "Czarnowiejska2", city: "Krakow2 ");  
Category category1 = new Category( name: "Cat1");  
Category category2 = new Category( name: "Cat2");  
  
sessionFactory = getSessionFactory();  
Session session = sessionFactory.openSession();  
Transaction tx = session.beginTransaction();  
session.save(product1);  
session.save(product2);  
session.save(product3);  
session.save(supplier);  
session.save(category1);  
session.save(category2);  
supplier.addProduct(product1);  
supplier.addProduct(product2);  
supplier.addProduct(product3);  
product1.addCategory(category1);  
product2.addCategory(category2);  
product3.addCategory(category1);
```

```
select * from PRODUCT;
```

PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORYFK	SUPPLIERFK
1	Prod1	10	5	4
2	Prod2	0	6	4
3	Prod3	5	5	4

```
select * from CATEGORY;
```

CATEGORYID	NAME
5	Cat1
6	Cat2

```
Hibernate:
/* insert Category
*/ insert
into
    Category
    (name, CategoryID)
values
    (?, ?)
```

```
Hibernate:
/* update
Product */ update
Product
set
    CategoryFK=?,
    productName=?,
    SupplierFK=?,
    unitsOnStock=?
where
    productID=?
```

```
Hibernate:
/* update
Product */ update
Product
set
    CategoryFK=?,
    productName=?,
    SupplierFK=?,
    unitsOnStock=?
where
    productID=?
```

```
Hibernate:
/* update
Product */ update
Product
set
    CategoryFK=?,
    productName=?,
    SupplierFK=?,
    unitsOnStock=?
where
    productID=?
```

Wyszukiwanie produktu z danej kategorii i kategorii dla której należy dany produkt.

```
TypedQuery<Product> prodByCat = session.createQuery( s: "from Product as product" + " where lower(product.category.name)=:categoryName", Product.class);
prodByCat.setParameter( s: "categoryName", "cat1");
for (Product product: prodByCat.getResultList()){
    System.out.println(product.getProductName());
}
TypedQuery<Category> catByProd = session.createQuery( s: "from Category as category" + " where :product member of category.products", Category.class);
catByProd.setParameter( s: "product", product1);
for (Category category: catByProd.getResultList()){
    System.out.println(category.getName());
}

session.close();
```

```
Product / update
Product
set
    CategoryFK=?,
    productName=?,
    SupplierFK=?,
    unitsOnStock=?
where
    productID=?
Hibernate:
/*
from
    Product as product
where
    lower(product.category.name)=:categoryName */ select
    product0_.productID as product1_1_,
    product0_.CategoryFK as Category4_1_,
    product0_.productName as product2_1_,
    product0_.SupplierFK as Supplier5_1_,
    product0_.unitsOnStock as unitsOnS3_1_
from
    Product product0_,
    Category category1_
where
    product0_.CategoryFK=category1_.CategoryID
    and lower(category1_.name)=?
Prod1
Prod3
Hibernate:
/*
from
    Category as category
where
    :product member of category.products */ select
    category0_.CategoryID as Category1_0_,
    category0_.name as name2_0_
from
    Category category0_
where
    ? in (
        select
            products1_.productID
        from
            Product products1_
        where
            category0_.CategoryID=products1_.CategoryFK
    )
Cat1
```

7. Relacja wiele do wielu

Dodajemy klasę Invoice i mapujemy relację wiele do wielu z klasą Product.

```
5  @Entity
6  public class Invoice {
7      @Id
8      @GeneratedValue(strategy = GenerationType.AUTO)
9      private int invoiceID;
10
11     private int invoiceNumber;
12     private int quantity;
13     @ManyToMany
14     private List<Product> products= new ArrayList<>();
15
16     public Invoice(int invoiceNumber, int quantity) {
17         this.invoiceNumber = invoiceNumber;
18         this.quantity = quantity;
19     }
20
21     public Invoice() {
22     }
23
24     public void addProduct(Product product){
25         if(product.getUnitsOnStock() > 0){
26             products.add(product);
27             product.getInvoices().add(this);
28             product.setUnitsOnStock(product.getUnitsOnStock()-1);
29             this.quantity++;
30         }
31     }
32
33     public int getInvoiceNumber() {
34         return invoiceNumber;
35     }
36
37     public void setInvoiceNumber(int invoiceNumber) {
38         this.invoiceNumber = invoiceNumber;
39     }
40
41     public int getQuantity() {
42         return quantity;
43     }
44
45     <mapping class="Invoice"></mapping>
```



```

public Product() {
}

public List<Invoice> getInvoices() {
    return invoices;
}

public void addInvoice(Invoice invoice){
    invoice.addProduct(this);
}

```

```

@ManyToMany(mappedBy = "products")
private List<Invoice> invoices = new ArrayList<>();

```

Dodajemy do starego maina, 2 faktury i dodajemy “sprzedajemy” produkty na odpowiednich transakcjach. Zauważmy, że wykonujemy 4 transakcje sprzedaży, natomiast product2 ma wartość 0 unitOnStock, dlatego sprzedaż produktu 2 nie powinna się powieść, a w bazie powinny pojawić się tylko 3 rekordy.

```

10 public class Main {
11     private static SessionFactory sessionFactory = null;
12
13     public static void main(String[] args) {
14         Product product1 = new Product( productName: "Prod1", unitsOnStock: 10);
15         Product product2 = new Product( productName: "Prod2", unitsOnStock: 0);
16         Product product3 = new Product( productName: "Prod3", unitsOnStock: 5);
17         Supplier supplier = new Supplier( companyName: "AGH2", street: "Czarnowiejska2", city: "Krakow2 ");
18         Category category1 = new Category( name: "Cat1");
19         Category category2 = new Category( name: "Cat2");
20         Invoice invoice1 = new Invoice( invoiceNumber: 1);
21         Invoice invoice2 = new Invoice( invoiceNumber: 2);
22
23         sessionFactory = getSessionFactory();
24         Session session = sessionFactory.openSession();
25         Transaction tx = session.beginTransaction();
26         session.save(product1);
27         session.save(product2);
28         session.save(product3);
29         session.save(supplier);
30         session.save(category1);
31         session.save(category2);
32         session.save(invoice1);
33         session.save(invoice2);
34         supplier.addProduct(product1);
35         supplier.addProduct(product2);
36         supplier.addProduct(product3);
37         product1.addCategory(category1);
38         product2.addCategory(category2);
39         product3.addCategory(category1);
40         invoice1.addProduct(product1);
41         invoice1.addProduct(product3);
42         invoice2.addProduct(product2);
43         invoice2.addProduct(product3);
44         tx.commit();
45         session.close();

```

```
select * from INVOICE_PRODUCT;
```

Output APP.INVOICE_PRODUCT ×		
3 rows ▾		
	INVOICES_INVOICEID ▾	PRODUCTS_PRODUCTID ▾
1	7	1
2	7	3
3	8	3

```
select * from INVOICE;
```

Output APP.INVOICE ×			
2 rows ▾			
	INVOICEID ▾	INVOICENUMBER ▾	QUANTITY ▾
1	7	1	2
2	8	2	1

```
select * from PRODUCT;
```

Output APP.PRODUCT ×					
3 rows ▾					
	PRODUCTID ▾	PRODUCTNAME ▾	UNITSONSTOCK ▾	CATEGORYFK ▾	SUPPLIERFK ▾
1	1	Prod1	9	5	4
2	2	Prod2	0	6	4
3	3	Prod3	3	5	4

```

Hibernate:
    /* update
      Invoice */ update
      Invoice
    set
      invoiceNumber=?,
      quantity=?
    where
      invoiceID=?
Hibernate:
    /* update
      Invoice */ update
      Invoice
    set
      invoiceNumber=?,
      quantity=?
    where
      invoiceID=?
Hibernate:
    /* insert collection
      row Invoice.products */ insert
    into
      Invoice_Product
      (invoices_invoiceID, products_productID)
    values
      (?, ?)
Hibernate:
    /* insert collection
      row Invoice.products */ insert
    into
      Invoice_Product
      (invoices_invoiceID, products_productID)
    values
      (?, ?)
Hibernate:
    /* insert collection
      row Invoice.products */ insert
    into
      Invoice_Product
      (invoices_invoiceID, products_productID)
    values
      (?, ?)

```


Faktury w ramach produktów i produkty w ramach faktur:

```
TypedQuery<Product> prodByInv = session.createQuery( s: "from Product as product" +  
    " where :invoice member of product.invoices", Product.class);  
prodByInv.setParameter( s: "invoice", invoice1);  
for (Product product: prodByInv.getResultList()){  
    System.out.println(product.getProductName());  
}  
  
TypedQuery<Invoice> invByProd = session.createQuery( s: "from Invoice as invoice" +  
    " where :product member of invoice.products", Invoice.class);  
invByProd.setParameter( s: "product", product3);  
for (Invoice invoice: invByProd.getResultList()){  
    System.out.println(invoice.getInvoiceNumber());  
}
```

```
Hibernate:  
/*  
from  
    Product as product  
where  
    :invoice member of product.invoices */ select  
    product0_.productID as productI1_3_,  
    product0_.CategoryFK as Category4_3_,  
    product0_.productName as productN2_3_,  
    product0_.SupplierFK as Supplier5_3_,  
    product0_.unitsOnStock as unitsOnS3_3_  
from  
    Product product0_  
where  
    ? in (  
        select  
            invoices1_.invoices_invoiceID  
        from  
            Invoice_Product invoices1_  
        where  
            product0_.productID=invoices1_.products_productID  
    )  
Prod1  
Prod3  
Hibernate:  
/*  
from  
    Invoice as invoice  
where  
    :product member of invoice.products */ select  
    invoice0_.invoiceID as invoiceI1_1_,  
    invoice0_.invoiceNumber as invoiceN2_1_,  
    invoice0_.quantity as quantity3_1_  
from  
    Invoice invoice0_  
where  
    ? in (  
        select  
            products1_.products_productID  
        from  
            Invoice_Product products1_  
        where  
            invoice0_.invoiceID=products1_.invoices_invoiceID  
    )  
1  
2
```

9. JPA

Do folderu src dodajemy META-INF/persistence.xml, który wygląda następująco, prersistance-unit name="JPA_DB", więc tak będziemy się odwoływać.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0">
  <persistence-unit name="JPA_DB">
    <properties>
      <property name="hibernate.connection.driver_class"
        value="org.apache.derby.jdbc.ClientDriver"/>
      <property name="hibernate.connection.url" value="jdbc:derby://localhost/DMaichrowskiJPA"/>
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.format_sql" value="true" />
      <property name="hibernate.hbm2ddl.auto" value="create-drop" />
    </properties>
  </persistence-unit>
</persistence>
```

Dodajemy klasę MainJPA, w którym edytujemy punkt VI, dodawanie do bazy wygląda następująco:

```
1 import javax.persistence.*;
2
3 public class MainJPA{
4
5     public static void main(String[] args) {
6         Product product1 = new Product( productName: "Prod1", unitsOnStock: 10);
7         Product product2 = new Product( productName: "Prod2", unitsOnStock: 0);
8         Product product3 = new Product( productName: "Prod3", unitsOnStock: 5);
9         Supplier supplier = new Supplier( companyName: "AGH2", street: "Czarnowiejska2", city: "Krakow2 ");
10        Category category1 = new Category( name: "Cat1");
11        Category category2 = new Category( name: "Cat2");
12
13        EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "JPA_DB");
14        EntityManager em = emf.createEntityManager();
15        EntityTransaction etx = em.getTransaction();
16        etx.begin();
17        em.persist(product1);
18        em.persist(product2);
19        em.persist(product3);
20        em.persist(supplier);
21        em.persist(category1);
22        em.persist(category2);
23        supplier.addProduct(product1);
24        supplier.addProduct(product2);
25        supplier.addProduct(product3);
26        product1.addCategory(category1);
27        product2.addCategory(category2);
28        product3.addCategory(category1);
29        etx.commit();
30        em.close();
31    }
32 }
```

3 `select * from PRODUCT;`

Output APP.PRODUCT x					
	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORYFK	SUPPLIERFK
1	1	Prod1	10	5	4
2	2	Prod2	0	6	4
3	3	Prod3	5	5	4

```

Hibernate:
insert
into
    Product
    (CategoryFK, productName, SupplierFK, unitsOnStock, productID)
values
    (?, ?, ?, ?, ?)
Hibernate:
insert
into
    Product
    (CategoryFK, productName, SupplierFK, unitsOnStock, productID)
values
    (?, ?, ?, ?, ?)
Hibernate:
insert
into
    Product
    (CategoryFK, productName, SupplierFK, unitsOnStock, productID)
values
    (?, ?, ?, ?, ?)
Hibernate:
insert
into
    Supplier
    (city, companyName, street, SupplierID)
values
    (?, ?, ?, ?)
Hibernate:
insert
into
    Category
    (name, CategoryID)

```

Natomiast pobieranie kategorii i produktów następująco

```

29     etx.commit();
30
31     TypedQuery<Product> prodByCat = em.createQuery( s: "from Product as product" +
32         " where lower(product.category.name)=:categoryName", Product.class);
33     prodByCat.setParameter( s: "categoryName", v: "cat1");
34     for (Product product: prodByCat.getResultList()){
35         System.out.println(product.getProductName());
36     }
37     TypedQuery<Category> catByProd = em.createQuery( s: "from Category as category" +
38         " where :product member of category.products", Category.class);
39     catByProd.setParameter( s: "product", product1);
40     for (Category category: catByProd.getResultList()){
41         System.out.println(category.getName());
42     }
43
44     em.close();

```

```

        SupplierFK=?,
        unitsOnStock=?
    where
        productID=?
Hibernate:
    update
        Product
    set
        CategoryFK=?,
        productName=?,
        SupplierFK=?,
        unitsOnStock=?
    where
        productID=?
Hibernate:
    select
        product0_.productID as productI1_3_,
        product0_.CategoryFK as Category4_3_,
        product0_.productName as productN2_3_,
        product0_.SupplierFK as Supplier5_3_,
        product0_.unitsOnStock as unitsOnS3_3_
    from
        Product product0_,
        Category category1_
    where
        product0_.CategoryFK=category1_.CategoryID
        and lower(category1_.name)=?
Prod1
Prod3
Hibernate:
    select
        category0_.CategoryID as Category1_0_,
        category0_.name as name2_0_
    from
        Category category0_
    where
        ? in (
            select
                products1_.productID
            from
                Product products1_
            where
                category0_.CategoryID=products1_.CategoryFK
        )
Cat1

```

Możemy zaobserwować, że różnica między Hiberneta, a JPA to plik konfiguracyjny, który już nie wymaga mapowania. Natomiast w mainie, to inne nazwy metod oraz sesji, natomiast pozostała część kodu pozostaje bez zmian.

10. Kaskady

Dodajmy kaskady dla produktów oraz faktów.

```
@ManyToMany(mappedBy = "products", cascade = {CascadeType.PERSIST})  
private List<Invoice> invoices = new ArrayList<>();
```

```
@ManyToMany(cascade = {CascadeType.PERSIST})  
private List<Product> products = new ArrayList<>();
```

W mainie widzimy (w porównaniu do rozwiązania bez kaskady), że nie musimy już pisać `em.persist({produkt})`, jeżeli zrobimy to dla faktów, persystencja zostanie wykonana kaskadowo.

```
public static void main(String[] args) {  
    Product product1 = new Product( productName: "Prod1", unitsOnStock: 10);  
    Product product2 = new Product( productName: "Prod2", unitsOnStock: 0);  
    Product product3 = new Product( productName: "Prod3", unitsOnStock: 5);  
    Supplier supplier = new Supplier( companyName: "AGH2", street: "Czarnowiejska2", city: "Krakow2 ");  
    Category category1 = new Category( name: "Cat1");  
    Category category2 = new Category( name: "Cat2");  
    Invoice invoice1 = new Invoice( invoiceNumber: 1);  
    Invoice invoice2 = new Invoice( invoiceNumber: 2);  
  
    EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "JPA_DB");  
    EntityManager em = emf.createEntityManager();  
    EntityTransaction etx = em.getTransaction();  
    etx.begin();  
    invoice1.addProduct(product1);  
    invoice1.addProduct(product3);  
    invoice2.addProduct(product2);  
    invoice2.addProduct(product3);  
    em.persist(invoice1);  
    em.persist(invoice2);  
}
```

```
select * from PRODUCT;  
select * from INVOICE;  
select * from INVOICE_PRODUCT;
```

APP.PRODUCT					
PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORYFK	SUPPLIERFK	
1	3 Prod1	9	7	6	
2	4 Prod3	3	7	6	
3	5 Prod2	0	8	6	

11. Klasy wbudowane

- Klasa Address zostaje wbudowana do Supplie

```
@Embedded  
private Address address;
```

```

3  @Embeddable
4  public class Address {
5      private String street;
6      private String city;
7
8      public Address() {
9      }
10
11     public Address(String street, String city) {
12         this.street = street;
13         this.city = city;
14     }

```

```

Supplier supplier = new Supplier( companyName: "AGH2",
    new Address( street: "Czarnowiejska2", city: "Krakow2 "));

EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "JPA_DB");
EntityManager em = emf.createEntityManager();
EntityTransaction etx = em.getTransaction();
etx.begin();
em.persist(supplier);

```

```

Hibernate:
insert
into
Supplier
(city, street, companyName, SupplierID)
values
(?, ?, ?, ?)

```

```
1  select * from SUPPLIER;
```

Output APP.SUPPLIER x				
Tx: Auto				
	SUPPLIERID	CITY	STREET	COMPANYNAME
1	1	Krakow2	Czarnowiejska2	AGH2

- Dane adresowe znajdują się w tabeli dostawców

```

12  Invoice invoice2 = new Invoice( invoiceNumber: 2);
13  Supplier supplier = new Supplier( companyName: "AGH2",
14      street: "Czarnowiejska2", city: "Krakow2 ");
15
16  EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "JPA_DB");
17  EntityManager em = emf.createEntityManager();
18  EntityTransaction etx = em.getTransaction();
19  etx.begin();
20  em.persist(supplier);
21  invoice1.addProduct(product1);

```



```

4
5 @Entity
6 @SecondaryTable(name="ADDRESS_TBL")
7 public class Supplier {
8     @Id
9     @GeneratedValue(strategy = GenerationType.AUTO)
10    private int SupplierID;
11    private String companyName;
12    @Column(table="ADDRESS_TBL")
13    private String street;
14    @Column(table="ADDRESS_TBL")
15    private String city;
16
17    @OneToMany(mappedBy = "supplier")
18    private Set<Product> products = new HashSet<>();
19
20    public Supplier() {
21    }
22
23    public Supplier(String companyName, String street, String city) {
24        this.companyName = companyName;
25        this.street = street;
26        this.city = city;
27    }
28
29    public int getSupplierID() { return SupplierID; }

```

Obserwujemy logi, widzimy dodanie drugiej tabeli

Hibernate:	Hibernate:
<pre> alter table ADDRESS_TBL add constraint FKcp31om0h5hkqjoodxm6e44992 foreign key (SupplierID) references Supplier </pre>	<pre> create table ADDRESS_TBL (city varchar(255), street varchar(255), SupplierID integer not null, primary key (SupplierID)) </pre>

```

Hibernate:
insert
into
    Supplier
    (companyName, SupplierID)
values
    (?, ?)
Hibernate:
insert
into
    ADDRESS_TBL
    (city, street, SupplierID)
values
    (?, ?, ?)

```

Następnie dla potwierdzenia sprawdzamy zawartość bazy danych

```
1 select * from SUPPLIER;
```

Output APP.SUPPLIER x	
SUPPLIERID	COMPANYNAME
1	AGH2

```
5 select * from INVOICE_PRODUCT;
6 select * from ADDRESS_TBL;
```

Output APP.ADDRESS_TBL x		
CITY	STREET	SUPPLIERID
Krakow2	Czarnowiejska2	1

12. Dziedziczenie

- Jedna tabela

```
3 @Entity
4 @Inheritance(strategy = InheritanceType.SINGLE_TABLE)
5 public class Company {
6     @Id
7     @GeneratedValue(strategy = GenerationType.AUTO)
8     private int CompanyID;
9
10    private String companyName;
11    private String street;
12    private String city;
13    private String zipCode;
14
15    public Company() {
16    }
17
18    public Company(String companyName, String street, String city, String zipCode) {
19        this.companyName = companyName;
20        this.street = street;
21        this.city = city;
22        this.zipCode = zipCode;
23    }
24 }
```

```

4  @Entity
5  @DiscriminatorValue(value = "C")
6  public class Customer extends Company{
7      private float discount;
8
9      public Customer() {
10         super();
11     }
12     public Customer(float discount, String companyName, String street, String city, String zipCode) {
13         super(companyName, street, city, zipCode);
14         this.discount = discount;
15     }

```

```

4  @Entity
5  @DiscriminatorValue(value = "S")
6  public class Supplier extends Company {
7      private String bankAccountNumber;
8
9      @OneToMany(mappedBy = "supplier")
10     private Set<Product> products = new HashSet<>();
11
12     public Supplier() {
13         super();
14     }
15
16     public Supplier(String bankAccountNumber, String companyName, String street, String city, String zipCode) {
17         super(companyName, street, city, zipCode);
18         this.bankAccountNumber = bankAccountNumber;
19     }

```

```

3  public class MainJPA{
4
5      public static void testCompany(){
6          Supplier supplier1 = new Supplier( bankAccountNumber: "bankAcc1", companyName: "AGH1",
7              street: "Czarnowiejska1", city: "Krakow1", zipCode: "30-049");
8          Supplier supplier2 = new Supplier( bankAccountNumber: "bankAcc2", companyName: "AGH2",
9              street: "Czarnowiejska2", city: "Krakow2", zipCode: "30-049");
10         Customer customer1 = new Customer( discount: 0, companyName: "AGH2",
11             street: "Czarnowiejska2", city: "Krakow2", zipCode: "30-049");
12         Customer customer2 = new Customer( discount: 0, companyName: "AGH2",
13             street: "Czarnowiejska2", city: "Krakow2", zipCode: "30-049");
14
15         EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "JPA_DB");
16         EntityManager em = emf.createEntityManager();
17         EntityTransaction etx = em.getTransaction();
18         etx.begin();
19         em.persist(supplier1);
20         em.persist(supplier2);
21         em.persist(customer1);
22         em.persist(customer2);
23         etx.commit();
24         TypedQuery<Supplier> supplierQuery = em.createQuery( s: "from Supplier as supplier" +
25             " where lower(supplier.class)=:supplierClass", Supplier.class);
26         supplierQuery.setParameter( s: "supplierClass", o: "s");
27         TypedQuery<Customer> customerQuery = em.createQuery( s: "from Customer as customer" +
28             " where lower(customer.class)=:customerClass", Customer.class);
29         supplierQuery.setParameter( s: "supplierClass", o: "s");
30         customerQuery.setParameter( s: "customerClass", o: "c");
31         for (Supplier supplier: supplierQuery.getResultList()){
32             System.out.println(supplier);
33         }
34         for (Customer customer: customerQuery.getResultList()){
35             System.out.println(customer);
36         }
37
38         em.close();
39     }
40
41     public static void main(String[] args) {
42         testCompany();

```

SELECT * FROM ADDRESS_VEE;
SELECT * FROM COMPANY;

COMPANY
INVOICE
INVOICE_PRODUCT

Output APP.COMPANY X

4 rows

	DTYPE	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	DISCOUNT	BANKACCOUNTNUMBER
1	S	1	Krakow1	AGH1	Czarnowiejska1	30-049	<null>	bankAcc1
2	S	2	Krakow2	AGH2	Czarnowiejska2	30-049	<null>	bankAcc2
3	C	3	Krakow2	AGH2	Czarnowiejska2	30-049	0	<null>
4	C	4	Krakow2	AGH2	Czarnowiejska2	30-049	0	<null>

Hibernate:

```
select
    supplier0_.CompanyID as CompanyI2_1_,
    supplier0_.city as city3_1_,
    supplier0_.companyName as companyN4_1_,
    supplier0_.street as street5_1_,
    supplier0_.zipCode as zipCode6_1_,
    supplier0_.bankAccountNumber as bankAcco8_1_
from
    Company supplier0_
where
    supplier0_.DTYPE='S'
    and lower(supplier0_.DTYPE)=?
```

Supplier@3e6f3bae

Supplier@272a179c

Hibernate:

```
select
    customer0_.CompanyID as CompanyI2_1_,
    customer0_.city as city3_1_,
    customer0_.companyName as companyN4_1_,
    customer0_.street as street5_1_,
    customer0_.zipCode as zipCode6_1_,
    customer0_.discount as discount7_1_
from
    Company customer0_
where
    customer0_.DTYPE='C'
    and lower(customer0_.DTYPE)=?
```

Customer@7c2a69b4

Customer@375b5b7f

- Tabele Łączone

(W stosunku do joinów, zmiana strategy w Company, oraz usunięcie DiscriminatorValue)

```
2
3
4
5
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Company {
    @Id
```

```
@Entity
public class Supplier extends Company {
    private float discount;
```

```
@Entity
public class Customer extends Company{
    private float discount;
```


Main dodający oraz wyciągający rekordy z bazy.

```
41 public static void testCompany2(){
42     Supplier supplier1 = new Supplier( bankAccountNumber: "bankAcc1", companyName: "AGH1",
43     street: "Czarnowiejska1", city: "Krakow1", zipCode: "30-049");
44     Supplier supplier2 = new Supplier( bankAccountNumber: "bankAcc2", companyName: "AGH2",
45     street: "Czarnowiejska2", city: "Krakow2", zipCode: "30-049");
46     Customer customer1 = new Customer( discount: 0, companyName: "AGH2",
47     street: "Czarnowiejska2", city: "Krakow2", zipCode: "30-049");
48     Customer customer2 = new Customer( discount: 0, companyName: "AGH2",
49     street: "Czarnowiejska2", city: "Krakow2", zipCode: "30-049");
50
51     EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "JPA_DB");
52     EntityManager em = emf.createEntityManager();
53     EntityTransaction etx = em.getTransaction();
54     etx.begin();
55     em.persist(supplier1);
56     em.persist(supplier2);
57     em.persist(customer1);
58     em.persist(customer2);
59     etx.commit();
60     TypedQuery<Supplier> supplierQuery = em.createQuery( s: "from Supplier as supplier", Supplier.class);
61     TypedQuery<Customer> customerQuery = em.createQuery( s: "from Customer as customer", Customer.class);
62     for (Supplier supplier: supplierQuery.getResultList()){
63         System.out.println(supplier);
64     }
65     for (Customer customer: customerQuery.getResultList()){
66         System.out.println(customer);
67     }
68
69     em.close();
70 }
71
72 public static void main(String[] args) {
73     testCompany2();
74 }
```

SELECT * from CUSTOMER;

Output APP.CUSTOMER X

	DISCOUNT	COMPANYID
1	0	3
2	0	4

6

select * from SUPPLIER;

Output APP.SUPPLIER X

	BANKACCOUNTNUMBER	COMPANYID
1	bankAcc1	1
2	bankAcc2	2

SELECT * from COMPANY;

Output APP.COMPANY x

4 rows

	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE
1	1	Krakow1	AGH1	Czarnowiejska1	30-049
2	2	Krakow2	AGH2	Czarnowiejska2	30-049
3	3	Krakow2	AGH2	Czarnowiejska2	30-049
4	4	Krakow2	AGH2	Czarnowiejska2	30-049

```
(?, ?)
Hibernate:
select
    supplier0_.CompanyID as CompanyI1_1_,
    supplier0_1.city as city2_1_,
    supplier0_1.companyName as companyN3_1_,
    supplier0_1.street as street4_1_,
    supplier0_1.zipCode as zipCode5_1_,
    supplier0_.bankAccountNumber as bankAcco1_6_
from
    Supplier supplier0_
inner join
    Company supplier0_1_
        on supplier0_.CompanyID=supplier0_1_.CompanyID
Supplier@31ff1390
Supplier@28cb9120
Hibernate:
select
    customer0_.CompanyID as CompanyI1_1_,
    customer0_1.city as city2_1_,
    customer0_1.companyName as companyN3_1_,
    customer0_1.street as street4_1_,
    customer0_1.zipCode as zipCode5_1_,
    customer0_.discount as discount1_2_
from
    Customer customer0_
inner join
    Company customer0_1_
        on customer0_.CompanyID=customer0_1_.CompanyID
Customer@25c5e994
Customer@69b2f8e5
```

- Tabela na klasę
- (Jedyna zmiana to zmiana strategii w stosunku do tabeli łączonych)

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Company {
```


(Main bez zmian w stosunku do tabeli łączonych)

```
7 select * from CUSTOMER;
8
```

	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	DISCOUNT
1	3	Krakow2	AGH2	Czarnowiejska2	30-049	0
2	4	Krakow2	AGH2	Czarnowiejska2	30-049	0

```
select * from SUPPLIER;
```

	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	BANKACCOUNTNUMBER
1	1	Krakow1	AGH1	Czarnowiejska1	30-049	bankAcc1
2	2	Krakow2	AGH2	Czarnowiejska2	30-049	bankAcc2

Tym samym wszystkie zadania laboratoryjne, jako zadanie domowe tworzymy aplikację Webową do zamawiania produktów.

Aplikacja do zamawiania produktów

Opis aplikacji:

W aplikacji mamy możliwość tworzenia zamówień, na podstawie produktów dostępnych w bazie (nie możemy ich dodawać). W danym zamówieniu możemy zamówić dla każdego produktu od 0 do ilości produktów na stanie. W każdym zamówieniu musi znajdować się co najmniej jeden zamówiony produkt.

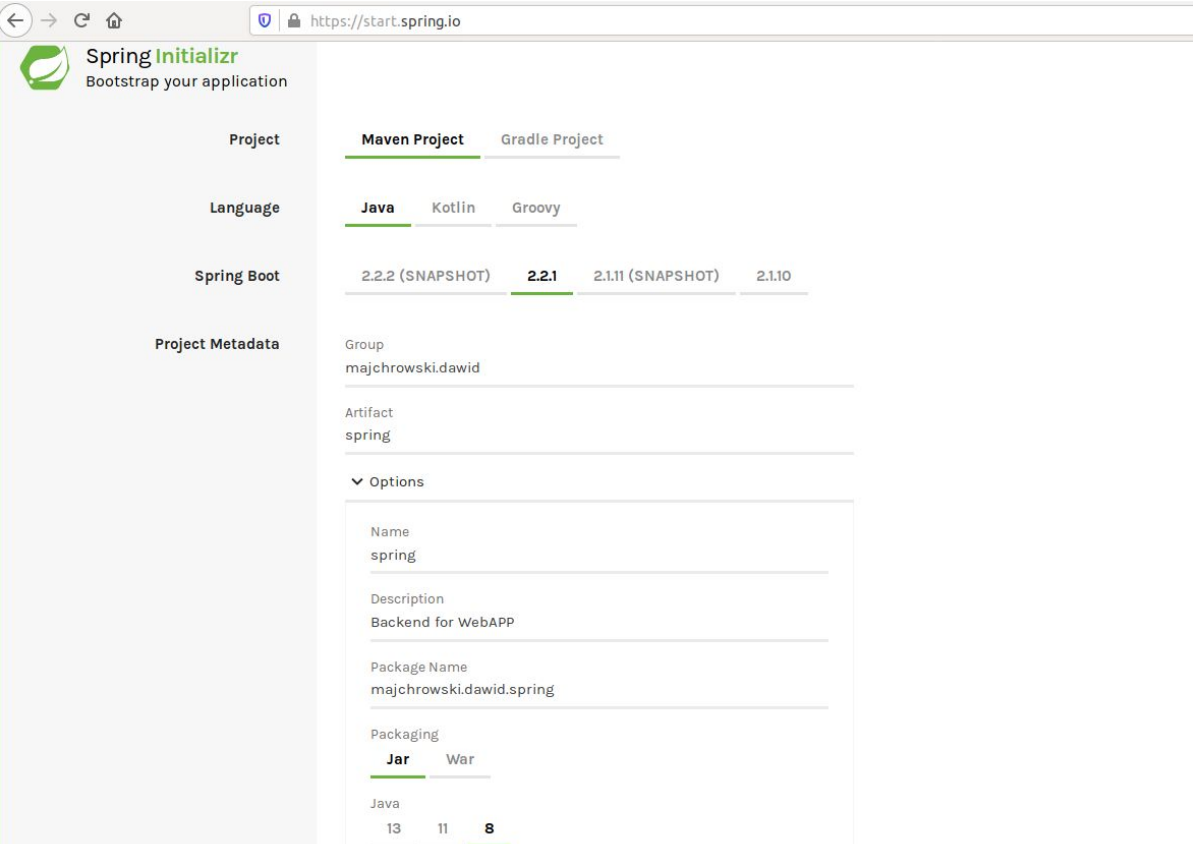
Aplikacja Webowa

- Backend (Spring Boot + Maven + JPA + DerbyDB)
- Frontend (Angular)

Backend:

Odpalamy serwer Derby (tak samo jak na labach)

Tworzymy projekt Spring Boot, korzystając z narzędzia Spring Initializr, jako manager projektu wybieramy Mavena, oraz nie specyfikujemy żadnych dodatkowych zależności.



The screenshot shows the Spring Initializr web application interface. The browser address bar displays <https://start.spring.io>. The application has a sidebar on the left with the following sections:

- Project**: Maven Project (selected), Gradle Project
- Language**: Java (selected), Kotlin, Groovy
- Spring Boot**: 2.2.2 (SNAPSHOT), 2.2.1 (selected), 2.1.11 (SNAPSHOT), 2.1.10
- Project Metadata**: Group (majchrowski.dawid), Artifact (spring)

The main content area is titled "Options" and contains the following fields:

- Name**: spring
- Description**: Backend for WebAPP
- Package Name**: majchrowski.dawid.spring
- Packaging**: Jar (selected), War
- Java**: 13, 11, 8 (selected)

Po czym wypakowujemy projekt w odpowiednie miejsce.

Oprócz już załadowanych zależności, będziemy potrzebować spring-web do obsługi restowych zapytań, jpa do obsługi bazy danych oraz derbyclienta do połączenia.

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>

  <dependency>
    <groupId>org.apache.derby</groupId>
    <artifactId>derbyclient</artifactId>
    <version>10.14.2.0</version>
  </dependency>
</dependencies>
```

Po ściągnięciu zależności w pliku src/main/resources/application.properties dodajemy konfigurację bazy danych.

```
1  spring.datasource.url = jdbc:derby://localhost/DMajchrowskiJPA
2
3  spring.jpa.show-sql = true
4  spring.jpa.hibernate.ddl-auto = update
5  spring.datasource.driver-class-name=org.apache.derby.jdbc.ClientDriver
6
```

Na początku dodajemy model bazy danych. Będziemy potrzebowali faktury i klientów z relacją wiele do wielu. Różnica w stosunku do laboratorium jest taka, że teraz na każdej fakturze może być kilka tych samych produktów, zatem pole "quantity" będziemy modelować w tabeli łącznikowej. Zatem relację wiele do wielu rozbijemy na 2 relacje jeden do wielu do strony tabeli łącznikowej z dodatkowym polem ilościowym. Model wygląda następująco.

```
package majchrowski.dawid.spring.bean;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Invoice {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int invoiceID;

    public Invoice() {
    }

    public void setInvoiceID(int invoiceID) { this.invoiceID = invoiceID; }

    public int getInvoiceID() { return invoiceID; }
```

```

4
5  @Entity
6  public class OrderDetails {
7
8      @Id
9      @GeneratedValue(strategy = GenerationType.AUTO)
10     private int id;
11
12     @ManyToOne(fetch = FetchType.LAZY)
13     private Invoice invoice;
14
15     @ManyToOne(fetch = FetchType.LAZY)
16     private Product product;
17
18     private int quantity;
19
20     @
21     public OrderDetails() {
22     }
23
24     @
25     public OrderDetails(Invoice invoice, Product product, int quantity) {
26         this.invoice = invoice;
27         this.product = product;
28         this.quantity = quantity;
29     }

```

```

7
8  @Entity
9  public class Product {
10
11     @Id
12     @GeneratedValue(strategy = GenerationType.AUTO)
13     private int productID;
14
15     private String productName;
16     private Integer unitsOnStock;
17     private Double unitPrice;
18
19     @
20     public Product(String productName, Integer unitsOnStock, Double unitPrice) {
21         this.productName = productName;
22         this.unitsOnStock = unitsOnStock;
23         this.unitPrice = unitPrice;
24     }
25
26     @
27     public Product() {
28     }
29

```

Mając model każdej tabeli tworzymy repozytoria.

```

public interface InvoiceRepository extends JpaRepository<Invoice, Integer> {
}

public interface OrderDetailsRepository extends JpaRepository<OrderDetails, Integer> {
}

```

```
public interface ProductRepository extends JpaRepository<Product, Integer> {
}
```

W głównej części aplikacji dodajemy dodawanie produktów do bazy danych przy uruchomieniu aplikacji. (symulujemy istnienie produktów w bazie danych)

```
10
11 @SpringBootApplication
12 public class Application {
13
14     @Autowired
15     private ProductRepository productRepository;
16
17     public static void main(String[] args) { SpringApplication.run(Application.class, args); }
18
19     @Bean
20     InitializingBean sendDatabase() {
21         return () -> {
22             productRepository.save(new Product( productName: "Product1", unitsOnStock: 1, unitPrice: 23.5));
23             productRepository.save(new Product( productName: "Product2", unitsOnStock: 15, unitPrice: 1.));
24             productRepository.save(new Product( productName: "Product3", unitsOnStock: 0, unitPrice: 1500.));
25             productRepository.save(new Product( productName: "Product4", unitsOnStock: 5, unitPrice: 325.));
26             productRepository.save(new Product( productName: "Product5", unitsOnStock: 3, unitPrice: 11.));
27             productRepository.save(new Product( productName: "Product6", unitsOnStock: 100, unitPrice: 25.));
28         };
29     }
30 }
31
32
33
34
```

Dodajemy jedną klasę pomocniczą, która będzie potrzebna do tworzenia Restowych kontrolerów.

```
public class ProductOrder {
    private int productID;
    private int quantity;

    public int getProductID() { return productID; }

    public int getQuantity() { return quantity; }
}
```

Tworzymy 2 kontrolery, do obsługi produktów oraz zamówień. Dla zamówień potrzebujemy jedynie metodę get, zatem będzie on dość prosty.

```
14 @RestController
15 public class ProductController {
16
17     @Autowired
18     ProductRepository productRepository;
19
20     @CrossOrigin
21     @RequestMapping(
22         method = RequestMethod.GET,
23         path = "/products",
24         produces = MediaType.APPLICATION_JSON_VALUE
25     )
26     public List<Product> getAllProducts() { return productRepository.findAll(); }
27
28 }
29
```

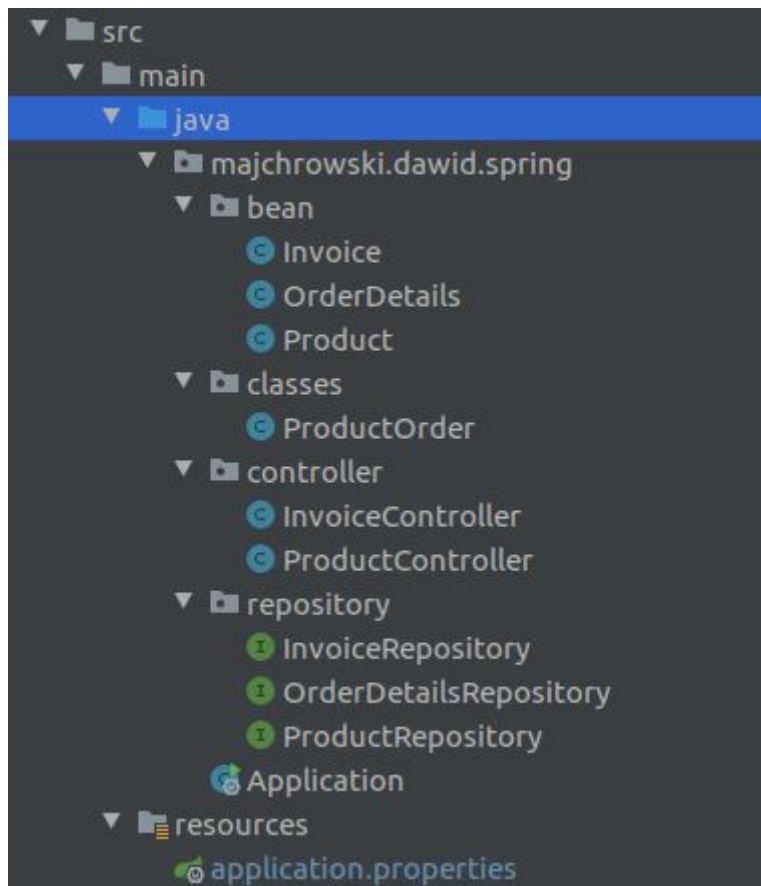
Korzystamy ze springowego restowego kontrolera, oraz zwracamy listę wszystkich produktów, dla zapytania GET /products.

Dla kontrolera Zamówień będziemy potrzebować 2 metody get oraz post oraz wszystkie 3 repozytoria oraz wcześniej zdefiniowaną klasę pomocniczą.

```
16
17 @RestController
18 public class InvoiceController {
19
20     @Autowired
21     InvoiceRepository invoiceRepository;
22
23     @Autowired
24     OrderDetailsRepository orderDetailsRepository;
25
26     @Autowired
27     ProductRepository productRepository;
28
29     @CrossOrigin
30     @RequestMapping(
31         method = RequestMethod.GET,
32         path = "/orders",
33         produces = MediaType.APPLICATION_JSON_VALUE
34     )
35     public List<Invoice> getAllInvoices() { return invoiceRepository.findAll(); }
36
37
38     @CrossOrigin
39     @RequestMapping(
40         method = RequestMethod.POST,
41         path = "/orders",
42         produces = MediaType.APPLICATION_JSON_VALUE,
43         consumes = MediaType.APPLICATION_JSON_VALUE
44     )
45     @PostMapping
46     public Invoice addInvoice(@RequestBody List<ProductOrder> productOrders) {
47         Invoice invoice = invoiceRepository.save(new Invoice());
48         for (ProductOrder productOrder : productOrders) {
49             int quantity = productOrder.getQuantity();
50             Product product = productRepository.getOne(productOrder.getProductID());
51             product.setUnitsOnStock(product.getUnitsOnStock() - quantity);
52             OrderDetails orderDetails = new OrderDetails(invoice, product, quantity);
53             orderDetailsRepository.save(orderDetails);
54         }
55         return invoice;
56     }
57 }
```

Metoda get niczym się nie różni, natomiast w metodzie post, przyjmujemy zamówienie w postaci listy {productID, ilość} oraz odpowiednio dodajemy wszystkie zamówienia do bazy.

Tym samym kończymy definiowanie strony backendowej. Struktura całego springowego projektu wygląda następująco



Zajmiemy się teraz frontendem. Strona zostanie wykonana w angularze oraz zostaną użyte niektóre części wspólne z ćwiczenia z EF.

Frontend:

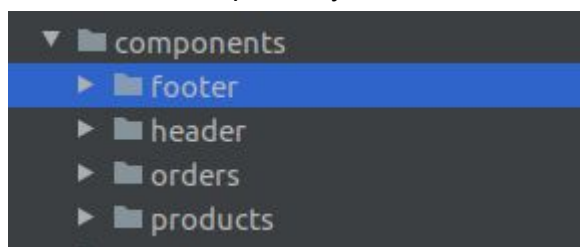
Będziemy potrzebować 2 głównych klas (produktów oraz zamówień)

```
1 export class Product {  
2     productID: number;  
3     productName: string;  
4     unitsOnStock: number;  
5     unitPrice: number;  
6     selectedUnit?: number;  
7 }  
8
```

```
1 export class Order {  
2     invoiceID: number;  
3 }
```

W polu zamówienia jest też opcjonalne pole selectedUnit, które odpowiada za ilość danego produktu, który chcemy zamówić.

Prosta struktura komponentów (footer i header z EF), oraz komponent odpowiedzialny za zamówiania oraz produkty.



Zamówienia (tabela faktur oraz przełączanie widoku po naciśnięciu przycisku “New Order”)

```
orders.component.html x
1 <div>
2   <app-header [title]="title"></app-header>
3   <div *ngIf="view === 0">
4     <div *ngIf="orders" class="order-div">
5       <table mat-table [dataSource]="orders" class="mat-elevation-z8">
6         <!-- Name Column -->
7         <ng-container matColumnDef="invoiceID">
8           <th mat-header-cell *matHeaderCellDef>Invoice Number</th>
9           <td mat-cell *matCellDef="let element"> {{element.invoiceID}} </td>
10        </ng-container>
11        <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
12        <tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
13      </table>
14      <mat-toolbar-row>
15        <span class="fill-remaining-space"></span>
16        <button mat-raised-button color="primary" (click)="onNewOrderClick()">New Order</button>
17        <span class="fill-remaining-space"></span>
18      </mat-toolbar-row>
19    </div>
20  </div>
21  <app-products *ngIf="view !== 0" (invoiceEmitter)="addInvoice($event)"></app-products>
22 </div>
23
```

```
export class OrdersComponent implements OnInit {
  orders: Order[];
  displayedColumns: string[] = ['invoiceID'];
  title = 'Your Orders';
  view = 0;

  constructor(private orderService: OrderService) {}

  ngOnInit() {
    this.getOrders();
  }

  private getOrders() {
    this.orderService.getOrders().subscribe(
      next: orders => this.orders = orders
    );
  }

  private onNewOrderClick() {
    this.title = 'New Order';
    this.view = 1;
  }

  addInvoice(order: Order) {
    if (order) {
      this.orders.push(order);
    }
    this.title = 'Your Orders';
    this.view = 0;
  }
}
```

Component produktów, wyświetla listę wszystkich produktów pobranych z bazy danych. Jeżeli liczba produktów na stanie jest 0 to wyłącza możliwość zamówienia produktu. Pozwala na złożenie zamówienia w przypadku wybrania co najmniej 1 produktu. Pozwala też powrócić do widoku zamówień.

```
products.component.html
1 <div *ngIf="products" class="product-div">
2   <table mat-table [dataSource]="products" class="mat-elevation-z8">
3
4     <!-- Position Column -->
5     <ng-container matColumnDef="ID">
6       <th mat-header-cell *matHeaderCellDef> Product ID.</th>
7       <td mat-cell *matCellDef="let element"> {{element.productId}} </td>
8     </ng-container>
9
10    <!-- Name Column -->
11    <ng-container matColumnDef="name">
12      <th mat-header-cell *matHeaderCellDef> Name</th>
13      <td mat-cell *matCellDef="let element"> {{element.productName}} </td>
14    </ng-container>
15
16    <!-- Unit Column -->
17    <ng-container matColumnDef="unitsOnStock">
18      <th mat-header-cell *matHeaderCellDef> Number of Unit</th>
19      <td mat-cell *matCellDef="let element">
20        <mat-form-field>
21          <mat-select panelClass="example-panel-dark-blue" [(value)]="element.selectedUnit"
22            [disabled]="!unitExists(element)">
23            <mat-option *ngFor="let unit of createArray(element.unitsOnStock)"
24              [value]="unit"> {{unit}}
25          </mat-option>
26        </mat-select>
27      </mat-form-field>
28    </td>
29  </ng-container>
30
31  <!-- Price Column -->
32  <ng-container matColumnDef="unitPrice">
33    <th mat-header-cell *matHeaderCellDef> Unit Price</th>
34    <td mat-cell *matCellDef="let element">{{element.unitPrice}} </td>
35  </ng-container>
36
37  <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
38  <tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
39 </table>
40 <mat-toolbar-row>
41   <span class="fill-remaining-space"></span>
42   <button mat-raised-button color="primary" [disabled]="!correctOrder()" (click)="placeOrder()">Place Order</button>
43   <span class="margin"></span>
44   <button mat-raised-button color="warn" (click)="cancelOrder()">Cancel</button>
45   <span class="fill-remaining-space"></span>
46 </mat-toolbar-row>
47 </div>
```

```
products.component.html × products.component.ts ×
1 import {Component, EventEmitter, OnInit, Output} from '@angular/core';
2 import {Product} from '../classes/product';
3 import {ProductService} from '../services/product.service';
4 import {Order} from '../classes/order';
5 import {OrderService} from '../services/order.service';
6
7 @Component({
8   selector: 'app-products',
9   templateUrl: './products.component.html',
10  styles: []
11 })
12 export class ProductsComponent implements OnInit {
13
14   products: Product[];
15   displayedColumns: string[] = ['ID', 'name', 'unitsOnStock', 'unitPrice'];
16
17   @Output() invoiceEmitter = new EventEmitter<Order>();
18
19   constructor(
20     private productService: ProductService,
21     private orderService: OrderService
22   ) {
23   }
24
25   createArray(n: number) {
26     return [...Array(n + 1).keys()];
27   }
28
29   ngOnInit() {
30     this.getOrders();
31   }
32 }
```

```
33   private unitExists(product: Product): boolean {
34     return product.unitsOnStock > 0;
35   }
36
37   private correctOrder(): boolean {
38     return this.products.some(product => product.selectedUnit > 0);
39   }
40
41   private placeOrder() {
42     const preparedProducts = [];
43     this.products.filter(product => product.selectedUnit > 0).forEach(product => {
44       preparedProducts.push({
45         productID: product.productID,
46         quantity: product.selectedUnit
47       });
48     });
49     this.orderService.postInvoice(preparedProducts).subscribe( next result => {
50       this.invoiceEmitter.emit(result);
51     }, error: error => {
52       alert('Failed to place order');
53       this.invoiceEmitter.emit( value: null);
54     });
55   }
56
57   private cancelOrder() {
58     this.invoiceEmitter.emit( value: null);
59   }
60
61   private getOrders() {
62     this.productService.getProducts().subscribe( next products => {
63       this.products = products;
64       this.products.forEach(product => product.selectedUnit = 0);
65     });
66   }
67 }
```

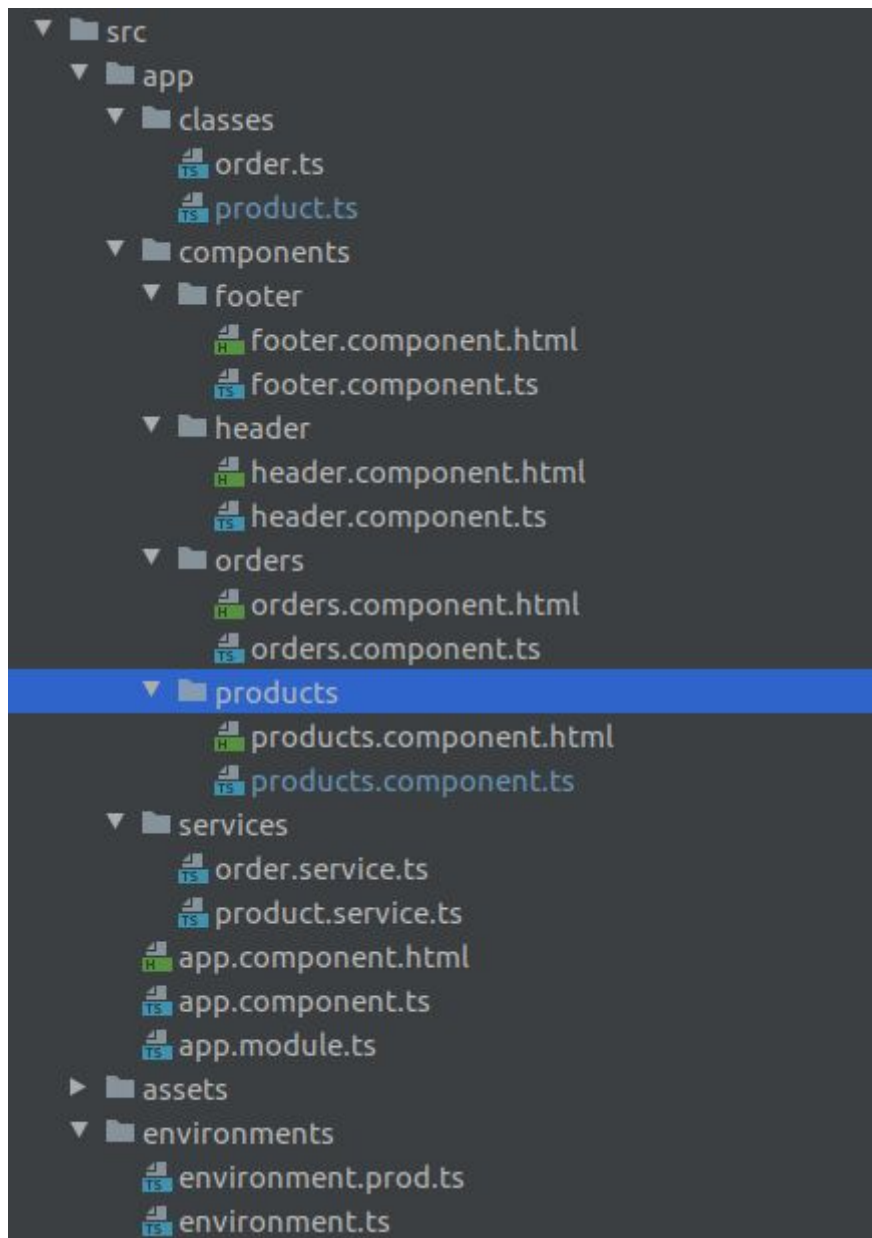
Ostatnie 2 klasy typescriptowe to OrderService oraz ProductService, wykonują zapytania get/post na url ustawiony w zmiennych środowiskowych (environment.ts). Zwracają Observable, które są wykorzystywane przy inicjowaniu komponentów korzystających z owych serwisów.

```
export const environment = {  
  production: false,  
  backendURL: 'http://localhost:8080'  
};  
/*
```

```
7  @Injectable({  
8    providedIn: 'root'  
9  })  
10  
11  export class OrderService {  
12    readonly rootURL = `${environment.backendURL}/orders`;  
13  
14    constructor(private http: HttpClient) {  
15    }  
16  
17    public getOrders(): Observable<Order[]> {  
18      return this.http.get<Order[]>(this.rootURL);  
19    }  
20  
21    public postInvoice(data): Observable<Order> {  
22      console.log(data);  
23      return this.http.post<Order>(this.rootURL, data);  
24    }  
25  }  
26
```

```
7  
8  @Injectable({  
9    providedIn: 'root'  
10  })  
11  export class ProductService {  
12    readonly rootURL = `${environment.backendURL}/products`;  
13  
14    constructor(private http: HttpClient) {  
15    }  
16  
17    public getProducts(): Observable<Product[]> {  
18      return this.http.get<Product[]>(this.rootURL);  
19    }  
20  
21  }  
22
```


Struktura frontendu:



Działanie aplikacji:

Odpalamy backend (serwer derby chodzi oraz baza danych jest stworzona):

```

Hibernate: create table invoice (invoiceid integer not null, primary key (invoiceid))
Hibernate: create table order_details (id integer not null, quantity integer not null, invoice_invoiceid integer, product_productid integer, primary key (id))
Hibernate: create table product (productid integer not null, product_name varchar(255), unit_price double, units_on_stock integer, primary key (productid))
Hibernate: alter table order_details add constraint FKt1y5wu0g9ddd6q97p7ec3hhoh foreign key (invoice_invoiceid) references invoice
Hibernate: alter table order_details add constraint FKt364hvs03sod3e288tbk63rk foreign key (product_productid) references product

```

```

Hibernate: values next value for hibernate_sequence
Hibernate: insert into product (product_name, unit_price, units_on_stock, productid) values (?, ?, ?, ?)
Hibernate: values next value for hibernate_sequence
Hibernate: insert into product (product_name, unit_price, units_on_stock, productid) values (?, ?, ?, ?)
Hibernate: values next value for hibernate_sequence
Hibernate: insert into product (product_name, unit_price, units_on_stock, productid) values (?, ?, ?, ?)
Hibernate: values next value for hibernate_sequence
Hibernate: insert into product (product_name, unit_price, units_on_stock, productid) values (?, ?, ?, ?)
Hibernate: values next value for hibernate_sequence
Hibernate: insert into product (product_name, unit_price, units_on stock, productid) values (?, ?, ?, ?)

```


5 `select * from product;`

Output APP.INVOICE x APP.PRODUCT x

6 rows Tx: Auto DDL

	PRODUCTID	PRODUCT_NAME	UNIT PRICE	UNITS ON STOCK
1	7	Product1	23.5	1
2	8	Product2	1	15
3	9	Product3	1500	0
4	10	Product4	325	5
5	11	Product5	11	3
6	12	Product6	25	100

6 `select * from ORDER_DETAILS;`

Output APP.INVOICE x APP.ORDER_DETAILS x

0 rows Tx: Auto DDL

ID	QUANTITY	INVOICE_INVOICEID	PRODUCT_PRODUCTID
----	----------	-------------------	-------------------

7 `select * from invoice;`

8

Output APP.INVOICE x APP.INVOICE 2 x

0 rows Tx: Auto DDL

INVOICEID

Stan początkowy aplikacji, stworzone 3 tabele, oraz tabela produkty wypełniona 6 produktami, bez żadnych faktur. Odpalamy frontend.

Your Orders

Invoice Number

New Order

David Majchrowski 2019

Strona startowa bez żadnych faktur, klikamy "New Order".

New Order

Product ID	Name	Number of Item	Unit Price
7	Product1	0	23.5
8	Product2	0	1
9	Product3	0	1500
10	Product4	0	325
11	Product5	0	11
12	Product6	0	25

Place Order Cancel

David Majchrowski 2019

Obserwujemy brak możliwości złożenia zamówienia, gdyż nie wybraliśmy żadnego produktu oraz zablokowaną możliwość zamówienia produktu nr 3 (0 produktów na stanie). Dodajemy zamówienie, żeby wyglądało następująco

New Order

Product ID	Name	Number of Stock	Unit Price
7	Product1	1	23.5
8	Product2	2	1
9	Product3	5	1500
10	Product4	4	325
11	Product5	2	11
12	Product6	3	25

Place Order

Cancel

Dawid Majchrowski 2019

Widzimy, że mamy możliwość złożenia zamówienia, więc klikamy “Place Order”.
Rezultat na frontendzie:

Your Orders

Invoice Number

13

New Order

Dawid Majchrowski 2019

Rezultat na backendzie:

```
Hibernate: select product0_productid as product11_2_, product0_product_name as product_2_2_, product0_unit_price as unit_pri3_2_0_, product0_units_on_stock as units_on4_2_0_ from product product0_
Hibernate: values next value for hibernate_sequence
Hibernate: insert into invoice (invoiceid) values (?)
Hibernate: select product0_productid as product11_2_0_, product0_product_name as product_2_2_0_, product0_unit_price as unit_pri3_2_0_, product0_units_on_stock as units_on4_2_0_ from product product0_ where product0_productid=7
Hibernate: values next value for hibernate_sequence
Hibernate: insert into order_details (invoice_invoiceid, product_productid, quantity, id) values (?, ?, ?, ?)
Hibernate: update product set product_name=?, unit_price=?, units_on_stock=? where productid=?
Hibernate: select product0_productid as product11_2_0_, product0_product_name as product_2_2_0_, product0_unit_price as unit_pri3_2_0_, product0_units_on_stock as units_on4_2_0_ from product product0_ where product0_productid=7
Hibernate: values next value for hibernate_sequence
Hibernate: insert into order_details (invoice_invoiceid, product_productid, quantity, id) values (?, ?, ?, ?)
Hibernate: update product set product_name=?, unit_price=?, units_on_stock=? where productid=?
Hibernate: select product0_productid as product11_2_0_, product0_product_name as product_2_2_0_, product0_unit_price as unit_pri3_2_0_, product0_units_on_stock as units_on4_2_0_ from product product0_ where product0_productid=7
Hibernate: values next value for hibernate_sequence
Hibernate: insert into order_details (invoice_invoiceid, product_productid, quantity, id) values (?, ?, ?, ?)
Hibernate: update product set product_name=?, unit_price=?, units_on_stock=? where productid=?
Hibernate: select product0_productid as product11_2_0_, product0_product_name as product_2_2_0_, product0_unit_price as unit_pri3_2_0_, product0_units_on_stock as units_on4_2_0_ from product product0_ where product0_productid=7
Hibernate: values next value for hibernate_sequence
Hibernate: insert into order_details (invoice_invoiceid, product_productid, quantity, id) values (?, ?, ?, ?)
Hibernate: update product set product_name=?, unit_price=?, units_on_stock=? where productid=?
```

select * from invoice;

Output

APP.INVOICE ×

APP.INVOICE 2 ×

1 row

INVOICEID

113

6

7

8

select * from ORDER_DETAILS;

select * from invoice;

Output

APP.INVOICE ×

APP.ORDER_DETAILS ×

5 rows

ID

QUANTITY

INVOICE_INVOICEID

PRODUCT_PRODUCTID

1

14

1

13

7

2

15

3

13

8

3

16

4

13

10

4

17

2

13

11

5

18

3

13

12

```

5 select * from product;
6 select * from APP.INVOICE;

```

Output APP.INVOICE x APP.PRODUCT x				
Tx: Auto ✓ DDL				
	PRODUCTID	PRODUCT_NAME	UNIT PRICE	UNITS_ON_STOCK
1	7	Product1	23.5	0
2	8	Product2	1	12
3	9	Product3	1500	0
4	10	Product4	325	1
5	11	Product5	11	1
6	12	Product6	25	97

Widzimy, że dane zostają poprawnie dodane do bazy. (Dodanie faktury, danych do tabeli łącznikowej oraz zmiana pola UNITS_ON_STOCK w bazie)

Kilkamy ponownie “New Order” na frontendzie

New Order			
Product ID	Name	Number of Unit	Unit Price
7	Product1	<input type="text" value="0"/>	23.5
8	Product2	<input type="text" value="0"/>	1
9	Product3	<input type="text" value="0"/>	1500
10	Product4	<input type="text" value="0"/>	325
11	Product5	<input type="text" value="0"/>	11
12	Product6	<input type="text" value="0"/>	25
		<input type="button" value="Place Order"/>	<input type="button" value="Cancel"/>

Dawid Majchrowski 2019

Obserwujemy zablokowany Produkt1, ze względu na brak jego dostępności w bazie.

Dodajemy jeszcze 2 zamówienia, tak aby pozbyć się wszystkich produktów z magazynu oraz obserwujemy stan aplikacji.

Frontend: (3 faktury, brak możliwości złożenia zamówienia, ze względu na brak produktów)

Your Orders			
Invoice Number			
13			
19			
24			
		<input type="button" value="New Order"/>	

Dawid Majchrowski 2019

New Order			
Product ID	Name	Number of Unit	Unit Price
7	Product1	<input type="text" value="0"/>	23.5
8	Product2	<input type="text" value="0"/>	1
9	Product3	<input type="text" value="0"/>	1500
10	Product4	<input type="text" value="0"/>	325
11	Product5	<input type="text" value="0"/>	11
12	Product6	<input type="text" value="0"/>	25
		<input type="button" value="Place Order"/>	<input type="button" value="Cancel"/>

Dawid Majchrowski 2019

Backend: (0 produktów w magazynie, 3 faktury oraz poprawna tabela łącznikowa)

```
5 select * from product;
```

Output APP.INVOICE × APP.PRODUCT ×				
	PRODUCTID	PRODUCT_NAME	UNIT_PRICE	UNITS_ON_STOCK
1	7	Product1	23.5	0
2	8	Product2	1	0
3	9	Product3	1500	0
4	10	Product4	325	0
5	11	Product5	11	0
6	12	Product6	25	0

```
7 select * from invoice;
```

Output APP.INVOICE × APP.INVOICE 2 ×	
	INVOICEID
1	13
2	19
3	24

```
6 select * from ORDER DETAILS;
```

Output APP.INVOICE × APP.ORDER_DETAILS ×				
	ID	QUANTITY	INVOICE_INVOICEID	PRODUCT_PRODUCTID
1	14	1	13	7
2	15	3	13	8
3	16	4	13	10
4	17	2	13	11
5	18	3	13	12
6	20	11	19	8
7	21	1	19	10
8	22	1	19	11
9	23	96	19	12
10	25	1	24	8
11	26	1	24	12

Wszystkie założone funkcjonalności co do aplikacji zostały spełnione.

Kod projektu: <https://github.com/majchrow/Databases/tree/master/assignment3/WebAPP>