

Dokumentacja bazy danych

Skład grupy: Krzysztof Gołuchowski i Krystian Sienkiewicz

Temat: Prosty sklep internetowy z suplementami dla sportowców

SZBD: PostgreSQL

Technologia: Java/Hibernate

Opis projektu

Zaprojektowaliśmy system do obsługi sklepu internetowego. Funkcjonalność obejmują takie operacje jak:

- rejestracja oraz logowanie użytkowników
- przeglądanie zawartości sklepu
- filtrowanie zawartości sklepu wobec kategorii produktów
- zarządzanie zawartością koszyka przez użytkownika (kontrola zasobów)
- złożenie zamówienia przez użytkownika

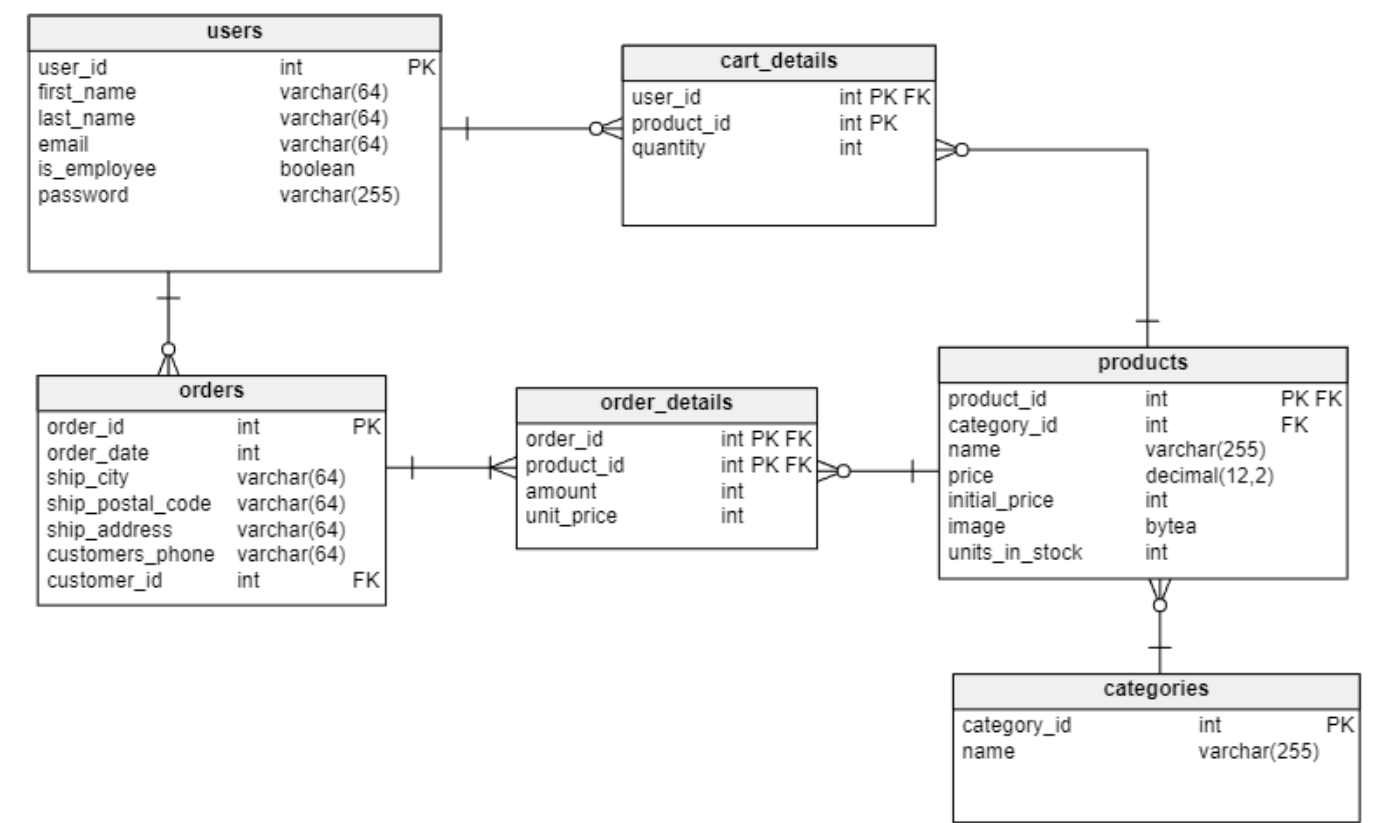
Dla zalogowanego użytkownika, który posiada uprawnienia pracownika:

- możliwość edycji produktów w bazie (zmiana ceny, wprowadzenie promocji)
- możliwość dodania nowego produktu do bazy
- wygenerowanie raportu ze sprzedaży dla poprzednich lat i miesięcy

Synchronizację oraz transakcyjność operacji zaimplementowaliśmy na przykładzie składania zamówień przez użytkownika.

Frontend stworzyliśmy w JavaScript, przy użyciu framework'u React.

Database diagram



Data Model

Users table

```
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "user_id")
    private Long id;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "email")
    private String email;

    @Column(name = "is_employee")
    private Boolean isEmployee;

    @Column(name = "password")
    private String password;
}
```

Products table

```
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "product_id")
    private Long id;

    @ManyToOne
    @JoinColumn(name = "category_id", referencedColumnName = "category_id")
    private Category category;

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

    @Column(name = "price")
    private double price;

    @Column(name = "initial_price")
    private double initialPrice;

    @Column(name = "image")
    private byte[] image;

    @Column(name = "units_in_stock")
    private int unitsInStock;

    public void removeUnitsInStock(int quantity) {
        unitsInStock -= quantity;
    }
}
```

Order table

```
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "order_id")
    private Long orderId;

    @Column(name = "orderdate")
    private Date orderDate;

    @Column(name = "shipcity")
    private String shipCity;

    @Column(name = "shippostalcode")
    private String shipPostalCode;

    @Column(name = "shipaddress")
    private String shipAddress;

    @Column(name = "customersphone")
```

```
    private String customersPhone;

    @ManyToOne
    @JoinColumn(name = "customer_id", referencedColumnName = "user_id")
    private User customer;
}
```

Order Details table

```
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "order_id")
    private Long orderId;

    @Column(name = "orderdate")
    private Date orderDate;

    @Column(name = "shipcity")
    private String shipCity;

    @Column(name = "shippostalcode")
    private String shipPostalCode;

    @Column(name = "shipaddress")
    private String shipAddress;

    @Column(name = "customersphone")
    private String customersPhone;

    @Column(name = "customer_id")
    private Long customerId;
}
```

Category table

```
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "category_id")
    private Long id;

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

Cart Details table

```
public class CartDetails {  
    @EmbeddedId  
    private CartDetailsId cartId;  
  
    @ManyToOne  
    @MapsId("userId")  
    @JoinColumn(name = "user_id")  
    private User user;  
  
    @ManyToOne  
    @MapsId("productId")  
    @JoinColumn(name = "product_id")  
    private Product product;  
  
    @Column(name = "quantity")  
    private Integer quantity;  
  
    public void incrementQuantity() {  
        quantity++;  
    }  
  
    public int decrementQuantity() {  
        if (quantity > 0)  
            quantity--;  
        return quantity;  
    }  
}
```

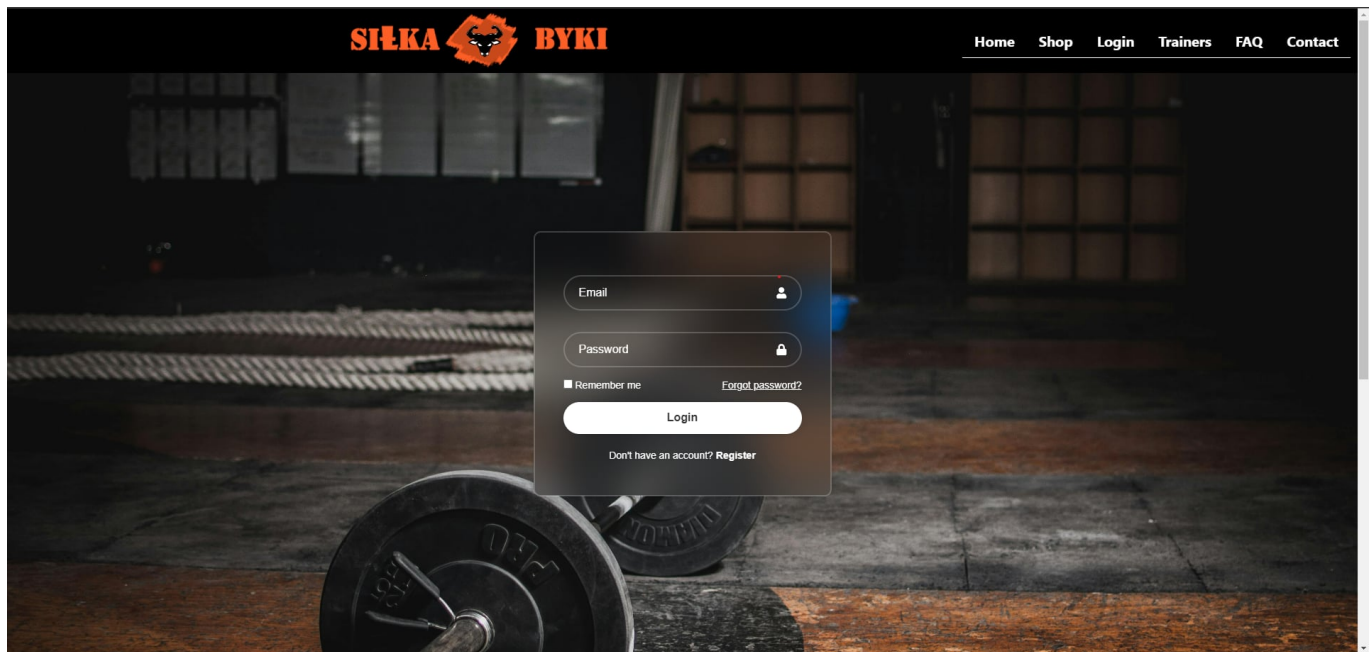
Cart Details Id

```
@Embeddable  
  
public class CartDetailsId implements Serializable {  
    private Long userId;  
    private Long productId;  
}
```

Backend endpoints

User Controller

Login User – POST „/users/login“



Body:

```
{
  "email": "krzysiu123@gmail.com",
  "password": "siema"
}
```

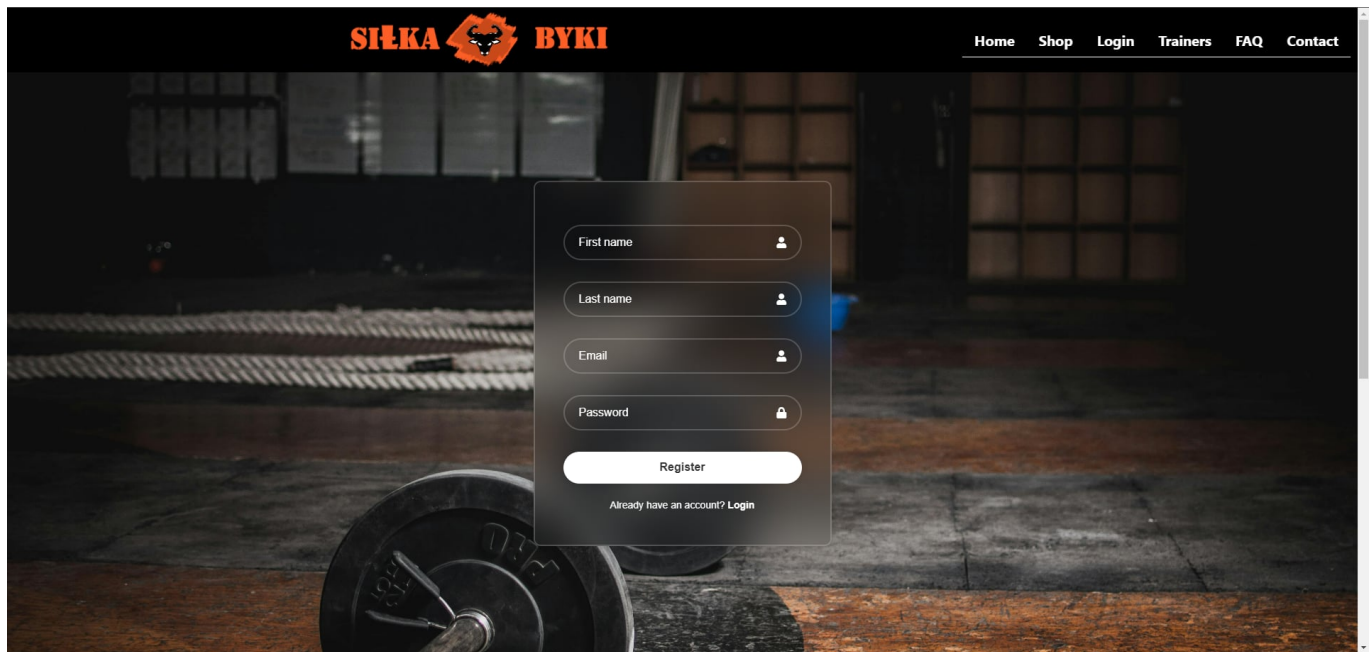
Result:

```
{
  "isEmployee": true,
  "message": "Zalogowano pomyślnie!",
  "loggedUserId": 2
}
```

W przypadku podania nieprawidłowych danych zwracane jest

`ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null);`

Register User – POST „/users/register”



Body:

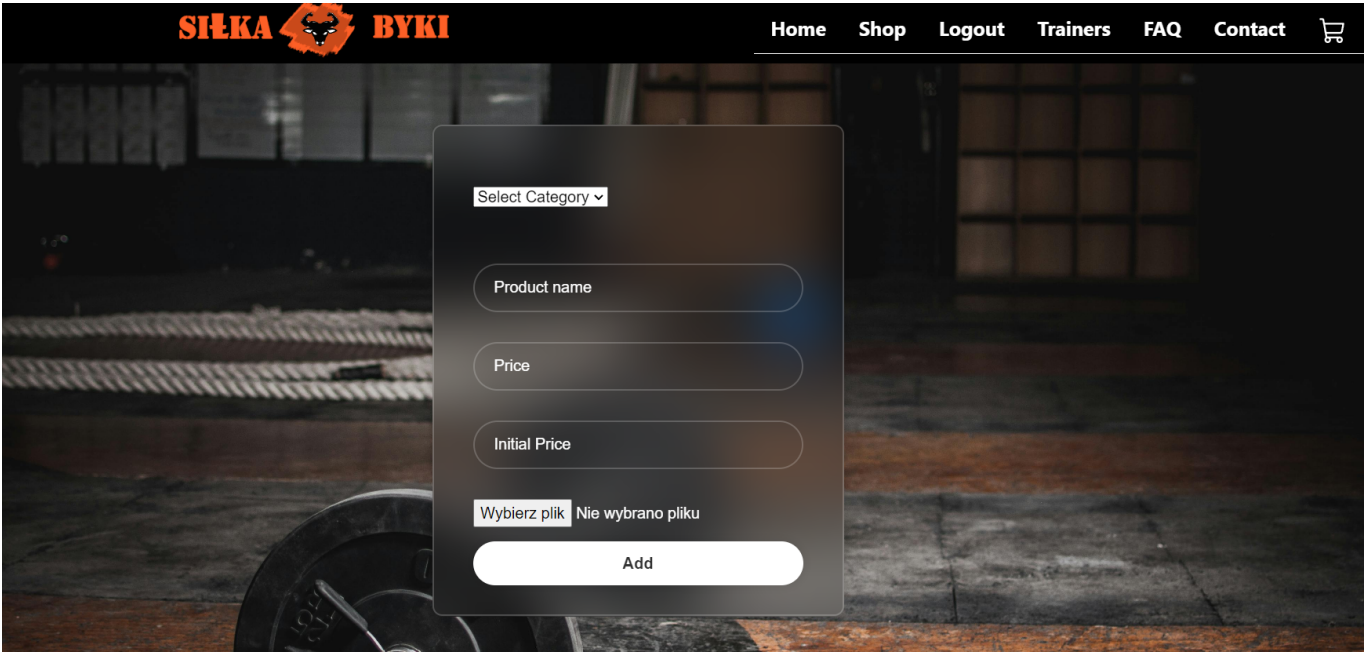
```
{
  "firstName": "Krzysztof",
  "lastName": "Goluchowski",
  "email": "krzychu@gmail.com",
  "isEmployee": true,
  "password": "butter123"
}
```

Result:

```
Zarejestrowano pomyslnie
Witaj Krzysztof!
```

Product Controller

Create Product – POST „/products/add”



Body:

```
{
  "category_id": "1",
  "name": "Najlepsze białko",
  "price": 100,
  "initial_price": 89.99,
  "image": "Wybrane zdjęcie z komputera"
}
```


Result:

```
DODANO!
```

Get All Products – GET „/products/all”

Strona sklepu

SIŁKA



BYKI

Home


Shop

Logout

Trainers

FAQ

Contact




Wszystko

Białko

Węglowodany

Witaminy




Białko dla studentów

\$10.99

\$10

9%

Add to Cart




Extra Doładowanie

\$99.99

\$20

80%

Add to Cart



Białko dla byków

\$20.99

Add to Cart

Body:

```
{}
```

Result:

```
[
  {
    "id": 4,
    "categoryID": 1,
    "name": "Białko dla studentów",
    "price": 10.99,
    "initialPrice": 10.99,
    "image": "iVBORw0KGgoAAAANSUhEUgAAAS0 ...",
    "unitsInStock": 2
  },
  ... Reszta produktów
]
```

Delete Product – DELETE „/products/{id}”

Body:

```
{}
```


Result:

Product deleted successfully!

Update Product Price – PUT „/products/edit-price/{id}”

SIŁKA BYKI

HomeShopLogoutTrainersFAQContact




Initial price:
\$10.99

Current price:
\$10

Cena:
10

EdytujUsun




Initial price:
\$99.99

Current price:
\$20

Cena:
20

EdytujUsun



Initial price:
\$20.99

Current price:
\$20.99

Cena:
20.99

EdytujUsun

Body:

```
{
  "price": 10
}
```

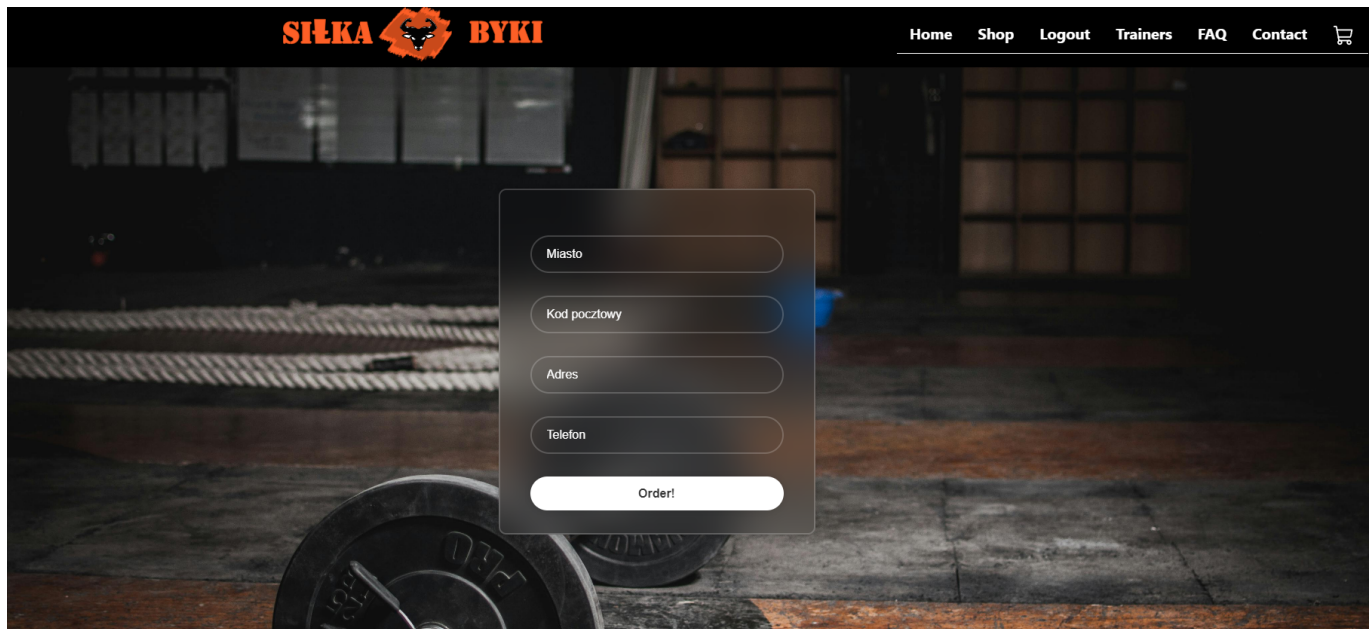
Result:

Product updated successfully!

Order Controller

Place Order – PUT „/orders/place” TODO

10 / 19



Body:

```
{
  "orderDate": "2024-06-03T15:08:06.445+00:00",
  "shipCity": "Krakow",
  "shipPostalCode": "30-055",
  "shipAddress": "Kawiory 21",
  "customersPhone": 123456789,
  "customerId": 2
}
```

Result:

Pomyślnie złożono zamówienie!

Get Monthly Order Report – GET „/orders/monthly-report”

Annual Sales Report - Sales in 2023

- Month: Styczeń, Total Sales: \$575.00
- Month: Luty, Total Sales: \$75.00
- Month: Marzec, Total Sales: \$21.75
- Month: Kwiecień, Total Sales: \$18.00
- Month: Maj, Total Sales: \$0.00
- Month: Czerwiec, Total Sales: \$24.75
- Month: Lipiec, Total Sales: \$56.00
- Month: Sierpień, Total Sales: \$13.00
- Month: Wrzesień, Total Sales: \$10.00
- Month: Październik, Total Sales: \$0.00
- Month: Listopad, Total Sales: \$10.00
- Month: Grudzień, Total Sales: \$89.25

Sales to Now - Sales in 2024

- Month: Styczeń, Total Sales: \$13.00
- Month: Luty, Total Sales: \$10.00
- Month: Marzec, Total Sales: \$0.00
- Month: Kwiecień, Total Sales: \$10.00
- Month: Maj, Total Sales: \$89.25
- Month: Czerwiec, Total Sales: \$658.91

Body:

```
{}
```

Result:

```
[
  [
    1,
    25.0
  ],
  [
    2,
    75.0
  ],
  ... reszta miesięcy
  [
    12,
    89.25
  ]
]
```

Get Current Year Sales – GET „orders/current-year-sales” Dla miesiąca Maj

Body:

```
{}
```

Result:

```
[
  [
    1,
    13.0
  ],
  [
    2,
    10.0
  ],
  [
    3,
    195.0
  ],
  [
    4,
    10.0
  ],
  [
    5,
    89.25
  ]
]
```

Categories Controller

Get All Categories – GET „/categories/all”

Body:

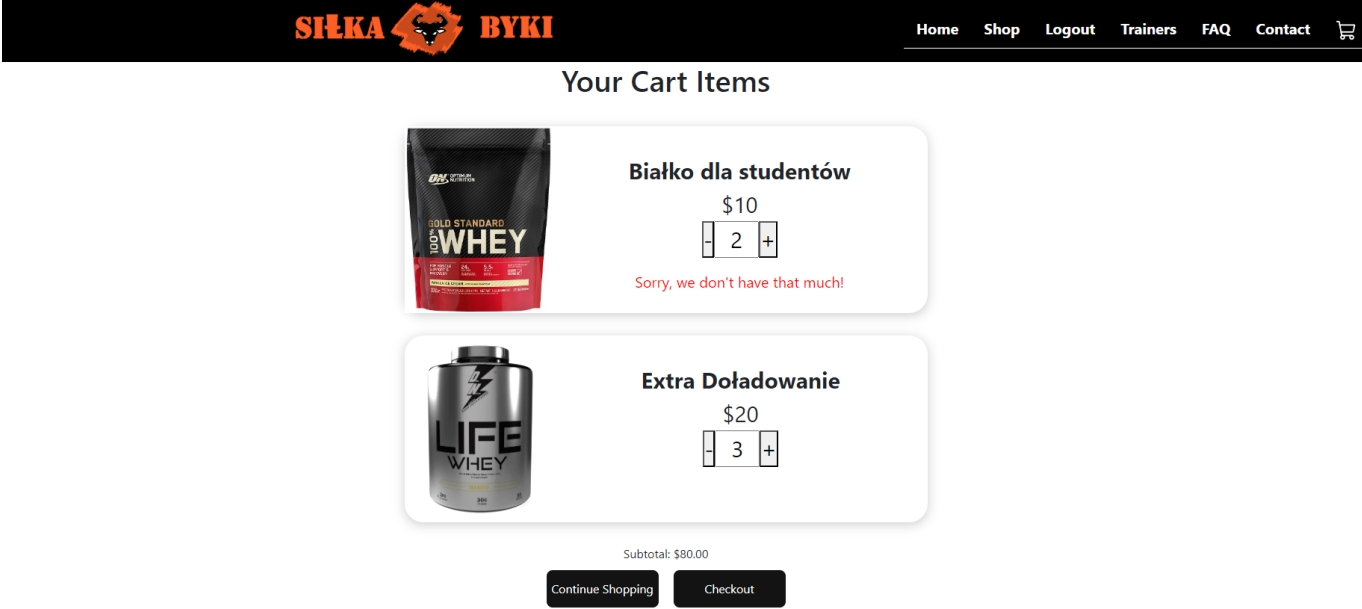
```
{}
```

Result:

```
{
  "Białko": 1,
  "Węglowodany": 2,
  "Witaminy": 3
}
```

Cart Details Controller

Get All Cart Items – POST „/cart/all”



Body:

```
{
  "id": 3
}
```

Result:

```
[
  {
    "userId": 3,
    "productId": 3,
    "quantity": 2
  },
  {
    "userId": 3,
    "productId": 5,
    "quantity": 2
  },
  {
    "userId": 3,
    "productId": 1,
    "quantity": 1
  },
  {
    "userId": 3,
    "productId": 7,
    "quantity": 1
  }
]
```

Body:

```
{
  "userId": 3,
  "productId": 5
}
```

Result:

```
{
  "userId": 3,
  "productId": 5,
  "quantity": 3
}
```

Remove Product – PUT „/cart/remove“

Body:

```
{
  "userId": 3,
  "productId": 5
}
```

Result:

```
{
  "userId": 3,
  "productId": 5,
  "quantity": 2
}
```

Set Product Quantity – PUT „/cart/set?quantity=\${newAmount}“

Dla newAmount = 10

Body:


```
{
  "userId": 3,
  "productId": 5
}
```

Result:

```
{
  "userId": 3,
  "productId": 5,
  "quantity": 10
}
```

Operacje o charakterze raportującym

SIŁKA  BYKI

HomeShopLogoutTrainersFAQContact

Annual Sales Report - Sales in 2023

- Month: Styczeń, Total Sales: \$575.00
- Month: Luty, Total Sales: \$75.00
- Month: Marzec, Total Sales: \$21.75
- Month: Kwiecień, Total Sales: \$18.00
- Month: Maj, Total Sales: \$0.00
- Month: Czerwiec, Total Sales: \$24.75
- Month: Lipiec, Total Sales: \$56.00
- Month: Sierpień, Total Sales: \$13.00
- Month: Wrzesień, Total Sales: \$10.00
- Month: Październik, Total Sales: \$0.00
- Month: Listopad, Total Sales: \$10.00
- Month: Grudzień, Total Sales: \$89.25

Sales to Now - Sales in 2024

- Month: Styczeń, Total Sales: \$13.00
- Month: Luty, Total Sales: \$10.00
- Month: Marzec, Total Sales: \$0.00
- Month: Kwiecień, Total Sales: \$10.00
- Month: Maj, Total Sales: \$89.25
- Month: Czerwiec, Total Sales: \$658.91

Roczny raport sprzedażowy za poprzedni rok z podziałem na miesiące

```
SELECT
  EXTRACT(MONTH FROM o.orderdate) AS month,
  SUM(od.quantity * od.unit_price) AS totalValue
FROM
  orders o
JOIN
  order_details od
ON
  o.order_id = od.order_id
WHERE
  o.orderdate >= DATE_TRUNC('year', CURRENT_DATE) - INTERVAL '1 year' AND
  o.orderdate < DATE_TRUNC('year', CURRENT_DATE)
GROUP BY
  EXTRACT(MONTH FROM o.orderdate)
ORDER BY
  month;
```

Raport sprzedażowy za bieżący rok z podziałem na miesiące


```
SELECT
    EXTRACT(MONTH FROM o.orderdate) AS month,
    SUM(od.quantity * od.unit_price) AS totalValue
FROM
    orders o
JOIN
    order_details od
ON
    o.order_id = od.order_id
WHERE
    o.orderdate >= DATE_TRUNC('year', CURRENT_DATE) AND
    o.orderdate <= CURRENT_DATE + INTERVAL '1 day'
GROUP BY
    EXTRACT(MONTH FROM o.orderdate)
ORDER BY
    month;
```

Transakcje

W OrderService:

```
@Transactional
@Override
public String placeOrder(OrderDto orderDto){
    List<CartDetailsDto> allCartDetailsDto =
    cartDetailsService.getCartDetailsByUserId(orderDto.getCustomerId());

    if (!checkAllProductsInStock(allCartDetailsDto)){
        return "Nie ma takiej ilości w magazynie";
    }

    OrderDto savedOrderDto = createOrder(orderDto);

    List<OrderDetailsDto> allOrderDetailsDto =

    cartDetailsService.mapAllCartDetailsToOrderDetailsDto(allCartDetailsDto,
    savedOrderDto);

    allOrderDetailsDto.forEach(this::createOrderDetails);

    for (CartDetailsDto cartDetailsDto : allCartDetailsDto) {
        productService.removeFromStock(cartDetailsDto.getProductId(),
        cartDetailsDto.getQuantity());
    }

    cartDetailsService.emptyCart(orderDto.getCustomerId());

    return "Pomyślnie złożono zamówienie!";
}
```

Metoda `placeOrder` wykonuje wiele operacji, które muszą być traktowane jako jedna całość. Jeśli którakolwiek z operacji zakończy się niepowodzeniem (zostanie wyrzucony wyjątek `RuntimeException`), cała transakcja zostanie wycofana (`rollback`).

```
// [...]\n\npublic class OrderController {\n\n    private OrderService orderService;\n\n    // [...]\n\n    @ExceptionHandler(RuntimeException.class)\n    public ResponseEntity<String> handleRuntimeException(RuntimeException e) {\n        return\n        ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Something went wrong\n        :(");\n    }\n\n    // [...]\n}\n
```

Gdy nastąpi taka sytuacja, że zostanie wyrzucony wyjątek `RuntimeException` oraz zostanie wykonana operacja `rollback`, metoda oznaczona `@ExceptionHandler` wychwyci to, oraz zwróci do klienta odpowiedni komunikat.

W `CartDetailsService`:

```
@Transactional\n@Override\npublic CartDetailsDto setProductQuantity(Long userId, Long productId, int\nquantity) {\n    CartDetails cartDetails = findCartDetailsOrNew(userId, productId);\n\n    if (quantity == 0) {\n        cartDetailsRepository.delete(cartDetails);\n        return null;\n    }\n\n    cartDetails.setQuantity(quantity);\n\n    CartDetails savedCartDetails = cartDetailsRepository.save(cartDetails);\n\n    return CartDetailsMapper.mapToCartDetailsDto(savedCartDetails);\n}\n
```

Metoda `setProductQuantity` również wykonuje wiele operacji, których wykonanie powinno być traktowane jako jedna całość, dlatego zastosowanie `@Transactional` jest uzasadnione.

Synchronizacja

W przypadku, gdy zostaną wysłane więcej niż jedna prośba utworzenia nowego zamówienia jednocześnie, dochodzi do sytuacji, że operacje aktualizacji bazy danych wykonują się jednocześnie.

O synchronizację tego wybranego *endpoint'u*, zadbaliliśmy przy użyciu *Lock*

```
// [...]  
  
public class OrderController {  
  
    private OrderService orderService;  
  
    private final Lock orderLock = new ReentrantLock();  
  
    // [...]  
  
    @PostMapping("/place")  
    public ResponseEntity<String> placeOrder(@RequestBody OrderDto orderDto) {  
        orderLock.lock();  
        String response = orderService.placeOrder(orderDto);  
        orderLock.unlock();  
        return ResponseEntity.ok(response);  
    }  
  
    // [...]  
}
```

Po "zablokowaniu" `orderLock` przez pierwszy wątek, inny "poczekaj", aż ten pierwszy skończy operację na bazie danych i go odblokuje.