

Akademia Górniczo-Hutnicza

Wydział Informatyki, Elektroniki i Telekomunikacji



Systemu realizujący wybrane operacje w przykładowej
bazie Northwind

Igor Dzierwa
Konrad Makuch
Adrian Nędza

Analiza zadania:

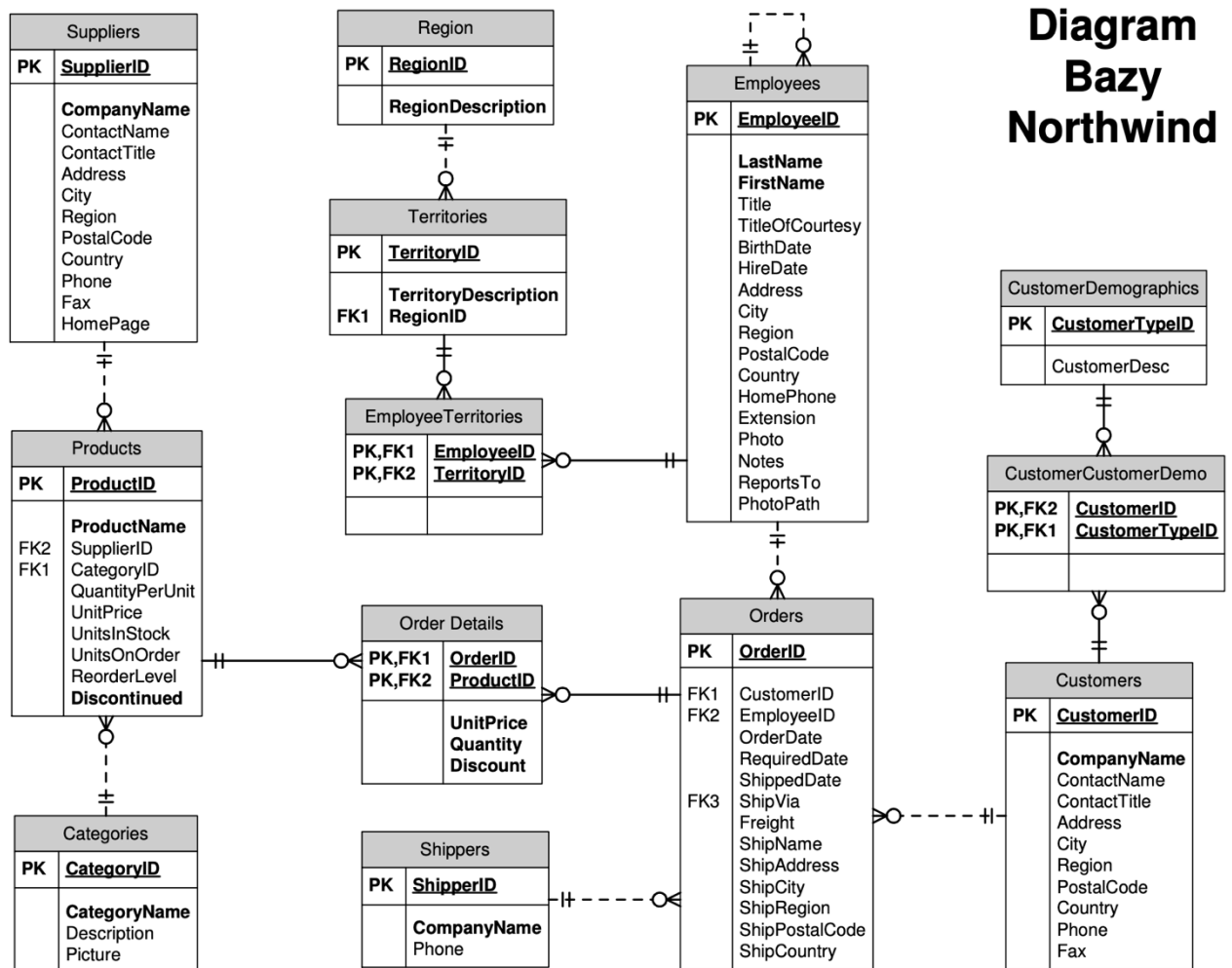
Celem zadania jest zaimplementowanie systemu realizującego wybrane podstawowe operacje w przykładowej bazie Northwind, w wybranej technologii.

Stos technologiczny:

- **MongoDB** – nierelacyjny system do zarządzania bazą danych Northwind
- **Spring Boot + Java11** – serwis realizujący podstawowe operacje na bazie Northwind.
- **Swagger** – automatyczna dokumentacja metod RestAPI.
- **Docker** – konteneryzacja aplikacji.

Model bazy danych Northwind:

- Schemat:



- Symulacja relacji w bazie danych nosql:
 - **Embedded way** — jeden dokument zostaje osadzony w innym dokumencie.
 - **Reference approach** — dokumenty przechowywane są oddzielnie, ale jeden dokument zawiera odniesienie do pola id drugiego dokumentu.
 - W naszej reprezentacji bazy Northwind zastosowaliśmy połączenie obu podejść. Relacje pomiędzy dokumentami symulowane są zarówno poprzez zagnieżdzenie dokumentów jak i referencje.
 - Referencje pozwalają ograniczyć redundancję niepotrzebnych danych przy każdym zwróconym zapytaniu, jak również dają pewność że nie przekroczymy rozmiaru dokumentu (16Mb).
 - Zagnieżdżone dokumenty pozwalają ograniczyć złożoność zapytań, często do jednego dokumentu.
 - Tabele łącznikowe takie jak: Order Details, Employee Territories oraz CustomerCustomerDemo nie zostały uwzględnione jako kolekcje w bazie danych.
- Dokument Supplier z kolekcji Suppliers:

```

@Document(collection = "suppliers")
@NoArgsConstructor
@Getter @Setter
@JsonPropertyOrder
public class Supplier {

    @Id
    private String id;

    private String companyName;

    private String contactName;

    private String contactTitle;

    private String address;

    private String city;

    private String region;

    private String postalCode;

    private String country;

    private String phone;

    private String fax;

    private String homePage;
  
```

```

{
  "id": "string",
  "companyName": "string",
  "contactName": "string",
  "contactTitle": "string",
  "address": "string",
  "region": "string",
  "postalCode": "string",
  "city": "string",
  "country": "string",
  "phone": "string",
  "fax": "string",
  "homePage": "string",
}
  
```

- Dokument Category z kolekcji Categories:

```
@Document(collection = "categories")
@NoArgsConstructor
@Getter @Setter
@JsonPropertyOrder
public class Category {
    @Id
    private String id;

    private String categoryName;

    private String description;

    private String picture;
}
```

```
{
  "id": "string",
  "categoryName": "string",
  "description": "string",
  "picture": "string"
}
```

- Dokument Product z kolekcji Products:
 - **Przełożenie relacji one-to-many pomiędzy Suppliers i Products** – uzyskane za pomocą referencji poprzez odniesienie do id dokumentu Supplier z kolekcji Suppliers – zagnieżdżenie dokumentu w tym wypadku nie jest optymalnym rozwiązaniem, gdyż nie jest to informacja często pozyskiwana i przez to występuje zbędna redundancja danych. Co więcej, wymagałoby to wymuszoną kaskadowość operacji w sytuacji aktualizacji dokumentu Supplier.
 - **Przełożenie relacji one-to-many pomiędzy Categories i Products** – uzyskane za pomocą zagnieżdżenia dokumentu Category w obrębie dokumentu Product – jako często pozyskiwana informacja, pozwala to na niwelację odwołań do kolekcji Categories.

```
@Document(collection = "products")
@NoArgsConstructor
@Getter @Setter
@JsonPropertyOrder
public class Product {
    @Id
    private String id;

    private String productName;

    private String supplierID;

    @DBRef
    private Category category;

    private int quantityPerUnit;

    private double unitPrice;

    private int unitsInStock;

    private int unitsInOrder;

    private int reorderLevel;

    private boolean discontinued;
}
```

```
{
  "id": "string",
  "supplierID": "string",
  "productName": "string",
  {
    "category": {
      "id": "string",
      "categoryName": "string",
      "description": "string",
      "picture": "string"
    },
    "quantityPerUnit": 0,
    "unitPrice": 0,
    "unitsInStock": 0,
    "unitsInOrder": 0,
    "reorderLevel": 0,
    "discontinued": true,
  }
}
```

- Dokument Region z kolekcji Regions:

```
@Document(value = "regions")
@NoArgsConstructor
@Getter @Setter
@JsonPropertyOrder
public class Region {
    @Id
    private String id;

    private String regionDescription;
}
```

```
{
  "id": "string",
  "regionDescription": "string"
}
```

- Dokument Territory z kolekcji Territories:
 - **Przełożenie relacji one-to-many pomiędzy Regions i Territories** – uzyskane za pomocą zagnieżdżenia dokumentu Region w obrębie dokumentu Territory (sytuacja analogiczna do przełożonej relacji one-to-many pomiędzy Categories i Products).

```
@Document(collection = "territories")
@NoArgsConstructor
@Getter @Setter
@JsonPropertyOrder
public class Territory {
    @Id
    private String id;

    private String territoryDescription;

    @DBRef
    private Region region;
}
```

```
{
  "id": "string",
  "territoryDescription": "string",
  "region": {
    "id": "string",
    "regionDescription": "string"
  },
}
```

- Dokument CustomerDemographic z kolekcji CustomerDemographics:

```
@Document(collection = "customerDemographics")
@NoArgsConstructor
@Getter @Setter
@JsonPropertyOrder
public class CustomerDemographic {
    @Id
    private String id;

    private String customerDesc;
}
```

```
{
  "id": "string"
  "customerDesc": "string",
}
```

- Dokument Employee z kolekcji Employees:
 - **Przełożenie relacji many-to-many pomiędzy Territories i Employees** – uzyskane za pomocą *one way embedding*, czyli osadzenia zagnieżdżenia po jednej stronie relacji, co optymalizuje wydajność odczytu relacji many-to-many.

```
@Document(collection = "employees")
@NoArgsConstructor
@Getter @Setter
@JsonPropertyOrder
public class Employee {
    @Id
    private String id;

    private String lastName;

    private String firstName;

    private String title;

    private String titleOfCourtesy;

    private Date birthDate;

    private Date hireDate;

    private String address;

    private String city;

    private String region;

    private String postalCode;

    private String country;

    private String homePhone;

    private String extension;

    private String photo;

    private String notes;

    private String reportsTo;

    private String photoPath;

    @DBRef
    private List<Territory> employeeTerritories = new ArrayList<>();
}
```

```
{
  "id": "string",
  "lastName": "string",
  "firstName": "string",
  "title": "string",
  "titleOfCourtesy": "string",
  "birthDate": "2020-12-09T11:38:34.146Z",
  "hireDate": "2020-12-09T11:38:34.147Z",
  "address": "string",
  "city": "string",
  "region": "string",
  "postalCode": "string",
  "country": "string",
  "homePhone": "string",
  "extension": "string",
  "photo": "string",
  "notes": "string",
  "reportsTo": "string",
  "photoPath": "string",
  "employeeTerritories": [
    {
      "id": "string",
      "territory": {
        "id": "string",
        "territoryDescription": "string",
        "region": {
          "id": "string",
          "regionDescription": "string"
        }
      }
    }
  ],
}
```

- Dokument Customer z kolekcji Customers:
 - **Przełożenie relacji many-to-many pomiędzy Customers i CustomerDemographics** – uzyskane za pomocą *one way embedding*, czyli osadzenia zagnieżdżenia po jednej stronie relacji, co optymalizuje wydajność odczytu relacji many-to-many.

```
@Document(collection = "customers")
@NoArgsConstructor
@Getter @Setter
@JsonPropertyOrder
public class Customer {
    @Id
    private String id;

    private String companyName;

    private String contactName;

    private String contactTitle;

    private String address;

    private String city;

    private String region;

    private String postalCode;

    private String country;

    private String phone;

    private String fax;

    @DBRef
    private List<CustomerDemographic> customerDemographics = new ArrayList<>();
}
```

```
{
  "id": "string",
  "companyName": "string",
  "contactName": "string",
  "contactTitle": "string",
  "address": "string",
  "city": "string",
  "region": "string",
  "postalCode": "string",
  "country": "string",
  "phone": "string",
  "fax": "string",
  "customerCustomerDemo": [
    {
      "id": "string"
      "customerDemographic": {
        "id": "string"
        "customerDesc": "string",
      },
    }
  ],
}
```

- Dokument Shipper z kolekcji Shippers:

```
@Document(collection = "shippers")
@NoArgsConstructor
@Getter @Setter
@JsonPropertyOrder
public class Shipper {
    @Id
    private String id;

    private String companyName;

    private String phone;
}
```

```
{
  "id": "string",
  "companyName": "string",
  "phone": "string"
}
```

- Dokument Order z kolekcji Orders:
 - **Przełożenie relacji one-to-many pomiędzy Customers i Orders** – uzyskane za pomocą referencji poprzez odniesienie do id dokumentu Customer z kolekcji Customers .
 - **Przełożenie relacji one-to-many pomiędzy Employees i Orders** – uzyskane za pomocą referencji poprzez odniesienie do id dokumentu Employee z kolekcji Employees.
 - **Przełożenie relacji one-to-many pomiędzy Shippers i Orders** – uzyskane za pomocą referencji poprzez odniesienie do id dokumentu Shipper z kolekcji Shippers.
 - Dokumenty: Product, Customer, Employee i Shipper nie są zagnieżdżone w obrębie dokumentu Order ze względu na uniknięcie duplikacji danych, ponieważ, wiemy że dokumenty Order będą tworzone z dużą częstotliwością, dodatkowo wpływ mają na to operacje aktualizacji poszczególnych dokumentów – zmieniamy je wówczas tylko w jednym miejscu.

```

@Document(collection = "orders")
@NoArgsConstructor
@Getter @Setter
@JsonPropertyOrder
public class Order {
    @Id
    private String id;

    private String customerID;

    private String employeeID;

    private String shipperID;

    private Date orderDate;

    private Date requiredDate;

    private Date shippedDate;

    private String freight;

    private String shipName;

    private String shipAddress;

    private String shipCity;

    private String shipRegion;

    private String shipPostalCode;

    private String shipCountry;

    private List<OrderDetails> orderDetails = new ArrayList<>();

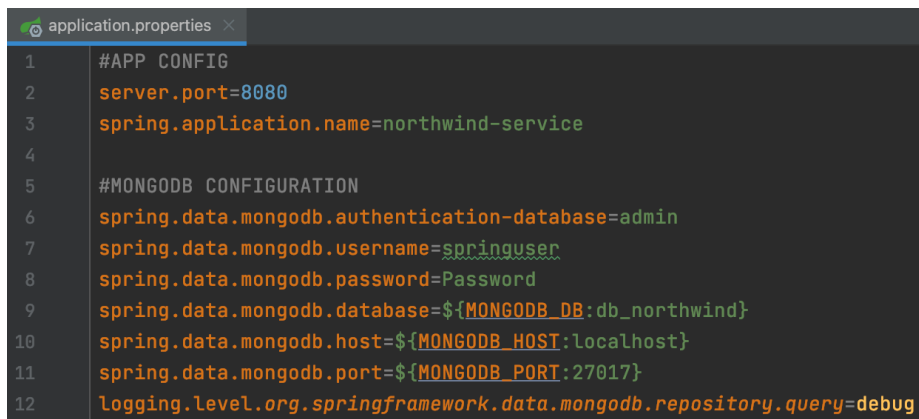
```

```

{
  "id": "string",
  "customerID": "string",
  "employeeID": "string",
  "orderDate": "2020-12-09T12:22:51.964Z",
  "requiredDate": "2020-12-09T12:22:51.964Z",
  "shippedDate": "2020-12-09T12:22:51.964Z",
  "shipperID": "string",
  "freight": "string",
  "shipName": "string",
  "shipAddress": "string",
  "shipCity": "string",
  "shipRegion": "string",
  "shipPostalCode": "string",
  "shipCountry": "string",
  "orderDetails": [
    {
      "productName": "string",
      "unitPrice": 0,
      "quantity": 0,
      "discount": 0
    }
  ]
}

```


Plik konfiguracyjny bazy danych MongoDB, wspierany przez Spring Boot:



```
1 #APP CONFIG
2 server.port=8080
3 spring.application.name=northwind-service
4
5 #MONGODB CONFIGURATION
6 spring.data.mongodb.authentication-database=admin
7 spring.data.mongodb.username=springuser
8 spring.data.mongodb.password=Password
9 spring.data.mongodb.database=${MONGODB_DB:db_northwind}
10 spring.data.mongodb.host=${MONGODB_HOST:localhost}
11 spring.data.mongodb.port=${MONGODB_PORT:27017}
12 logging.level.org.springframework.data.mongodb.repository.query=debug
```

- **spring.data.mongodb.authentication-database** – nazwa autentyfikacji bazy danych.
- **spring.data.mongodb.username** – nazwa użytkownika w serwerze MongoDB.
- **spring.data.mongodb.password** – hasło do konta użytkownika w serwerze MongoDB.
- **spring.data.mongodb.database** – nazwa bazy danych.
- **spring.data.mongodb.host** – nazwa hosta serwera MongoDB.
- **logging.level.org.springframework.data.mongodb.repository.query** – pozwala włączyć logi dotyczące wykonywanych zapytań na bazie danych.

Operacje CRUD na wybranej kolekcji - Products:

Spring Data dostarcza interfejs `MongoRepository`, który umożliwia zarządzanie obiektami danych, dla większości typowych operacji na kolekcjach dokumentów, takich jak: dodawanie, usuwanie, aktualizowanie oraz wyszukiwanie po określonym polu.

- **Przy tworzeniu/aktualizowaniu dokumentu Product** uczestniczą dokumenty takich kolekcji jak: Categories oraz Suppliers.
Do odwołań, do konkretnych dokumentów z kolekcji, wykorzystujemy repozytoria:
 - **Categories** – wyszukanie konkretnego dokumentu Category następuje za pośrednictwem wartości pola categoryName:

```
@Repository
public interface CategoriesRepository extends MongoRepository<Category, String> {
    List<Category> findAll();
    Category findByCategoryName(String categoryName);
}
```

- **Suppliers** – wyszukanie konkretnego dokumentu Supplier następuje za pośrednictwem wartości pola companyName.

```
@Repository
public interface SuppliersRepository extends MongoRepository<Supplier, String> {
    List<Supplier> findAll();
    Supplier findByCompanyName(String companyName);
}
```

- **Products** – wyszukanie konkretnego dokumentu Product następuje za pośrednictwem wartości pola productName lub id.

```
@Repository
public interface ProductsRepository extends MongoRepository<Product, String> {
    List<Product> findAll();
    Product findByProductName(String productName);
    List<Product> findAllByCategory(Category category);
    List<Product> findAllBySupplierID(String supplierID);
}
```

- Tworzenie nowego dokumentu Product w kolekcji Products:

```
@PostMapping(value = "/api/product")
@ResponseBody
public ResponseEntity<String> addNewProduct(@RequestBody ProductsRequestBody productsRequestBody){
    Product product = new Product(productsRequestBody);
    Category category = categoriesRepository.findByCategoryName(productsRequestBody.getCategoryName());
    Supplier supplier = suppliersRepository.findByCompanyName(productsRequestBody.getSupplierName());
    if (category != null && supplier != null) {
        product.setCategory(category);
        product.setSupplierID(supplier.getId());
        productsRepository.save(product);
        return ResponseEntity.ok("{\"status\": \"added\"}");
    } else if (category == null && supplier != null) {
        return ResponseEntity.ok("{\"status\": \"category does not exist\"}");
    } else if (category != null && supplier == null) {
        return ResponseEntity.ok("{\"status\": \"supplier does not exist\"}");
    }
    return ResponseEntity.ok("{\"status\": \"supplier and category does not exist\"}");
}
```

- Przekazywane dane:

POST	/api/product	addNewProduct
Parameters		
No parameters		
Request body		
Example Value Schema		
<pre>{ "categoryName": "string", "discontinued": true, "productName": "string", "quantityPerUnit": 0, "reorderLevel": 0, "supplierName": "string", "unitPrice": 0, "unitsInOrder": 0, "unitsInStock": 0 }</pre>		

- Logi z konsoli:

- 1) find using query: { "categoryName" : "string"} fields: Document{{{}} for class: class com.agh.northwindproject.Categories.Category in collection: categories
- 2) find using query: { "companyName" : "string"} fields: Document{{{}} for class: class com.agh.northwindproject.Suppliers.Supplier in collection: suppliers
- 3) Inserting Document containing fields: [productName, supplierID, category, quantityPerUnit, unitPrice, unitsInStock, unitsInOrder, reorderLevel, discontinued, _class] in collection: products

- Aktualizacja wybranego dokumentu Product w kolekcji Products:

```
@PutMapping(value = "/api/product/{productID}")
@ResponseBody
public ResponseEntity<String> updateProduct(@PathVariable String productID,
                                           @RequestBody ProductsRequestBody productsRequestBody) {
    if(productsRepository.findById(productID).orElse( other: null) != null) {
        Product product = new Product(productsRequestBody);
        product.setId(productID);
        Category category = categoriesRepository.findByCategoryName(productsRequestBody.getCategoryName());
        Supplier supplier = suppliersRepository.findByCompanyName(productsRequestBody.getSupplierName());
        if (category != null && supplier != null) {
            product.setCategory(category);
            product.setSupplierID(supplier.getId());
            productsRepository.save(product);
            return ResponseEntity.ok("{\"status\": \"updated\"}");
        } else if (category == null && supplier != null) {
            return ResponseEntity.ok("{\"status\": \"category does not exist\"}");
        } else if (category != null && supplier == null) {
            return ResponseEntity.ok("{\"status\": \"supplier does not exist\"}");
        }
        return ResponseEntity.ok("{\"status\": \"supplier and category does not exist\"}");
    }
    return ResponseEntity.ok("{\"status\": \"product does not exist\"}");
}
```

- Przekazywane dane:

PUT /api/product/{productID} updateProduct	
Parameters	
Name	Description
productID * required string (path)	productID
<input type="text" value="productID - productID"/>	
Request body	
Example Value Schema	
<pre>{ "categoryName": "string", "discontinued": true, "productName": "string", "quantityPerUnit": 0, "reorderLevel": 0, "supplierName": "string", "unitPrice": 0, "unitsInOrder": 0, "unitsInStock": 0 }</pre>	

- Logi z konsoli:

- 1) findOne using query: { "id" : "6004c97c767fd149c118d768"} fields: Document{{{}} for class: class com.agh.northwindproject.Products.Product in collection: products
- 2) findOne using query: { "_id" : { "\$oid" : "6004c97c767fd149c118d768"}} fields: {} in db.collection: db_northwind.products
- 3) find using query: { "categoryName" : "string"} fields: Document{{{}} for class: class com.agh.northwindproject.Categories.Category in collection: categories
- 4) find using query: { "companyName" : "string"} fields: Document{{{}} for class: class com.agh.northwindproject.Suppliers.Supplier in collection: suppliers
- 5) Saving Document containing fields: [_id, productName, supplierID, category, quantityPerUnit, unitPrice, unitsInStock, unitsInOrder, reorderLevel, discontinued, _class]

- Wyszukanie konkretnego dokumentu Product w kolekcji Products po id dokumentu:

```
@GetMapping(value = "/api/product/id/{productID}")
@ResponseBody
public ResponseEntity<Product> getProductById(@PathVariable String productID){
    Product product = productsRepository.findById(productID).orElse( other: null);
    return ResponseEntity.ok(product);
}
```

- W parametrze ścieżki podawana wartość pola id dokumentu Product:

GET /api/product/id/{productID} getProductById	
Parameters	
Name	Description
productID * required string (path)	productID
<input type="text" value="productID - productID"/>	

- Logi z konsoli:

- 1) findOne using query: { "id" : "6004c97c767fd149c118d768"} fields: Document{{{}} for class: class com.agh.northwindproject.Products.Product in collection: products
- 2) findOne using query: { "_id" : { "\$oid" : "6004c97c767fd149c118d768"}} fields: {} in db.collection: db_northwind.products

- Wyszukanie konkretnego dokumentu Product w kolekcji Products po polu productName:

```
@GetMapping(value = "/api/product/{productName}")
@ResponseBody
public ResponseEntity<Product> getProductByProductName(@PathVariable String productName){
    return ResponseEntity.ok(productsRepository.findByProductName(productName));
}
```

- W parametrze ścieżki podawana wartość pola productName dokumentu Product:

GET /api/product/{productName} getProductByProductName	
Parameters	
Name	Description
productName * required string (path)	productName
<input type="text" value="productName - productName"/>	

- Logi z konsoli:

- 1) find using query: { "productName" : "string"} fields: Document{{{}} for class: class com.agh.northwindproject.Products.Product in collection: products

- Wyszukanie wszystkich dokumentów z kolekcji Products:

```
@GetMapping(value = "/api/products")
@ResponseBody
public ResponseEntity<List<Product>> getAllProducts() { return ResponseEntity.ok(productsRepository.findAll()); }
```

- Funkcja nie przyjmuje żadnych parametrów:

GET	/api/products	getAllProducts
Parameters		
No parameters		

- Logi z konsoli:

- find using query: {} fields: Document({}) for class: class com.agh.northwindproject.Products.Product in collection: products

- Usunięcie wybranego dokumentu Product w kolekcji Products:

```
@DeleteMapping(value = "/api/product/{productID}")
@ResponseBody
public ResponseEntity<String> deleteProduct(@PathVariable String productID){
    Product product = productsRepository.findById(productID).orElse( other: null);
    if(product != null){
        productsRepository.delete(product);
        return ResponseEntity.ok("{\"status\": \"removed\"}");
    }
    return ResponseEntity.ok("{\"status\": \"product already not existing, cannot remove\"}");
}
```

- W parametrze ścieżki podawana wartość pola id dokumentu Product:

DELETE	/api/product/{productID}	deleteProduct
Parameters		
Name	Description	
productID ★ required string (path)	productID	productID - productID

- Logi z konsoli:

- findOne using query: {"id" : "6004c97c767fd149c118d768"} fields: Document({}) for class: class com.agh.northwindproject.Products.Product in collection: products
- findOne using query: {"_id" : {"\$oid" : "6004c97c767fd149c118d768"}} fields: {} in db.collection: db_northwind.products
- Remove using query: {"_id" : {"\$oid" : "6004c97c767fd149c118d768"}} in collection: products.

Operacje na kolekcji Orders:

- **Przy tworzeniu nowego dokumentu Order** uczestniczą dokumenty takich kolekcji jak: Customers, Employees, Products oraz Shippers.

Do odwołań, do konkretnych dokumentów z kolekcji, wykorzystujemy repozytoria:

- **Customers** – wyszukanie konkretnego dokumentu Customer następuje za pośrednictwem wartości pola companyName.

```
@Repository
public interface CustomersRepository extends MongoRepository<Customer, String> {
    List<Customer> findAll();
    Customer findByCompanyName(String companyName);
}
```

- **Employees** – wyszukanie konkretnego dokumentu Employee następuje za pośrednictwem wartości pól lastName oraz firstName.

```
@Repository
public interface EmployeesRepository extends MongoRepository<Employee, String> {
    List<Employee> findAll();
    Employee findByLastNameAndFirstName(String lastName, String firstName);
}
```

- **Products** – wyszukanie konkretnego dokumentu Products następuje za pośrednictwem pola productName.

```
@Repository
public interface ProductsRepository extends MongoRepository<Product, String> {
    List<Product> findAll();
    Product findByProductName(String productName);
    List<Product> findAllByCategory(Category category);
    List<Product> findAllBySupplierID(String supplierID);
}
```

- **Shippers** – wyszukanie konkretnego dokumentu Shipper następuje za pośrednictwem wartości pola companyName.

```
@Repository
public interface ShippersRepository extends MongoRepository<Shipper, String> {
    List<Shipper> findAll();
    Shipper findByCompanyName(String companyName);
}
```

- **Orders** – wyszukanie konkretnego dokumentu Order może następować zarówno poprzez wartość pól customerID i EmployeeID, lub id.

```
@Repository
public interface OrdersRepository extends MongoRepository<Order, String> {
    List<Order> findAll();
    List<Order> findByCustomerIDAndEmployeeID(String customerID, String employeeID);
}
```

- Tworzenie nowego dokumentu Order w kolekcji Orders:
 - Oprócz utworzenia nowego dokumentu Orders, równocześnie następuje aktualizacja wartości pól: unitsInOrder oraz unitsInStock konkretnego dokument Product.

```

@PostMapping(value = "/api/order")
@ResponseBody
public ResponseEntity<String> addNewOrder(@RequestBody OrderRequestBody orderRequestBody){
    Customer customer = customersRepository.findByCompanyName(orderRequestBody.getCustomerCompanyName());
    if (customer == null) {
        return ResponseEntity.ok("{\"status\": \"customer does not exists\"}");
    }
    Employee employee = employeesRepository.findByLastNameAndFirstName(orderRequestBody.getEmployeeLastName(),
        orderRequestBody.getEmployeeFirstName());
    if (employee == null) {
        return ResponseEntity.ok("{\"status\": \"employee does not exists\"}");
    }
    Shipper shipper = shippersRepository.findByCompanyName(orderRequestBody.getShipperCompanyName());
    if (shipper == null) {
        return ResponseEntity.ok("{\"status\": \"shipper does not exists\"}");
    }

    Order order = new Order(orderRequestBody);
    order.setCustomerID(customer.getId());
    order.setEmployeeID(employee.getId());
    order.setShipperID(shipper.getId());
    order.setOrderDate( Calendar.getInstance().getTime());

    for(OrderDetailsRequestBody orderDetailsRequestBody : orderRequestBody.getOrderDetails()) {
        Product product = productsRepository.findByProductName(orderDetailsRequestBody.getProductName());
        if(product != null && product.isDiscontinued() == false) {
            if(product.getQuantityPerUnit() >= orderDetailsRequestBody.getQuantity()) {
                if(product.getUnitsInStock() - orderDetailsRequestBody.getQuantity() >= 0) {
                    OrderDetails orderDetails = new OrderDetails(product.getProductName(), orderDetailsRequestBody);
                    orderDetails.setUnitPrice(product.getUnitPrice() - orderDetails.getDiscount());
                    order.getOrderDetails().add(orderDetails);
                    product.setUnitsInOrder(product.getUnitsInOrder() + orderDetailsRequestBody.getQuantity());
                    product.setUnitsInStock(product.getUnitsInStock() - orderDetailsRequestBody.getQuantity());
                    productsRepository.save(product);
                } else {
                    return ResponseEntity.ok("{\"status\": \"not enough units in stock\": \" + product.getProductName()});
                }
            } else {
                return ResponseEntity.ok("{\"status\": \"invalid quantityPerUnit\": \" + product.getProductName()});
            }
        } else {
            return ResponseEntity.ok("{\"status\": \"product does not exists\": \" + product.getProductName()});
        }
    }

    ordersRepository.save(order);
    return ResponseEntity.ok("{\"status\": \"added\"}");
}

```


○ Przekazywane dane

POST `/api/order` addNewOrder

Parameters

No parameters

Request body

Example Value | Schema

```
{
  "customerCompanyName": "string",
  "employeeFirstName": "string",
  "employeeLastName": "string",
  "freight": "string",
  "orderDate": "2021-01-18T19:29:32.548Z",
  "orderDetails": [
    {
      "discount": 0,
      "productName": "string",
      "quantity": 0
    }
  ],
  "requiredDate": "2021-01-18T19:29:32.548Z",
  "shipAddress": "string",
  "shipCity": "string",
  "shipCountry": "string",
  "shipName": "string",
  "shipPostalCode": "string",
  "shipRegion": "string",
  "shippedDate": "2021-01-18T19:29:32.548Z",
  "shipperCompanyName": "string"
}
```

○ Logi z konsoli:

- 1) find using query: { "companyName" : "Testowa firma"} fields: Document{{{}} for class: class com.agh.northwindproject.Customers.Customer in collection: customers
- 2) find using query: { "lastName" : "Test", "firstName" : "Testowy"} fields: Document{{{}} for class: class com.agh.northwindproject.Employees.Employee in collection: employees
- 3) find using query: { "companyName" : "DHL"} fields: Document{{{}} for class: class com.agh.northwindproject.Shippers.Shipper in collection: shippers
- 4) find using query: { "productName" : "Macbook air"} fields: Document{{{}} for class: class com.agh.northwindproject.Products.Product in collection: products
- 5) Saving Document containing fields: [_id, productName, supplierID, category, quantityPerUnit, unitPrice, unitsInStock, unitsInOrder, reorderLevel, discontinued, _class]
- 6) find using query: { "productName" : "Macbook pro"} fields: Document{{{}} for class: class com.agh.northwindproject.Products.Product in collection: products
- 7) Saving Document containing fields: [_id, productName, supplierID, category, quantityPerUnit, unitPrice, unitsInStock, unitsInOrder, reorderLevel, discontinued, _class]
- 8) Inserting Document containing fields: [customerID, employeeID, shipperID, orderDate, requiredDate, shippedDate, freight, shipName, shipAddress, shipCity, shipRegion, shipPostalCode, shipCountry, orderDetails, _class] in collection: orders

- Wyszukanie konkretnego dokumentu Order w kolekcji Orders po id dokumentu:

```
@GetMapping(value = "/api/order/{orderId}")
@ResponseBody
public ResponseEntity<Order> getOrder(@PathVariable String orderId){
    Order order = ordersRepository.findById(orderId).orElse( other: null);
    return ResponseEntity.ok(order);
}
```

- W parametrze ścieżki podawana wartość pola id dokumentu Order:

Name	Description
orderId * required string (path)	orderId

orderId - orderId

- Logi:

- findOne using query: { "id" : "6005f6b9b095b86c3d3186d3"} fields: Document{{}} for class: class com.agh.northwindproject.Orders.Order in collection: orders
- findOne using query: { "_id" : { "\$oid" : "6005f6b9b095b86c3d3186d3"} } fields: {} in db.collection: db_northwind.orders

- Wyszukanie konkretnego dokumentu Order w kolekcji Orders po wartości pól customerCompanyName, employeeLastName, employeeFirstName:

```
@GetMapping(value = "/api/order/{customerCompanyName}/{employeeLastName}/{employeeFirstName}")
@ResponseBody
public ResponseEntity<List<Order>> getOrdersByCustomerAndEmployee(@PathVariable String customerCompanyName,
                                                                    @PathVariable String employeeLastName,
                                                                    @PathVariable String employeeFirstName){
    Customer customer = customersRepository.findByCompanyName(customerCompanyName);
    Employee employee = employeesRepository.findByLastNameAndFirstName(employeeLastName, employeeFirstName);
    if (customer != null && employee != null) {
        return ResponseEntity.ok(ordersRepository.findByCustomerIdAndEmployeeId(customer.getId(), employee.getId()));
    }
    return null;
}
```

- W parametrze ścieżki podawana jest wartości pól: customerCompanyName dokumentu Customer oraz employeeLastName i employeeFirstName dokumentu Employee.

Name	Description
customerCompanyName * required string (path)	customerCompanyName
employeeLastName * required string (path)	employeeLastName
employeeFirstName * required string (path)	employeeFirstName

customerCompanyName - customerCompan

employeeLastName - employeeLastName

employeeFirstName - employeeFirstName

- Logi:

- 1) find using query: { "companyName" : "Testowa firma"} fields: Document{{{}} for class: class com.agh.northwindproject.Customers.Customer in collection: customers
- 2) find using query: { "lastName" : "Test", "firstName" : "Testowy"} fields: Document{{{}} for class: class com.agh.northwindproject.Employees.Employee in collection: employees
- 3) find using query: { "customerID" : "6005f574b095b86c3d3186d0", "employeeID" : "6005f5ccb095b86c3d3186d1"} fields: Document{{{}} for class: class com.agh.northwindproject.Orders.Order in collection: orders

- Wyszukanie wszystkich dokumentów Order w kolekcji Orders:

```
@GetMapping(value = "/api/orders")
@ResponseBody
public ResponseEntity<List<Order>> getAllOrders() { return ResponseEntity.ok(ordersRepository.findAll()); }
```

- Funkcja nie przyjmuje żadnych parametrów:

GET	/api/orders	getAllOrders
Parameters		
No parameters		

- Logi:

- 1) find using query: {} fields: Document{{{}} for class: class com.agh.northwindproject.Orders.Order in collection: orders

- 2)

- Usunięcie wybranego dokumentu Order w kolekcji Orders:

```
@DeleteMapping(value = "/api/order/{orderId}")
@ResponseBody
public ResponseEntity<String> deleteOrder(@PathVariable String orderId){
    Order order = ordersRepository.findById(orderId).orElse( other: null);
    if(order != null){
        ordersRepository.delete(order);
        return ResponseEntity.ok("{\"status\": \"removed\""});
    }
    return ResponseEntity.ok("{\"status\": \"order not exists\""});
}
```

- W parametrze ścieżki podawana wartość pola id dokumentu Order:

DELETE	/api/order/{orderId}	deleteOrder
Parameters		
Name	Description	
orderId * required	orderId	
string (path)		orderId - orderId

○ **Logi:**

- 1) findOne using query: { "id" : "6005f702b095b86c3d3186d4"} fields: Document{{{}} for class: class com.agh.northwindproject.Orders.Order in collection: orders
- 2) findOne using query: { "_id" : { "\$oid" : "6005f702b095b86c3d3186d4"}} fields: {} in db.collection: db_northwind.orders
- 3) Remove using query: { "_id" : { "\$oid" : "6005f702b095b86c3d3186d4"}} in collection: orders

Konfiguracja aplikacji

Aplikacja została skonteneryzowana za pośrednictwem Dockera – wyróżnione zostały 3 kontenery:

- Kontener serwisu serwera
- Kontener skonfigurowanej bazy MongoDB

```
version: '3'
services:
  mongo:
    image: mongo:latest
    container_name: mongo
    environment:
      MONGO_INITDB_DATABASE: db_northwind
      MONGO_INITDB_ROOT_USERNAME: springuser
      MONGO_INITDB_ROOT_PASSWORD: Password
    ports:
      - "27017:27017"
    volumes:
      - mongodb-dbnorthwind-volume:/data/db
    networks:
      - northwind-service-network

  northwind-service:
    image: northwind-service
    container_name: 'northwind-service'
    build: northwind-service/.
    environment:
      MONGODB_HOST: mongo
    ports:
      - 8080:8080
    networks:
      - northwind-service-network

volumes:
  mongodb-dbnorthwind-volume:

networks:
  northwind-service-network:
```

- **Budowa kontenerów:** `docker-compose up -d` (instalacja zależności frontu może trwać nawet 10 minut).
- **Usunięcie kontenerów:** `docker-compose down`

Adres dokumentacji swagger: <http://localhost:8080/swagger-ui/index.html>