WAREHOUSE-REST

David Socaciu 4DHIT 16.09.2025

Grundlagen

Bevor ich angefangen habe, habe ich zuerst das Projekt auf Github geforked und anschließend in IntelliJ geklont. Anschließend habe ich das build.gradle ausgeführt und angefangen die Schritte zu erledigen.

Implementierung eines Generator zur Erzeugung der Standort- und Warendaten

Als erstes habe ich die Klasse ProductData erzeugt die Attribute, Konstruktor sowie die Setter- und Gettermethoden implementiert, folgende Attribute habe ich verwendet:

```
    private String productID;
    private String productName;
    private String productCategory;
    private int productQuantity;
    private String productUnity;
```

Anschließend habe ich die Methode getData aus der Klasse WarehouseSimulation bearbeitet um eine Generierung der Produkte zu ermöglichen.

Ich habe in einer Arrayliste 6 Produkte hinzugefügt und anschließend habe ich mithilfe der nextBoolean() Methode folgendes implementiert:

```
        while (actualProducts.size() < 2) {
            for(ProductData productData : productDataList) {
                if (random.nextBoolean()) {
                    actualProducts.add(productData);
                }
            }
        }
```

Ich habe eine zweite Arrayliste erstellt, in welche mindestens 2 der 6 Produkte hinzugefügt werden mithilfe der while Bedingung. Die nextBoolean() Methode hat mir geholfen um mir zufällig die Anzahl der Produkte auszuwählen.

Daten im JSON und XML Format zur Verfuegung stellen

Die Daten waren im JSON bereits abrufbar unter: http://localhost:8080/warehouse/001/json

Um die Daten im XML-Format anzuzeigen zu lassen, fehlt mir bislang eine Dependency die ich in die build.gradle einfügen musste

```
implementation 'com.fasterxml.jackson.dataformat:jackson-dataformat-xml'
```

und anschließend die build.gradle neu ausgeführt, sowie eine neue Methode im Controller die fast identisch mit der Methode für die JSON ist.

```
    @RequestMapping(value="/warehouse/{inID}/xml", produces =
MediaType.APPLICATION_XML_VALUE)
    public WarehouseData warehouseDataXML( @PathVariable String inID ) {
        return service.getWarehouseData( inID );
    }
```

Der Unterschied war das man beim Value /xml hinschreibt und bei MediaType XML statt JSON.

Implementierung der REST Schnittstelle

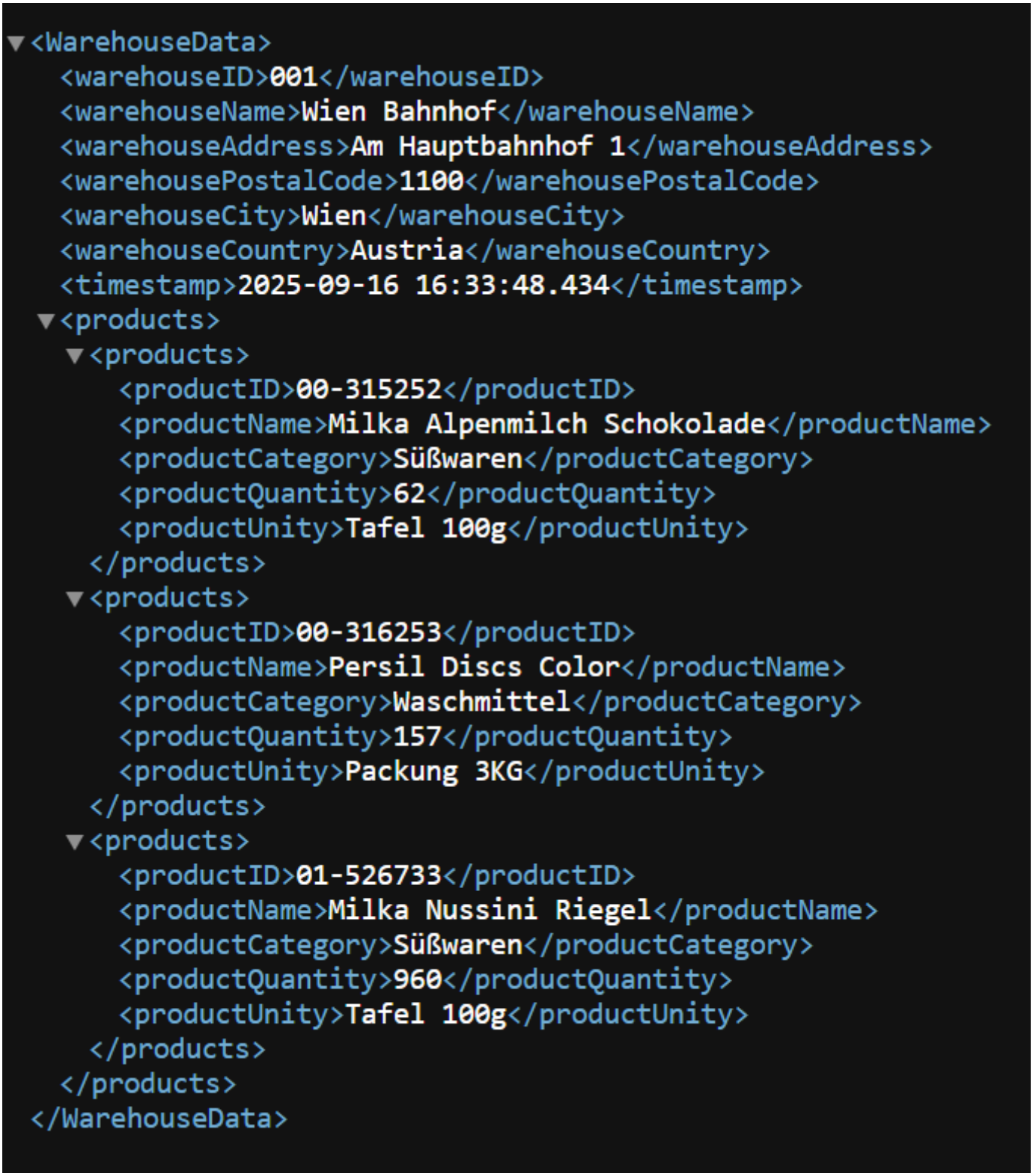
Nun kann man unter localhost:8080 folgendes sehen:

This is the warehouse application! (DEZSYS_WAREHOUSE_REST)

Link to JSON warehouse/001/json
Link to XML warehouse/001/xml

Hier sind 2 Beispiele, eines vom XML und eines vom JSON Dateiformat:

```
{
  "warehouseID": "001",
  "warehouseName": "Wien Bahnhof",
  "warehouseAddress": "Am Hauptbahnhof 1",
  "warehousePostalCode": 1100,
  "warehouseCity": "Wien",
  "warehouseCountry": "Austria",
  "timestamp": "2025-09-16 16:33:45.364",
  "products": [
    {
      "productID": "00-871895",
      "productName": "Bio Apfelsaft Gold",
      "productCategory": "Getraenk",
      "productQuantity": 293,
      "productUnity": "Packung 1L"
    },
    {
      "productID": "00-316253",
      "productName": "Persil Discs Color",
      "productCategory": "Waschmittel",
      "productQuantity": 383,
      "productUnity": "Packung 3KG"
    },
    {
      "productID": "00-315252",
      "productName": "Milka Alpenmilch Schokolade",
      "productCategory": "Süßwaren",
      "productQuantity": 941,
      "productUnity": "Tafel 100g"
    }
  ]
}
```



Erweiterte Grundlagen

Zuerst habe ich hierfür eine index.html Datei mit dem Grundgerüst erstellt und zuerst mich auf die Daten von einem Warehouse und noch nicht die Produktdaten fokussiert habe.

```
<div id="warehouseinfos">
            <p id="id"></p>
            <p id="warehouseid"></p>
            <p id="name"></p>
            <p id="address"></p>
            <p id="postalcode"></p>
            <p id="city"></p>
            <p id="country"></p>
            <p id="timestamp"></p>

    </div>
```

Anschließend habe ich eine Tabelle erstellt welche später dann die Produkte im Warenhaus anzeigen wird.

```
        <table>
            <thead>
                <tr>
                    <th>Product-ID</th>
                    <th>Product-Name</th>
                    <th>Product-Category</th>
                    <th>Product-Quantity</th>
                    <th>Product-Unity</th>
                </tr>
            </thead>
            <tbody id="tablebody">

            </tbody>

        </table>
```

Dabei habe ich den Body der Tabelle mit einer ID ausgestattet und leer gelassen um später dann die Produkte mithilfe von Jquery ergänzen.

Als letztes aber am wichtigsten habe ich mithilfe von AJAX (JQuery) die Produktdaten geholt.

```
            $.ajax({
                url: 'http://localhost:8080/warehouse/001/json',
                method: 'GET',
                success: function(data) {
```

Als URL habe ich das 1. Warehouse als JSON genommen, da wir in diesem Beispiel nur ein Warehouse erstellen mussten, die Methode ist GET da ich Daten bekommen will und wenn es funktioniert passiert folgendes:

Zuerst werden die Daten für das Warehouse geladen, und anschließend werden die Produkte geladen auf die ich näher eingehe

```
for(let i = 0; i < data.products.length;i++){
   let row = $("<tr>");
   row.append($("<td>").text(data.products[i].productID));
   row.append($("<td>").text(data.products[i].productName));
   row.append($("<td>").text(data.products[i].productCategory));
   row.append($("<td>").text(data.products[i].productQuantity));
   row.append($("<td>").text(data.products[i].productUnity));
   $("#tablebody").append(row);
}
```

Mithilfe von einer for-Schleife gehe ich durch die ganzen Produkte durch und füge einer Zeile nacheinder die einzelnen Attribute hinzu.

Für die Filter habe ich eine Eingabe gemacht wo man nach einem Produkt suchen kann und die Eingabe wird mit den Produktnamen in der Tabelle verglichen wird und nur die Produkte in der Tabelle angezeigt werden die mit der Eingabe übereinstimmen.

Quellen

XML Serialization and Deserialization with Jackson | Baeldung