-->

# Milestone 0 (M0) - A simple shop system

Your software development company has been contracted to delevop an eCommerce system for a small online shop. The shop sells a number of **items**, which are not perishable and not fragile. Each item has a name, description, and size. In addition, it has a buying price, and a sales price. Shop admins can manually add or delete items to / from the catalogue.

The shop sells items to **users**. A user has a name, email address, and physical address (with street, city, and postal code). There is no username, i.e. a user can only be identified by his/her email address.

When a user buys items, he/she puts them in their personal **shopping cart** first. The shopping cart for a user consists of one or several **shopping cart parts**. A shopping cart part references exactly one item, and contains the quantity of that item. (At the time when items are put in the shopping cart, we assume that they are available in sufficient quantity. The actual availability is checked later.) The user can remove items from the shopping cart, or proceed to check out.

When a user checks out the shopping cart, an **order** is created. Similar to the shopping cart, each order consists of one or several **order parts**. A order part references exactly one item, and contains the quantity of that item in the order.

The shop has several **fulfillmentCenter centers**. A fulfillmentCenter center has name and physical address (with street, city, and postal code). Each order is served by a dedicated fulfillmentCenter center. This is determined by the first postal code digit of user who made the order. A fulfillmentCenter center has a range of postal code digits it serves. Let's make an example: Linda Mittelstedt lives in 41061 Mönchengladbach. She creates an order in the shop. The shop has three fulfillmentCenter centers A, B, and C. A has range "0 - 3", B has range "4 - 8", and C has "9 - 9". Then Linda's order is served by fulfillmentCenter center B. (If several fulfillmentCenter centers have a fitting range, the shop can take any of them.)

To each fulfillmentCenter center belongs a number of **stock levels**, each referring to exactly one item. A stock level contains the quantity of items that this fulfillmentCenter center currently stores. This quantity can also be 0, i.e. the items is not available in this fulfillmentCenter center. Shop admins can manually change the stock level quantity for items. Orders containing out-of-stock items will be cancelled and deleted automatically. (This will change in the future, but currently there is no switch to another fulfillmentCenter center.)

For simplicity reasons, item returns will not be handled by the system for now, but are done manually, outside the system between shop admin and user. So this is out of scope for this ecommerce system.

*(This is what you need to model in this first milestone. Additions come later.)*

### Glossary

To make things easier, here is an English-German glossary of the terms used above. Remember that
in your code, you need to use the **English** terms.

| English | German |
|---|---|
| item | zu verkaufende Ware |
| catalogue | Produktkatalog, -verzeichnis |
| perishable | verderblich |
| fragile | zerbrechlich |
| buying price | Einkaufspreis |
| sales price | Verkaufspreis |
| user | Kunde |
| email address | Email des Kunden |
| postal code | Postleitzahl |
| shopping cart | Warenkorb |
| to check out | zur Kasse gehen |
| order | Bestellung |
| order part | Bestellung-position |
| fulfillmentCenter center | Auslieferungszentrum |
| stock levels | Warenbestände |
| item returns | zu verkaufende Ware-Rücksendungen |

## Tasks in M0

### E1: Specify a Logical Data Model (Logisches Datenmodell, LDM)

Create a logical data model (LDM) for the store system. The LDM should map the entities, value objects and relationships described in the text above. Hint: The bold (fettgedruckt)

terms in the text above should help.

Store your solution in the `exercises` directory. There is already a template called `E01solution.uxf`.

**IMPORTANT**: You must adhere to a few rules, so that the test can check your solution:

- Keep the file name the same, otherwise the test will not find your diagram. Please also do not create variants, such as `E02_corrected.uxf` or similar.
- You do not have to upload a diagram this time - unless you have questions about your diagram.

## E2: Implement a minimal version (MVP) for the store system

Implement a prototype for the eCommerce system. The functionality of the prototype are defined by **four interfaces**. You will find them under `thkoeln.archilab.ecommerce.usecases`. You need to implement the methods of these interfaces.

The best way to do this is to implement one or more Spring Boot Services, e.g. like this:

```
@Service
public class MyFantasticShopSystem implements UserRegistrationUseCases {
    //...
}
```

How you implement the use cases is up to you in M0. However, you must implement the domain entities from the LDM (this will be tested). In addition, you must use Spring JPA to persist the data.

**IMPORTANT**: Please follow these rules so that the test can check your implementation:

1. The implementation code must located within the package `thkoeln.archilab.ecommerce.solution`. You can create any number of sub-packages and classes as you like.
    - In subsequent milestones, we will test for adherence to the DDD package structure guidelines that we use in ST2.
    - If you want to be prepared, follow the guidelines from the beginning.
    - In short: the package structure should reflect the domain model. One main package for each aggregate root (= "main business entity"),
      and below that sub-packages `domain` and `application` for the respective layers.
    - In the `domain` package, you have entities, value objects, and repositories.

    - In the `application` package, you have application services, DTOs, and controllers.
2. Your implementation must implement **all** interfaces from `thkoeln.archilab.ecommerce.usecases`.
3. All functional tests in this milestone are visible to you. You can therefore test locally.
4. The best starting point for your implementation is to follow the interfaces methods, and not necessarily look at the tests (they will be confusing and hard to understand at this point). A suitable start point is e.g. `UserRegistrationUseCases`. Pick the easiest interface (with least dependencies) first, and then go on from there.
5. Write your own tests, in addition to the ones provided. This will actually help you getting your tasks done quicker. And we will check on your own tests in subsequent milestones.
6. We assume that you use JPA, i.e. entities and repositories. The naming convention for classes is as introduced in ST1 (camel case). Repositories are named like the corresponding entities, with the suffix `Repository`.