Exercise 02 - Decomposing the ProductSrv

Required Services

The following services are involved and have to be started before the final exercise validation:

- CustomerSrv (`http://localhost:8000`)
- NotificationSrv (`http://localhost:8010`)
- OrderProcessSrv (`http://localhost:8020`)
- OrderSrv (`http://localhost:8030`)
- ProductSrvFacade (`http://localhost:8040`)
- ProductSrv (`http://localhost:8050`)
- CategorySrv (`http://localhost:8060`)
- WarehouseSrv (`http://localhost:8070`)
- WebUI (`http://localhost:5000`)
- Apache Zookeeper (`localhost:2181`, starts automatically in the provided Ubuntu VM)
- Apache Kafka (`localhost:9092`, starts automatically in the provided Ubuntu VM)

Description

The `ProductSrv` has grown over time and is now fairly large compared to the other services. It is responsible for several different entities, namely products, product categories, and the available amount of product copies in the warehouse. The following resources are currently provided:

```
GET     /categories (experiment.webshop.products.resources.ProductCategoryResource)
POST    /categories (experiment.webshop.products.resources.ProductCategoryResource)
DELETE  /categories/{id}
(experiment.webshop.products.resources.ProductCategoryResource)
GET     /categories/{id}
(experiment.webshop.products.resources.ProductCategoryResource)
PUT     /categories/{id}
(experiment.webshop.products.resources.ProductCategoryResource)

GET     /products (experiment.webshop.products.resources.ProductResource)
POST    /products (experiment.webshop.products.resources.ProductResource)
DELETE  /products/{id} (experiment.webshop.products.resources.ProductResource)
GET     /products/{id} (experiment.webshop.products.resources.ProductResource)
PUT     /products/{id} (experiment.webshop.products.resources.ProductResource)

GET     /products/{id}/availability
(experiment.webshop.products.resources.WarehouseResource)
PUT     /products/{id}/availability
(experiment.webshop.products.resources.WarehouseResource)
```

The lead developer has decided to split up the `ProductSrv` to increase maintainability and scaling efficiency. Two new services will be created: A `CategorySrv` handling product categories and a `WarehouseSrv` responsible for

product availability. The CRUD operations related to products will remain in the `ProductSrv`. Runnable skeleton projects for the new services have already been created, they just provide no resources yet.

Since this change has been expected some months ago, precautions have been taken. First, the different capabilities of the `ProductSrv` have already been decomposed in separate classes (one `Resource` and one `Repository` class per capability). And second, the `ProductSrvFacade` shields the `ProductSrv` from all consumers so that it will be the only component that has to be adjusted with the new URLs.

Tasks

1. **Move the product category related functionality.** Move all functionality related to product categories from the `ProductSrv` to the new `CategorySrv`. It already has a resource class (`experiment.webshop.categories.resources.ProductCategoryResource`) and a repository class (`experiment.webshop.categories.db.ProductCategoryRepository`) that have to be replaced (you have to adjust the package name of the copied file though). All necessary model classes are already present in `experiment.webshop.categories.api` (you simply have to adjust the import statements in the copied files). Be sure to also remove the instantiation of the now missing functionality from the `ProductSrv`, i.e. from the `run()` method of the `experiment.webshop.products.ServiceApplication` class. In the end, the following resources should be provided by the new `CategorySrv` instead:

```
GET     /categories (experiment.webshop.categories.resources.ProductCategoryResource)
POST    /categories (experiment.webshop.categories.resources.ProductCategoryResource)
DELETE  /categories/{id}
(experiment.webshop.categories.resources.ProductCategoryResource)
GET     /categories/{id}
(experiment.webshop.categories.resources.ProductCategoryResource)
PUT     /categories/{id}
(experiment.webshop.categories.resources.ProductCategoryResource)
```

2. **Move the product availability related functionality.** Move all functionality related to product availability from the `ProductSrv` to the new `WarehouseSrv`. It already has a resource class (`experiment.webshop.warehouse.resources.WarehouseResource`) and a repository class (`experiment.webshop.warehouse.db.WarehouseRepository`) that have to be replaced (you have to adjust the package name of the copied file though). All necessary model classes are already present in `experiment.webshop.warehouse.api` (you simply have to adjust the import statements in the copied files). Be sure to also remove the instantiation of the now missing functionality from the `ProductSrv`, i.e. from the `run()` method of the `experiment.webshop.products.ServiceApplication` class. In the end, the following resources should be provided by the new `WarehouseSrv` instead:

```
GET     /products/{id}/availability
(experiment.webshop.warehouse.resources.WarehouseResource)
PUT     /products/{id}/availability
(experiment.webshop.warehouse.resources.WarehouseResource)
```

3. **Fix the `ProductSrvFacade`.** Since we moved functionality to new endpoints, we need to adapt the invocation URLs for service consumers. This can be done centrally at the `ProductSrvFacade` in the

`experiment.webshop.products.resources.ProductFacadeResource`. In addition to a `PRODUCT_SRV_ENDPOINT` variable, endpoint variables for the new services already exist. For category and warehouse operations, these new endpoint variables have to be used when building the HTTP requests.

Validation

When you are finished with all tasks, make sure all required services (see Required Services) and the exercise validation UI is up and running (if not, execute `exercise-validation/build-and-run-validation-ui.sh`) and then navigate to `http://localhost:5001` (**it is important to start from this page, because it will determine which version you are working on**). Click on `Exercise 02` and then on `Start Validation`. If every check is successful (`status: true`), pause your stopwatch and notify an experiment admin for the manual validation part and to write down your time.