# Documentation

Table of Contents

[Documentation 1](#_Toc35260690)

[Setup 2](#_Toc35260691)

[Importing the project 2](#_Toc35260692)

[Host and Port 2](#_Toc35260693)

[Jars and copying 2](#_Toc35260694)

[Application 3](#_Toc35260695)

[The services 4](#_Toc35260696)

[HTTP Methods 4](#_Toc35260697)

## Setup

### Importing the project

Please unzip the project folder and import the project as a whole into IntelliJ IDEA (as gradle project).

Next go into project settings -> modules, remove the parent module and import all the modules individually into the project as gradle modules.

After this is done, in the gradle menu, click on reimport all gradle projects (refresh icon).

Up next add a run/debug configuration for tomcat server (local). Under deployment, add all artifacts (war files) to be deployed at server startup. These are: products, users, webshop, and orders.

Make their Application context match as: /products, /users, /webshop, and /orders. Apply and confirm the made changes.

Now you should be able to press the run button and tomcat will be started with the configurations in place.

For the client GUI, it is important to go into preferences -> build, execution, deployment -> build tools -> gradle.

On the right, under gradle projects, select client and select build and run using IntelliJ IDEA such that the GUIs can be constructed based on the design tool that was used with IntelliJ IDEA.

### Host and Port

To set up the application’s host address and port please look into the gradle module named: request-response. This module contains a class named Url that holds all the information to create URLs for specific services. So in case you host your tomcat service somewhere else, please change them accordingly.

Also do not forget to create the JAR of that module and copy it over to the other modules, in case the default host and port were modified.

### Jars and copying

#### Users

Users has a custom task to create a jar file of its distributed models. The users module also has several copying tasks that should be used to copy the jar into the modules that use it:

* webshop (copyModelJarToShop)
* client (copyModelJarToClient)

#### Request-response

This library is used to reduce copying and pasting code that has to do with providing user made descriptions or error classes into a response. Besides that, it is used as the central point for managing URL related configuration.

The JAR is created by a regular jar task as all classes inside it are shipped to other modules. The request-response jar needs to be copied over to several modules aswell:

* client (copyClassJarToClient)
* orders (copyClassJarToOrders)
* products (copyClassJarToProducts)
* users (copyClassJarToUsers)

#### Products

Products has a custom task to create a jar file of its distributed models. The product module is required in several other modules and copied by (task) respectively:

* client (copyModelJarToClient)
* webshop (copyModelJarToWebShop)
* orders (copyModelJarToOrders)

#### Orders

Orders has a custom task to create a jar file of its distributed models. The orders module is required in several other modules and copied by (task) respectively:

* client (copyModelJarToClient)
* webshop (copyModelJarToShop)

## Application

The application consists of three GUIs. The application would start with the UserAccount application that provides login functionality, the user can choose to register by clicking a button which will open the RegisterAccount. After registering successfully, the user can return to the login form and log in to make use of the shop (OrderApplication).

Any newly created account for now will have a Role of customer, this results in the user not being able to manage products.

The tester of the system can however run the order application’s main method directly to enter the shop as a user with a CATEGORY\_MANAGER role, allowing to play around with all functionality.

All provided GUI forms/applications can be run individually, while the UserAccount (login form) is the main entry point and allows the user to experience all functionality.

For this reason and for testing purposes, one hard coded user account with a CATEGORY\_MANAGER role is added for the user to test functionality.

## The services

The application consists of three services in total, of which: users, orders, and products. These three are combined and in turn provided by the webshop service which serves as a composite service, providing the client (java GUIS) with appropriate responses.

### HTTP Methods

All methods provided by the webshop are also available through their direct services with the same Uris except that they are of the form: <resource>/rest/<addition>

Thus ex: /products/rest/find?name=soep

|  |  |  |
| --- | --- | --- |
| **Webshop** | | |
| ***Method*** | ***URI*** | ***Description*** |
| GET | /webshop/rest/products | Returns a collection of all available products in the shop |
| POST | /webshop/rest/products | Add a new product to the shop's available products |
| GET | /webshop/rest/products/sort | Returns a sorted collection of products based on query parameter ex: method=price |
| GET | /webshop/rest/products/find | Returns a collection of products that match the query parameter string ex: name=soep |
| GET | /webshop/rest/products/{id} | Returns the product with matching id |
| PUT | /webshop/rest/products/{id} | Updates the product with matching id based on provided form parameters: name, unit, quantity, price. Not all are required |
| DELETE | /webshop/rest/products/{id} | Removes the product with matching id from the shop's available products |
| POST | /webshop/rest/products/discount/{id} | Applies the given discount (int) given by form parameter: percentage to product matching the id |
|  |  |  |
| GET | /webshop/rest/users | Returns the collection of all registered users |
| POST | /webshop/rest/users/login | Requires two parameters: name and password, returning the user object if successful login |
| POST | /webshop/rest/users | Creates a new user based on for parameters: name, password, confirm. Returning the user object on success |
|  |  |  |
| GET | /webshop/rest/orders/{orderNumber} | Returns the order object for given orderNumber |
| POST | /webshop/rest/orders | Creates a new empty order object for user matching form parameter: userId |
| POST | /webshop/rest/orders/{orderNumber} | Stores the received order object into the collection of orders |