

Enginyeria del Programari de
Components i Sistemes
Distribuïts

Project Stub IDE Example

Laboratory



Grau en Enginyeria Informàtica (GEI)
Application Development (GTAS)
Computer Science and Multimedia Faculty
March 2025
Version 5.0en

Table of Contents

About this project	3
Docker	4
Installation	4
Docker Desktop / Docker Compose installation	4
Basic infrastructure (dockers).....	4
Microservices IDE.....	6
Open JDK	6
Eclipse	6
Installation	6
Spring Boot Tools.....	7
Java JDK.....	7
Lombok Project	8
IntelliJ IDEA Ultimate	9
Installation	9
Microservices Stubs.....	10
Run with Eclipse.....	10
Run with IntelliJ	11
Test if they work.....	12
How to generate the .jar.....	13
From Eclipse.....	13
From IntelliJ	14
Postman / Swagger Installation	15
Postman	15
Installing and running Tests	19
Installing	19
Running	20
Running from Eclipse.....	21
Running from IntelliJ	22

About this project

This is the lab project for the SA course at the UOC. This project is available in the GitHub organization <https://github.com/UOC-EPCSD-SPRING-2025>.

Lab Project is made up of 3 elements (each one in its own GIT repository):

- A [docker-compose.yml](#) file to start up the basic infrastructure needed to run the services
- A folder for the [ProductCatalog](#) microservice
- A folder for the [User](#) microservice
- A folder for the [Notification](#) microservice

Made with:

- [Docker](#) / [Docker Compose](#)
- [Spring](#) / [Spring Boot](#)
- [Maven](#)
- [Apache Kafka](#)
- [PostgreSQL](#) / [Adminer](#)

Before starting:

To set up the containers that are part of the basic infrastructure of the project, the following ports will be used:

- 22181 - Apache Kafka (Zookeeper)
- 19092 - Apache Kafka (Server)
- 54320, 54321 - PostgreSQL
- 18080 - Adminer
- 18081 - Used by the productcatalog microservice
- 18082 - Used by the user microservice

To avoid conflicts with other installed applications, the default ports of all applications have been modified. Still, if there is a conflict over a port already in use, simply modifying the ports specified in the [docker-compose.yml](#) file will fix the problem. This link to the official docker compose documentation explains how to modify this configuration using the *ports*: [Networking in Compose](#) option.

IMPORTANT NOTICE: *The modified ports will also have to be changed in the microservices configuration (usually defined in the Spring application.properties file).*

Docker

Installation

Docker Desktop / Docker Compose installation

Proceed to install Docker Compose following the steps described in the following guide: <https://docs.docker.com/compose/install/> (according to your OS).

Under Windows, registration may be required, as [Docker Desktop](#) requires it for educational/personal/non-commercial projects. On the plus side, it will not be necessary to install anything else because it already includes *Compose*.

It is important that you carefully review the hardware and software requirements described in the installation guides. If your system fails to meet them, even after a successful installation, you will see errors when trying to start containers. An alternative for those with slightly older systems is [Docker Toolbox](#) or also [Podman](#).

Once Docker Compose is installed, we will continue with the project stub. It is recommended to set up a folder structure like so:

```
Spring-2025
├─ README.md
├─ docker-compose.yml
├─ spring-2025-notification
├─ spring-2025-productcatalog
└─ spring-2025-user
```

Basic infrastructure (dockers)

Download the code in ZIP format or just clone the [spring-2025](https://github.com/UOC-EPCSD-SPRING-2025/spring-2025) (<https://github.com/UOC-EPCSD-SPRING-2025/spring-2025>) repository in the working folder (spring-2025 if the recommendation has been followed).

From the work folder, run the command:

```
docker compose up -d (Win)
docker-compose up -d (Linux)
```

The following containers should start:

- *spring-adminer_1* (adminer, an SQL client)
- *spring-kafka_1* (the Kafka server)
- *spring-productdb_1* (the PostgreSQL database for the productcatalog service)
- *spring-userdb_1* (the PostgreSQL database for the user service)

- *spring-zookeeper_1* (Kafka zookeeper)

To verify that all containers are up and running, we will execute the following command in a different terminal session:

```
docker ps -a
```

We should see something like this:

```
C:\SPRING-2025>docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
9a5b02e0873d	confluentinc/cp-kafka:latest	"/etc/confluent/docker..."	About a minute ago	Up 17 seconds	0.0.0.0:19092->19092/tcp, 0.0.0.0:29092->9092/tcp	spring-2025-kafka-1
f24e074e44ea	postgres	"docker-entrypoint.s..."	About a minute ago	Up 18 seconds	0.0.0.0:54320->5432/tcp	spring-2025-productdb-1
7906d6d1c1b6	adminer	"entrypoint.sh docke..."	About a minute ago	Up 18 seconds	0.0.0.0:18080->8080/tcp	spring-2025-adminer-1
f4eb5a42f488	postgres	"docker-entrypoint.s..."	About a minute ago	Up 18 seconds	0.0.0.0:54321->5432/tcp	spring-2025-userdb-1
f073e7c0d978	confluentinc/cp-zookeeper:latest	"/etc/confluent/docker..."	About a minute ago	Up 18 seconds	2888/tcp, 3888/tcp, 0.0.0.0:22181->22181/tcp	spring-2025-zookeeper-1

```
C:\SPRING-2025>
```

To check the operation, you can access the *Adminer* panel at <http://localhost:18080/> and make a query against the PostgreSQL DB that we have just instantiated with the following connection data:

- *System: PostgreSQL*
- *Server: productdb*
- *Username: product*
- *Password: product*
- *Database: product*

Language: English ▼

Adminer 4.8.1

(PostgreSQL) product@productdb -

Login

System	PostgreSQL ▼
Server	productdb
Username	product
Password
Database	product

☐ Permanent login

Microservices IDE

In order to carry out the Microservices part, **you can set up the work environment most appropriate to your preferences or knowledge.**

Here, and as an example, a work environment based on Eclipse another on IntelliJ is presented.

To do this, we need to install the following software:

- Open JDK
- Eclipse
- IntelliJ
- Spring Tools
- Lombok project

Open JDK

This IDE uses OpenJDK version 11.0.20.8. Any other version will surely work too.

To install it, you must first download the program from its website: <https://jdk.java.net/archive/>, or from <https://www.openlogic.com/openjdk-downloads>

For the Windows version you can find the .msi at:

<https://builds.openlogic.com/downloadJDK/openlogic-openjdk-jre/11.0.20+8/openlogic-openjdk-jre-11.0.20+8-windows-x64.msi> or the ZIP at:

https://download.java.net/java/GA/jdk11/9/GPL/openjdk-11.0.2_windows-x64_bin.zip

Eclipse

Installation

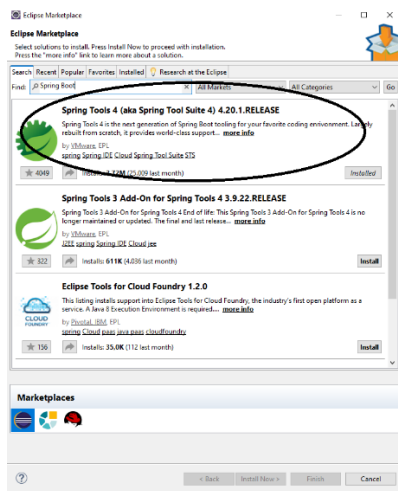
Eclipse can be obtained on the web: <https://www.eclipse.org/downloads/packages/>
Specifically, this tutorial uses the *Eclipse IDE for Enterprise Java and Web Developers version 2024-12-R*, which can be found at:

https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2024-12/R/eclipse-jee-2024-12-R-win32-x86_64.zip

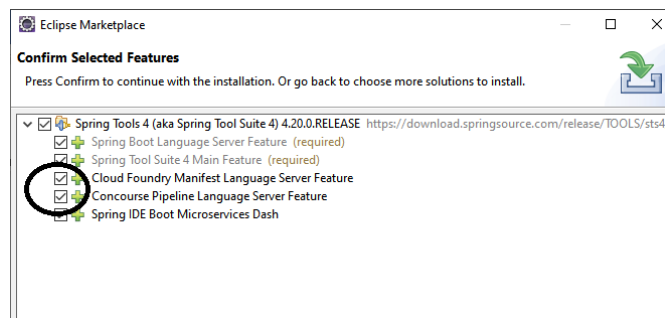


Spring Boot Tools

To install *Spring Boot Tools*, we go to the Eclipse Menu and select: *Help* → *Eclipse Marketplace* → *Find* → *Spring Tools 4*

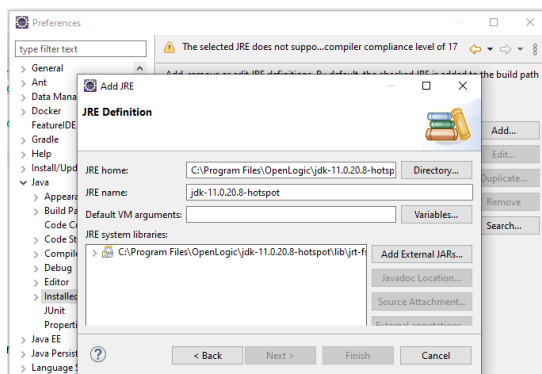


During the installation it is necessary to select the features *Cloud* and *Concourse for Language Server Feature* and finally *Select all Trust Authorities*.



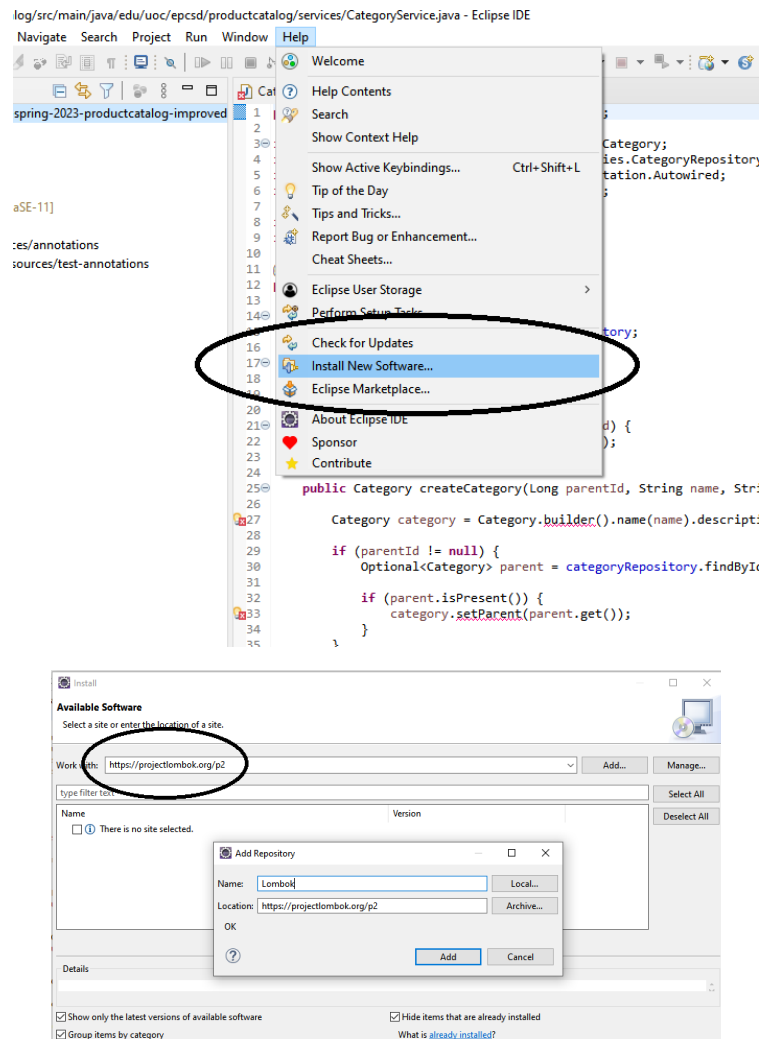
Java JDK

We also need to indicate which Java JDK we are going to use. For this we are going to: *Window* → *Preferences* → *Java* → *Installed JREs* → *Add* → *Standard VM* → *Directory* where our *JDK* is:



Lombok Project

We also need to install Lombok Project. For this we are going to: *Help* → *Install New Software* → *Add* → *Name: Lombok* → *Location: <https://projectlombok.org/p2>*

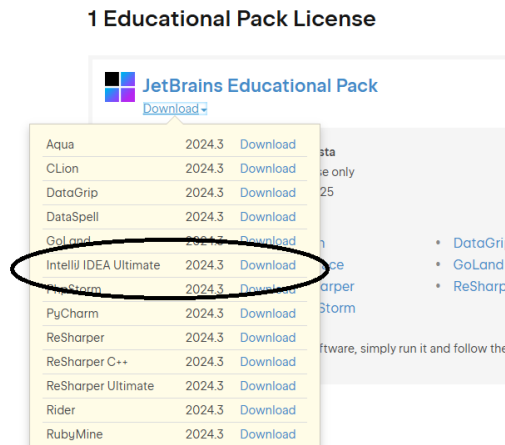


We can get more help at: <https://projectlombok.org/setup/eclipse>

IntelliJ IDEA Ultimate

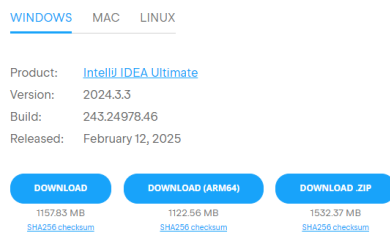
Installation

For the correct functioning of our work environment, we need the version of IntelliJ *Ultimate*, which is the one that allows us to use Spring Tools and Hibernate. This version is paid, but there is a student license with which we can download *Ultimate* for free. To do this, we need to go to <https://www.jetbrains.com/community/education/> and register.

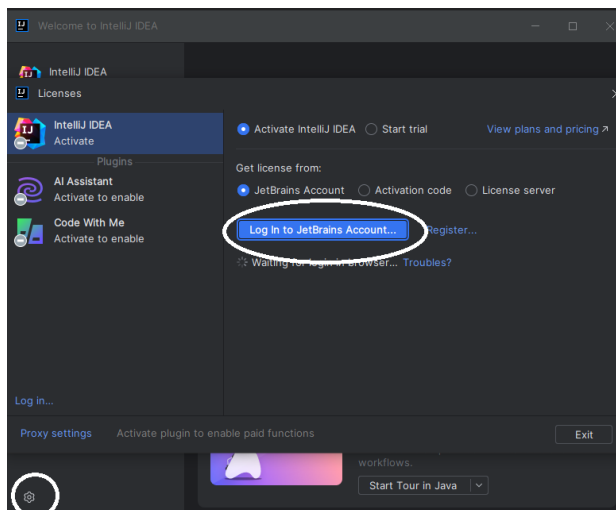


From there we will proceed to download and install the *Ultimate* version:

Download IntelliJ IDEA Ultimate 2024.3.3



The license must be activated before it can be used:



Microservices Stubs

To insert, modify, deploy, and run the microservices from Eclipse and IntelliJ, we will follow the next steps:

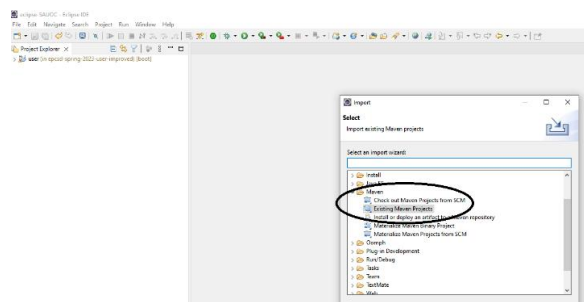
- Download the code in ZIP format or just clone from <https://github.com/UOC-EPCSD-SPRING-2025> , in the repositories:
 - [spring-2025-productcatalog](#)
 - [spring-2025-user](#)
 - [spring-2025-notification](#)

into the working folder on your computer (*spring-2025* if the recommendation has been followed)

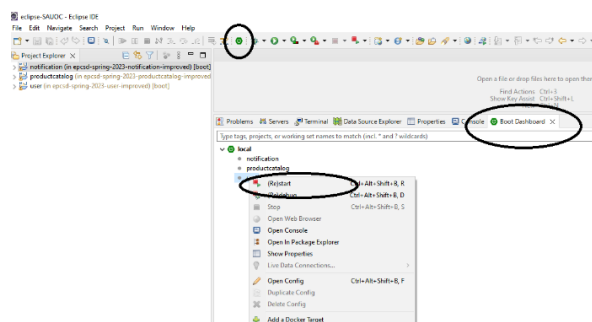
- Make sure that the Docker containers are running.

Run with Eclipse

- Import each microservice into Eclipse as a Maven project: *File* → *New* → *Project* → *Import existing Maven Project*, from your folder computer.

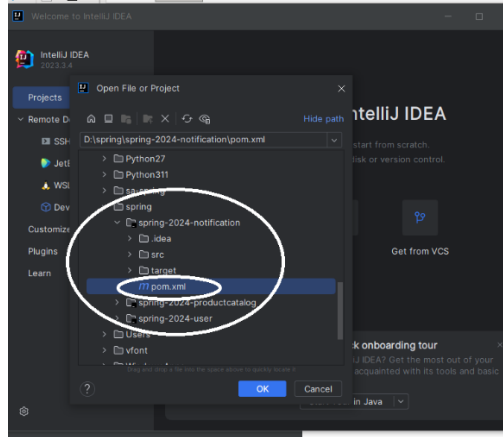


- Boot each microservice as a *Spring Boot* project using the Eclipse option *Boot Dashboard*:

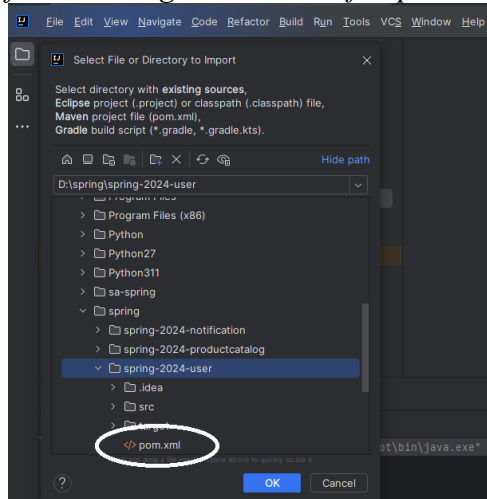


Run with IntelliJ

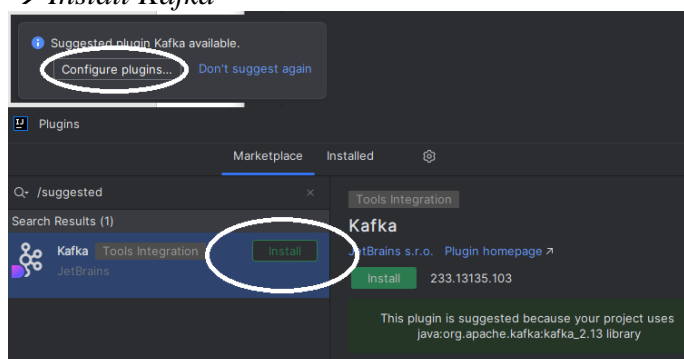
- In the *Welcome* window click on: *Projects* → *Open* → *Search your Microservice project pom.xml file*:



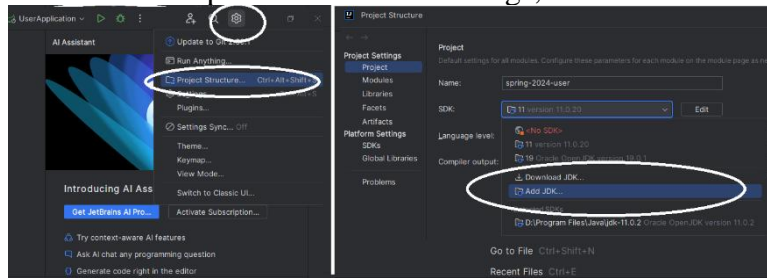
- For the rest of the Microservices click on: *Main Menu* → *File* → *New* → *Project from existing sources...* → *file pom.xml from Microservice* → *New window*.



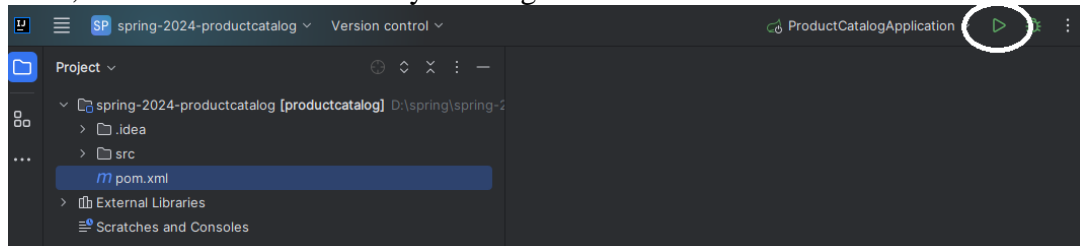
- If the system suggests installing Kafka plugin, you click on: *Configure plugins* → *Install Kafka*



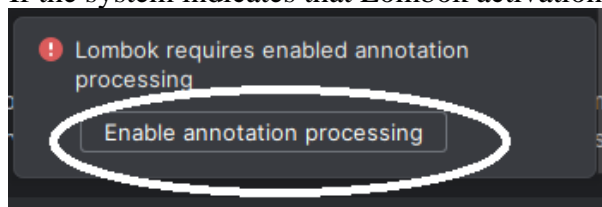
- To select our OpenSDK click on settings, and look for the folder where it is:



- Now, run each Microservices by clicking on its *Run* button:



- If the system indicates that Lombok activation is required, click on “*Enable...*”



Test if they work

Verify that they work correctly by calling the following URLs:

- <http://localhost:18081/swagger-ui/index.html>
- <http://localhost:18082/swagger-ui/index.html>

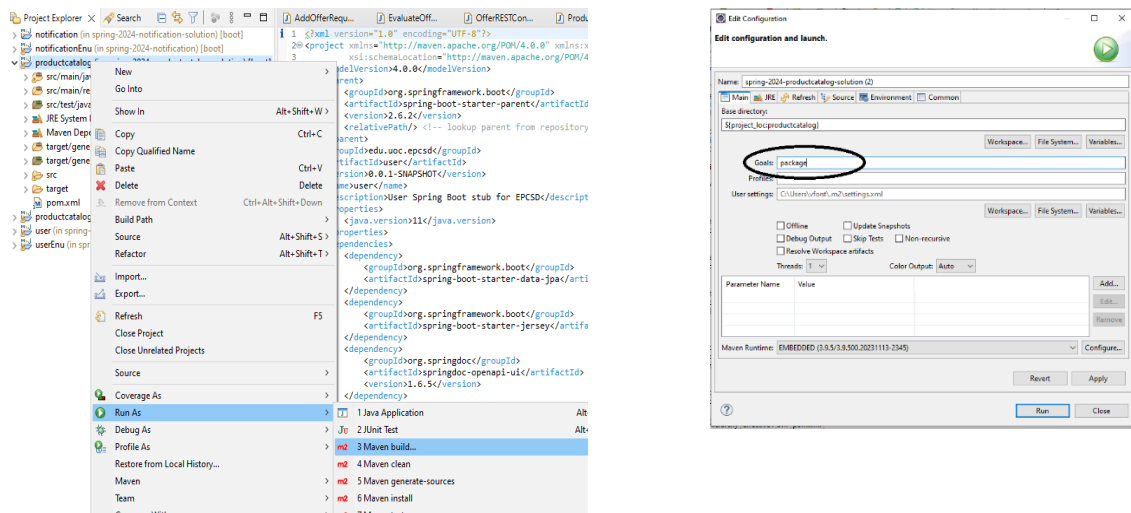
How to generate the .jar

To generate the jar it is necessary to have the Docker images running.

From Eclipse

To generate the jar from Eclipse, open the "Maven build" option in "Run as", and in the Goal option you put "package". Once executed, you can find your jar in the target directory of your project: *C:\spring-2025-productcatalog-solution\target*.

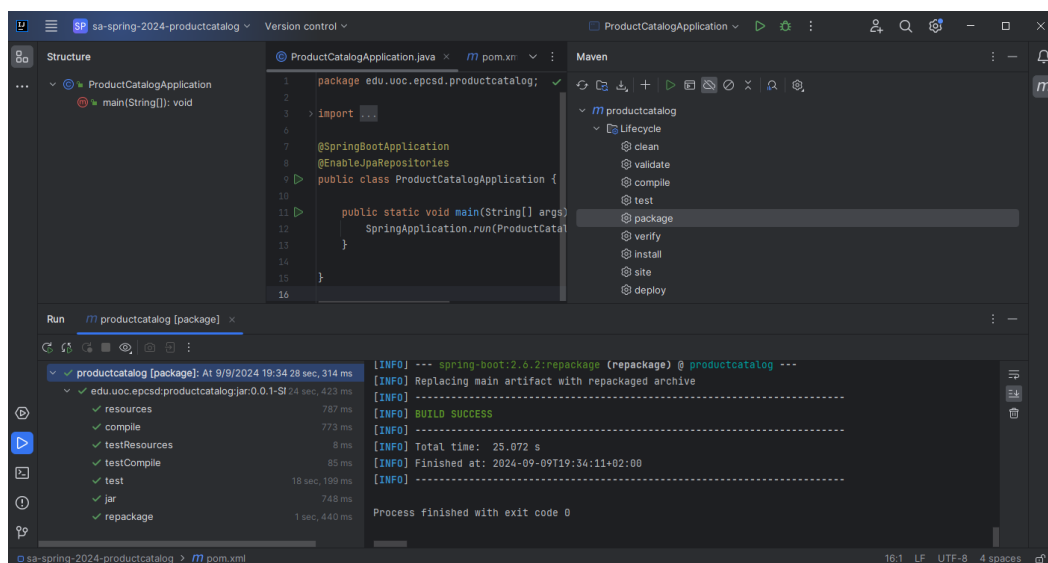
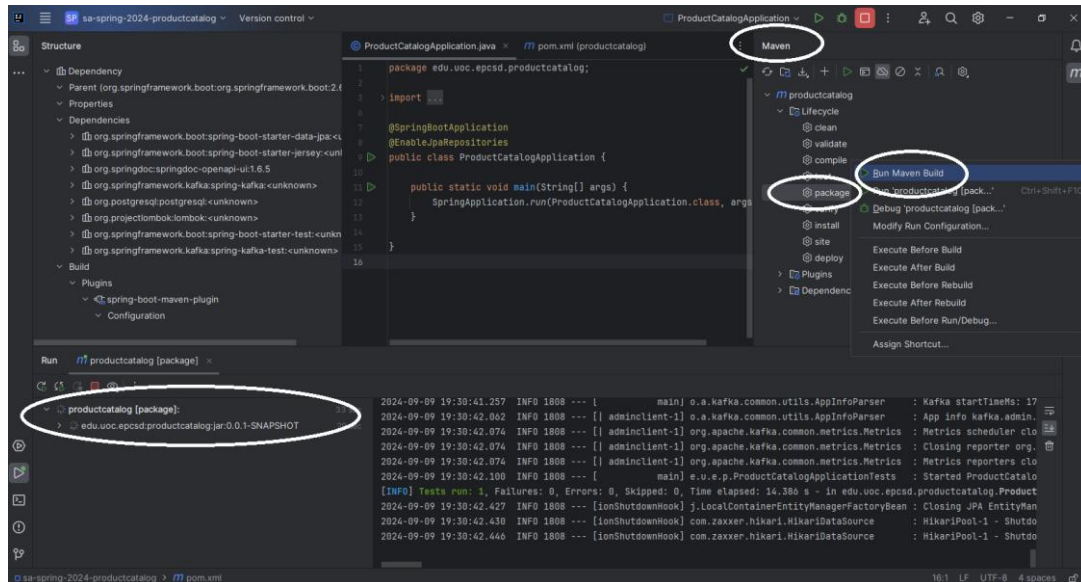
Executing, for example: *C:\spring-2025-productcatalog-solution\target>java -jar productcatalog-0.0.1-SNAPSHOT.jar* should work for you.



Porta-retalls	Organitza	Crea	Obre	Selecciona
← → ↕ ↗ Aquest ordinador > Dades (D:) > prova > spring-2024-productcatalog > target				
OneDrive	Nom	Data de modificació	Tipus	Mida
OneDriveTemp	classes	15/5/2024 12:36	Carpeta de fitxers	
Program Files	generated-sources	15/5/2024 12:36	Carpeta de fitxers	
Program Files (x86)	generated-test-sources	15/5/2024 12:36	Carpeta de fitxers	
prova	maven-archiver	15/5/2024 12:37	Carpeta de fitxers	
spring-2024-productcatalog	maven-status	15/5/2024 12:36	Carpeta de fitxers	
Python	surefire-reports	15/5/2024 12:37	Carpeta de fitxers	
Python27	test-classes	15/5/2024 12:36	Carpeta de fitxers	
Python311	productcatalog-0.0.1-SNAPSHOT	15/5/2024 13:13	Fitxer JAR	65.885 kB
sa-spring	productcatalog-0.0.1-SNAPSHOT.jar.origi...	15/5/2024 13:13	Fitxer ORIGINAL	41 kB

From IntelliJ

To generate the jar from IntelliJ, you must go to the Maven window, expand Lifecycle and select: *Package* → *right click* → *Run Maven Build*



Once executed, you can find your jar in the target directory of your project: *D:\spring-2025-productcatalog\target*.

Executing, for example: *D:\spring-2025-productcatalog\target>java -jar productcatalog-0.0.1-SNAPSHOT.jar*, should work for you.

Postman / Swagger Installation

In addition, taking advantage of the fact that the stubs have their own embedded HTTP server, a *springdoc-openapi* module has been added to auto-generate and publish the microservice definition (API) file in *JSON/OpenAPI v3* format, as well as a simple *SwaggerUI* web interface that allows making calls to the operations of the microservices.

The URLs to access these functionalities are:

- service definition files:
 - *Product Catalog*: <http://localhost:18081/v3/api-docs>
 - *User*: <http://localhost:18082/v3/api-docs>
- SwaggerUI web interfaces for testing:
 - *Product Catalog*: <http://localhost:18081/swagger-ui/index.html>
 - *User*: <http://localhost:18082/swagger-ui/index.html>

Postman

As we have already said, once the implementation is completed, you can obtain the “contract” or definition of each of the services in JSON format at v3/api-docs.

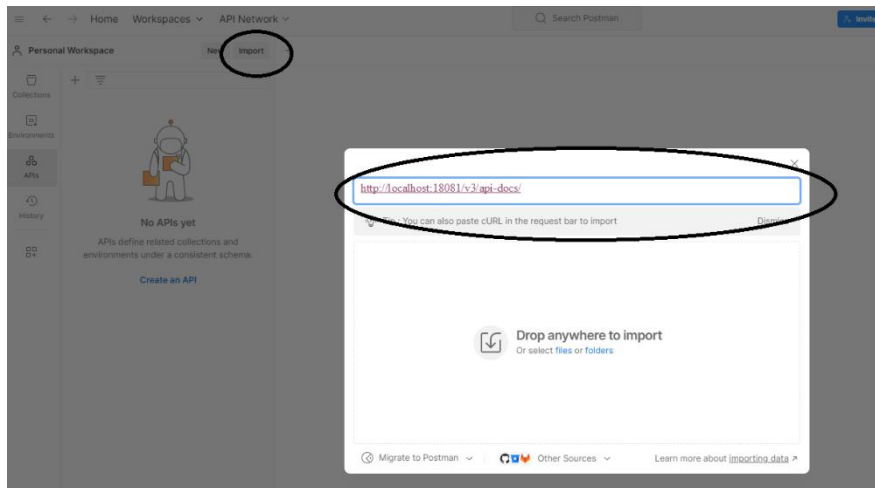


This JSON file can be downloaded or imported directly into an external tool, such as Postman, from where it will be much easier to execute all the necessary tests to verify the correct operation of the different services.

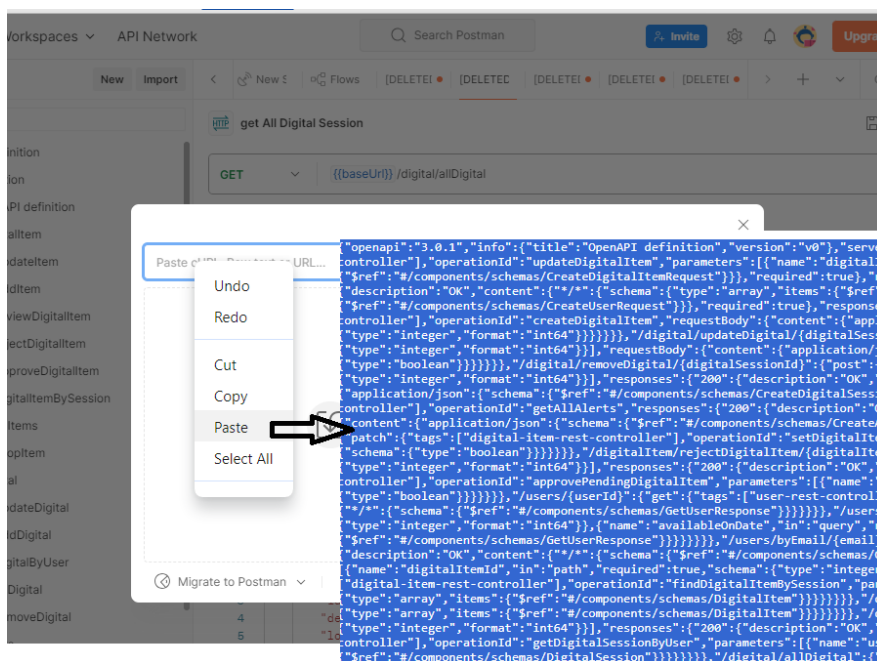
You can find information about Postman in <https://www.postman.com/>, and if you want to use the Postman App, you can find it in:

https://www.postman.com/downloads/?utm_source=postman-home

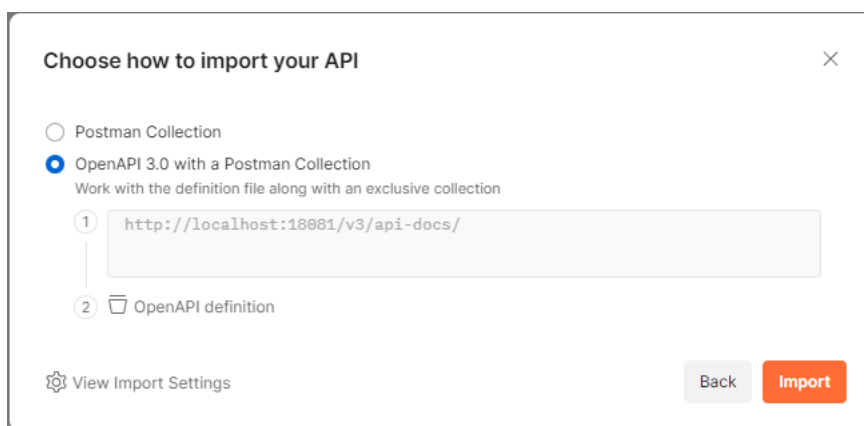
For example, you can import the service definition file of Product Catalog like this:



Or Paste de json file:



You can choose how to import your API:



And then calling: *get All Products*:

The screenshot shows the Postman interface with the 'GET get All Products' request selected in the left sidebar. The URL is `http://localhost:18081/products`. The response is a JSON array of two product objects:

```
1 [
2   {
3     "id": 1,
4     "name": "Canon 5000",
5     "description": "Cámara de fotos Canon 5000",
6     "dailyPrice": 100.0,
7     "brand": "Canon",
8     "model": "5000",
9     "categoryId": 2
10  },
11  {
12    "id": 2,
13    "name": "Canon EOS 80D",
```

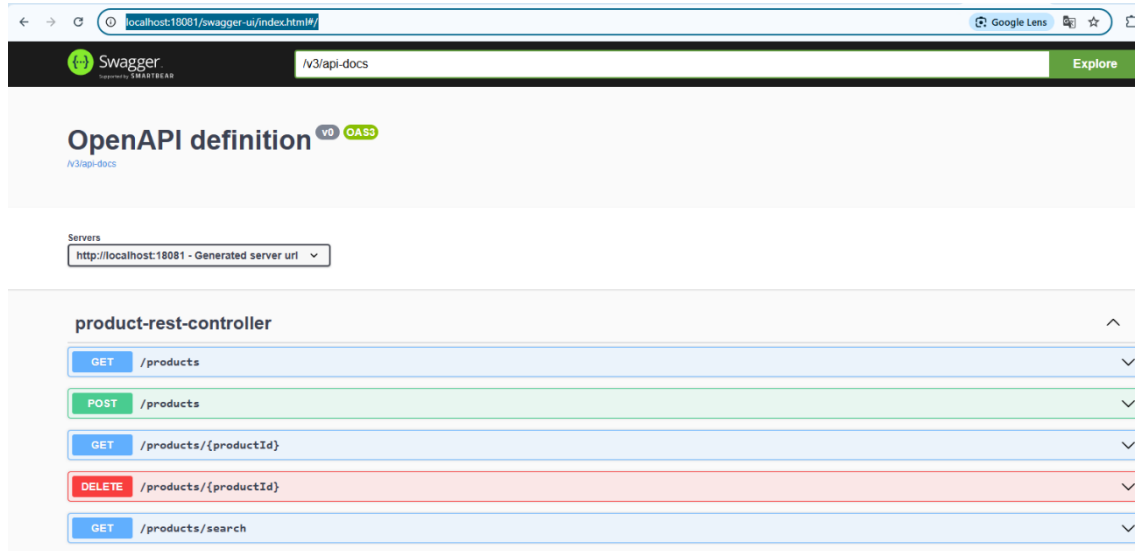
Or *get Products By Id*:

The screenshot shows the Postman interface with the 'GET get Product By Id' request selected in the left sidebar. The URL is `http://localhost:18081/products/productId`. The 'Path Variables' section shows `productId` with the value `1`. The response is a JSON object for a product with id 1:

```
1 {
2   "id": 1,
3   "name": "Canon 5000",
4   "description": "Cámara de fotos Canon 5000",
5   "dailyPrice": 100.0,
6   "brand": "Canon",
7   "model": "5000",
8   "categoryId": 2
9 }
```

Swagger

In order to test the operations of the microservices with Swagger, you just need to click on the link. For example: <http://localhost:18081/swagger-ui/index.html>



Installing and running Tests

Installing

In order to run the tests, it is necessary to have the following programs installed with their respective dependencies: **JUnit**, **Spring Boot Test**, **Mockito**, **AssertJ**, and **ArchUnit**.

You can install them by indicating their dependencies in the file *pom.xml* of the corresponding microservice project.

For example:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>com.tngtech.archunit</groupId>
  <artifactId>archunit-junit5</artifactId>
  <version>1.3.0</version>
  <scope>test</scope>
</dependency>
```

For more information, you can access to these links:

- <https://github.com/anirban99/hexagonal-architecture/tree/master/src/test/java/com/example/hexagonal/architecture>
- <https://github.com/eugenp/tutorials/blob/master/spring-boot-modules/spring-boot-testing/src/test/java/com/baeldung/boot/testing/EmployeeServiceImplIntegrationTest.java>
- <https://spring.io/guides/gs/testing-web/>
- <https://docs.spring.io/spring-boot/docs/current/reference/html/features.html#features.testing.spring-boot-applications.with-mock-environment>
- <https://github.com/eugenp/tutorials/tree/master/spring-boot-modules/spring-boot-testing/src/test/java/com/baeldung/boot/testing>
- https://www.archunit.org/userguide/html/000_Index.html#introduction
<https://github.com/tng/archunit-examples/tree/main/example-junit5/src/test>

Running

Now, as an example, we will create a class that tests the context loads. The name of file is *UserApplicationTests.java*, and put it in `|src|test|java|edu|uoc|epcsd|user :`

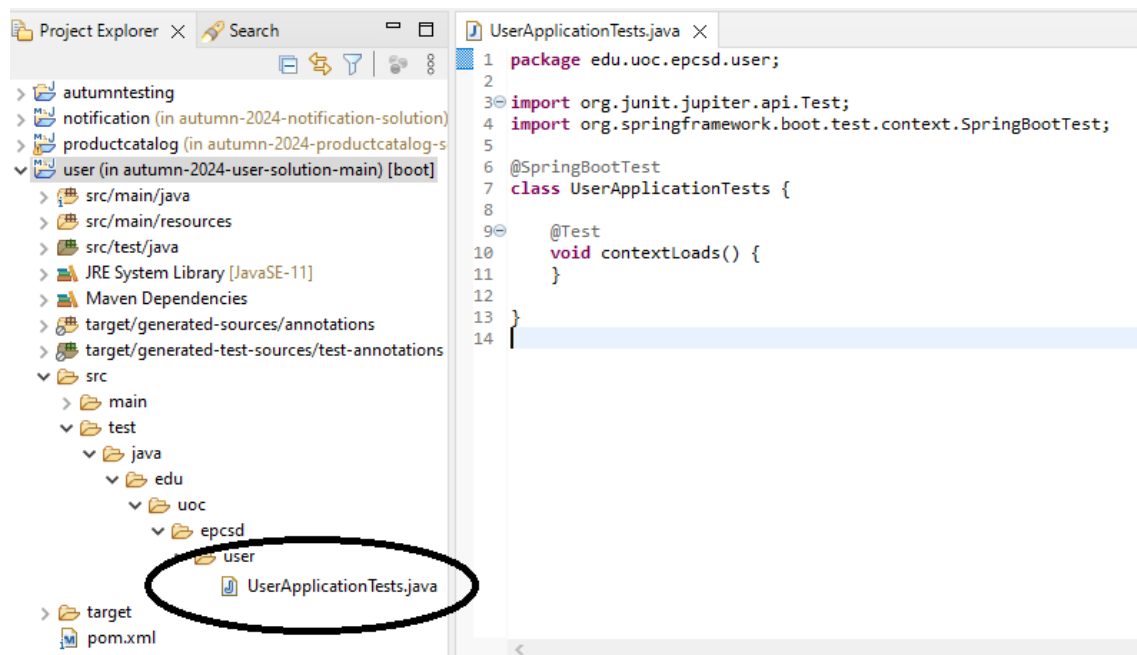
```
package edu.uoc.epcsd.user;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class UserApplicationTests {

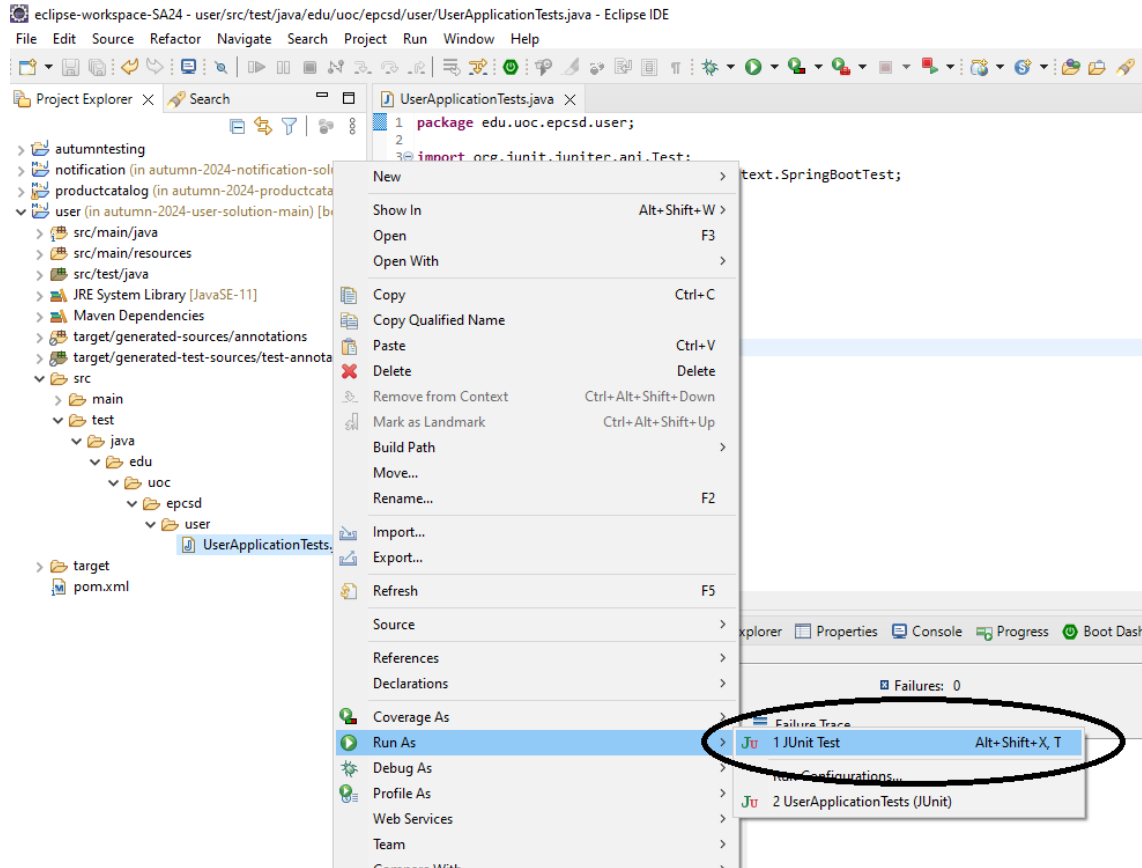
    @Test
    void contextLoads() {
    }

}
```

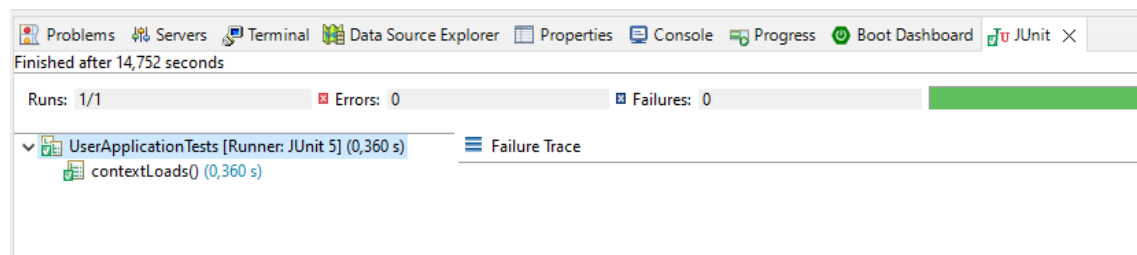


Running from Eclipse

If you want to run the test with Eclipse, you need to click on the project: *Run as* → *JUnit Test*



And you can see the result in the tab JUnit:



Running from IntelliJ

If you want to run the test in IntelliJ, you must click on the project, and then you must click on *Run 'All Tests'*:

