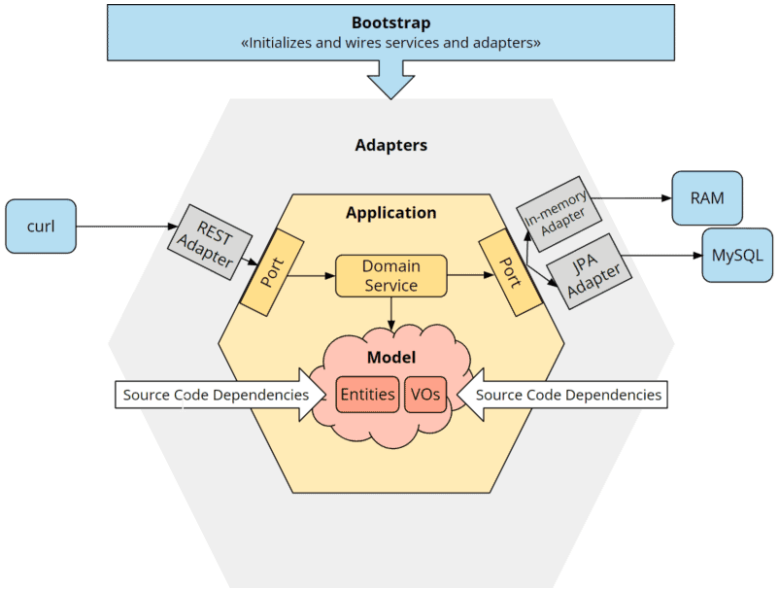
Practice 1 Hexagonal Architecture

Information Systems Design. TecnoCampus.

This first DSI practice aims to become familiar with hexagonal architecture through an experiential challenge.

The base project you will work with is structured excellently through modules. The hexagonal architecture diagram is presented below so that domain-driven design (DDD) has been applied to the core of the hexagon:



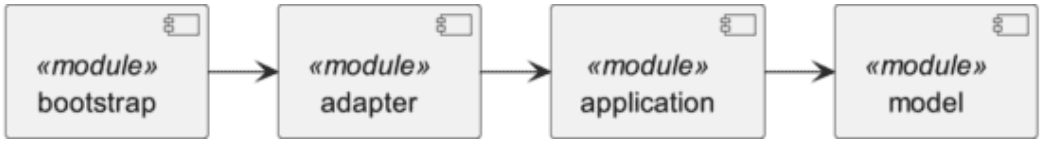
From the point of view of source code, the project has been structured through Maven modules. A Maven module has been created for each component type of the hexagonal architecture + domain driven design:

Hexagonal Architecture – Module Structure

model	Contains the domain model, i.e. those classes that represent the shopping cart and the products. According to domain-driven design, we will distinguish here between *Entities* (have an identity and are mutable) and *Value Objects* (have *no* identity and are immutable).

application	Contains a) the ports and b) the domain services that implement the use cases. *application* and *model module* together form the application core.
adapter	Contains the REST and persistence adapters.
bootstrap	Instantiates adapters and domain services and starts the built-in Undertow web server.

We define the dependencies between the Maven modules as follows, analogous to the project structure graphic above:



Dependencies of the Maven modules

Through this practice, we will demonstrate that it is possible to reuse the entire core of the hexagonal architecture even if we use an entirely different database to achieve the persistence of the domain's entities. In this case, the change that is going to occur is going to be the database. However, we could apply the same principle to any infrastructure technology. For example, a prevalent situation occurs when a use case of the application module needs to communicate with an external API to obtain some data.

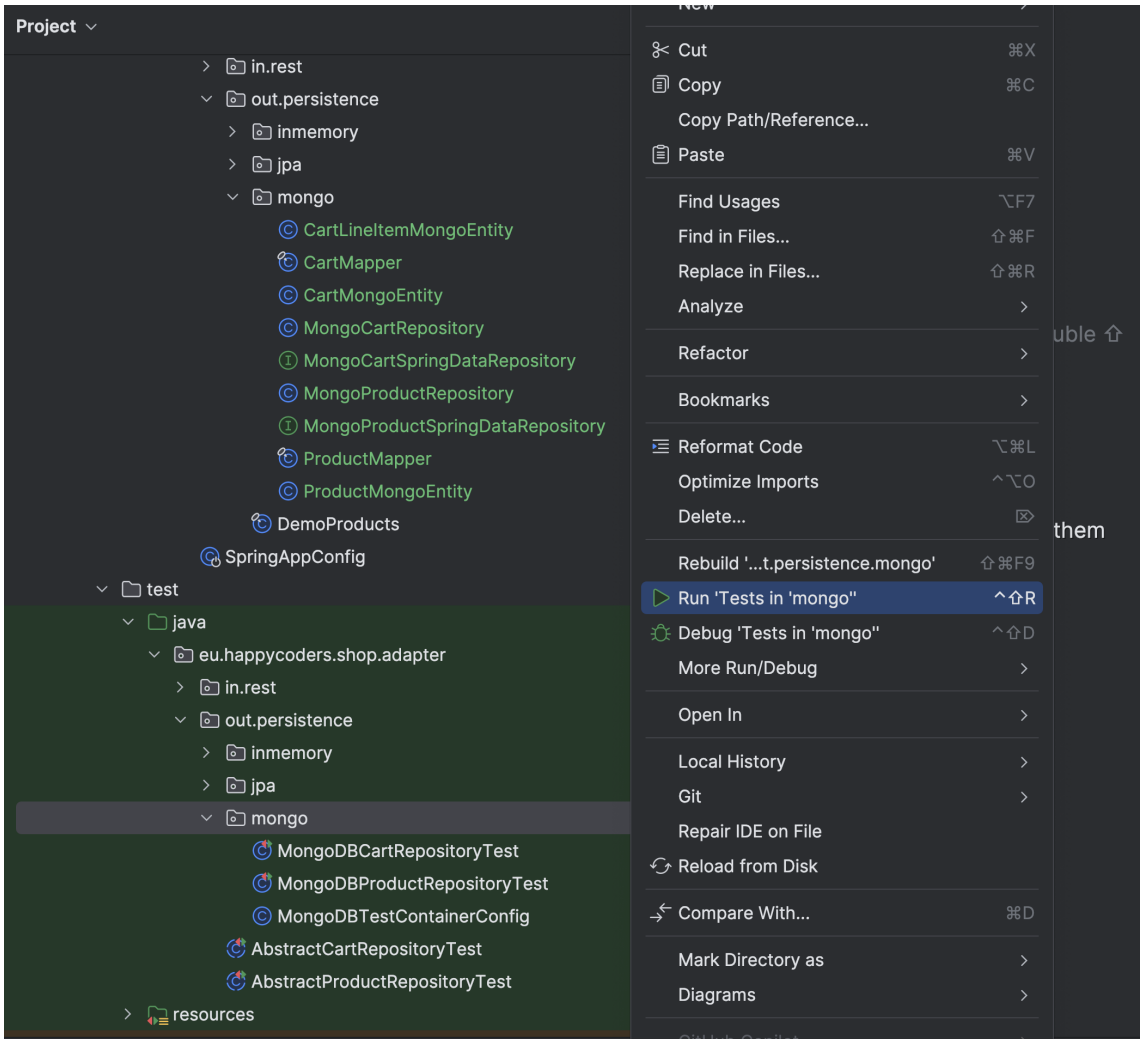
You can imagine that the use case must communicate with an API offered through SOAP technology at any time. Later, the SOAP service will no longer be offered to provide a REST API or GraphQL service. In other words, the same data is provided from a functional point of view; however, the specific communication technology with the service changes. We want to ensure that the hexagonal architecture's core is independent of these infrastructure changes. We are looking to improve the maintenance of our software significantly.

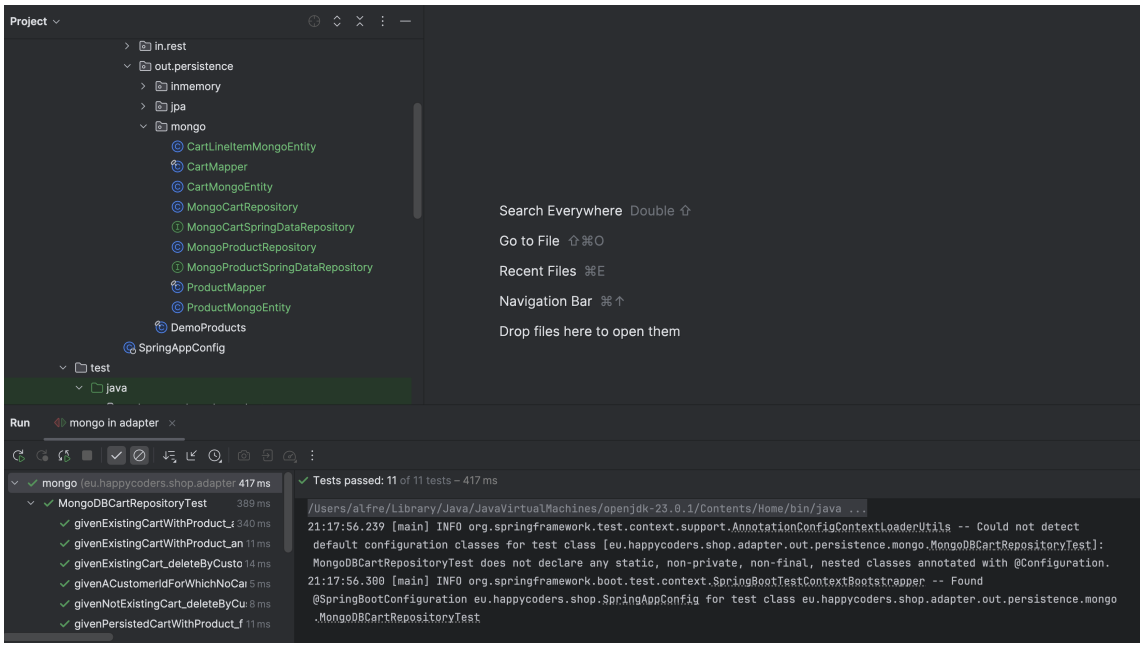
Adding a persistence adapter that uses a non-relational technology is relevant, as it further demonstrates that the rich domain can be fully reused. This architecture is favourable for improving software maintenance in the long run.

MongoDB is a non-relational database system that is very successful in the industry and integrates well with the Spring ecosystem. For this reason, it has been selected as an additional adapter in this excellent example of hexagonal architecture. All tests, both unit tests and end-to-end (e2e) tests, must be passed satisfactorily.

The first part of the practice involves implementing MongoDB adapters. You must pass the MongoDB tests to pass this first part successfully.

Here are some screenshots that demonstrate a scenario where MongoDB adapters have already been deployed, and tests are passed successfully:





As you can see, the 11 tests have been passed satisfactorily. The Mongo package will be empty in your repository—that's normal. Your work will try implementing all the classes and interfaces needed to offer the MongoDB adapters.

You must implement four types of components:

1. MongoDB Documents
2. Spring Data MongoDB Repositories
3. Mappers between MongoDB Documents and domain entities.
4. The classes that implement the output ports of the hexagonal architecture precisely rely on the three types of previous components.

You should read the following article to understand the types of components better. This article has shown how the persistence adapters for JPA have been implemented:

https://www.happycoders.eu/software-craftsmanship/ports-and-adapters-java-tutorial-db/

It's not about directly translating all JPA entities into MongoDB following the same structure. Instead, we should focus our design on implementing persistence

entities so that they optimally use the specific database technology. Consciously use @DBRef to implement relationships whenever strictly necessary. MongoDB's main advantage over relational databases is that the data model is more flexible and less restrictive, significantly increasing the system's overall scalability.

The teachers provided you with the following videos, which we recorded expressly to present the practice in a clear and well-guided way. We hope you enjoy your learning experience and get the most out of it.

Explanatory videos of the hexagonal example code:

Part 1: https://vimeo.com/1043914792/d1d843364f

Part 2: https://vimeo.com/1043944907/f48cdc4fe9

Part 3: https://vimeo.com/1043951273/dad507cf10

Part 4: https://vimeo.com/1043957055/146ef346f3

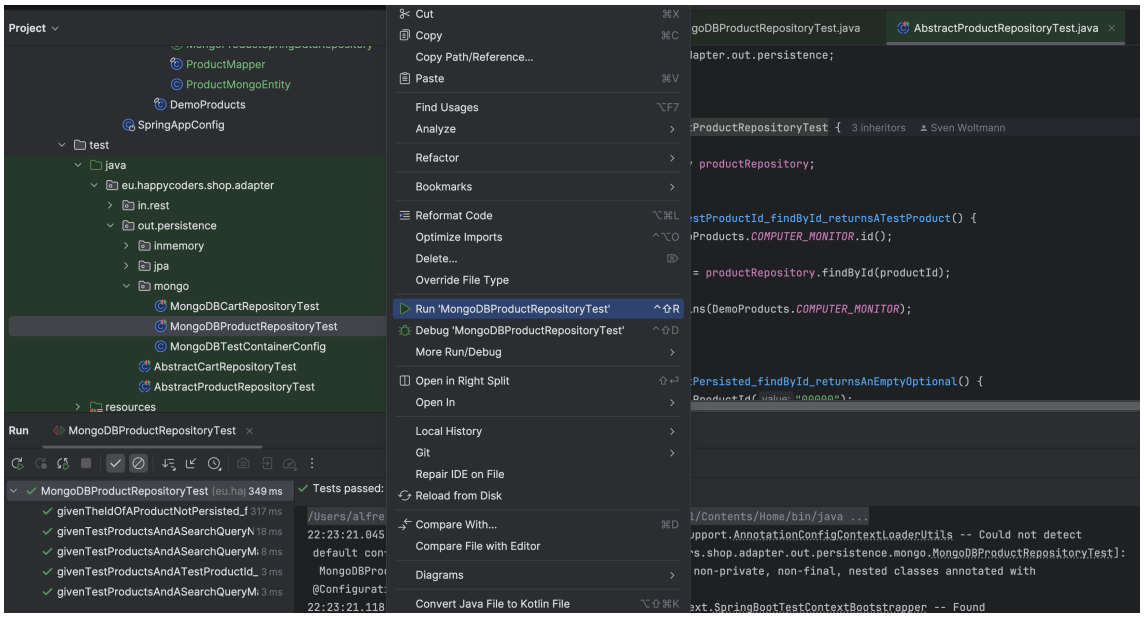
Part 5: https://vimeo.com/1043964130/2e7ba582dd

Part 6: https://vimeo.com/1046199174/ebf463c549

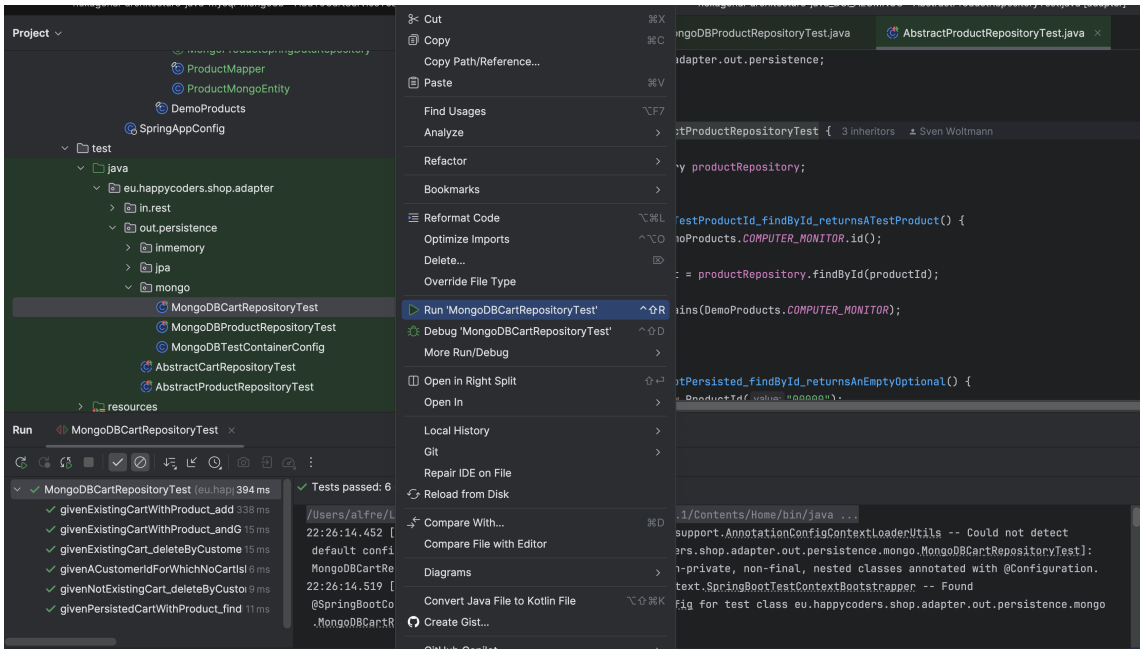
Part 7: https://vimeo.com/1046213773/0bcce0fac6

We recommend that you start with the tests of the product entity.

Here is an example of the successful execution of the tests with the product entity:



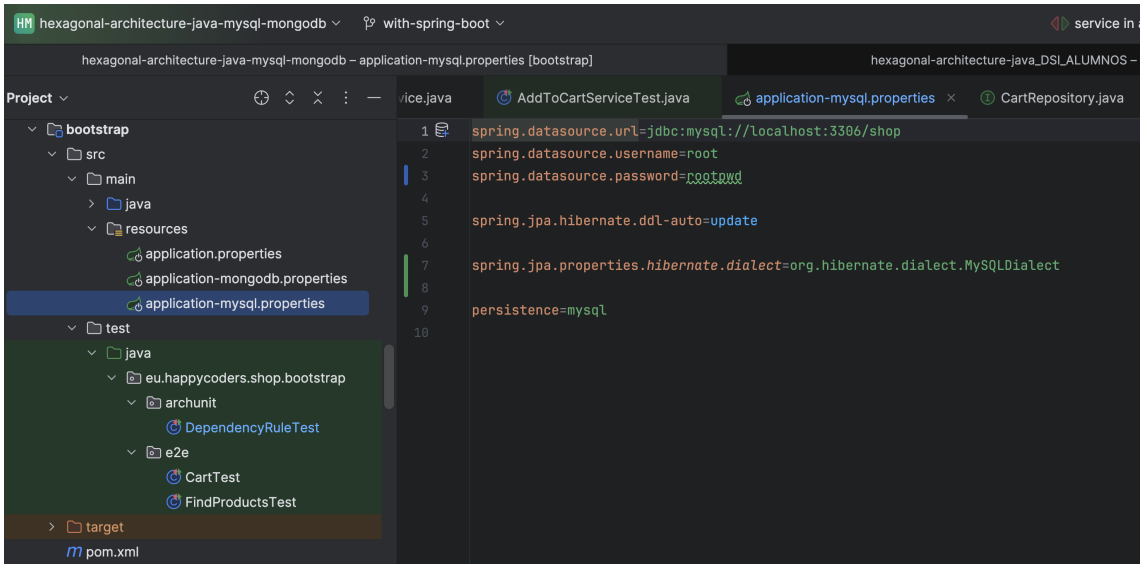
Below, you can continue with the Cart tests:

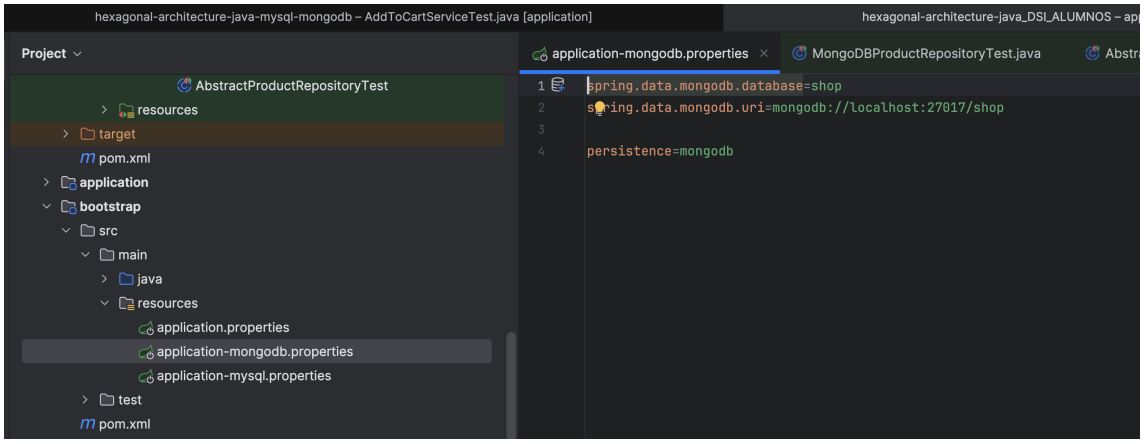


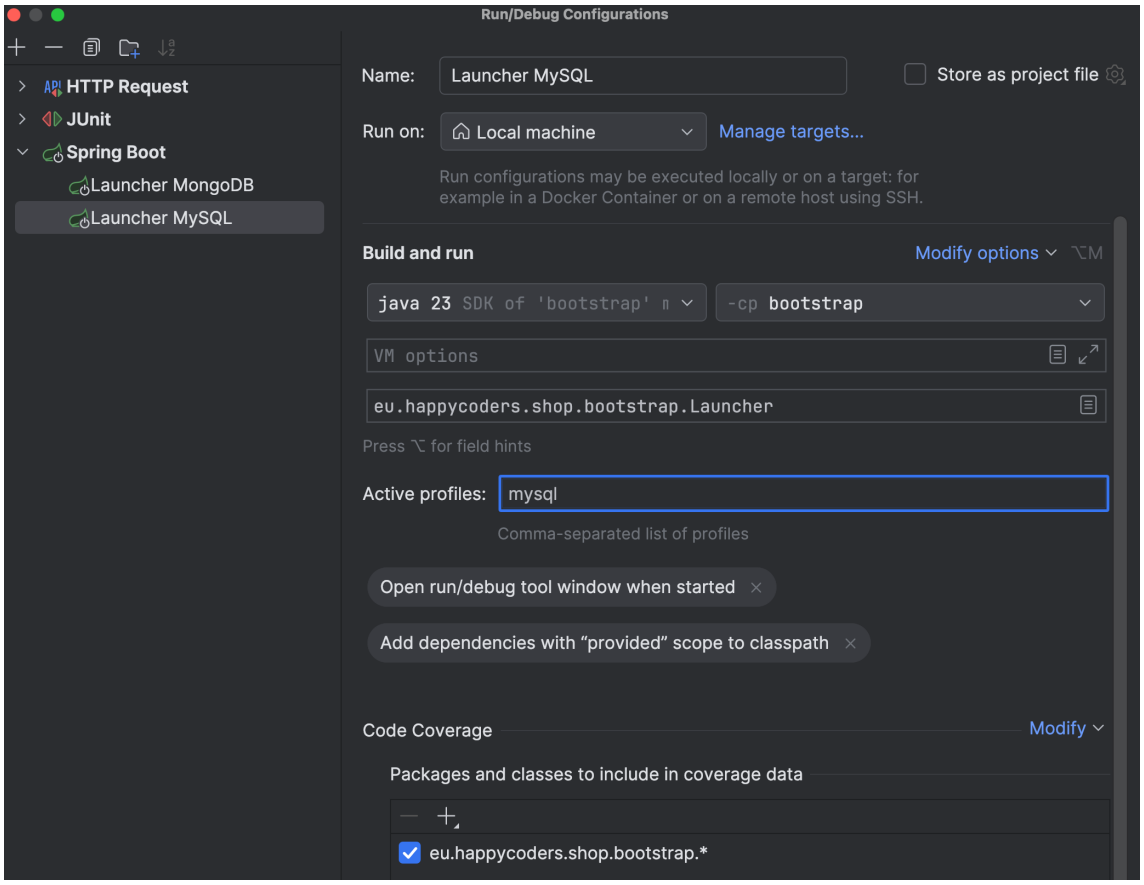
During your development tasks, it is strongly recommended that you run the application manually via the Bootstrap module. The videos explain in detail how to run the application using MySQL and MongoDB. Another aspect that is also well explained in the videos is that the project provides a file with example HTTP requests. This will allow you to test your use cases manually to complement automatic testing. Teachers believe it is an excellent idea to approach the project through these manual executions. Are you wondering why? The thing is, this project uses wildly sophisticated technology to automate the testing environment. Instead

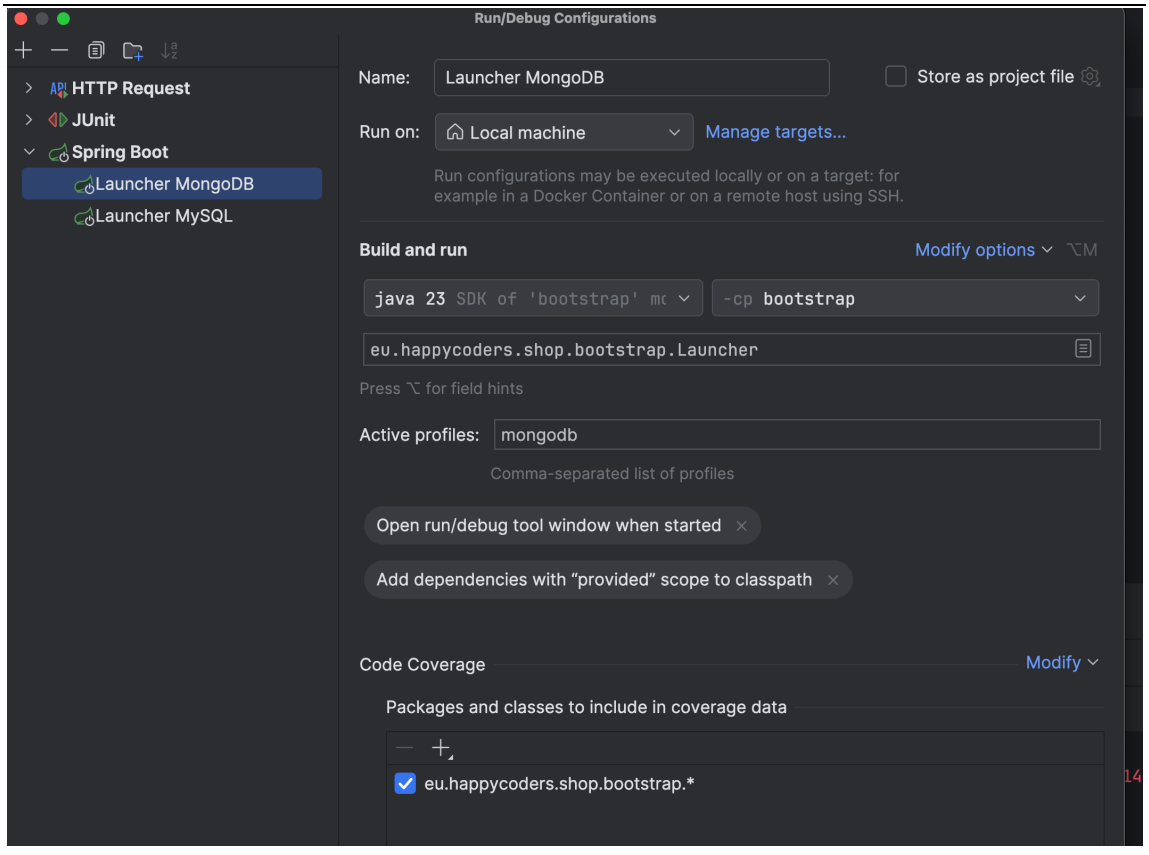
of manually lifting the database, it is the test itself through a library called Test Containers, which is responsible for creating the Docker container with the database and launching it. In addition, our project configures the Spring Data component to talk to the database server that has been automatically launched using TestContainers. This makes everything too automatic and magical. Or, as some joke says, "automagic". These manual executions and configuring and starting the database manually will help you better understand the whole process.

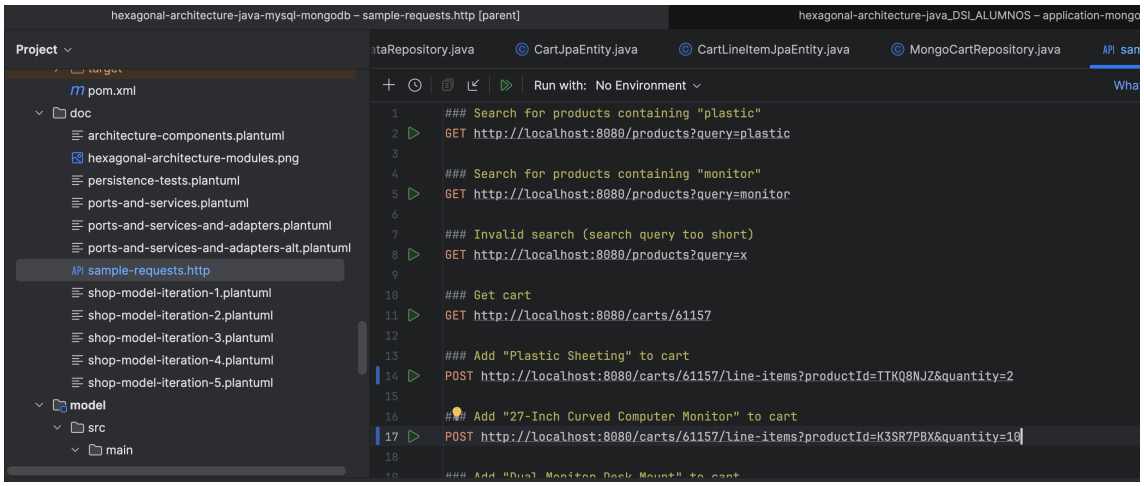
You should install the MongoDB Compass tool to see how changes are reflected in the database. Below is a screenshot of the document with the HTTP requests, for example, and the Compass GUI App, which visualises the database's contents.

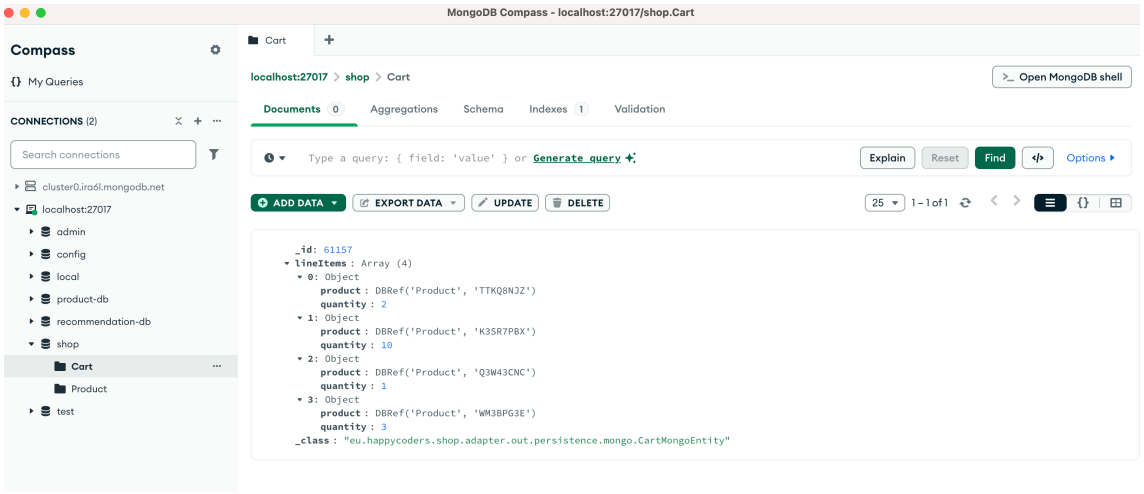





Run/Debug Configurations
HTTP Request
JUnit
Spring Boot
Launcher MongoDB
Launcher MySQL
Name: Launcher MySQL
Store as project file
Run on: Local machine
Manage targets...
Run configurations may be executed locally or on a target: for example in a Docker Container or on a remote host using SSH.
Build and run
Modify options ⌥M
java 23 SDK of 'bootstrap' m
-cp bootstrap
VM options
eu.happycoders.shop.bootstrap.Launcher
Press ⌥ for field hints
Active profiles: mysql
Comma-separated list of profiles
Open run/debug tool window when started
Add dependencies with "provided" scope to classpath
Code Coverage
Modify
Packages and classes to include in coverage data
eu.happycoders.shop.bootstrap.*


Run/Debug Configurations
HTTP Request
JUnit
Spring Boot
Launcher MongoDB
Launcher MySQL
Name: Launcher MongoDB
Store as project file
Run on: Local machine
Manage targets...
Run configurations may be executed locally or on a target: for example in a Docker Container or on a remote host using SSH.
Build and run
Modify options ⌥M
java 23 SDK of 'bootstrap' mc
-cp bootstrap
eu.happycoders.shop.bootstrap.Launcher
Press ⌥ for field hints
Active profiles: mongodb
Comma-separated list of profiles
Open run/debug tool window when started
Add dependencies with "provided" scope to classpath
Code Coverage
Modify
Packages and classes to include in coverage data
eu.happycoders.shop.bootstrap.*





In the screenshots we have shown, the following information can be seen:

- The configuration files for Spring Data connect to databases running on the local operating system.
- The IntelliJ IDEA configuration files will run our backend through the Bootstrap module. Please note that we activate the appropriate MySQL or MongoDB profiles by defining an option for the Java Virtual Machine.
- The definition files of the sample HTTP requests that interact with our Java backend. Several post-type requests have been executed to fill in the shopping cart for this demonstration.
- Finally, the last screenshot shows the contents of the database using the MongoDB Compass client application.

We wish you great success with your practice and that you enjoy your learning process as much as we teachers have enjoyed preparing it.