deti universidade de aveiro departamento de eletrónica, telecomunicações e informática

Lab 2 - Java at the server-side and the role of application containers

Updated: -09-.

Introduction to the lab

Learning outcomes

- Deploy a java web application into an application container (servlets).
- Start a simple web application with Spring Boot and Spring Initializr.

Submission

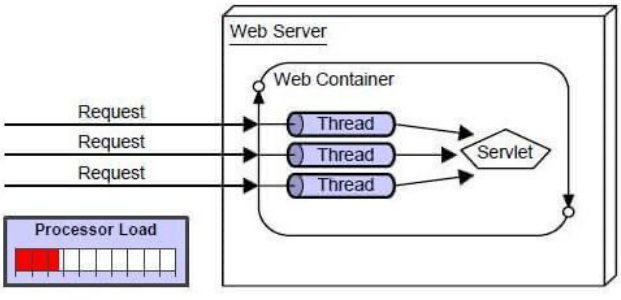
This is a two-week lab. You may submit as you go but be sure to complete and submit all activities up to 48h after the second class.

- Add your work to the "lab2" folder in your personal Git repository for IES and prepare a subfolder for each relevant section of the lab (e.g.: lab2/**lab2_1**, lab2/**lab2_2**,…).
- you will submit your work by committing to the main line in your repository. You are required to have at least one commit per week. The final commit for each lab should be include the message "**Lab x completed.**" (replacing *x* with the lab number.)
- in addition to the requires coding projects, you should include a "notebook" prepared for each lab, placed in the root folder of that lab, e.g. in 🗁lab2/README.md . This notebook should be actively used during the exercise activities, for you to take note of key concepts, save some important/useful links, maybe paste some key visuals on the topics being addressed, etc. This should be a notebook one could study from.

2.1 Server-side programming with servlets

Java Servlet (also known as the Jakarta Servlet) is the foundation web specification in the Java Enterprise environment. A Servlet is a Java class that runs at the server, handles (client) requests, processes them, and reply with a response. A servlet must be deployed into a (multithreaded) **Servlet Container** to be usable. Containers[1] handle multiple requests concurrently. *Servlet* is a generic interface and the *HttpServlet* is an extension of that interface (the most used type of Servlets).

When an application running in a web server receives a request, the Server hands the request to the Servlet Container which in turn passes it to the target Servlet.
Servlet Container is a part of the usual set of services that we can find in Java Application Server.



[1] Servlet Containers and Docker Containers are different concepts! You use Docker Containers to deploy virtualized runtimes, for any kind of services; Servlet Containers provide a runtime to execute server-side web-related Java code (no relation with virtualization).

For some use cases, you may prefer to run the web container from within your app. In this case, you will be using an "embedded server", since its lifecycle (start, stop) and the deployment of the artifacts is controlled by your own application code.

a) Adapting from the sample in embedded server example using the Jetty server implement a Servlet example using Jetty. → Step **#3.4** in the example **not** required; be sure to **complete step #3.5**,

   Note that the project should be based on a standard Java application.

b) Adapt the example so the user can optionally send a message to be printed as a parameter (if nothing is sent, prints a default message). Note: explore the *request* object.

   http://127.0.0.1:8680 → print a default message

   http://127.0.0.1:8680/?msg="Hard workers welcome!" → print the *msg* parameter

   http://127.0.0.1:8680/?msg=%22Hard%20workers%20welcome!%22 → the same; special characters "escaped"

   Note: remember to update your lab "notebook"…

2.2 Server-side programming and application servers (Tomcat)

For most enterprise scenarios, you will not use embedded servers, rather high-performance, dedicated application servers to run web artifacts. There are several production-ready application servers you can choose from (e.g.: Apache Tomcat, RedHat WildFly, Payara, Glassfish,...). For this exercise, we will consider Apache Tomcat.

There are multiple ways to run the application server service and deploy artifacts into it; for now, we will run Tomcat in a Docker container.

c) Start by creating a new Jakarta EE application, based on the Web profile.

   You may use IntelliJ wizard and start from scratch; otherwise, you may get an empty project generate by IntelliJ you may use to get started (consider download as a zip and copy/adapt the included JakartaWebStarter project).

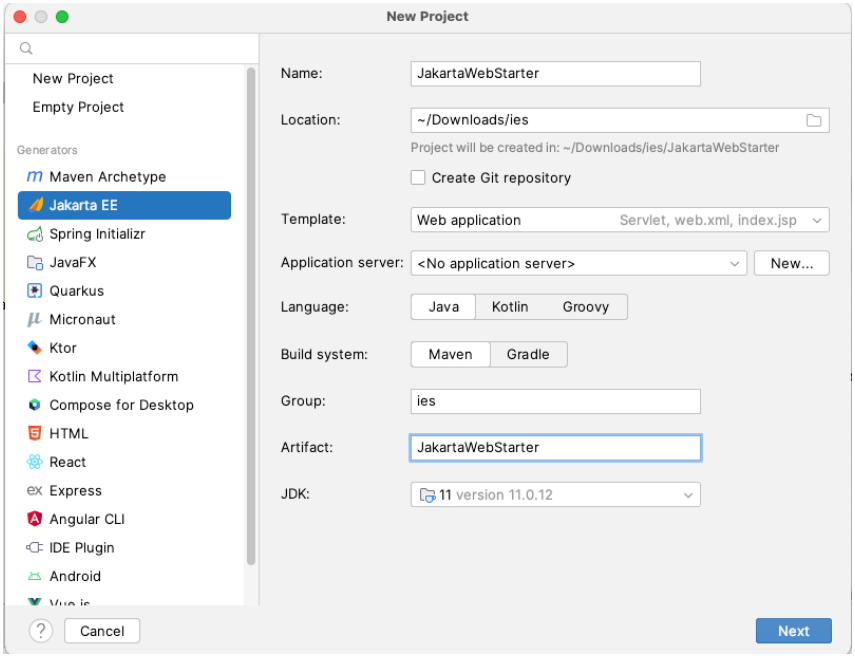


Figure 1- Creating an enterprise application using the Jakarta EE framework, in IntelliJ.

Note: alternatively, you may start your project from a related maven archetype.

d) Adapt the existing code to implement a similar function to the previous exercise (print a custom message using parameters passed to the Servlet in the URL; check #6 in this related sample).

Build/package the project.

Note that the resulting artifact is packaged as a "**.war**" (Web Archive), which is meant to be deployed in an application server.

Deploy the packed application (.war) into the application server using the same strategy discussed in this article, using docker-compose. [Or this alternative, with Dockerfile, but in this approach you would need to recreate the image to propagate later updates…]

Note: you do not need to follow the article contents… in fact, you just need to understand/adapt the **docker-compose.yml** to prepare a Docker based deployment.

Take a minute to verify the configuration in docker-compose.yml . You will notice that the local *./target* folder (in which the war will be prepared) is being mapped to the default webapps location of Tomcat (upon start, Tomcat will deploy the wars found in this folder).

Run the component (services) by issuing:

$ docker-compose up

e) Confirm that your application was successfully deployed in Tomcat inside the Docker container:

e.g.: http://127.0.0.1:8080/ (TomCat should produce 404 error)

e.g.: http://127.0.0.1:8080/*JakartaWebStarter-1.0-SNAPSHOT*/hello-servlet

Note: you should stop the deployment if redeploying your app (*docker-compose down*).

2.3 Introduction to web development with a full-featured framework (Spring Boot)

Spring Boot is a rapid application development platform built on top of the popular Spring Framework. By assuming opinionated choices by default (e.g.: assumes common configuration options without the need to specify everything), Spring Boot is a convention-over-configuration addition to the Spring platform, useful to get started with minimum effort and create stand-alone, production-grade applications.

a) Use the Spring Initializr[2] to create a new (maven-supported, Spring Boot) project for your web app.

Be sure to **add** the "**Spring web**" **dependency**. Spring Initializr templates contain a collection of all the relevant transitive dependencies that are needed to start a particular functionality and will simplify the setup of the POM.

Download the "starter kit" generated by Spring Initializr.

You should be able to build your application using the regular Maven commands. The generated seed project also includes a convenient Maven wrapper script (mvnw). To run you Spring Boot web application:

```
$ mvn install -DskipTests && java -jar target\webapp1-0.0.1-SNAPSHOT.jar
or
$ ./mvnw spring-boot:run
```

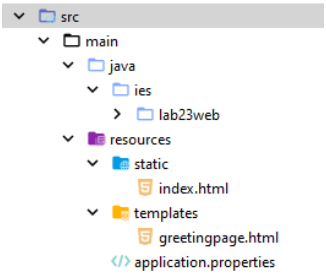
Access your browser at http://localhost:8080/; you should retrieve a page produced by Spring Boot (white label error).

Note: mind the **ports in use**! You will get an error if 8080 is already in use (maybe there is a running Tomcat?...).

b) Build a simple application to serve web content as detailed in this Getting Started article. Start the project from the scratch (instead of downloading the available source code). Try to type the code (so you learn the annotations).

Change the default port (look in application.properties) of the embedded server to something different from 8080 (default).

You may need to include the Thymeleaf starter in the dependencies (artifact co-ordinates: org.springframework.boot:spring-boot-starter-thymeleaf).



Note: The implementation of Spring MVC relies on the Servlets engine, however, you do not need to "see" them. The abstraction layers available will provide the developer with more convenient, higher-level interfaces.

c) In the previous example, you created a web application that intercepts the HTTP request and redirects to a page, with a custom message (be sure to change the optional parameter "name").

---

[2] Spring Initializr is integrated with intelliJ and can also be used with VS Code.

Extend you project to create a REST endpoint, which listen to the HTTP request, and answer with a JSON result (a greeting message). You may find this guide helpful.

Beside the @Controller, now you will use a **@RestController**.

Be sure to access the endpoint from the command line, using the curl utility (or use the Postman tool for API development).

Note: mind the URL path; it should be **a different path** from the previous task.

2.4 Wrapping-up & integrating concepts

Jakarta EE allows to create complex and robust applications for demanding scenarios. Often, we do not need a full-featured Application Server, rather just enough resources to run web applications/web services. Several servers will offer a "trimmed" version associated to a "Web Profile" which is appropriate, for example, to deploy an API over HTTP, as exemplified in the previous "RESTful guide".

a) Create a web service (REST API) to offer random quotes from movies/shows. You should support the three endpoints listed below.

Notes:

- you do not need to use a real database; consider using "static" information, for now. Need inspiration?
- all responses should be formatted as JSON data.

Endpoints:

Method	Path	Description
GET	api/quote	Returns a random quote (show not specified).
GET	api/shows	List of all available shows (for which some quote exists). For convenience, a show should have some identifier/code.
GET	api/quotes?show=<show_id>	Returns the existing quotes for the specified show/film.

(Don't forget to push the changes upstream!)