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

Updated: 2021-10-26.

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.

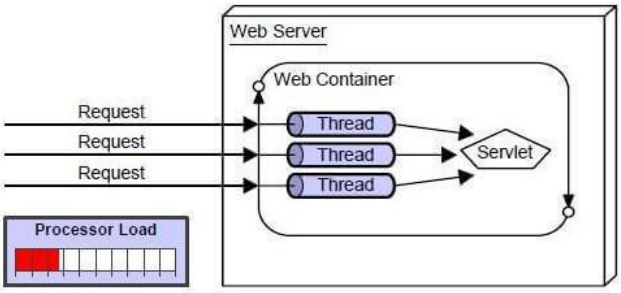
Submission

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 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.

a) Prepare an application server to run your web projects.

[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 virtualization).

There are several production-ready application servers you can choose from (e.g.: Apache Tomcat, RedHat WildFly, Payara, Glassfish,...).

For this exercise, consider using the **Apache Tomcat**, which is open source. Just download (Tomcat v9 → core → zip) and expand to a local folder[2].

Run the application server (use the startup script inside <path to Tomcat>/bin folder).

Confirm that Tomcat server is running. Here are 3 alternatives to do it:

i) Curl tool

```
$ curl -I 127.0.0.1:8080
```

ii) Access the default page in the browser: http://localhost:8080

iii) Observe the server log:

```
$ tail logs/catalina.out
```

b) The Tomcat installation includes a management environment ("Manager app") you can use to control the server, including deploying and un-deploying applications you develop (http://localhost:8080/manager).

To use the manager app, **you need to register at least one role** in conf/tomcat-users.xml. Restart tomcat. (see users configuration sample in step #6 here if needed)

```
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<user username="admin" password="secret" roles="manager-gui,manager-script"/>
```

From the web Manager environment, explore the Examples application already installed in Tomcat. Inspect the Examples --> Servlet --> **Request Parameters**. Check the related source code; note the superclass (in "extends").

c) Let us now create a web application and deploy it into Tomcat. Start by creating a maven-based **web application** project (not a regular Java application…).

For that, you need to use a suitable Maven archetype, for example, use this template from *codehaus'*:

archetypeGroupId=org.codehaus.mojo.archetypes
archetypeArtifactId=**webapp-javaee7**
archetypeVersion=1.1

(You may need to add this additional archetype to your IDE.)

d) This base project already includes am default index page.

Build the project and ensure there are no errors (mvn install).

Confirm that you have a **.war** file in <project folder>/target. This is your application packaged as a Web Archive (you may inspect its contents in a regular Zip tool, though not required).

e) Deploy the packed application (.war) into the application server.

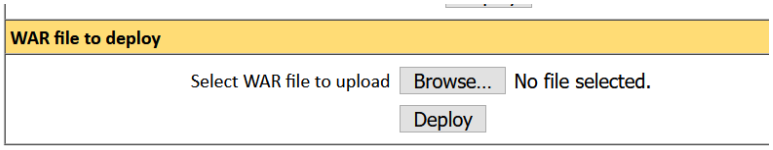
Use the Tomcat management interface (Tomcat manager application) to upload and deploy a .war file (http://localhost:8080/manager).

or

copy the .war file into <Tomcat root>/webapps

---

[2] For a production environment, you would install Tomcat as a system service/daemon or, even better, as a "dockerized" service. In this case, we will just start and stop from the command line, in the installation folder.



f) Confirm that your application was successfully deployer in Tomcat: it should be listed in the Manager app and you should get the page "Hello World!" displayed in the browser (e.g.: http://localhost:8080/*your-web-app-name*/)

g) [**Optional**] Deploying using the Tomcat Manager page has some disadvantages: it is not "connected" with the IDE and is specific to Tomcat. The productive alternative is to use the **IDE integrated deployment support**.

   You may choose to configure the Tomcat server integration in your IDE, e.g., for IntelliJ[3]

   In this example, the artifact to deploy is the <your project>.war file.

   Important: when deploying from IDE for development purposes, it is better that Tomcat has not been already started from outside the IDE (instead, it will be started/stopped by the IDE as needed).

h) Add a **basic servlet** to your project that takes the name of the user, passed as a parameter in the HTTP request and prints a customized message.

   You may adapt these instructions (from section "Develop Servlet with @WebServlet Annotation" to section "Handling Servlet Request and Response").

   Note: in older versions of the Servlet Container specifications, the developer needed to write a **web.xml** file with the configuration descriptors, including the mapping of Servlet classes and URL paths. The current implementations, however, use annotations (check for the presence of @WebServlet annotation), which makes the use of web.xml not required (in most of the cases).

i) Be sure to include a deliberate runtime error in your app (e.g.: a NullPointerException) caused when the servlet methods are used.

   Look for the exception (and the stack trace) in the application container [localhost] log. (Then, correct the error.)

j) [**Optional**] Given that Java application containers are based on open specifications, you may run your project using a different server, without code modifications. The deployment methods, however, will be different.

   You may try other popular servers, e.g.: Payara Server (web profile), instead of Tomcat.

2.2 Server-side programming with embedded servers

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 application code.

a) Adapting from the sample in embedded server example using the Jetty server (Step **#3.4** in the example not required; be sure to **complete step #3.5**): implement the Servlet example from the previous exercise, but with an embedded server approach.

[3] If you are using the IntelliJ Ultimate edition, the plugin "Tomcat and TomEE" is already installed.

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

2.3 Introduction to web apps 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[4] 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).

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

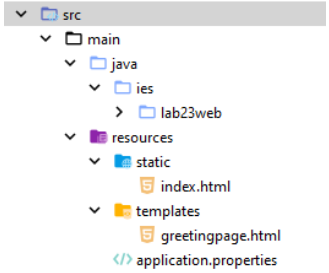
Access your browser at http://localhost:8080/; you should retrieve a page produced by Spring Boot (white label error).

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.

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

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").

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

Java Enterprise 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 may check/adapt the logic from an existing API (like this one).
- you do not need to use a real database; consider using "static" information, for now.
- all responses should be formatted as JSON data.

Endpoints:

Method	Path	Description
GET	api/quote	Returns a random quote from a random show/film.
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 a random quote for the specified show/film.

Review questions

Answer the following questions in a final section, in your lab notebook (README.md).

A. What are the responsibilities/services of a "servlet container"?

B. Explain, in brief, the "dynamics" of Model-View-Controller approach used in Spring Boot to serve web content. (You may exemplify with the context of the previous exercises.)

C. Inspect the POM.xml for the previous Spring Boot projects. What is the role of the "starters" dependencies?

D. Which annotations are transitively included in the @SpringBootApplication?

E. Search online for the topic "Best practices for REST API design". From what you could learn, select your "top 5" practices, and briefly explain them in you own words.

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