Module 4: Data & Information

Tools installation tutorial session

25 April 2017
13:45-17:30

The objective of these exercises is to make you acquainted with some of the tools that will be used in this module. These exercises have been also designed to help you install the tools and give you initial hands-on experience with them.

1 Installing and configuring JDK 8

You have probably already installed JDK 8 in your machine for Module 2, but it is a good idea to check the installation.

1.1 Downloading and installing JDK 8

JDK 8 can be downloaded from http://www.oracle.com/technetwork/java/javase/downloads/. Make sure you download and install a JDK that matches your operating system and hardware architecture (e.g., Windows x64 for Windows 8/7/Vista 64 bits). JDK 8 can be installed by running the installation program downloaded from this location. Make sure you have the proper administrative rights to run an installation program in your machine.
After that test the Java installation by opening a Command prompt (Terminal in Linux/MacOS) and typing `java -version`. This command should return something like

```
java version "1.8.0_51"
Java(TM) SE Runtime Environment (build 1.8.0_51-b16)
Java HotSpot(TM) 64-Bit Server VM (build 25.51-b03, mixed mode)
```

1.2 Setting the JAVA_HOME and PATH environment variables

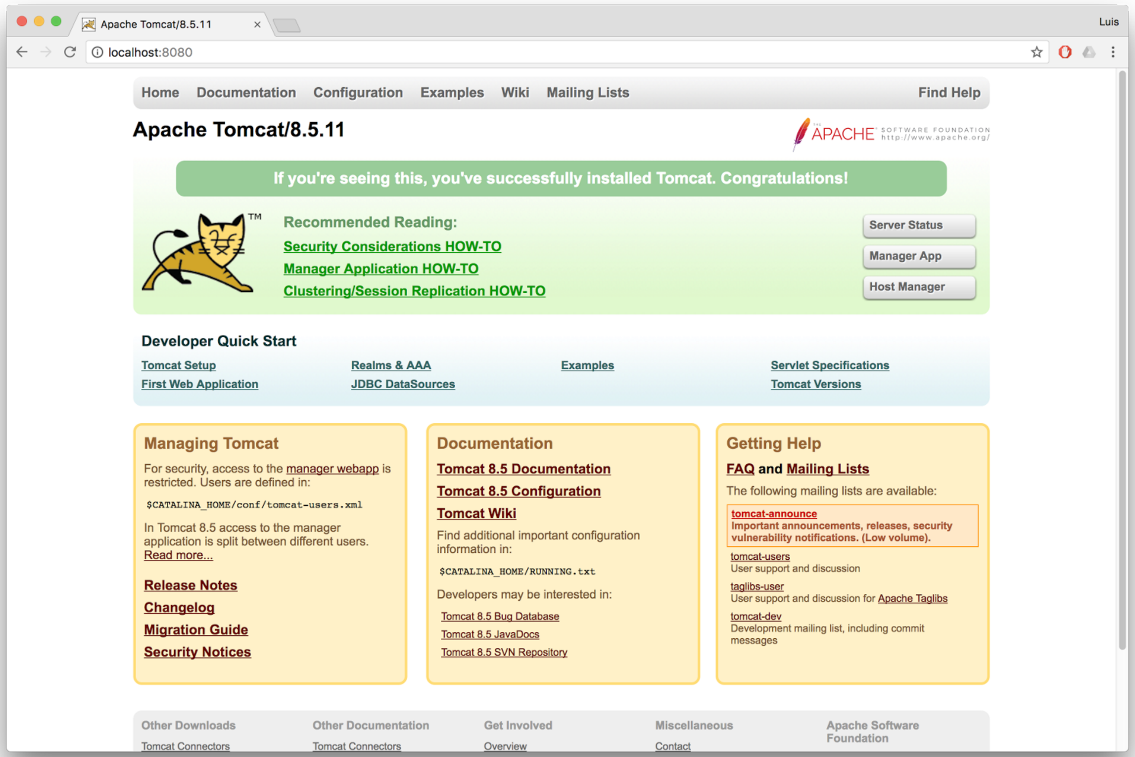
After downloading and installing JDK 8 you may have to set the JAVA_HOME and the PATH environment variables. You should do this for sure if the test of the Java installation with `java -version` fails. Setting these environment variables is often necessary in Windows machines.

Environment variables are explained in http://en.wikipedia.org/wiki/Environment_variable. Each operating system (e.g., Windows 7 / 8 / 10, MacOS and Linux) has a particular way of setting environment variables. The page mentioned above also contains pointers to pages in which instructions to set environment variables in the most popular operating systems can be found. Make sure you learn how this can be done in the operating system running in your laptop.

Setting the `JAVA_HOME` environment variable is advisable (even if not necessary) because many tools make use of this environment variable to find out where your default Java installation can be found. First of all you have to find out the directory in which Java has been installed. For example, JDK 8 has been installed in my Windows 7 64 bits machine in directory `C:\Program Files\Java\jdk1.8.0_51`. This string should then be assigned to environment variable `JAVA_HOME`. Finally you have to add the `bin` directory of the `JAVA_HOME` directory to the list of directories where your system looks for programs, so that the Java tools can be properly found. This can be done by copying the path to this directory (complete directory name) to the beginning of the value of the `PATH` environment variable. A good practice is to use the value of `JAVA_HOME` for that. In this way, whenever you install a new JDK you only have to change the value of the `JAVA_HOME` environment variable. For example in Windows 7 you can do this by adding `%JAVA_HOME%\bin;` to the beginning of the `Path` environment variable in Control Panel → System → Advanced System Settings → Environment variables... → System variables

2 Installing Tomcat 8.5

In order to run, Web applications require a so called Application server, which can be considered as a special web server that knows how to handle the components of a web application (https://en.wikipedia.org/wiki/Application_server). In this module we use the most popular Application server for Java applications, namely Apache Tomcat (http://tomcat.apache.org/). In order to remain compatible with the code supplied in this course we require that you download and use Tomcat 8.5, which can be downloaded from http://tomcat.apache.org/download-80.cgi. You should download the ‘Core’ version that matches your OS and hardware (and Java installation), preferably one of the `.zip` files, and extract the contents of this `.zip` file to a directory that does not contain spaces in its name (Java does not like spaces). For example, you can create a directory `Development` in the root of the `C:` disk and extract the Tomcat installation to this directory. In Windows, after extracting Tomcat 8.5 to the `C:\Development\` directory today (21 April 2017), the path to the Tomcat 8.5 installation becomes `C:\Development\apache-tomcat-8.5.14`.



Apache Tomcat/8.5.11

If you're seeing this, you've successfully installed Tomcat. Congratulations!

Recommended Reading:
Security Considerations HOW-TO
Manager Application HOW-TO
Clustering/Session Replication HOW-TO

Developer Quick Start
Tomcat Setup
First Web Application
Realms & AAA
JDBC DataSources
Examples
Servlet Specifications
Tomcat Versions

Managing Tomcat
For security, access to the manager webapp is restricted. Users are defined in:
`$CATALINA_HOME/conf/tomcat-users.xml`
In Tomcat 8.5 access to the manager application is split between different users. Read more...
Release Notes
Changelog
Migration Guide
Security Notices

Documentation
Tomcat 8.5 Documentation
Tomcat 8.5 Configuration
Tomcat Wiki
Find additional important configuration information in:
`$CATALINA_HOME/RUNNING.txt`
Developers may be interested in:
Tomcat 8.5 Bug Database
Tomcat 8.5 JavaDocs
Tomcat 8.5 SVN Repository

Getting Help
FAQ and Mailing Lists
The following mailing lists are available:
tomcat-announce
Important announcements, releases, security vulnerability notifications. (Low volume).
tomcat-users
User support and discussion
taglibs-user
User support and discussion for Apache Taglibs
tomcat-dev
Development mailing list, including commit messages

Other Downloads
Tomcat Connectors
Other Documentation
Tomcat Connectors
Get Involved
Overview
Miscellaneous
Contact
Apache Software Foundation

Figure 1. Tomcat 8.5 initial page after successful installation.

You can test the Tomcat installation by starting up the Application server and accessing the welcome page of the server through a browser. The script programs to start up and shutdown Tomcat can be found in the `bin` directory of the Tomcat installation (e.g., `startup.sh` and `startup.bat` for Linux/MacOS and Windows, respectively). Start up Tomcat by running the script that matches your machine. If everything goes well, Tomcat will return the values of environment variables `CATALINA_BASE`, `CATALINA_HOME`, etc. before starting up. You can now test the server by starting a browser and going to the page `http://localhost:8080`. If Tomcat 8.5 is properly installed you should see a page similar to the one in Figure 1.

After that don't forget to shutdown Tomcat with the proper script program (`shutdown.sh` and `shutdown.bat` for Linux/MacOS and Windows, respectively). Make sure you always properly shutdown the server when you want to stop it.

3 Installing Eclipse IDE for Java EE Developers

You have used Eclipse in Module 2, but we strongly recommend that you install a fresh version of Eclipse for this module. This is because you will need some specific plugins for developing web applications and using Maven and Git, and we want your workbench to remain stable, which cannot be guaranteed if you start from an existing Eclipse installation.

The plugins you need are available in the Eclipse IDE for Java EE Developers distribution ('distro' in the Eclipse jargon), which can be downloaded from https://www.eclipse.org/downloads/. There you should choose 'Download packages' and select the version that matches your Java installation (32 or 64 bits). After downloading Eclipse, extract the contents of the `.zip` file to a directory that has a name without spaces. In Windows, the `C:\Development\` directory mentioned above is an excellent location for the Eclipse installation. We also suggest that you rename the `eclipse` directory generated when you extract the contents of the `.zip` file to some name that indicates the Eclipse version, like, e.g., `eclipse-neon3-j2ee`. This is not so important now, but this will be useful later when you get more Eclipse installations (and this will probably happen!). The current Eclipse version (4.6.3) is also known as 'Neon.3'.

In Windows you can start Eclipse now by double-clicking on `eclipse.exe` (click on `Eclipse.app` in MacOS). It is also advisable to make a shortcut to this program. The first time you start Eclipse you have to choose a location for the workspace. Since in future you may have many Eclipse installations (you probably already have two!), we strongly advise you to choose a workspace location inside the Eclipse installation, so that it does not interfere with other Eclipse installations. If you followed our advices for the file locations so far, your workspace should be located at `C:\Development\eclipse-neon3-j2ee\workspace`. Make this workspace default and Eclipse will never bother you with this question again.

4 Connecting Eclipse to Tomcat

Eclipse allows web application developers to connect Eclipse Java EE projects to Tomcat, and run these applications inside Eclipse for the purpose of testing and debugging, so that it is not necessary to deploy these applications ('application deployment' is discussed later in the module). In order to do this, we have to create a server in Eclipse, so that we can assign our projects (web applications) to this server, and start and stop it whenever we want. This can be done by selecting File → New → Other… → Server → Server, and selecting Apache → Tomcat v8.5 Server. The wizard will ask for the Tomcat installation directory, and if you followed the suggestions concerning file locations this should be something like `C:\Development\apache-tomcat-8.5-14`. If the Server has been installed properly, the workbench should look as in Figure 2. By right-clicking on the server in the Servers view, you can start and stop the Tomcat server under the control of Eclipse.

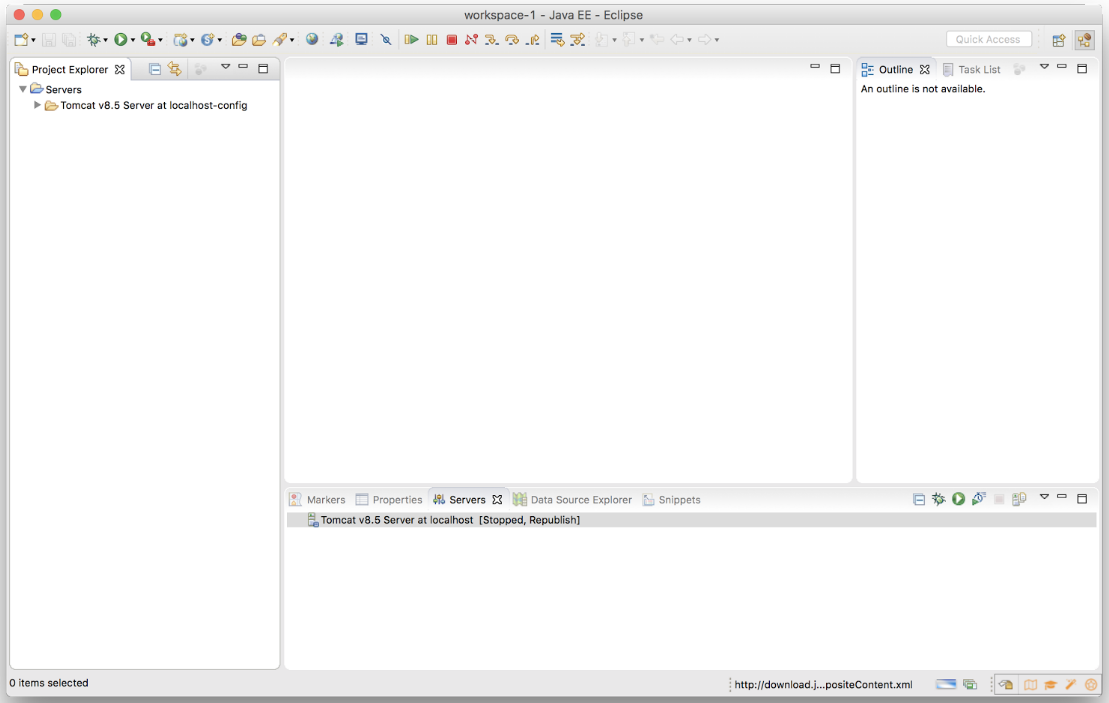

workspace-1 - Java EE - Eclipse
Quick Access
Project Explorer
Servers
Tomcat v8.5 Server at localhost-config
Outline
Task List
An outline is not available.
Markers
Properties
Servers
Data Source Explorer
Snippets
Tomcat v8.5 Server at localhost [Stopped, Republish]
0 items selected
http://download.j...positeContent.xml

Figure 2. Workbench after Tomcat 8.5 has been connected to Eclipse.

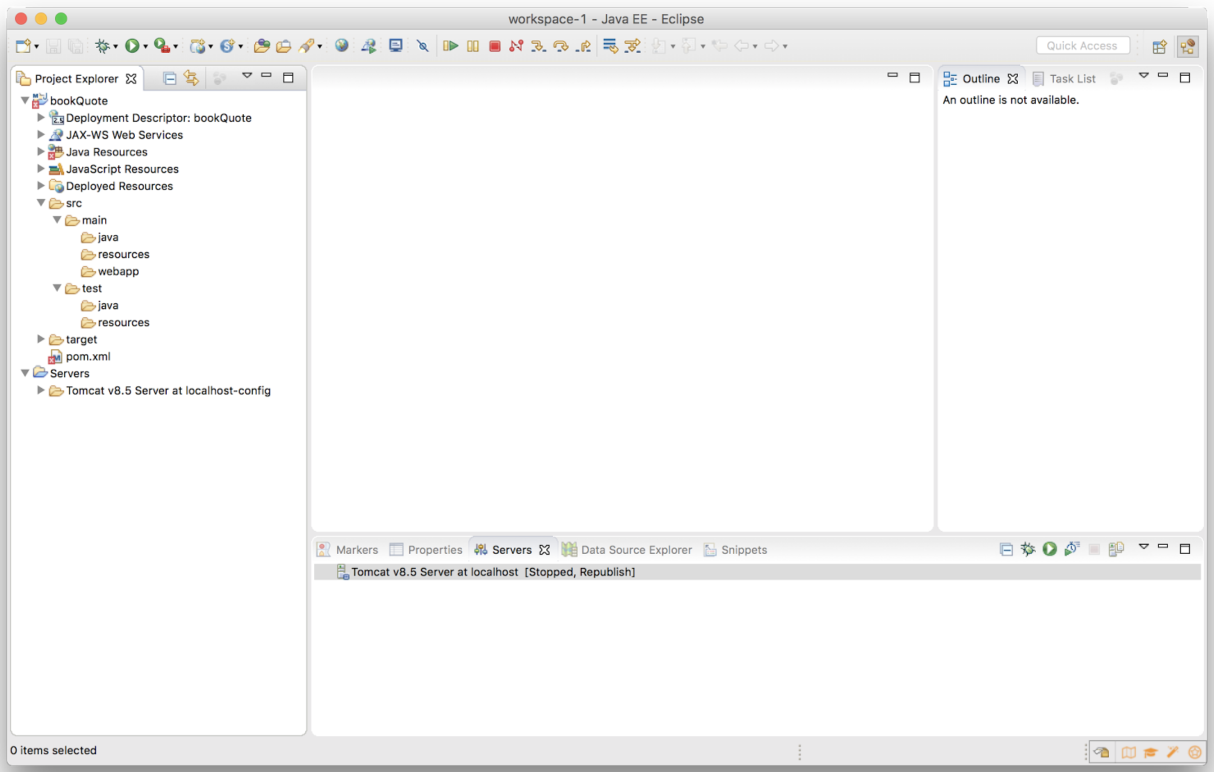
5 Creating a Maven web application project in Eclipse

In this module you will use Maven (http://maven.apache.org/), which is a software project management tool. Maven is extremely useful because it allows the management of code dependencies, like dependencies of libraries that are necessary to compile and run the software applications. In a sense Eclipse already does that, but Maven makes this possible even when Eclipse is not being used.

Libraries are (binary) code that can be reused in a program, so that the programmer can concentrate on the task at hand. You should already know what a library is. Think about the `java.utils` package that you have to import in order to use the Java collection classes. This package is quite stable, but other packages have more specific versions that you have to load to guarantee the compilation and execution of your code. Maven not only allows you to define the dependencies, but also downloads these packages from the Maven repository at (http://mvnrepository.com/), so it is extremely handy.

Maven prescribes a specific structure for the files in a project (source code, web resources like HTML and CSS files, class files, etc.) and a `pom.xml` file that describes the Maven project. In order to explain this structure we start by creating an empty Maven project in Eclipse by selecting File → New → Maven project in the Java EE perspective, and we make the following choices:

- Set Create a simple project, since we do not want to use archetypes for the time being.
- Set Use default Workspace location, make sure the Location field is empty and press Next >.
- Define a group id (for example, `nl.utwente.di`).
- Define the artifact id, which is going to be the name of the application (for example, `bookQuote`).
- Select `war` as packaging. This is necessary to indicate to Maven that this is a web application.
- Keep the default value `0.0.1-SNAPSHOT` for the version.
- Give a name and a description to your Maven project if you want (not mandatory), and press Finish.



An outline is not available.

Tomcat v8.5 Server at localhost [Stopped, Republish]

0 items selected

Figure 3. Eclipse Maven project generated with the Maven project generation wizard.

A Maven project will be generated with the artifact id, a directory structure and a `pom.xml` file, like in Figure 3. The project may contain errors, but don't worry about them for the moment. We will fix them later.

Figure 3 shows the directory structure of a Maven project. Directory `src` contains all the source code, including Java files (in `main/java`), web resources, like HTML and CSS files (in `main/webapp`) and other resources (in `main/resources`). Directory `target` contains temporary files that are generated by Maven, like class files, deployment packages, etc. Figure 3 also shows the Eclipse POM editor.

Important

Now you are asked to download the `di-prog-lab-i.zip` file from Blackboard, which contains the resources (files) that are necessary for this session.

5.1 Configuring the Maven project

Once the Maven project is generated, we still have to perform some steps to be able to use the newer JDK and Servlet versions (JDK 8 and Servlet 3.1, respectively) than the versions supported as default by Maven. All necessary information about a Maven project is in its `pom.xml` file, therefore in order to facilitate these steps we give you a pre-configured `pom.xml` file to replace the `pom.xml` file generated by the wizard. This file can be found in the `di-prog-lab-i.zip` file. Perform the following steps:

1. Extract the `pom.xml` and `web.xml` files the from the `di-prog-lab-i.zip` file.
2. Copy the `pom.xml` file to the root directory of the Maven project, which should replace the `pom.xml` file generated by the wizard.
3. Create a directory called `WEB-INF` in `src/main/webapp` and copy the file `web.xml` from the `di-prog-lab-i.zip` file to this directory (this may take a while to return).

5.2 Synchronising the Eclipse and Maven projects

The next step is to configure the Eclipse project and synchronise these settings with the Maven project. This should remove the errors in your project. In order to connect Tomcat 8.5 to the Maven project, right-click on the project and choose Properties from the context menu, choose Project Facets, then:

- Set Java version 1.8.
- Set Dynamic Web Module version 3.1 and click on Apply.
- Still in this menu, click on Runtimes (on the right), select Apache Tomcat 8.5 and click on Apply and OK.

Your project probably still has an error because the Eclipse project is not synchronised with the Maven project, so right-click on the project and choose Maven → Update project. You now have a Maven / Eclipse project that we will use to build up, test and deploy a simple web application.

6 Test-based design in Maven

You probably noticed the directory `src/test` in Figure 3, where Maven keeps the test resources. Maven supports test-based design since it automates tests based on JUnit. In order to illustrate how it works you should take the following steps.

In a browser go to http://mvnrepository.com/ and look for JUnit. Select the newest version and copy the dependency definition to the dependencies list in the `pom.xml` file, just below the `<dependencies>` tag. You should get the following code:

```
...
      <dependencies>
       <dependency>
             <groupId>junit</groupId>
             <artifactId>junit</artifactId>
             <version>4.12</version>
       </dependency>
...
```

Now create the directory hierarchy `nl/utwente/di/bookQuote` in directory `src/test/java/` in your Eclipse project, and define a class `TestQuoter` in this directory with the following code:

```
package nl.utwente.di.bookQuote;

import org.junit.Assert;
import org.junit.Test;

/** *  Tests the Quoter */
public class TestQuoter {

  @Test
  public void testBook1() throws Exception {
    Quoter quoter = new Quoter();
    double price = quoter.getBookPrice("1");
    Assert.assertEquals("Price of book 1", 10.0, price, 0.0);
  }
}
```

Add a `Quoter` class to `src/main/java/nl/utwente/di/bookQuote` that has a dummy implementation of the `getBookPrice()` method (e.g., with **`return`** `0;` in the body) so that the compilation errors of the project disappear.

To let Maven run the test you have to right-click on the project in Eclipse and choose Run As ... → Maven test.

Now answer the following questions: Which result did you obtain? How do you explain it?

Give a body to the `getBookPrice()` method in the `Quoter` class that gives the following results depending on the input:

isbn	1	2	3	4	5	others
result	10.0	45.0	20.0	35.0	50.0	0.0

This can be done easily by defining a `HashMap<String,Double>` object to store these values in the `Quoter` class, and using this `HashMap` object to query for the proper values.

Run the test again with `Run As ... → Maven test`. Did you get the expected result?

7 Implementing the Book Quote web application

Until now we have one class that returns the price of books based on their ISBN, but this is not a web application. In order to implement a web application we need a Servlet, which is an object that handles HTTP messages coming from the network. Take the following steps:

- Extract the `BookQuote.java` file from the `di-prog-lab-i.zip` file and copy it to the `nl.utwente.di.bookQuote` package in Eclipse. `BookQuote.java` extends the HTTP Servlet for this specific application. Inspect this file and answer the question below: How does the BookQuote Servlet work, i.e., what is the input and the output of method `doGet()`?
- Our web application will also need an initial page from which we can reach the `BookQuote` Servlet. This page and its formatting can be also found in the `di-prog-lab-i.zip` file (`index.html` and `styles.css`, respectively). Copy these files to directory `src/main/webapp` of your Maven project.

Now we will inspect file `web.xml` that you copied to directory `src/main/webapp/WEB-INF` a while ago. This file is called a *deployment descriptor*, and defines the mappings of URLs to Servlets that allows messages to be forwarded to the `BookQuote` Servlet in our application. Check the `web.xml` file and answer the following question: Which information does this file give to the Application server?

Now right-click on the project again and choose `Run As ... → Maven install`. Observe the result of this command, especially what happened with directory `target`. In this directory, a directory called `bookQuote` and a file `bookQuote.war` should have been generated, which will be used later when the web application is deployed. Don't worry about this for the time being.

Your application is ready to be deployed and tested!

8.1 Deployment in Eclipse

We mentioned before that Eclipse can be used to test a web application before deployment because it runs the Application server (Tomcat 8.5) internally. This can be done by adding an Eclipse project to the Tomcat installation. The following steps should be taken for that:

1. In the Java EE perspective, select the Servers tab, right-click on Tomcat 8.5 Server at localhost and choose Add and Remove…
2. Select the `bookQuote` project and click on Add > and Finish.
3. Start the server by right-clicking on the server again and choosing Start.

Your first web application should be running now.

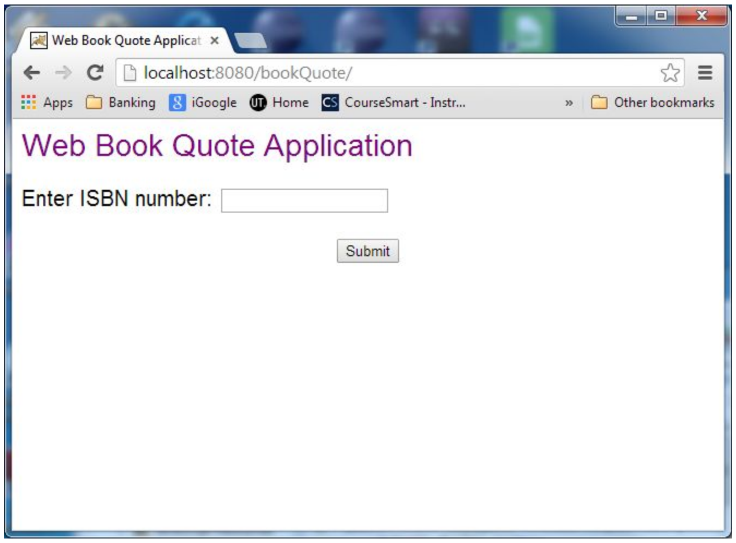
8.2 Deployment in Tomcat

Alternatively you can deploy the application in Tomcat. This can be done by copying the `bookQuote.war` file generated by Maven to the `webapps` directory of the Tomcat installation and starting the server. The server then unpacks the `bookQuote.war` file to a `bookQuote` directory and runs the application (an `.war` file is actually a `.zip` file).

9 Testing the application

Web applications get an URL that starts with the address of the Application Server, followed by the name of the web application itself and the name of index file (`index.html` or `index.htm` do not need to be mentioned).

In a browser, go to the address http://localhost:8080/bookQuote/. You should get the page shown in Figure 4, which is the initial page of your web application.

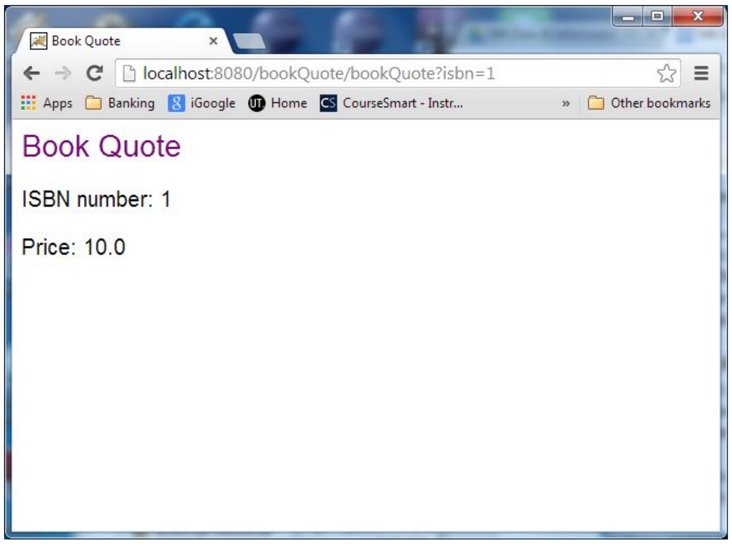


Web Book Quote Application

Enter ISBN number:

Figure 4. Book Quote application index page.

After inserting value 1 and pressing Submit the page shown in Figure 5 should be displayed.



Book Quote

ISBN number: 1

Price: 10.0

Figure 5. Result of the Book Quote application request.

Congratulations! Your first web application is up and running!

10 Uploading the Maven project to GitLab

Git (http://git-scm.com/) is the version control system that you will use in this module. There is a Git plugin for Eclipse, but in order to properly learn how Git works in this tutorial we run it from the command line. GitLab is an open source Git repository and you are going to store the deliverables produced in this module in the GitLab repository supported by SNT. As an exercise in this tutorial you will save the Maven project (the contents of the `src` directory and the `pom.xml` file) to your local Git repository and after that to a GitLab project.

10.1 Preparation

In order to prepare for this you should take the following steps:

1. Study Chapter1 “Getting Started” and Chapter 2 “Git Basics” of the Git documentation at http://www.gitscm.com/doc
2. Install Git in your computer by following the instructions in the documentation (Chapter 1).
3. Sign in to https://git.snt.utwente.nl/ with your UT student number.
4. Create a project bookQuote in GitLab.

10.2 Creating and populating the local Git repository

Now you have to create the local Git repository and add the files from the Eclipse project to this repository. This can be done with the following steps:

1. Create a directory in your laptop called for example `git` where your local Git repositories will be located.
2. Clone project bookQuote in this directory from GitLab (check the Git documentation to learn how this can be done).

3. Copy the `pom.xml` file to the `bookQuote` directory and add this file to the local Git repository (check the Git documentation to learn how this can be done). The `pom.xml` file can be found in `<workspace>/<project-name>`, where `<workspace>` is the directory where the Eclipse workspace is located and `<project-name>` is the name of the Eclipse project. You can also copy this file directly from Eclipse with Copy and Paste.
4. Copy the `src` directory of the Eclipse project to the `booQuote` directory and add this directory to the local Git repository.
5. Commit all the changes to your local Git repository.

10.3 Pushing the contents of your local repository to the remote repository

Finally you have to push the contents of your local repository to the remote repository so that the instructors of the course can inspect your work. Check the Git documentation before doing this.

Remark

In this exercise we have asked you to copy the contents of the Eclipse workspace to another directory where the local Git repository is created. Alternatively you could create your repository in the Eclipse workspace directory, and in this case no copying is necessary. The downside is that your workspace will then be the location of your local Git repository, so two different applications will manipulate the same directory. Find a workable alternative as soon as possible so that you can work in a productive way.

An alternative to manipulating your Git repository with the command line is to use the Eclipse Git plugin. A tutorial on how to do this can be found at
http://www.vogella.com/tutorials/EclipseGit/article.html.
Yet another alternative is to use the a Git GUI client. Make sure you use a Git GUI client that is not bound to GitHub, which we are not using in this module.

11 Celsius to Fahrenheit translator

Based on the tutorial so far, implement a web application that gets a temperature value in Celsius and translates it to its corresponding Fahrenheit value. Create a new project so that you don't 'cannibalise' the results of the first part of this session.

You will have to check/modify at least the following parts of the code:

1. Servlet implementation, which should handle the application logic (the calculation);
2. Index page (index.html), which should ask for a temperature in Celsius;
3. Deployment descriptor, which defines the URL for the servlets and connects these URLs to the proper Java classes.

Make sure your web application works as expected. When you are done create a local Git project called, for example, `temperature` and push it to your remote Git repository.

12 Additional exercises

1. Make a list of the terms (referring to techniques, tools, concepts, etc.) from the instructions in this exercise that you don't understand.
2. Look for resources on the Internet (blogs, whitepapers, videos, conversation threads, etc.) that contain information on these terms.
3. After studying these materials, rank them from the most useful to the least useful.

Comments and suggestions for improving this tutorial session are welcome and can be sent to mailto:l.ferreirapires@utwente.nl.