WIRTSCHAFTSUNIVERSITÄT WIEN
Vienna University of Economics and Business

Bachelor's Thesis

Titel of Bachelor's Thesis (english)	Developing Jakarta Server Pages as a Front-End Developer - An Alternative Approach to Developing Server-Side with Node.js
Titel of Bachelor's Thesis (german)	Entwicklung von Jakarta-Serverseiten als Frontend-Entwickler - Ein alternativer Ansatz zur serverseitigen Entwicklung mit Node.js
Author (last name, first name):	Besenbäck, Simon
Student ID number:	11801501
Degree program:	Bachelor of Science (WU), BSc (WU)
Examiner (degree, first name, last name):	ao. Univ.-Prof. Dr Rony G. Flatscher

I hereby declare that:

1. I have written this Bachelor's thesis myself, independently and without the aid of unfair or unauthorized resources. Whenever content has been taken directly or indirectly from other sources, this has been indicated and the source referenced.
2. This Bachelor's Thesis has not been previously presented as an examination paper in this or any other form in Austria or abroad.
3. This Bachelor's Thesis is identical with the thesis assessed by the examiner.
4. (only applicable if the thesis was written by more than one author): this Bachelor's thesis was written together with

   The individual contributions of each writer as well as the co-written passages have been indicated.

08/02/2023
Date

Besenbäck Simon
Unterschrift

Abstract

Usually server-side web development requires front-end developers to learn at least one additional programming language like Python, PHP, or Java. However, the process of learning the syntax of a new language is normally very time consuming and especially for novice coders extremely exhausting. Though, due to the invention of tag libraries it is possible to use scripting languages like JavaScript for developing JSP's. Therefore, this bachelor thesis elaborates the theoretical background of developing web-server-client-applications with tag libraries hosted on an Apache Tomcat server. Furthermore, several web applications running on an Apache Tomcat Server get developed, demonstrated, and explained. The web server of choice for those applications is Apache Tomcat as it is a very popular open-source web server and able to run servlets and Jakarta Server Pages (earlier Java Server Pages) (Vukotic & Goodwill, 2011). The tag library JSR-223 enables that the scripting language JavaScript is appliable in the web server. The developed web applications can be downloaded under *https://github.com/simon-1337/thesis-projects*.

Content

Listings

Figures

1 Introduction

In 2022 JavaScript has been the most used programming language again (GitHub Inc., n.d.). Furthermore, 72% of all companies are looking for JavaScript developers (GeeksforGeeks, 2022). Thus, many developers are competent in this language and a lot of people will start to learn it as there are many job opportunities. However, vanilla JavaScript (without frameworks, etc.) is usable for front-end development only. Therefore, developers who are only capable of the language JavaScript cannot develop a fully functional web application without learning a second language like Python, Java or PHP, which requires a lot of effort and time.

Having said this, the framework Node.js is getting more and more popular over the last years (Fireart Studio, 2019). Node.js is a cross-platform runtime environment with the purpose to execute JavaScript code. This allows developers to run JavaScript code on the server-side and thus build web applications using JavaScript on the client-side as well as in the back end. Following that, Node.js adds additional functionality to JavaScript, which is needed for back-end development (Sheldon & Denman, n.d.).

However, there is a second but less popular approach by which JavaScript can be used to develop a complete web application from the front end to the back end. The intended approach is developing Jakarta Server Pages (JSP) using a scripting engine like Rhino or Nashorn. For these thesis projects the scripting engine Nashorn was chosen. Unfortunately, some required server-side language functionalities are not implemented in JavaScript and Nashorn. Though, the tag library jsr-223.tld developed by Rony G. Flatscher can be used to circumvent this issue (Flatscher, 2021). The server used for providing the built web applications was Apache Tomcat.

The goals of this paper are to use those technologies to develop example web applications, to demonstrate how such development can be performed and to ease the first steps for developers interested in using those technologies. The example applications are of increasing complexity and are inspired by the programs developed by Lux (2021). The applications developed can be downloaded under *https://github.com/simon-1337/thesis-projects*.

The rest of the paper is structured as follows. The first part of the paper will discuss the needed theoretical backgrounds to understand the paper independent of the level

of expertise in this field. To ensure that the developed programs are reproducible by any reader the next part describes the necessary installation and setup steps in detail. After that the basic nutshell examples developed will be explained. This section will provide some fundamental knowledge regarding the development of JSP. The fourth section of the paper discusses the development of an example online shop application. Next, two more advanced examples are presented. In this section a program able to send emails as well as one including a file upload are explained. The last part of the paper presents a conclusion of the key aspects. Moreover, remaining open questions which open a further field for future research are discussed in this section.

2 Theoretical Background

The purpose of this section is to present a short overview of some essential background knowledge necessary to understand the topic.

2.1 What is Web Development?

Web development is the practice of creating and maintaining websites. This involves the design, layout, content, and functionality of the website. Web development is a broad field that can be divided into several subcategories. These include front-end, back-end and full-stack development. Each of these subcategories deals with different aspects of a website. (BrainStation, n.d.).

2.2 Front-end vs. Back-end Development

This part emphasizes the differences between front-end and back-end development.

2.2.1 Front-end Development

Front-end development is a part of web development that focuses on the user-facing side of a website. It is the process of ensuring that visitors can easily interact with and navigate the site. Front-end developers use tools such as programming languages, design skills, and other tools to create the drop-down menus, layouts, and designs of a website. The most used technologies in front-end development are HTML, CSS and JavaScript. HTML is a markup language used to create the structure and layout of the website, CSS is used for the design of the website, and JavaScript is used for creating features which enable users to interact with the website. However, it is not very common for front-end developers to know other languages like Python, Ruby or Java (Simmons, 2022).

2.2.2 Back-end Development

Back-end development is about working on the server-side software of a website, which is everything that is not seen when visiting it. Back-end developers make sure that the website works correctly by working on back-end logic, databases and servers. By writing code, they enable browsers to interact with databases and perform actions such as storing, understanding, and deleting data. Back-end developers create the foundation of a website or mobile app by using languages such as Python, Java, and

Ruby. Ensuring the back-end works smoothly, quickly and effectively in response to requests from the front-end is also a main part of their working field. (Coursera, 2022).

2.3 What are Jakarta Server Pages

Jakarta Server Pages (JSP) is a technology that developers use to create dynamic web pages for Java web applications. It is based on Java servlet specification, and it is a part of Jakarta EE. In JSP, developers write a client-side script or markup and then use JSP tags to connect the page to the back-end Java code. Therefore, JSP can be compared to giving HTML superpowers to interact with the back end.

Usually, HTML is sent to the client and dynamically changed on the client side with JavaScript. However, with JSP, HTML content is dynamic by pre-processing it with special commands to access the server capabilities and then the individually adapted and compiled page is sent to the client (Tyson, 2022).

2.4 Tag Libraries

A custom tag library is a useful tool for web designers who want to enhance their website without having to know Java. These libraries consist of a set of custom tags that can invoke custom actions in a JSP file. The use of tag libraries can help to separate the presentation of a website from its implementation, which makes maintaining and reusing it easy. Another significant benefit is that complex actions are simplified as Java coded functions are provided without the need for coding in Java. Additionally, custom tag libraries can be used to dynamically generate page content and control the flow of a website. Overall, custom tag libraries offer a number of benefits for web developers looking to add functionality to their website (International Business Machines Corporation (IBM), 2021).

2.5 Why Develop JSPs with Nashorn?

The question is: “Why would anyone develop a web application by using JSP and Nashorn?”. To answer this question it is important to understand what Nashorn is first. Nashorn is a JavaScript engine introduced in Java 8 and was part of the JDK until Java 14. However, Nashorn development continues as a standalone OpenJDK project on GitHub (Wikimedia Foundation, 2022b). It allows developers to run JavaScript code within the Java Virtual Machine (JVM). This enables Java and JavaScript to interact

with each other and allows developers to invoke JavaScript from Java applications as well as to use JavaScript for the development of JSPs (Bandara, 2018).

As explained in section 2.3 JSP allows easy interaction with the server. Furthermore, it is possible to write JavaScript code instead of Java, by using the Nashorn JavaScript engine. Therefore, it is easy for front-end developers who know JavaScript to get into back-end development by using JSP and Nashorn. That way developers can use JavaScript to develop both the front end and back end of an application.

Additionally, JSP and Nashorn can be integrated with other Java technologies such as JDBC which allows for easy interaction with databases and other data sources. This gives developers even more power and flexibility when building server-side applications.

2.6 script-jsr223.tld

Unfortunately, Nashorn and JavaScript do not support XMLHttpRequests. XMLHttpRequests are used to handle data sent from a client to the server. Thankfully, the tag library script-jsr223.tld (section 2.7.4 explains where it can be downloaded) developed by Rony G Flatscher offers the functionality needed for a web server so that any scripting language can be used for back-end development. Therefore, this tag library provides the attributes request and response necessary for interacting with a client (Flatscher, 2021).

In conclusion Nashorn together with the tag library fulfills the requirements to develop full stack web applications. This enables developing complete web applications as a front-end developer, who is solely capable of the programming language JavaScript.

2.7 Installation and Setup of Software Components

This section provides a detailed installation guide for all necessary software components to ensure that readers of all different backgrounds are able to reproduce the developed web applications.

2.7.1 OpenJDK

To start the installation process a Java Development Kit needs to be installed. However, Java does not support Nashorn in Java 15 and later. Though, fortunately there is

an OpenJDK project which is developing an open source JDK which still has support for Nashorn. The process of installing OpenJDK is as follows. The first step is to go to the website https://jdk.java.net/. On this page the button *OpenJDK Early Access Builds* needs to be clicked to open the required page.

The next step is to select the latest JDK version found in the section “Ready for use”. The latest version by the time this thesis was written was JDK 19. Selecting the version forwards the visitor to a page showing a lot of information about the selected version. Among others the builds which can be downloaded are shown approximately in the middle of the screen. The correct version according to the operating system needs to be donwloaded. In the case of this thesis, the used operating system was Windows, so the next steps are relevant only for Windows users.

To install OpenJDK on Windows it is required to download the ZIP archive file with the description *Windows/x64* next to it. After the file is downloaded it needs to be extracted to *C* drive. To do so, the user needs to right-click on the file and then click on the *Extract all…* menu item. Now it is required to select an extract destination for the JDK files. Within this thesis the destination of extraction chosen was C:\jdk-19 (*How to Download and Install OpenJDK 11 on Windows 10 PC for Aleph*, 2020).

After the installation process is completed, a slightly more complex step follows. In order to use the JDK installation it is required to set up an environment variable which points to the OpenJDK installation.

> An environment variable is a dynamic "object" on a computer, containing an editable value, which may be used by one or more software programs in Windows. Environment variables help programs know what directory to install files in, where to store temporary files, and where to find user profile settings (*What Is an Environment Variable?*, 2018, para. 1).

This process is described in a very detailed manner. However, if anything is unclear and a user does not know what exactly is to do, it is recommended to get help by a person with some knowledge in this area or to do some more research, as the behavior of some applications or the whole system might change when environment variables get altered.

To correctly set up the environment variable it is necessary to open the system properties. The fastest and easiest way to do so is to click on the search button in the Windows taskbar and enter *env*. The next step is to click on the shortcut *Edit system environment variables*. Now after the system properties have opened, the user sees the enhanced register of the system properties. Searching for the button called *environment variables* needs to be done next. This button can be found on the bottom right of the system properties window. After clicking this button, the environment variables window opens. Once opened a table containing the user variables and another one containing the system variables are visible. The later one is the one that matters for this case. To proceed, the next thing to do is to click on the *new* button belonging to the system variables. Now the user is required to enter a variable name and a variable value. For the variable name one needs to enter *JAVA_HOME* and for the variable value it is important to enter the exact path of the Java installation directory that was chosen before. In case of this thesis the correct installation directory is C:\jdk-19. After clicking on *Ok*, the new system variable is visible in the list of system variables (*How to Download and Install OpenJDK 11 on Windows 10 PC for Aleph*, 2020). This step was fulfilled correctly, if the system variables have an entry looking like the marked one in Figure 1.

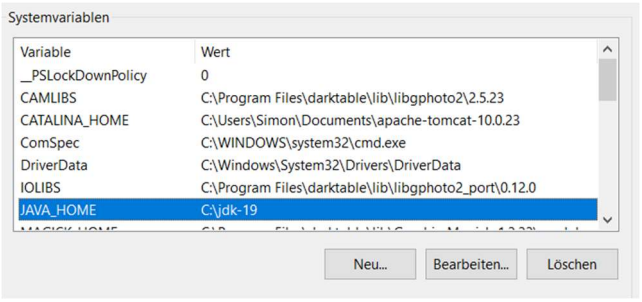

Figure 1: Screenshot of the system variables

After the successful creation of a new system variable one last step is needed to ensure that Java is fully functional. For this, it is needed to stay in the opened window and search for the system variable *Path*. As the next step it is required to select the *Path* variable and click on *edit*. In the window which has been opened a button named *new* can be found. The next step is to click on this button, enter *%JAVA_HOME%\bin* and click *move up* until the new path is on top. Figure 2 shows the path variable after the previous steps have been done. Finally, the only thing left to do is closing the open windows by clicking on *ok*.

For the case that there cannot be found an environment variable called *Path*, the user is required to create one. The same procedure as creating the *JAVA_HOME* variable can be applied. However, the user needs to enter *Path* as variable name and *%JAVA_HOME%\bin* as variable value. For the case that the operating system is Windows 7 there is no additional window which can be opened by selecting *Edit*. Instead the user is required to append *;%JAVA_HOME%\bin* at the end of the variable value (How to Download and Install OpenJDK 11 on Windows 10 PC for Aleph, 2020).

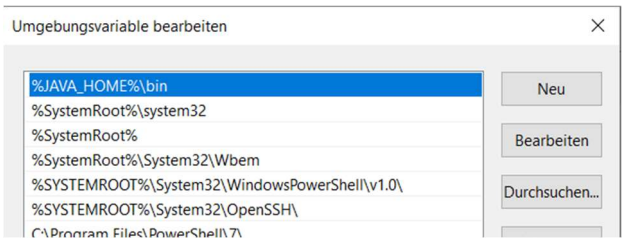

Figure 2: Screenshot of the path environment variable

Important Note: The *path* variable as well as the *system variables* will look a bit different on every system. Figure 1 and Figure 2 solely present an example how the basic structure of the variables roughly looks like. The user should not perform any other changes then the one described in the previous section.

Once all above steps have been completed, Java is ready to use. To test if everything works as expected the CMD (integrated command-line interpreter by Windows) needs to be opened. To do so the easiest way is to click on the search icon in the Windows task bar and enter *cmd* and click on the *Command Prompt* shortcut. Once the command line or also called console is opened the command *java -version* can be entered. After pressing enter the console prints the installed OpenJDK version. If the console correctly prints out the version of the installed OpenJDK the process of installing and setting up OpenJDK is successfully accomplished (*How to Download and Install OpenJDK 11 on Windows 10 PC for Aleph*, 2020).

2.7.2 Apache Tomcat & Eclipse IDE

In this thesis an Apache Tomcat server was used to develop web applications in the Eclipse IDE. This section describes the necessary installation and configuration steps for this setup. If a developer is using a different combination of server and IDE, some steps described below might differ.

To start with, the Tomcat server as well as the Eclipse software need to be downloaded from their homepages. The latest Eclipse software version can be found on their homepage eclipse.org. On the webpage the user will immediately notice the download button for the latest Eclipse IDE. To download the latest available stable version of Apache Tomcat, the user needs to visit the webpage tomcat.apache.org. Once the website is visible in the browser, a download area within a navigation section on the left of the homepage is displayed. Identifying the latest stable version is quite simple. The user only needs to look for the version with the highest version number. Versions without alpha or beta in brackets are stable versions (Apache Software Foundation, n.d.-a). When the latest version was clicked on a new website will open. On this webpage the user needs to look for the section *Binary Distributions*. This Binary Distribution part contains a subitem core. To download Apache Tomcat the link named zip, placed below the subitem, has to be clicked. As soon as the download is completed it is required to extract the files inside the zip archive to the directory where the server should be stored. In the case of this thesis the documents folder was chosen.

After that the Eclipse IDE which is already downloaded needs to be installed and opened. In the bottom middle of the Eclipse IDE the tab servers can be found. The servers tab will display that there are no servers available as well as the link "*click this link to create a new server…"*. Clicking this link opens a new window, where the user needs to select the server type. After the correct server is selected the user can click on finish to close the window. Next a new window will open which requires the user to select the directory of the Apache Tomcat installation and select the JRE. For JRE, the button *installed JREs* needs to be clicked on and the OpenJDK installation needs to be selected. If the instructions were followed exactly this should be C:\jdk-19 (or newer than 19). If everything was done correctly the Tomcat server will be visible as *[Stopped, Republish]* under the *Servers* tab in the bottom of your Eclipse window. In order to know the port, on which the webpages on this server can be accessed, the user needs to double click on the Tomcat server. After a few seconds a new window will open which displays an overview of the server settings. Within this window a section called *ports* can be found. Under this section in the row with the port name *HTTP/1.1* the port number can be seen. The default value for the HTTP port is 8080.

Now the user can try if the server is already working as desired. Yet it is not uncommon that there are still some changes needed to be made. To determine if the server is already fully functional, the user needs to right click on the server and select *Start*. As soon as the server in the servers tab is displayed as *Started, Synchronized*. The server can be accessed via the URL: http://localhost:8080/ in any Browser (Shah, 2019a). Figure 3 displays how the website will look like if everything is already working.

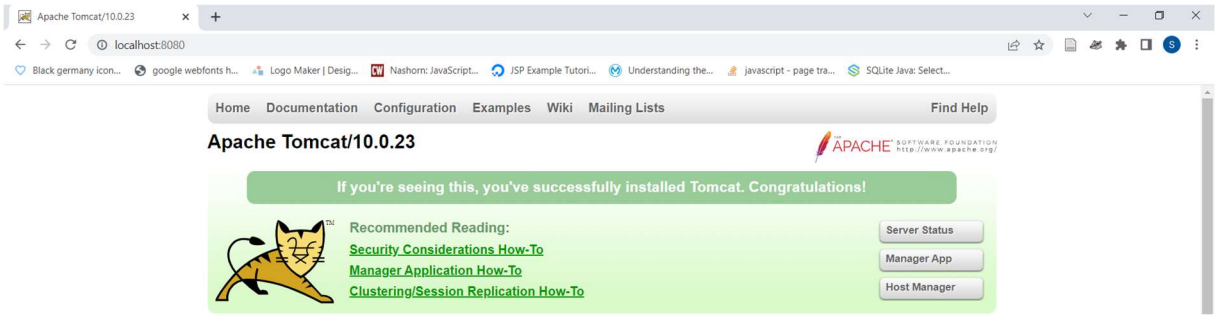

Figure 3: Apache Tomcat Server

However, the user needs to configure a few more things if the error *404 – Page not found* occurs. The first step is to go back to the Eclipse IDE and double click on the Tomcat server again. Once the overview page is opened the user needs to go to the section called *Server Locations*. There it is necessary to select the checkbox *Use Tomcat installation (takes control of Tomcat installation)*. After this step is done the user needs to save the change before closing the tab. To test if it is working now, the server needs to be restarted by right clicking on it and selecting *Restart* (Shah, 2019b). Now entering the URL: http://localhost:8080/ in the Browser will open the website shown in Figure 4.

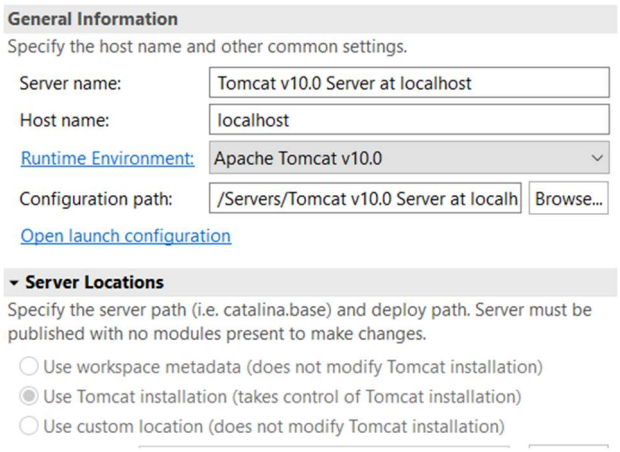

Figure 4: Server Overview - "Server Location"

2.7.3 Configuring Manager Application Access

For a programmer to be able to access the developed JSPs, it is required to configure a username and password by which the manager application of a Tomcat web server can be accessed. By default, the access to this manager application is strictly forbidden for the user. This is a necessary safety measure to ensure that instead of anyone on the internet, only authenticated users can access the manager application of a web-server. To grant access to the Manager web application, a developer can either create a new username and password, then assign one of the manager-xxx roles to it. Or it is also possible to add one of the manager-xxx roles to an existing username and password combination. Creating new usernames and passwords as well as new roles is done in the tomcat-users.xml file of the Apache Tomcat server. (Apache Software Foundation, n.d.-c). This file can be found inside the *conf* directory of your Tomcat installation. Again, in the example of this thesis the Apache Tomcat folder can be found inside the documents directory.

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

Listing 1: tomcat-users.xml

Listing 1 displays the configuration necessary to access the manager application with the username *test* and the password *test*. However, when configuring this file a developer needs to make sure that the configured lines are not within a comment. A comment in an XML-File is everything between "*<!--"* and "*-->"*.

However, if Eclipse is used as IDE as soon as the server is closed and later restarted the configuration done before will be lost and therefore the manager application is not accessible anymore. To prevent this from happening the configuration needs to be done in the tomcat-users.xml in the eclipse-workspace (Flynn, 2013). This is because Eclipse makes a copy of the server configuration files in the workspace (Mihn, 2019). For the development of this project this means the configuration needs to be done in the tomcat-users.xml file found under this path: *C:\eclipse-workspace\Servers\Tomcat v10.0 Server at localhost-config*.

To check if everything is working as intended the manager application button on the starting page of the Tomcat server needs to be clicked. Following that the user is

requested to enter the username and password chosen in the tomcat-users.xml file. If the credentials are entered correctly and the configuration above was done in a correct manner the user will be forwarded to an overview of the different web projects currently available on the server. By clicking on a link of a project the user can now open the web application and will be directed to the index.html or index.jsp page of the application.

2.7.4 script-jsr223.tld and jakarta.ScriptTagsLibs.jar

To allow the usage of Nashorn code for developing JSPs it is necessary to download two specific files, namely *the jakarta.ScriptTagLibs.jar* as well as the *script-jsr223.tld* from the URL https://sourceforge.net/projects/bsf4oorexx/files/Sandbox/rgf/taglibs/ga/. The *jakarta.ScriptTagsLibs.jar* needs to be added to the *lib* directory inside the Apache Tomcat folder. Afterwards the path to this folder should look somewhat like this: *C:\Users\Simon\Documents\apache-tomcat-10.0.23\lib*. The *script-jsr223.tld* file needs to be added to the *WEB-INF* folder inside a Dynamic Web Project later (Flatscher, 2021). However, this will be explained in more detail when the creation of a new Project gets described in section 3.1.

2.7.5 nashorn-core-15.4.jar

In order to enable the usage of the Nashorn scripting engine, downloading the Nashorn jar file is required. As Nashorn is not included in JDK since Java 15, the jar file needs to be downloaded from the URL: https://search.maven.org/artifact/org.openjdk.nashorn/nashorn-core/15.4/jar (*Nashorn Engine*, 2020/2022). The dropdown on the upper left can be clicked to check if a newer version is available. Once the latest version of the Nashorn scripting engine is selected, the jar file can be downloaded. To do so the user is required to select the downloads button on the upper right of the page. After clicking the button, a dropdown opens where jar needs to be selected.

When the file is downloaded it can either be added to the *lib* directory of the server or to the *WEB-INF* folder inside a project. By adding it to the project, other developers can use the project without having to add the jar files to their server first. However, adding the jar file to the *lib* directory of the Tomcat server enables the usage of Nashorn in each new Project without any further actions (Lux, 2021). Therefore, the latter option was chosen.

2.7.6 ASM

ASM is a framework that allows for the manipulation and analysis of Java bytecode. With it, you can modify existing classes and create new classes directly in binary form (*ASM*, n.d.).

According to Szegedi (2022), the Nashorn scripting engine depends on the ASM 7.3.1 files. To enable the usage of Nashorn in Apache Tomcat for developing JSPs it is necessary to download all 4 jar files and add them to the *lib* directory in the Apache Tomcat server. The required jar files are *asm-7.3.1.jar*, *asm-commons-7.3.1.jar*, *asm-tree-7.3.1.jar* and *asm-util-7.3.1.jar*.

A user can download those ASM jar files from the OW2 Maven repository under https://repository.ow2.org/nexus/#welcome. On this page one simply needs to search for the 4 jar files by entering their names in the search bar (*ASM*, n.d.).

3 Basic Nutshell Example

The applications demonstrated in this section provide an introduction into developing JSPs with Nashorn. And are of lower complexity than the programs developed later in this paper. Like all the applications developed in this thesis, the projects discussed in this section are based on the projects developed by Lux (2021) too.

3.1 Create a Dynamic Web Application Project

First the user needs to open the Eclipse application. Once the Eclipse IDE has opened the easiest way to create a new dynamic web project is to click the right button of the mouse somewhere in the project explorer of Eclipse (on the left side). Then the next step is to click on *New* and select *Dynamic Web Project*. Now a Project name needs to be chosen and afterwards the check for *Use default location* needs to be removed. If this check is not removed, the application will be stored into the eclipse-workspace. After that it's necessary to click on *Browse* and search for the Apache Tomcat server. Which is a folder normally named *apache-tomcat* followed by the version number (e.g., apache-tomcat-10.0.23). The project needs to be saved inside the webapps folder of the Apache Tomcat server. Thus, the user needs to click on webapps and afterwards right click inside the window and create a new folder. It is reasonable to choose the same name for the project and the folder. The newly created folder needs to be selected as the project location. Next *finish* needs to be clicked and the web project will be created. If the newly created project is now visible in the project explorer of Eclipse the process was successful.

Figure 5: Example of a new created dynamic web project

Following the successful creation of a new project the next step is to add the downloaded *script-jsr223.tld* file to the *WEB-INF* folder of the created project. As the folder is already existing inside every project, it is not necessary to create a new one. However, the path to this folder is long (e.g., new_project/src/main/webapp/WEB-INF).

Therefore, it makes sense to create a new folder with the name directly inside the new created project (e.g., new_project/WEB-INF). Now the file script-jsr223.tld must be placed within this folder. If the newly created dynamic web project looks like the one displayed in Figure 5 the steps have been executed correctly. To create new files like JSP, HTML, CSS, and so on, the user now can click the right mouse when on the newly created project and select the type of file. After that the created files can be accessed via localhost:8080/new_project.

Important Note: It is crucial to add the new files directly inside the new created project. By default, the files are often saved inside *src/main/webapp*.

However, if the projects provided on the GitHub account with the name *simon-1337* are used the only thing to do is to add the .war file to the webapps folder of the Apache Tomcat server and start the server. Afterwards the project will be automatically unpacked, and a new folder will be created with the name of the .war file. To look at the code or even adapt it, the user can now click *File* on the upper left in Eclipse and then select *Open Project from File System*. Here it is required to navigate to the webapps folder inside the Apache Tomcat server again and select the folder which was automatically created by the .war file. Figure 6 displays how the helloworld project will look like after it is unpacked and opened in Eclipse. The following sections are discussing the files in this project.

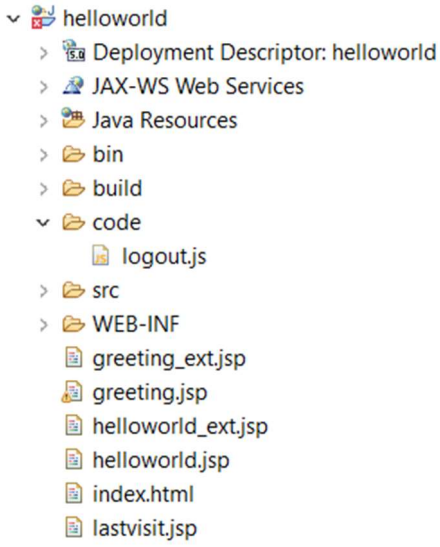

Figure 6: helloworld project in project explorer

3.2 Introduction to JSP

The technology of Jakarta Server Pages or short JSP, is already introduced in section 2.2. The goal of the following paragraphs is to present the most important characteristics and features one needs to know when developing JSPs.

3.2.1 Structure of JSP Files

The following part discusses the required structure of JSPs in more detail.

helloworld.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib uri="./WEB-INF/script-jsr223.tld" prefix="s" %>
```

Listing 2: JSP directives

The first extremely crucial parts of a JSP are the directives or more precisely in this case the *page* and the *taglib* directives. Those directives are shown in listing 2. The *page* directives define several properties dependent on the current page. These properties are then communicated to the JSP container. The first property of the *page* directives specifies that the used language is a scripting language based on Java. The *contentType* property defines the MIME type as well as the character encoding, which is in our case a standard html file and UTF-8 encoded (*Jakarta Server Pages*, n.d., Sec 1.10). MIME Type is short for Multipurpose Internet Mail Extensions and describes which type of data gets sent (*MIME-Type/Übersicht – SELFHTML-Wiki*, 2022).

However, as those *page* directives are set by default when a new JSP is created, there is no effort needed by the programmer because the page directives in listing 2 fulfill the requirements for the applications of this work. Though this is different in the case of the *taglib* directive which is needed to declare where the tag library used in this page can be found. By using the *uri* attribute the tag library chosen is uniquely identified. The prefix defines the symbol that specifies the library that should be used in a particular section (*Jakarta Server Pages*, n.d., Sec 1.10). In our case the *uri* needs to point to the script-jsr223.tld file. If the instructions of this work were followed the required path will be *./WEB-INF/script-jsr223.tld*.

The next part of the *helloworld.jsp* file is HTML structure. Here again the HTML code visible in listing 3 is inside every newly created .jsp file by default. The first line

describes the type of the current document, which is *html* in this case. This is a crucial part as every HTML document must start with this declaration (W3Schools, n.d.-b).

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Hello, World</title>
</head>
<body>

</body>
</html>
```

Listing 3: HTML backbone

The markup language consists of many tags. Almost every tag has a start and an end tag. Opening tags always consist of a *less-than sign* followed by the name of the tag and end with a *greater-than sign* (e.g., <*tag*>). Closing tags look nearly the same, except for the small difference that between the less-than sign and the tag a dash is required (e.g., <*/tag*>). After the declaration of the document type the first tag needed is the <*html*> tag. This one is the root of the HTML document. The space between the start tag (<*html*>) and the end tag (<*/html*>) is the container for all other HTML elements (W3Schools, n.d.-g). The next tag which should be present in any HTML page is the head. This element is placed between <*html*> and <*body*> and is the container which is responsible for metadata. This means inside the head is any data that describes the HTML document and is important for the document to correctly be displayed. However, the metadata itself is not displayed in the document (W3Schools, n.d.-e). There are several tags that can be used inside the head. For example, the title is strictly required in any HTML document. The text inside this tag is shown in the title bar of the browser or in the page's tab. Thus a user is only allowed to write text inside this tag (W3Schools, n.d.-h).

After the head is closed the next tag in the example is the <*body*> tag. The body is the container that is filled with all contents of an HTML document that are visible for a visitor of the webpage. Those contents include headings, paragraphs, hyperlinks, buttons, images and many more (W3Schools, n.d.-a)

```
<s:script type="javascript">
      var greeting = "Hello, world! (Sent from Nashorn)";
      print("<div>" + greeting + "</div>")
</s:script>
```

Listing 4: helloworld script tag

The file helloworld.jsp also consists of a short scripting code inside the body. This code is inside of the *<script>* tag, which determines that everything within these tags is written in a scripting language. The *'s:'* before script needs to be equivalent to the prefix chosen above in the directives for the tag library. Adding this prefix before *script* enables that the customized tags of the tag library can be used inside the area between the opening and closing script tags. As listing 4 shows, a developer needs to declare which scripting language is used inside these tags. In the case of this project the language that needs to be assigned to the *type* attribute is *javascript*. From now on it is possible to write JavaScript/Nashorn code inside the script tag. Furthermore, due to the prefix a developer can also use all customized tags of the tag library. A full list of all attributes that can be used by this tag library can be found under this link: https://sourceforge.net/projects/bsf4oorexx/files/Sandbox/rgf/taglibs/ga/ (Flatscher, 2021).

In the first line inside the script, the string on the right side of the equal sign is saved in the variable greeting. In the second line the print statement prints the content inside the brackets to the HTML file at the current position. This means everything inside the brackets is added to the HTML document at the position of the script. Thus, what happens is that once the user requests this HTML document the script will be executed on the server side and the content is added in the body of the file. After that the dynamically generated document is sent to the client.

Writing HTML code inside this print requires to put either single or double quotation marks around the code. However, inside those quotation marks a developer cannot access variables declared before. Therefore, the code needs to be split into the html code and JavaScript code to access variables. In the case of this thesis. this is done by using the *'+'* sign, which can be seen in listing 4. The code in this example uses the HTML tag * * (a container that can be filled with any content). Inside this container is the variable greetings, which is a normal string when the HTML page gets generated. Once a client requests the document, the file received by the client is looking like a

normal HTML document containing a short text and it is not noticeable that this page was preprocessed on the server. Figure 7 displays a Screenshot of the processed HTML page that the client receives. As one can see, the client is not able to tell that the document has been dynamically processed. It looks exactly as if this was a normal HTML file with static code which got sent to the Client. Furthermore, the code that was sent to the client is shown in figure 8. However, even there is no difference to a normal HTML document noticeable.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Hello, World</title>
</head>
<body>
<div>Hello, world! (Sent from Nashorn)</div>


</body>
</html>
```

Figure 8: helloworld.jsp - generated html file

Important Note: Using JavaScript on the client side, a developer can create a variable by using the keyword *let* before the name of the variable. However, for the development of JSPs with Nashorn *let* is not working.

3.2.2 Printing Dynamically Generated Content

Using JSP it is not possible to dynamically manipulate content on the client side of a web application. Instead, the HTML files need to be rendered on the server side and afterwards a static but dynamically and uniquely rendered HTML document gets sent to the client.

helloworld_ext.jsp

The file helloworld_ext.jsp showcases a small example of a web application which dynamically generates HTML code by printing the current date and time into the HTML document when a client requests the document. This is done by using the command *Date()* which by default creates an object date that uses the current time zone. This object is then saved as a string into a variable (Olawanle, 2022). However, as the goal only is to print the current time and date into the HTML document, it is not necessary to save the new object in a variable. Instead, it is possible to use the *expr* tag instead

of *script* as shown in listing 5. Due to that the current date and time is returned and fetched as a string by the expression (Flatscher, 2021). This string is then printed inside the <*p*> tag.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib uri="./WEB-INF/script-jsr223.tld" prefix="s" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Hello, world</title>
</head>
<body>
<header>
<s:script type="javascript">
var greeting = "Hello, world! (Sent from Nashorn)";
print('<h1>' + greeting + '</h1>');
</s:script>
</header>
<body>
<p>The time right now: <s:expr type="javascript">Date()</s:expr></p>
</body>
</html>
```

Listing 5: helloworld_ext.jsp

Looking at listing 5 one will notice another change in comparison to the helloworld.jsp file. The script which greets a client is not anymore in the body tag. Instead, it is placed inside the <*header*> tag. Very important to mention is that this tag is totally different from the head (which must be present in every HTML document and contains meta data of a document). The header element can be seen as a container for introductory content. For example, a header typically contains elements like headings, icons or logos (W3Schools, n.d.-f). The result of the code in listing 5 is presented in figure 9.

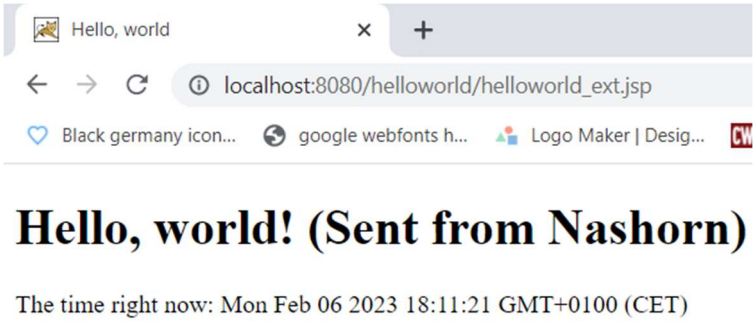

Figure 9: helloworld_ext.jsp in the web browser

In the following example applications, not every HTML element will be explained in such detail like in the two previous examples. The reason for that is that most of those

tags are self-explanatory by their tag name. However, if there is something unclear the website w3schools offers a list of available tags with a brief description available to every element. The link to the list of HTML elements is: https://www.w3schools.com/tags/default.asp.

3.3 Cookies

When a server sends data to a user's web browser, a small piece of information called an HTTP cookie gets sent too. The browser can store this cookie and send it back to the server with any subsequent requests. Essentially, an HTTP cookie is used to identify whether two requests are coming from the same browser. This is useful for keeping a user logged in, for example. Cookies are used to remember stateful information for the stateless HTTP protocol. Typically, cookies are used for three purposes, namely session management, personalization, and tracking. Session management is used for remembering logins, shopping carts, game scores and so on. Personalization is used for user preferences, themes, and other settings. Lastly, tracking is used for recording and analyzing user behavior (Mozilla Foundation, 2023b).

As one will notice cookies provide a lot of extremely useful opportunities to develop user-friendly web applications. By the examples in the following sections, it will be demonstrated how one can set cookies. Furthermore, it gets examined how they can be used to save data provided by a client and how cookies can be deleted. In the applications within this thesis cookies are used to store names of clients in order to individually greet them on the webpage. Moreover, they are used to store information about the time when the webpage was visited previously by them. However, an even more advanced usage of cookies is presented in the e-commerce example described in section 4.

3.3.1 Setting Cookies

Here the creation of cookies is explained in further detail.

lastvisit.jsp

To be able to set a new cookie the first step necessary is to request all Cookies from the client's browser. Those cookies are then saved into the variable *allCookies*. As

already explained in section 2.65.1 those XMLHttpRequest methods are possible due to adding the script-jsr223.tld tag library.

```
<s:script type="javascript" throwException="true">
//reqeuest and response procedures
//first do the request and response because response won't work if something was
//already printed

//request
var lastVisit;
var allCookies = request.getCookies();

//response to add the current time
var today = new Date();
var time = today.getHours() + ":" + today.getMinutes() + ":" + today.getSeconds();
timeString = String(time);
var Cookie = Java.type("jakarta.servlet.http.Cookie");
var newCookie = new Cookie("lastVisit", timeString);
newCookie.setMaxAge(60 * 60 * 24);
newCookie.setPath("/");
response.addCookie(newCookie);
</s:script>
```

Listing 6: lastvisit.jsp - request and response (cookies)

After requesting the cookies saved in the browser, the next step is to get the current time as a string. For this a new object of the type of date was created and saved in a variable called today. By default, the value of this date will be set to the current date and time (Olawanle, 2022). However, the format of this date is not what is required for this application. The current value of *today* is formatted like the date and time in figure 9. As only the hours, minutes and seconds are needed, some more coding is required. Fortunately, JavaScript provides an easy method to retrieve single values from a date object (Mozilla Foundation, 2022b). For example, to get value hour, the method *getHours()* is used. Furthermore, an equal method exists for minutes and seconds (Mozilla Foundation, 2022c). It is necessary to save these values as string in one variable. This is important to be able to add those values to the response in form of a cookie. Listing 6 displays a possible solution how this can be accomplished. First all values are saved into the variable called *time* and afterwards this variable is converted to a string. Now it is time to create the cookie and send it back to the user. Therefore, the first step is to access the Java class cookie. In Nashorn importing Java classes works a bit different then in Java. To access a class the method *Java.type()* can be used. Therefore, a developer needs to add the name of the class inside the brackets and then assign this to a new variable. In the *lastvisit.jsp* application the name chosen

for this variable is *Cookie*. Now it is possible to create new instances of this class. In the example, an object named *newCookie* was created. The variable that should hold the object needs to be written on the left side of the *equal sign*. On the right side the *new* operator is followed by the name of the class. The attributes that should be forwarded are needed in the brackets (Oracle, n.d.-d).

Now that the object *newCookie* is created, next the method *setMaxAge()* is used. This determines how long the cookie should be saved by the browser. In the case of this thesis, it was chosen that this cookie should be save for one day. The second method that is used on the new cookie is *setPath()*. The reason why the path is set to "/" is because this enables cookies being available application wide (BalusC, 2016). As everything necessary has been accomplished, the last thing this script does is to add the cookie to the response that gets sent back to the client. By looking at figure 10 one can see the cookie that has been sent to the web browser. It is now saved in the browser's storage.

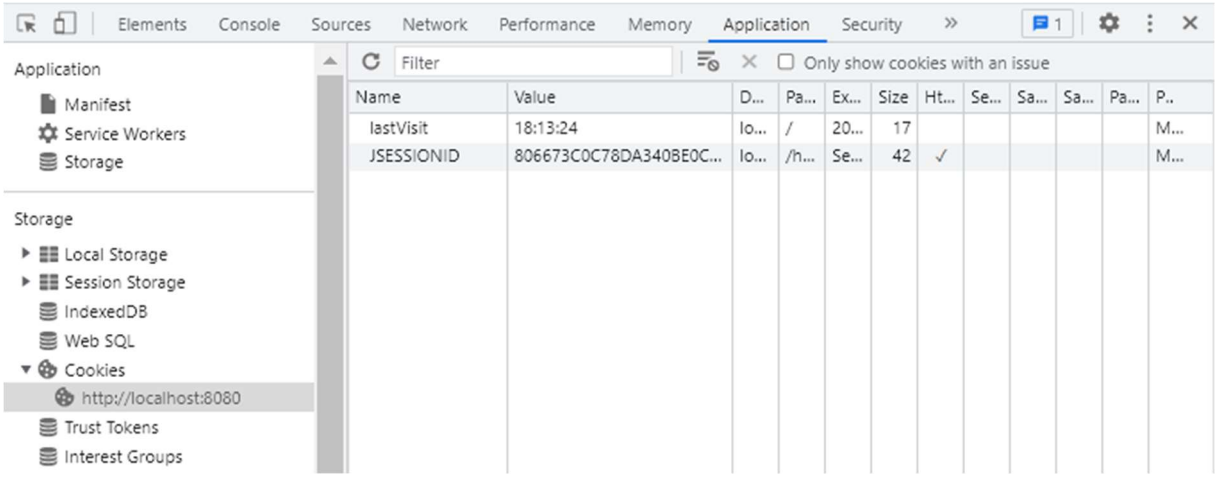

Figure 10: Cookies in web browser

What is still missing for this application is to communicate to the client when the last visit was. For the case that no cookie named *lastVisit* is stored in the browser, the text '*This is your first visit'* is displayed in the client's browser. To do so a second script is executed after the one presented in listing 6. This second script, which is shown in listing 7, first checks with an if statement, whether the variable *allCookies* is not equal to null. If this is not the case and consequently *allCookies* is empty the code inside this if statement is not executed and the execution directly jumps to the next if statement.

Yet, when the if statement is true a for loop gets executed which iterates over *allCookies*. For each iteration the cookie with the index of the current iteration gets saved into the variable *c* and another if statement gets executed. By this process it gets checked whether the name of the cookie currently saved in *c* equals *lastVisit*. If this condition becomes true, the value of the cookie which is currently saved in *c* gets written to the variable *lastVisit* and the for loop is stopped by the break statement. The break statement is not indispensable, though as the searched cookie has already been found it would be waste of computing power to execute the whole for loop.

The last task of the second script is to either print the time of the last visit or to print *'This is your first visit'*. To accomplish this, another if statement which checks whether the variable *lastVisit* is empty is needed. If this is the case, it means that there is no cookie with the name *lastVisit* saved in the browsers storage and thus it is the first visit of this user within the last 24 hours. Due to a *print()* statement this gets displayed in the clients web browser. However, when the if statement is false the code inside the else statement will be executed and the time of the last visit will be printed.

```
<s:script type="javascript">
//Print the last visit or if it is the first visit

if (allCookies != null) {
	for (var i = 0; i < allCookies.length; i++) {
		var c = allCookies[i];
		if (c.getName() == 'lastVisit') {
			lastVisit = c.getValue();
			break;
		}
	}
}

if (lastVisit == null) {
	print("This is your first visit!");
} else {
	print("Your last visit was at " + lastVisit);
}
</s:script>
```

Listing 7: lastvisit.jsp - printing time of last visit

Important Note: For whatever reason the execution of *print()* anywhere before a response in the script will prevent the response from being executed. To avert this issue from happening, all executions of the *print()* statement were placed in the later script. Therefore, when the first *print()* is executed, the response has already been sent to the client and no problems occurs. However, it is very unpractical and for the next

applications even impossible to always execute the response before a *print()* statement. Thankfully, within this thesis it has been discovered that using *out.print()* or *out.println()* instead of *print()* does not cause these problems.

3.3.2 Requesting User Input

An easy yet powerful method to develop interactive web applications is to combine user input with cookies. By doing so, it enables creating websites customized to a specific user. Within this section an example web application, which greets every user with their username, gets demonstrated.

greeting.jsp

The file greeting.jsp is normal JSP consisting of HTML and three JavaScript scripts inside the body. The whole visible content of this page is generated with JavaScript, which enables creating customized webpages for each client.

```
<!-- Request the Cookies and get the Value of the Cookie username -->
<s:script type="javascript">
var username;
var allCookies = request.getCookies();
if (allCookies != null) {
	for (var i = 0; i < allCookies.length; i++) {
		var c = allCookies[i];
		if (c.getName() == 'username') {
			username = c.getValue();
		}
	}
}
</s:script>
```

Listing 8: greeting.jsp - cookie request

The first script that is displayed in listing 8 is used to request the cookies stored in the client's browser. Furthermore, it is responsible for accessing the value stored in the cookie named username. A similar approach can be seen in the example from before where the last visit of the user is displayed.

```
<!-- Printing either the input field or the 'greeting' -->
<s:script type="javascript">
if (username == null) {
      out.println('<p>Hello what is your name?</p>');
      out.println('<form>' +
                        '<label for="username">Username:</label>' +
                        '<input type="text" name="username" required>' +
                        '<input type="submit" value="Ok">' +
                  '</form>');
} else {
out.println('<p>Welcome back, ' + username + '!</p>')
}
</s:script>
```

Listing 9: greeting.jsp - printing customized HTML code

The next script in this JSP is responsible for printing either the input field in which the user can enter a username or the greeting. As shown in listing 9 an if statement is used to accomplish this. First the if statement checks whether a username is equal to null. When this condition is true the client's browser has not stored a cookie named username. In this case a HTML form with two input fields inside is used. The *<form>* tag is required when a web application should facilitate users to enter data that is then sent to the web server for further processing and storage in a database (W3Schools, n.d.-d). In the case of this application two input fields are used. The first input field is of the type *text*. This means that empty text area is displayed in the user's browser where a username can be entered. Important here is to add the attribute *required* and a name to the input field. The attribute *required* prevents the form from being sent with an empty input field and the name is necessary to request the sent data on the server side. The second input has the type *submit*. Input fields of type *submit* are displayed as buttons. Figure 11 displays how the website looks in the browser of the user.

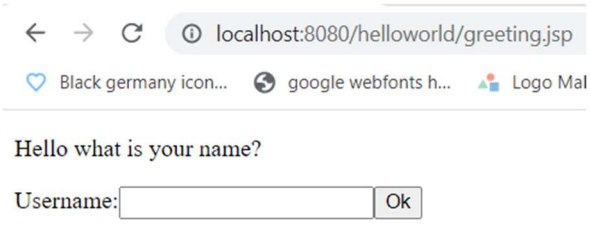

Figure 11: greeting.jsp - form in web browser

Once the user clicks the button the data is sent to the URL specified in the form with the attribute *action*. However, if no such attribute is defined the data is sent to current URL. This means the website reloads but stays at the same page. Furthermore, if not specified differently the GET method is used for sending the data (Mozilla Foundation,

2022a). Later in section 4.2.3 it is described what the GET method is and what other option exists. However, for the moment it is enough to understand that in this application the page sends the entered data to the current URL (http://localhost:8080/helloworld/greeting.jsp) and the page reloads itself.

When the condition in the if statement is false, it means that the variable *username* holds the value that has been priorly stored in the cookie *username*. This leads to the execution of the else statement and the personalized greeting is displayed in the client's browser. The way the website looks once the cookie username is saved in the browser is presented in figure 12.

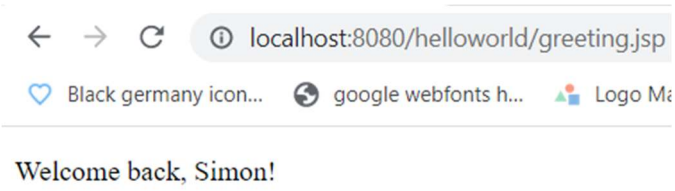

Figure 12: greeting.jsp - greeting in web browser

The third script is of interest after the user entered a value in the input field and submitted the form. The task of this script is to create and set the cookie. In order to access user input sent to an URL the *request.getParameter()* method is required. The name of the input field that should be requested needs to be written inside the brackets. Due to that method the value of this particular input field is returned. Every time the page is reloaded, the if statement checks whether the function *request.getParameter('username')* returns a value that is not equal to null. When this is the case the condition of this if statement is true and thus the code inside the brackets gets executed. However, this only happens when the user entered a value in the input field and submitted the form.

Now that the user filled in the form, the value entered on the client's side is saved into the variable called *uname*. Afterwards, an instance of the class *Cookie* is created, the value of the variable *uname* is saved in the cookie and the cookie gets sent to the client within the response. Though this last line the function *response.sendRedirect()* gets called. This function is able to redirect the user to another URL either inside or outside the server (yaminitalisetty, 2021). However, in the case of this example the attribute given to this function is another function, namely *request.getRequestURI()*. This function will return the URI of the current website and therefore the user is redirected to the

current page. The reason why this second reload is necessary will be explained in the next paragraph.

"Login" process explained

To better understand the logic behind the code explained in this section, the process executed once the user enters a name in the input field and clicks the submit button will be briefly explained. The first thing that happens once the user selects *Ok* is that the page reloads. Now the data entered by the user is available by using the function *request.getParameter()*. What happens is that the condition of the third if statement (displayed in listing 10) gets true and thus the code inside this statement gets executed. This means the cookie gets created and sent to the client. However, when the first two scripts were executed, there was still no cookie saved in user's web browser. Therefore, the first two scripts will be executed as if nothing had changed and the form to enter a username would be visible again. Now, to solve this problem another reload needs to be done and after that the page will look as intended.

```
<!-- creating cookie and adding it to the response -->
<s:script type="javascript">
if (request.getParameter('username') != null) {
	var uname = request.getParameter('username');
	var Cookie = Java.type("jakarta.servlet.http.Cookie");
	var newCookie = new Cookie("username", uname);
	newCookie.setMaxAge(60 * 60 * 24);
	newCookie.setPath("/");
	response.addCookie( newCookie );
	response.sendRedirect(request.getRequestURI())
}
</s:script>
```

Listing 10: greeting.jsp - username cookie

3.3.3 Deleting Cookies

The following section expands the previous greeting.jsp to demonstrate two further features. It gets determined how an external script can be accessed within a JSP. This presents a way to build the web applications in a more structured and clearer way, as the JavaScript code is then written in a separate file. However, the main topic of this section will be to demonstrate how cookies can be deleted. In this example the user input which is called username is used to greet a client in a customized manner. The process of deleting this cookie is thus called logout, as the username required is much

likely the usernames used in login. However, it is important to emphasize that this is just a demonstration of the usage of cookies and is far from a “normal” login.

greeting_ext.jsp

The JSP file greeting_ext.jsp is very similar to the file greeting.jsp that has been described in the previous section. However, an important difference can be found inside the else statement and is presented in listing 11. Similarly to the code in greeting.jsp this code is executed when the condition of the previous if statements is false. If this is the case, there must be a cookie named username stored in the browser of the client. Therefore, the username is used to print a customized *‘Welcome back’* message. This is still the same as in greeting.jsp. However, what is new is that now a button is printed which deletes the cookie *username* when it gets clicked. To accomplish this the button together with a second input field of type *hidden* is located inside a form. Figure 13 demonstrates how the website will look like when a cookie named username is stored on the client’s web browser.

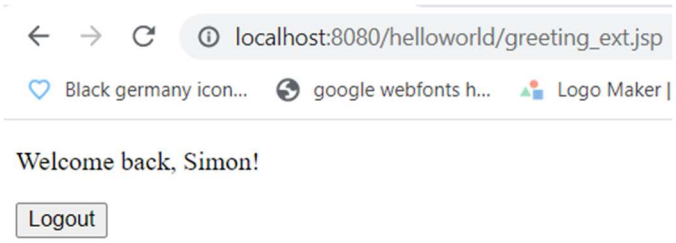

Figure 13: greeting_ext.jsp - greeting and logout button in web browser

Especially crucial here is to assign the hidden input a name and a value. This will be important when checking whether the user clicked the button whenever the page is reloaded. Like in the form where the cookie is created in listing 9, no attributes are assigned to the form in listing 11. This means that by default the data is again sent to the URL the form is present at by using the *GET* method. Now the web application always displays one of the two forms in the JSP depending on whether the username cookie is stored in a client’s browser or not.

```
else {
      out.println('<p>Welcome back, ' + username + '!</p>')
      out.println('<form>' +
                        '<input type="hidden" name="logoutButton" value="1">' +
                        '<input type="submit" value="Logout">' +
                  '</form>');
}
```

Listing 11: greeting_ext.jsp - else statement

The second difference compared to greeting.jsp can be found when the user clicks one of the form buttons. Once the page reloads it is necessary to check if the user wants the name to be saved or if the user is willing to logout and therefore the cookie needs to be deleted. The logic necessary to check whether one of this two cases is true could have been implemented directly inside a script in the JSP. Though instead it was chosen to implement it in an external JavaScript file and access it from the JSP. Listing 12 displays the code necessary to access the external script. In this case the script is stored inside a folder named code which is in the same web project as the JSP file. For the application to correctly operate it is crucial to provide the correct path for accessing the script. This path needs to be assigned to the *src* attribute.

```
<s:script type="javascript" src="code/logout.js" cacheSrc="false" />
```

Listing 12: greeting_ext.jsp - accessing external script

logout.js

The file logout.js is only responsible for two thigs, either to create and set a cookie or to remove a cookie. The first part checks whether a username has been sent to the URL and if so, creates a cookie and sends it back to the user. This is the exact same code as in the previous greeting.jsp example and can be seen in listing 10. The only difference is that here it is not written into the JSP but included from an external script.

However, what is new is the second part of this external script which is shown in listing 13. The first thing that happens is the checking of whether the condition of the if statement is true or false. In this case it is checked if an input field named *logoutButton* was sent to the current URL and is unequal to null. As already shown before the invisible input field inside the form has the name *logoutButton* and a value of *1* assigned to it. Therefore, if the user submits this form by selecting logout, the condition will be true and the code inside this if statement will be executed.

The code necessary to create a new cookie and set its path is the same as in previous examples. However, what changes here is that that the *maxAge* of the cookie is set to zero. Equally important is that the name of the cookie needs to be *username*. Once this cookie is sent to the browser it overwrites the one that is already stored in the client's browser. Furthermore, as the value of *maxAge* is set to zero the users web browser will delete the cookie immediately after overwriting the old one. The last step necessary is to reload the website by using the function sendRedirect() like in the examples above in order to display the HTML page correctly.

```
if (request.getParameter('logoutButton') != null) {
        var Cookie = Java.type('jakarta.servlet.http.Cookie');
        var removerCookie = new Cookie('username', '');
        removerCookie.setPath('/');
        removerCookie.setMaxAge(0);
        response.addCookie(removerCookie);
        response.sendRedirect(request.getRequestURI());
}
```

Listing 13: logout.jsp - remove cookie

4 E-Commerce Example

The following examples work together to build a simple web shop. Developers can use this as a starting point for their own online shop or as orientation of how the development of such a web shop could look like. Again the examples below are all inspired by and based on the applications developed by Lux (2021).

The first part will discuss the software that is required to build this web shop. Therefore, the installation and setup of the software components will be illustrated. After this the second section will discuss some of the most important security aspects which need to be kept in mind when developing a full functional web shop. Then in the section 4.3 all aspects regarding the database will be examined. Each section following 4.3 will discuss a feature of the web shop and explain the code to develop it in detail.

4.1 Required Software Components and Libraries

The purpose of this part is to introduce the additionally required software as well as explain the necessary installation and set-up steps. The first section is about the database system used for the online shop. The second part describes the hashing algorithm Bcrypt and the required steps to enable using the algorithm.

4.1.1 SQLite

SQLite is a free, open-source library that contains a relational database system. It is used in many different types of applications such as mobile phones, browsers and Skype. It is the most widely used and deployed database system in the world. SQLite supports most of the SQL language commands as defined in the SQL-92 standard. SQLite library can be directly integrated into applications, so there is no need for additional server software. This is the main difference from other database systems. By integrating the library the application is extended with database functions without relying on external software packages. SQLite has a simpler setup process compared to other databases that rely on a client-server architecture. It requires less configuration to be used in an application.

Furthermore, it has some unique features compared to other databases: the library is only a few hundred kilobytes in size. An SQLite database consists of a single file that contains all tables, indexes, views, triggers, etc. This simplifies the exchange between

different systems, even between systems with different byte orders. Each column can contain data of any type, and conversion is performed at runtime if necessary. However, there is no management of user and access rights at the database level. Instead the file access rights of the file system apply to the database files (Wikimedia Foundation, 2022c).

To sum up, the main reasons why SQLite was chosen is its simplicity and ease of use. It is lightweight and can be directly integrated into applications without the need for additional server software and it is open source and therefore free to use.

Installation and setup

The process of installing SQLite is very simple and straightforward. First, a new folder needs to be created. As it will be opened often via the path it is recommended to create the folder directly inside the *C:*\ directory. Thus, the path of the SQLite folder would be *C:**sqlite*. Once this is done, one needs to go to the download page of SQLite. The website can be found under the URL *https://www.sqlite.org/*. On this website a visitor needs to open the download page. As in the previous installation guides, this explanation is referred to using Windows as operating system.

On the download page the user needs to scroll down until the section with the headline *Precompiled Binaries for Windows* is reached. Here the user must download the command-line shell program to be able to work with SQLite directly on Windows. The program needed can be found by looking in the description next to the zip files (“How To Download & Install SQLite Tools,” n.d.). The description of the required zip file looks like the following: “A bundle of command-line tools for managing SQLite database files, including the command-line shell program, the sqldiff.exe program, and the sqlite3_analyzer.exe program” (*SQLite Download Page*, n.d., Precompiled Binaries for Windows Section).

Now that the zip file has been downloaded successfully the content of the file needs to be extracted to the folder *sqlite* that has been created before. If everything is done correctly the three programs shown in figure 14 should be visible inside the *sqlite* folder.

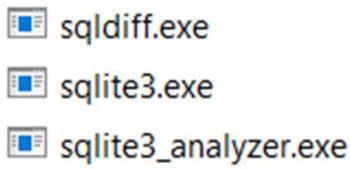

Figure 14: Sqlite program files

Finally, to test if SQLite is ready to use, the user can open the program in the console. To open the console the user needs to search for *cmd*. The first suggestion will be command prompt which needs to be selected. After that the command line window will be opened. Here the user can navigate to the sqlite program. To open a file or a folder the user needs to enter *cd* followed by the name. Via the command *cd ..* the console switches to the directory located immediately above the current. The command *dir* is used to display all folders and files inside the current directory.

In the case of this project, it is necessary to move to the directory above until the current directory is *C\:*. Then the next step is to use the command *cd sqlite*. In the *sqlite* folder the user can use the command *sqlite3* to start the SQLite. Now in the first line the version of SQLite is visible, which means that sqlite is started and it is ready to use. To see all available commands in sqlite the user can type *.help*. To quit the program the command *.quit* is used ("How To Download & Install SQLite Tools," n.d.). Figure 15 displays the commands required to start SQLite and the output in the console once SQLite is started. To create a new database the user needs to type *.open* followed by the name the database should have. If there is no such database inside the folder SQLite will automatically create a new one. For this project the chosen name for the database is shop.db.

```
C:\>cd sqlite

C:\sqlite>sqlite3
SQLite version 3.40.0 2022-11-16 12:10:08
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

Figure 15: starting sqlite

Connection with JSP

To connect the SQLite database with the JSPs some configurations need to be done. First it is necessary to download the SQLite JDBC Driver. JDBC stands for Java

Database Connectivity and is an API to connect a database and execute queries (*JDBC Tutorial | What Is Java Database Connectivity(JDBC) - Javatpoint*, n.d.). The .jar file of the SQLite JDBC Driver can be downloaded from https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc. After the download, the file needs to be added to the *lib* directory of the Tomcat server.

Furthermore a context.xml file needs to be added to the WEB-INF folder of the project. Inside this context.xml file the configuration seen in listing 14 needs to be done. The URL must point to the correct location of the database (Apache Software Foundation, n.d.-b, Sec. JDBC Data Sources). In most context.xml examples a username and a password will be set too. However, by default SQLite does not require authentication and thus username and password is not necessary (*SQLite: Documentation*, n.d.). When the provided fruitshop project is used creating and adding the context.xml file is not required as this is already done.

```
<Context>
 <Resource name="jdbc/sqlite" auth="Container"
        type="javax.sql.DataSource" driverClassName="org.sqlite.JDBC"
        url="jdbc:sqlite:C://sqlite/shop.db" maxTotal="100" maxIdle="10"
        maxWaitMillis="-1" removeAbandonedOnBorrow="true" removeAbandonedTimeout="60" />
</Context>
```

Listing 14: context.xml

As soon as the steps above are accomplished, a JSP can connect to a database by using the code displayed in listing 15. The first line of the code snippet imports the connection class from the java.sql.package. The second one is there to create the connection to the SQLite database. Using this code enables the JSP to read from and write to the database (*How to Connect to SQLite via JDBC*, n.d.).

```
// Import the Connection class from the java.sql package
var DriverManager = Java.type('java.sql.DriverManager');

// Create a connection to the SQLite database
var conn = DriverManager.getConnection("jdbc:sqlite:C://sqlite/shop.db");
```

Listing 15: Connecting to a database in JSP

4.1.2 Bcrypt

BCrypt is a way to secure user passwords by encrypting them using a technique called hashing. It was created by two computer scientists, Niels Provos and David Mazières

in 1999 and it is based on *Blowfish cipher*. It is designed to be resistant to attacks where someone tries to guess the password by trying many different options. Furthermore, it uses a random value called a *salt* to protect against a type of attack called rainbow table and it can adapt over time by making the encryption process slower, so it continues to be secure even as technology improves (Wikimedia Foundation, 2022d). The reason why such a hashing algorithm is crucial will be explained in section 4.2.

Installation and setup

The implementation of Bcrypt is very easy and newcomer friendly. To use Bcrypt there are several libraries out there ready to use. The one used in this example is the jBcrypt Java library provided by Damien Miller. To use this library the user simply needs to download the jar file from https://mvnrepository.com/artifact/org.mindrot/jbcrypt/0.4 and copy it into the lib directory of the Apache Tomcat server or into the lib folder inside the WEB-INF folder of the web application. In our example the jar file was copied into the lib directory of the Apache Tomcat server. Once those steps are accomplished, bcrypt can be used by the code shown in listing 15.

```
var bcrypt = Java.type("org.mindrot.jbcrypt.BCrypt");
```

Listing 16: "import" jbcrypt

4.2 Security Aspects

As web applications like online shops store sensitive user data, security is one of the most important aspects to think of. Yet the goal of this thesis is to demonstrate how a basic web application could look like and not to present ways to build highly secure web applications. However, this section will provide a small insight into web security and discuss some basic security measures. For a real-world web application which stores user data, a lot more needs to be thought of and an expert in this field is absolutely needed.

4.2.1 Storing Passwords Securely

To provide a login mechanism a web application needs to store user credentials in a table. The security strength of this depends on how the password is stored. Cleartext is the least secure way to store passwords. It is easy for an attacker to break into the database and steal the password table. This could give them access to each user

account. The problem is further compounded by the fact that many users re-use or use variations of a single password, which potentially allows the attacker to access other services too. A more secure way to store a password is to use hashing which changes the password into data that cannot be converted back to the original password (Arias, 2019, Sec. Storing Passwords is Risky and Complex)).

Hashing

A hash function is a mathematical process that takes any amount of data and changes it into a fixed-size string of bits (Arias, 2019, Sec. What's hashing about?). It is a great way to ensure that passwords are safe. One of the key properties of hashing is that it is irreversible, which means that the original password cannot be recovered from the hashed version. Additionally, hashing is deterministic, which means that the same input will always produce the same output. This is important for authentication because it allows to consistently verify user credentials. When an account is created, the password is transformed using a hashing algorithm and the username as well as the hashed password is saved in the database. When the user logs in, the provided password gets hashed and compared to the stored hash. If they match, the login is valid. The original password is never stored, only the hashed version. Further on a salt is a good measure to strengthen the hash. (Arias, 2019 Sec. Using Cryptographic Hashing for More Secure Password Storage).

Salting

A salt is a unique and randomly generated string that is added to each password as part of the hashing process. This makes it difficult for an attacker to crack large numbers of hashes at once, because they would have to use the respective salt for each hash, which would make the process a lot more time-consuming. Salting also protects against the use of pre-computed hashes as well as rainbow tables and makes it impossible to determine whether two users have the same password. In summary, salting makes cracking hashes more difficult and provides an extra layer of security for password storage (The Owasp Foundation, n.d., Sec. Password Storage Concepts).

4.2.2 Hypertext Transfer Protocol Secure (HTTPS)

HTTPS, also known as Hypertext Transfer Protocol Secure is an encrypted version of the standard HTTP protocol. It uses SSL or TLS to encrypt all communication between

a client and a server to provide protection against eavesdropping and tampering. It is highly recommended for all web applications that handle sensitive user data, as it allows clients to safely exchange sensitive data with a server. (Mozilla Foundation, 2023a). Transport Layer Security (TLS) is simply an updated version of Secure Socket Layer (SSL) that provides higher security. However, while nowadays most offered certificates are TLS, they are often referred to as SSL, as this term is more widely recognized. SSL and TLS are digital certificates that are used to establish a secure connection between a web server and a web browser. Such a certificate contains the public key and the identity of the website and it is issued by a trusted certificate authority (CA). To get a certificate, the website owner needs to apply for one from a trusted certificate authority (CA) and prove the ownership of the website (DigiCert, Inc, n.d.).

In actual web applications, the implementation of HTTPS through an SSL/TLS certificate is essential for maintaining security. However, as this is just an example project and not intended for public access, HTTPS has not been implemented.

4.2.3 Sending User Data (GET vs POST)

The GET and POST methods are both used for transferring data from a client to a server in the HTTP protocol. The POST method is used for sending additional data from the client to the server, which is placed in the message body. On the other hand, using the GET method all the data required by the server is included in the URL. However, while with GET the amount of data that can be sent is limited to 2048 characters, there are no such limitations with the POST method. Regarding security the GET method is less secure as the sent data is part of the URL (Educative, Inc, n.d.). The POST method sends data stored in the body of a HTTP request. Thus it is more secure as the parameters are not stored in browser history or web server logs (W3Schools, n.d.-i). For the following applications form data will be transmitted to the server using the POST method instead of the GET method.

4.3 SQLite Database Structure

In the example provided within this thesis a web shop selling fruits was developed. First the basic structure of the database had to be considered. A minimalistic entity-relationship model (ER model) was created to provide a clearer understanding of the database structure and the connections between tables in our web shop. This made

creating the tables simpler and less prone to errors. An ER model is a way to describe the relationships between different entities within a particular subject area. In the case of software engineering, ER models are often used to show the information required by a business in order to be able to perform its operations. Therefore, the ER model acts as an abstract data model, which outlines the structure of data that can be implemented in a database (Wikimedia Foundation, 2023). The erm presented in figure 18 was developed with the software draw.io. However, as this model was designed to identify the structure of the back end immediately, only the most essential information for the development is contained in it. For the creation and management of the tables in this database the Windows command line was used. The necessary SQLite commands to create the tables can be found in the Appendix of this work.

Figure 16: Entity Realtionship Model of the database tables

4.4 Reading Data from the Server

The first application developed in this e-commerce example is a simple list to display the products offered on the website. The goal of this is mainly to showcase reading data stored on a server and displaying it on a webpage.

productlist.jsp

The first new element in this JSP file is the *<link>* tag inside the head. This one is used to access an external CSS file. CSS files are used to style the elements inside an HTML file. The file is located inside a folder named *css*. Listing 17 displays how external CSS files can be added to a JSP file.

```
<link rel="stylesheet" href="css/style.css">
```

Listing 17: productlist.jsp - adding external CSS file

The other new features of this file can be found inside the <*script*> tag of the file. Here the first line of code imports the *DriverManager* class needed connect with the database. This class enables the use of useful methods like *getConnection().* The *getConnection()* method is used to establish a connection with specified URL (*Java DriverManager - Javatpoint*, n.d.). As it is shown in listing 18 this connection is saved in the variable *conn*. Now that the connection is established, the data needs to be read from the database. This is usually done with an SQL SELECT statement. The statement shown in listing 18 causes that all fields of the table *fruit* are returned. This is caused by the attribute **.* To restrict the output the statement can look somewhat like this: *SELECT price FROM fruit* (W3Schools, n.d.-j).

To execute the SQL statement, the variable call needs to call the method *createStatement()*. This method creates a statement object, which enables sending SQL statements to the connected database (Oracle, n.d.-a, Sec. Method Detail). In this example the object is saved in the variable *stmt*. From now on this variable can be used to execute a query. For the execution of the query, the method executeQuery() is used. When the script is executed, the executeQuery() method returns a single ResultSet object (Oracle, n.d.-c, Sec. Executing Queries). This object contains the data retrieved through the query. In the case of this example, the retrieved data is saved in the variable rs.

As the data is now accessible via the variable *rs* the next step is to iterate through the data and print it to the HTML page. The data of a ResultSet can be accessed through a cursor. The cursor, which is a pointer, is used to navigate through the rows of data. Initially, the cursor is positioned before the first row. You can move the cursor to different rows by calling various methods, such as *next()*. In the example of the *productlist.jsp* file, the data in the row where the cursor is currently positioned is outputted, each time *next()* is called (Oracle, n.d.-c, Sec. Processing ResultSet Objects). The intended look of the product list when it is opened in a web browser can be seen in figure 17.

```
// Import the DriverManager class from the java.sql package
var DriverManager = Java.type('java.sql.DriverManager');

// Create a connection to the SQLite database
var conn = DriverManager.getConnection("jdbc:sqlite:C://sqlite/shop.db");

// Execute a SELECT query to retrieve all data from table fruit
var stmt = conn.createStatement();
var rs = stmt.executeQuery('SELECT * FROM fruit;');

out.println('<ul>');
while (rs.next()) { //iterate through all rows in the table
      out.println('<li>' + rs.getString('name') + ': ' + rs.getString('price') +
'€</li>');
};
out.println('</ul>')
```

Listing 18: productlist.jsp - reading data from database

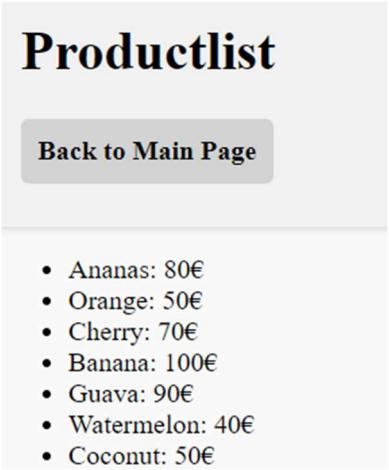

Figure 17: productlist.jsp - opened in web browser

4.5 Creating an Online Shop Main Page

A fully functional online shop requires a main page which acts as a harbor of the web application. By default, a user who visits the website will be directly headed to the index file of the application. From this stage the user needs to be able to see all important features and capabilities at one glance. Therefore, the website needs to offer a section where the important links are placed. In this case there are links directing to the sign-up page, to the login page and to the shopping cart of the online shop. For logged in users the possibility to log out needs to be available too. Moreover, it is important to present all available products to the user and offer the functionality to add them to the shopping cart. However, before the following sections will explain the development of this JSP in more detail, some preconditions need to be declared.

4.5.1 Prerequisites

This section declares which conditions need to be set in order to make the main page work properly.

Cookies vs Sessions

The functionality to add products to the shopping cart is offered to all users independent of their signed in status. Therefore, even users who are not logged in can add products to the shopping cart. However, it is a bad user experience if the page does not remember the products in the shopping cart. For example, the worst case scenario would be that a user wants to log in and after loading login.jsp file all progress of adding the required products is lost.

To prevent such experiences from happening, a method to store information without having user credentials is required. An example of such a method could be cookies, which were already introduced in section 3.3 on page 21. However, there is a more advanced method named sessions, which deliver some useful advantages compared to cookies. In principle, sessions also make use of cookie technology. However, the difference is that instead of storing the data in the browser on the client side, the data is stored on the server which brings some benefits along. First sessions in comparison to cookies can store a lot more data. cookies only possess the capability to store 4 KB of data, while sessions have a capacity of 128 MB. Furthermore, as cookies are stored on the server, sessions also provide a higher level of security (gittysatyam, 2021).

According to Lux (2021) Tomcat provides an easy way for the implementation of sessions. To create one the session attribute in the page directive (on top of every JSP file) needs to be set true. The HttpSession interface creates a unique session ID for each user and stores it in a cookie called JSESSIONID. This cookie is sent every time the page is requested. It allows to store and retrieve objects related to the user's session. If cookies are disabled, the URL is modified to include the session ID instead (Lux, 2021). However, a JSP in an Apache Tomcat server sets the session attribute to true by default. Therefore, using sessions does not require any configuration (no.good.at.coding, 2011).

Yet the default configuration of the session causes that the session will be deleted after 30 minutes. To prevent this, it is necessary to change this and increase the time to 1

day. To do so, <*session-config*> inside the web.xml file needs to be changed. Once a user found this tag, the value inside <*session-timeout*> needs to be changed to *24*60*. Listing 19 displays how the <*session-config*> needs to look like (Mihn, 2019).

Important Note: If Eclipse is used as an IDE, it is crucial to perform the change in the web.xml file located in the eclipse-workspace. Otherwise, the change will be lost once the server is started (Mihn, 2019).

```
<sesison-config>
  <session-timeout>24*60</session-timeout>
</sesison-config>
```

Listing 19: web.xml - session-timeout configuration

Product Images

The last thing to think of before developing the JSP files are the pictures used in the online shop. All images presented on the shop were distributed under the CC0 license, are free to use and do not require any attribution.

In the advanced examples (section 5) an application by which new products can be added will be presented. Thus, this will require the upload of new images. If all images would be stored in different locations it would be problematic to easily access all images without configuring a path for each picture by hand. Therefore, a new folder is needed in which all images are saved. In this example, the folder is named files and located in the Apache Tomcat folder. Figure 18 shows how the folder needs to look like.

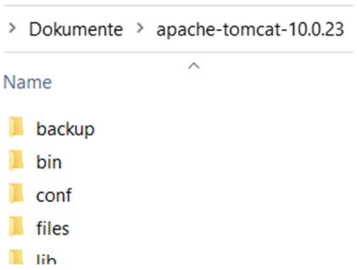

Figure 18: Apache Tomcat folder

To enable web applications to easily access the files folder additional configuration steps are required. For this a user needs to open the server.xml file. Once again if Eclipse is used the file needs to be changed inside the *eclipse-workspace*. In the bottom of this document the <*host*> element is located. This element needs to be

extended by the following line: *<Context docBase="C:\location-of-tomcat-server\apache-tomcat-10.0.23\files" path="/files" />* (Lux, 2021). The path inside must be changed to point to the correct location of the files folder. Figure 19 displays how the server.xml file looks like after this configuration step.

```
        <Context docBase="C:\Users\Simon\Documents\apache-tomcat-10.0.23\files" path="/files" />
      </Host>
    </Engine>
  </Service>
</Server>
```

Figure 19: server.xml - added context tag

After this configuration all web applications on the server can access the images inside this folder via the path */files*. This is equivalent to folders which are present inside a project (e.g., the *css* or *code* folder).

4.5.2 Main Page

The files required to build the main page are explained in the following paragraphs.

index.jsp

A user opening the web application will be directed to this page by default. The JSP file itself is nothing special. In principle it is a normal web document. However, there is no visible content present in the file. Instead, two external scripts which are responsible for the dynamic creation of the content are accessed. The first script, namely *userheader.js* is accessed inside the header of the document The second one, mainpage.js is accessed inside the body.

userheader.js

The task of this script is to correctly display the links to other pages like the login page or the shopping cart. It can be seen as the navigation area of the web application but it needs to present differing links depending on the circumstance. The first context that needs to be considered is which page is currently displayed. For example, if the user is currently in the login page, a link directing to the login page would be redundant. Though this is not necessary to be considered in here because *userheader.js* is only accessed on the main page (index.jsp). However, what needs to be managed by the script it to check if a user is logged in or not. Logged in users should see a button to log out, while others should see the login and sign-up button.

To accomplish this task, first it needs to be checked if a user is logged in. To do so, the developed web application utilizes sessions. Therefor, the customer_id of a user who has been logged in successfully is stored in the session of a web browser. This information is stored until a user selects to log out or 24 hours pass. However, the functionality of the login and sign-up process will be explained in section 4.6.

Now, the first step of the *userheader.js file* is to request the session. This is done via the method *request.getSession()*. The output of this method needs to be stored in a variable. After that calling the function *.getAttribute()* by this variable will either return the id of the logged in customer or null. This can be exploited to check if a user is logged in or not and depending on that display different links in the navigation area. Listing 20 displays the two if statements to check whether the user is logged in or not.

```
//// HEADER FOR A USER NOT LOGGED IN ////
if (session.getAttribute('logged') == null) {
      ...
}
//// HEADER FOR A LOGGED IN USER ///
if (session.getAttribute('logged') != null) {
      ...
}
```

Listing 20: userheader.js - checking login status

For users who are not logged in, the session is used to retrieve information about the number of products in the shopping cart. In case that the user is logged in, the database is used instead.

mainpage.js

The code inside this JavaScript file is responsible for the presentation of the offered products. Furthermore, it enables the customer to select a quantity and add the item to the shopping cart. First the script displays the products in a grid. The function to do so is like the one used to display a product list in section 4.2. Though the difference is that here not only the product names and prices are requested. The function that generates the HTML code to display a product is called in a loop to display all products available. Listing 21 shows this function. All the attributes forwarded to this function are retrieved from the database by using an SQL SELECT statement. Crucial is that the parameter picture needs to follow this pattern: /files/pictureName.jpg. However, it

is not the task of the file mainpage.js to provide the path in the correct way. This needs to be considered in section 5.1 and when the products are inserted in SQLite by hand.

```
//// HTML TEMPLATE ////
function templateProductItem(name, price, weight, picture, fruit_id) {
      out.println('<div class="product-container">' +
                  '<h2>' + name + '</h2>' +
                  '<img src="' + picture + '" height="300px" width="300px" />' +
                  '<p>Price: ' + price + ' Euro</p>' +
                  '<p>Weight: ' + weight + 'Kg</p>' +
                  '<form name="selection" method="post">' +
                       '<input type="hidden" name="selection" value="' +
                       fruit_id + '">' +
                       '<select name="quantity">' +
                             '<option value="1">1</option>' +
                             '<option value="2">2</option>' +
                             '<option value="3">3</option>' +
                             '<option value="4">4</option>' +
                             '<option value="5">5</option>' +
                             '</select>' +
                       '<input type="submit" style="cursor: pointer;" ' +
                      'value="Buy">' +
                '</form>' +
               '</div>');
}
```

Listing 21: mainpage.js - HTML template of product container

The second task of this JavaScript file is to adjust the shopping cart every time a user adds a new item. For that purpose, like in *userheader.js* the login status needs to be checked. If a user is logged in the update of the shopping cart needs to take place in the database. Else the session attribute *shopping_cart* needs to be created or updated if it is already existing. Figure 20 presents a screenshot of how the index.jsp of the online shop looks like.

Important Note: It is recommended to explicitly close the *Connection,* the *Statement* or *PreparedStatement* and the *ResultSet* when they are no longer needed (Oracle, n.d.-b).

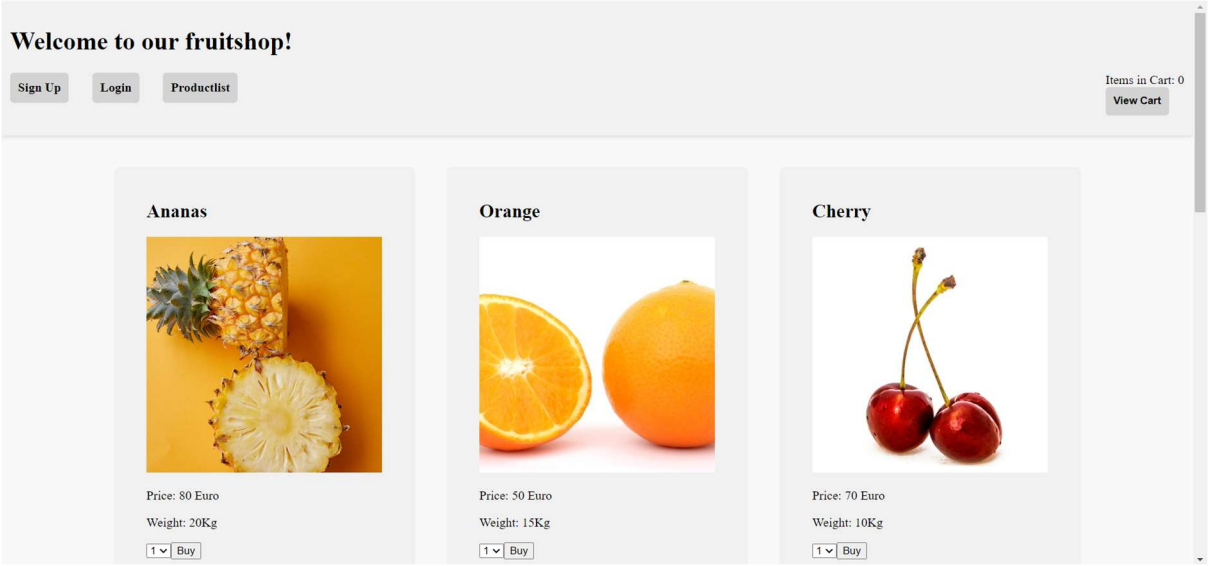

Figure 20: index.jsp - opened in web browser

4.6 Establishing Secure Registration and Login

For providing a realistic online shop experience it is essential to provide users the possibility to log in to our webpage with a registered account or to create a new account if the user does not possess one. In this regard storing the password in a secure and responsible manner is a key aspect in providing such services. Therefore, it is necessary to encrypt the passwords of registered users. In this example, the hashing algorithm of choice is the Bcypt algorithm. All requirements to use this algorithm are already discussed in section 4.1.2 on page 35.

4.6.1 Sign-Up

The following paragraphs declare how the registration of new customers works.

Particularity database value insertion

For this thesis the creation of the user accounts was done via the developed sign-up application. It is also possible to insert the rows in the table customer by hand in SQLite. However, if a salt is used the salt needs to be stored next to the password hash in the password file. Thankfully, using the Bcrypt algorithm this happens automatically. (Selzer, 2020, Sec. Salting a password). Furthermore, Bcrypt stores the number of rounds used to produce the hash too. The number of rounds is often referred to as the work factor.

Though even if the rounds as well as the salt need to be stored to check an entered password against a password in the database, it is possible to externally produce a hash with Bcrypt and insert it. The reason for that is that Bcrypt stores the salt and the work factor as well as the hashed passwords in the stored hash (erickson, 2011). Figure 21 displays how those hashes look like. Each row in this figure represents a different password but also holds the information about the used salt and the number of rounds used to produce it. After the first *$* the value *2a* defines the version of the used Bcrypt algorithm. Following the next *$* the value *12* determines the work factor used to generate the hash. The value after the third *$* represents the salt and the hashed password. The first 22 characters of this string represent the salt and the remaining characters represent the hashed password. This cipher text is the one that needs to be compared during the verification process (erickson, 2011).

```
sqlite> select password from customer;
$2a$12$CpYF.x.XRMGRLs28dYBEoua9XUzVHbLrsSfyi1Ags0yYwwpIJPvmG
$2a$12$PeogW1ip6FW25AOtT86/fOjGy4umMvpEdU81yz94mR/xDYjcZYD0C
$2a$12$dUZMUU1c6CNbTitfGqC.VeePgeoYB7yQ69ueScoNLjxt.nKRKiMh.
```

Figure 21: shop.db - hashed passwords

Therefore, it is not a problem if a different work factor was used to generate the password. The function to check if the passwords are identical automatically knows what work factor and what salt were used when the stored hash was generated. Thus, it is possible to either use an online hash generator or to write a script which does that locally. From a security perspective it will be better to not use an online hash generator. Therefore, listing 22 presents a script that can be used to generate a hash. This hash can then be inserted into the *shop.db*. Bcrypt uses a salt that is fixed in length and cannot be changed. The length of the salt used is 128 bits (Zhang, 2022). However, what is changeable is the work factor. By default, the work factor used is 10 but I in the script it has been increased to 12 as this will generate an even stronger hash (Miller, n.d.). The passwords chosen for the example customers were created with the following pattern: *username:* user@mail.com *password:* user.

```
var bcrypt = Java.type("org.mindrot.jbcrypt.BCrypt");
var plaintextPassword = 'example';
// Generate a new salt, cost of the later hashpw will be 12
var salt = bcrypt.gensalt(12);

// Hash the password with the salt
var hashedPassword = bcrypt.hashpw(plaintextPassword, salt);

out.println('The password hash is:' + hashedPassword)
```

Listing 22: Generating a password hash

signup.jsp

The page signup.jsp provides a form which is needed to get the necessary data from unregistered visitors of the web application, who are willing to create an account. The user needs to enter a username, a password and an identical repetition of the password. The username must be an email address. Entering this password twice has the purpose to reduce the risk of a typo by the customer. The filling of those three input fields is necessary to submit the form. To assure that the fields are entered correctly the attribute required is added to all three input fields and the username input is assigned to the type of email. Thus, none of these three input fields can be left out and the username input requires the scheme of a valid email address to proceed. However, the form also provides an additional input field which is displayed as a checkbox. This one is not required, and its purpose is to determine if a customer is willing to receive advertisement emails. Once the form gets submitted the data provided by the customer gets sent to the server via the POST method and is handled by the file create_user.js.

create_user.js

The first step to initiate the insertion of the provided user data into the database is to request the parameters *username*, *pwd1* and *pwd2*. Once this is done it is necessary to check if the entered passwords match each other. The code developed to do that can be seen in listing 23. In case the entered passwords are not identical, a warning is printed to the user that it did not work and retrying is necessary. If the passwords are the same nothing happens and the execution of the code is continued in the normal manner.

```
if (pwd1 != pwd2) {
      out.println('<p class="warning">Your entered Passwords do not match, please
try again!</p>');
}

if (username !== null && pwd1 !== null && pwd1 === pwd2) {
    insertIntoDB();
}
```

Listing 23: create_user.js compliance checks

The next step is to check if all input values are unequal to an empty string and if the two entered passwords match. As it is checked if the two entered passwords are identical it is not necessary to check both passwords against an empty string. As seen in listing 23 if the statement is true the function insertIntoDB() gets executed.

Everything from now on is inside the function insertIntoDB() and thus only executed if the input values are valid. The reason why the whole code is inside a function is that it offers the possibility to exit the code in a clean and easy way by using the return statement. The first task of the function is to establish a connection with the database shop. After that a *PreparedStatement* is needed to check if the user already exists. If this is the case and the username is already stored in the database, the return statement is used to stop the processing of the function after a warning is printed to the visitor of the website.

If the username is not in the database, the execution of the script can go on and the password hash is generated. After that the username and the hashed password are added to the database using the SQL statement *INSERT*. Moreover, it is checked if the customer is willing to receive emails. If this is the case the value in the column *receives_mail* is updated to *1*. By default, the value of this is *0* which means that the customer does not want to get additional emails. In this case *0* and *1* are used as boolean values instead of *true* and *false* because SQLite does not have a dedicated boolean type (Tandetnik, n.d.).

The reason why *PreparerdStatements* instead of *Statements* are used in this script is that it is more readable and especially more secure against SQL injections. SQL injections are especially dangerous when DML (Data Manipulation Language) queries are used (baeldung, 2022). SQL commands that are used to manipulate the data in a database are part of the DML. This includes the SQL statement INSERT, which will be

used to insert new data into the table customer later in the script (GeeksforGeeks, 2017).

Listing 24 demonstrates the code which is responsible for the insertion of the new customer. Furthermore, it displays the code that updates the boolean value in the column *receives_mail* for users who are willing to receive further emails.

```
prepstmt = conn.prepareStatement("INSERT INTO customer (username, password) "
"VALUES (?,?)");
prepstmt.setString(1, username);
prepstmt.setString(2, hashedPassword);
prepstmt.executeUpdate();  // add new user to database
prepstmt.close();

if (request.getParameter("newsletter") == 1) {
    prepstmt = conn.prepareStatement("UPDATE customer SET receives_mail=1 "
    "WHERE username=?");
    prepstmt.setString(1, username);
    prepstmt.executeUpdate();  // sign the user up for e-mails
    prepstmt.close();
}
conn.close();
```

Listing 24: create_user.js - adding new customer to database

4.6.2 Login

While the login.jsp file only is responsible for providing the form, where a user can enter the credentials to log in, the file login.js contains the logic behind the login process.

login.js

The first part of the script is responsible for the basic prerequisites. Therefore, the session is requested, the parameters sent by the form are written into variables and the connection is established. After this part which is nearly the same in almost all applications developed for this e-commerce example, the next step is to check if the entered credentials match with one from the database. The code that accomplishes this task is shown in listing 25. If the entered username exists and the cipher texts of the entered and the stored password are matching, a session attribute with the name *logged* is set. The *customer_id* of the user gets set as the value of this attribute. After that all pages in this project can identify the current user.

```
//check if password is correct
if (bcrypt.checkpw(pwd, hashedPassword)) {
	//store the login status is the session
	session.setAttribute('logged', id)
```

Listing 25: login.js - verification process

Once the session attribute is set, the last main task of this script is to transfer the shopping cart stored in the session to the shopping cart stored on the database. To do so, a for loop is used to iterate through the key-value pairs saved in the object *shopping_cart*. Each key in there is a different *fruit_id* and thus refers to a different item. By each iteration the variable *selection* will possess the value of the current key (*fruit_id*). By using this key, the value of the fruit can be accessed. This accessed value is the quantity, or precisely how often this item is in the cart. This logic is showed in listing 26.

```
//iterate through the shopping_cart that was stored in the session
for (var selection in shopping_cart) {
	quantity = shopping_cart[selection];
	quantity = parseInt(quantity);
	item = selection;
```

Listing 26: login.js - iteration through shopping cart stored in the session

Next step is to check if the fruit is already stored in a customer's database *shopping_cart*. If so, the quantity that is already stored in the database and the quantity of the session's shopping cart need to be aggregated. After this is done. The values are inserted into the database table *shopping_cart*. This whole process is repeated until all items in the sessions shopping cart are transferred to the database. Once all items are added to the database the session attribute shopping_cart needs to be removed. However, the code regarding the transfer of the shopping cart only gets executed if the session shopping cart is not empty.

Implementing this transfer of the items is extremely important for providing a satisfying user experience. For example, it might be that a user first adds products to the shopping cart and afterwards wants to log in to be able to check out. However, if the transfer

would not be implemented, the user would need to add the items again after logging in. Such a situation would be extremely annoying for any customer (Lux, 2021).

Important Note: The values stored inside a session attribute are stored as strings. Thus it is necessary to call the method *parseInt()* to transform the string into a number. After that it is possible to do mathematical operations with the values

4.6.3 Logout

Important to consider when developing a login mechanism is also to provide the option to log out. This is especially important for customers who are using a computer which is accessed by other people too.

logout.jsp

When a customer clicks the logout button, for example on the main page, the user is forwarded to the file logout.jsp. The purpose of this file solely is to invalidate the current session. Therefore, the session first is requested and afterwards deleted by calling the method *invalidate()*. Executing this method removes all objects that are bound to the session (Oracle, n.d.-c).

4.7 Creation of a Shopping Cart

The last application visible for a customer that is still not discussed is the implementation of the shopping cart. By clicking the link to the cart on the main page the user gets forwarded to the page shopping_cart.js. However, this file does not include any visible content apart from the header. This is indeed obvious, as the whole body needs to be dynamically generated, depending on the products that got added to the shopping cart. Therefore, the only element in the body is a <*script*> tag linking to an external JavaScript file. This file has the name shopping_cart.js and more lines of code than any other file in this online shop.

4.7.1 shopping_cart.js

The code in this script has the purpose to display the products that are already in the shopping cart in the correct way. This is independent from the login status of the customer. However, what depends on the login status is the ability to check out. Only logged in users can click on check out to order the products. Furthermore, the website

also provides the users with the facility to adapt the number of items in the shopping cart. Therefore, they can increase and decrease the quantity of an item or even completely remove it from the shopping cart with a single click. In figure 22 the look of the cart once some products are entered can be seen.

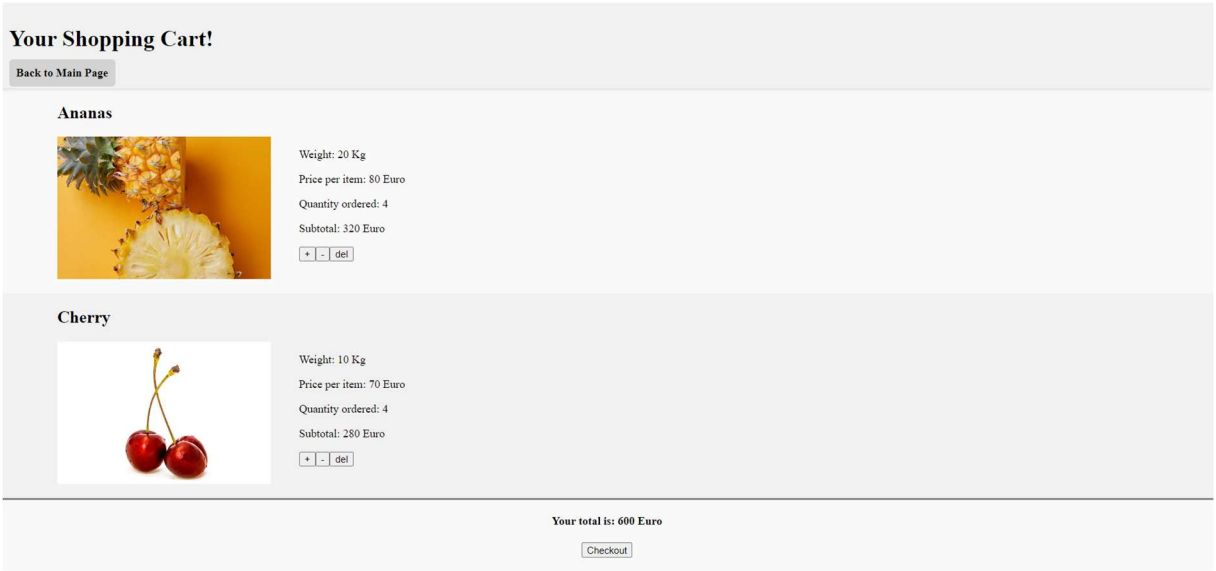

Figure 22: shopping_cart.jsp - opened in the browser

The information which products are in the shopping cart is stored in a different location depending on the login status of the customer. For logged in users the information is in the database table *shopping_cart*. For users who are currently not logged in this information is stored in the session. Therefore, the process of retrieving this information and displaying the products is different. The same is true for updating the quantity of a product. For that reason, the main functionality of the script is present twice but in different versions. One version is for users who are logged in and one for users who are not.

Displaying the products

For users who are not logged in the condition of the first if statement returns *true* and the code inside the brackets will be executed. Here the databases only purpose is to get the information and pictures of the products to display them. Though, the process to get the information which products are in the cart and with which quantity, is retrieved from the session. The mechanism to retrieve this information is the same as when transferring the session's shopping cart to the database's cart which is displayed in listing 26.

In case the user is logged in the code inside the second if statement (in line 63) gets executed. In contrast to the description above here the database plays a more important role. However, sessions are not used in this case. First the products that are added in the shopping cart get read from the table *shopping_cart*.

After the fruit_ids as well as the quantity of the added products are known the process is the same for both logged in and not logged in users. Now it is possible to select the products that are in the shopping_cart by filtering the SELECT statement with the *fruit_ids*. Those products are then displayed in the client's browser.

Adjusting the shopping cart

The logic behind the process in the if statement for logged in users and not logged in users here is almost the same. The main difference is where the change will be stored. For users that are logged in the change will be stored in the database using SQL statements. However, for the other visitors of the shop the update will be stored in the current session.

What is important here is that a special case needs to be prevented. A user might select the decrease by one button even if the quantity of this item is one. If this case is not considered in the development process the quantity would be zero afterwards. Even more problematic the quantity could go into negative numbers too. To prevent this from happening, an if statement needs to be implemented to check if the value is equal or smaller zero. When this is *true*, the same code as for deleting a product needs to be executed. Listing 27 displays this if statement for users who are not logged in.

```
if (quantity <= 0) {
    delete shopping_cart[id]; // delete product from cart if quantity goes below 1
}
```

Listing 27: shopping_cart.js - check if quantity is equal or smaller zero

Alternating background colors

The background color of the product containers is alternating between grey and white. The first one is white, the second one grey, the third one white, and so forth. To implement this the first step is to assign the number 0 to a variable (in his case named i) before the while loop that is generating the HTML code to display the products. Inside this while loop *i* is forwarded to the function *getBackgroundColor()*.

In this function the modulo operator is used to divide *i* by 2. If the result of this is 0 it means that *i* is an even number and a white color is returned. However, if the result is not 0 *i* must be an uneven number and a grey color is returned. The function getBackgroundColor() is shown in listing 28.

```
function getBackgroundColor(i) {
	if ((i % 2) == 0) {
		var backgroundColor = '#F9F9F9';
	} else {
		var backgroundColor ='#F1F1F1';
	}
	return backgroundColor;
}
```

Listing 28: shopping_cart.js - getBackgroundColor()

Inside the while loop the returned color is then saved into the variable *bgColor*. This variable is afterwards forwarded to the function *templateProductContainer* which is responsible for generating the HTML code to display the product. Now the color saved in *bgColor* is used as the background color of the created product container.

The last thing that needs to be done is to increase the variable *i* before the end of the while loop. Therefore, *i* will alternate between even and uneven numbers for each iteration and the background colors will also alternate. Listing 29 displays the while loop and the declaration of the variable *i*.

```
var i = 0;
out.println('<div class="shopping-cart-ctn">');
while (rs.next()) {
    totalprice = totalprice + (rs.getInt("price") * rs.getInt("quantity"));
    var bgColor = getBackgroundColor(i);
    templateProductContainer(rs.getString("name"), rs.getString("picture"), +
    rs.getString("weight"), rs.getInt("price"), rs.getInt("quantity"), +
    rs.getString("fruit_id"), bgColor);
    i++;
}
```

Listing 29: shopping_cart.js - while loop to generate the product containers

4.7.2 checkout.jsp

Users who are logged in and have at least one item in the shopping cart can order by clicking the checkout button in the bottom. However, as this is an example online shop, the checkout only deletes the products that are currently in the cart.

The first step of the JSP is to check whether the user is logged in or not. Therefor, the session is requested. Then it is checked inside the condition of an if statement if the logged attribute of the session is *null*. When this is the case, a warning is printed saying that the customer needs to log in first. Else the SQL statement *DELETE* is used to delete all rows in the table shopping cart where the customer_id equals the id from the session attribute. The else statement which has the purpose to clear the shopping cart of this customer is presented in listing 30.

```
} else {
	var qry = "DELETE FROM shopping_cart WHERE customer_id = ?;"
	var prepstmt = conn.prepareStatement(qry);
	prepstmt.setInt(1,session.getAttribute("logged"));
	prepstmt.executeUpdate();
	prepstmt.close();
	conn.close();
```

Listing 30: checkout.jsp - clearing shopping cart

5 Advanced Examples

For clarification it is necessary to once again emphasize that the applications developed by Lux (2021) were used as a model for the development of this program.

The examples discussed in this section offer a new approach to develop web applications. Until now it was either the case that the whole code was present in a single JSP file or that the JavaScript part of it was accessed from an external file. While this offers a good way to get into the development of such web applications, it also inherits some disadvantages. For example, the use of numerous if statements can make the program extremely complex and duplicated code execution might be the result. A more efficient approach would be to separate the handling of specific requests from the creation of content, which would reduce the amount of unnecessary processing and loading (Lux, 2021).

The applications located in the */admin* offer an alternative approach to develop web applications. However, because of this new folder the files inside it need to access external files in an adapted way. Listing 31 displays how the style.css file needs to be accessed by the programs inside the admin folder (Lux, 2021).

```
<link rel="stylesheet" href="../css/style.css">
```

Listing 31: Accessing style.css inside the admin folder

5.1 Developing a File Upload

To simplify the process of adding a new product to the online shop, a JSP that has the purpose to create new products was developed. To offer an ability for the user to create the product a form is needed. In this form the name, price and weight of the product need to be entered. While this is almost the same process as creating a new user, the hurdles to achieve this are low. The only difference from creating a user is that an image of the product is needed. Therefore, it was necessary to develop a program which uploads the chosen file from the local computer to the */files* folder located on the Apache Tomcat server.

The technology used for this program is called Jakarta Servlet (formerly known as Java Servlet). A Jakarta Servlet is a Java class. The instances of it receive and respond to requests from clients within a web server. The content of the responses can be

dynamically created at the time of the request and does not need to be available statically (such as in the form of an HTML page) for the web server (Wikimedia Foundation, 2022a). In this example the servlet is named *uploader* and maps to the JSP file *uploader.js*. The next sections will explain the development process step by step.

addproducts.html

This HTML file is responsible for providing a form for the user in which the product specifications can be entered as well as the image file selected. The main difference of the form in this file is the *action* attribute and *enctype* attribute. Listing 32 displays the opening form tag and its attributes. Instead of forwarding to a JSP file the action attribute forwards the data to the servlet *uploader*. The *enctype* is used to determine how the form data is encoded. To be able to upload files it is necessary to set this attribute to *multipart/form-data* (W3Schools, n.d.-c).

```
<form action="uploader"  style="margin: 10px" enctype="multipart/form-data"
method="post">
```

Listing 32: addproducts.html - <form> tag

web.xml configuration

To be able to send the data to the servlet some additional configuration is required. This configuration can either be hard coded in the servlet itself or the attributes can be added as a child element to the web.xml file. For the examples of this thesis, the second option was chosen (Fadatare, n.d.). The configuration that needs to be added to the web.xml file is shown in listing 33. In this file it gets declared that the servlet named *uploader* uses the code inside the JSP file *uploader.jsp*. Furthermore, the <*multipart-config*> attribute is used to enable file uploads (Oracle, n.d.-f Sec. The @MultipartConfig Annoation). Important too is the configuration of the <*servlet-mapping*>. This maps the servlet to the URL pattern */admin/uploader*. Therefore, it can be accessed by the URL localhost:8080/fruitshop/admin/uploader. This allows the servlet to handle requests that match that specific URL pattern (Joe, 2014).

```
<servlet>
    <servlet-name>uploader</servlet-name>
    <jsp-file>/admin/code/uploader.jsp</jsp-file>
    <multipart-config>
     <location>C:\Users\Simon\Documents\apache-tomcat-10.0.23\files\</location>
     <max-file-size>10000000</max-file-size>
     <max-request-size>10000000</max-request-size>
    </multipart-config>
   </servlet>

   <servlet-mapping>
    <servlet-name>uploader</servlet-name>
    <url-pattern>/admin/uploader</url-pattern>
   </servlet-mapping>
```

Listing 33: web.xml - multipart configuration

uploader.jsp

Once the form is submitted, the form data gets handled by the *uploader* servlet. This servlet is using the written code in the *uploader.jsp* file. In principle, most of the parts are equal to the other programs where user input was added to the database. However, two main things are different in this file. First as this is a servlet it is not necessary to have any HTML start tags. Instead after the declaration of the JSP directives the script immediately starts (Lux, 2021).

```
if (checkIfProductExists()) {
       var filename = name + ".jpg";
       var location = "/files/" + filename;
       request.getPart("file").write(filename);
```

Listing 34: uploader.jsp - writing a file to the server

The second difference is that a file gets uploaded. Therefore, the file needs to be written to the server and the path to find this newly stored file needs to be written to the database. Listing 34 shows the implementation of the code that is responsible for requesting the file and uploading it to the server in the desired location. To do so the method *getPart()* is used. By this method the file from the form data is returned (Oracle, n.d.-g). The file object is then directly written to the database by the method *write()*. The object that calls the method gets stored in the server. The location where it is stored is determined by the *<multipart-config>* attribute in the *web.xml* file. Therefore, the location is the */files* folder. The string inside the brackets is the name of the stored file object (baeldung, 2018). The location variable is later also used to access the stored image. For that the path gets saved on the database in the column pictures.

5.2 Sending Emails

This section presents an approach to build an application by which advertisement emails can be sent to customers automatically. Therefore, a form is developed in which the products that should be advertised are chosen. Those products are then shown in the email together with a link to the online shop. Furthermore, a link to unsubscribe from those emails needs to be implemented in the email too.

5.2.1 Prerequisites

This part is about the needed set ups to be able to develop the program.

Required .jar files

To enable sending emails, two jar files need to be downloaded and added to the lib directory of the web server. The first file that is required is called Jakarta Mail. However, this file depends on the second .jar file that is required, namely Jakarta Activation (Lux, 2021). Both files can be downloaded together in a zip formatted folder under the following link: *https://jar-download.com/artifacts/com.sun.mail/jakarta.mail/2.0.1/source-code*. After the download is complete the jar files need to be extracted and added to the *lib* folder.

MailHog

MailHog is an open-source tool that helps developers to test how their program sends emails. Instead of sending emails to a real email server, the application can send them to MailHog. The emails received by MailHog can be viewed and analyzed by accessing the URL where MailHog is running (Broda, n.d., Sec. What is MailHog?).

The installation and setup of MailHog can be done extremely fast. The first step is to go to the GitHub page of MailHog by following this link: *https://github.com/mail-hog/MailHog*. The section *Releases* can be found in the middle right side of this page. Now the latest version of MailHog should be selected. This will open a page with several *Assets*. According to the operating system of a visitor, the correct asset needs to be downloaded (Lux, 2021). After the download is complete the program is ready to be used and can be executed. By default, the SMTP server runs on port 1025. Therefore, the emails need to be sent to this port. The HTTP server starts on port 8025,

which means that it can be accessed in the browser via the URL: *localhost/8025* (Broda, n.d. Sec. Installation).

web.xml

To be able to use this application it is again required to declare a servlet in the *web.xml* file of the project. A developer can just copy and paste the configuration from the *uploader* discussed before and add it directly below it. Then only two things need to be changed. First every word *uploader* needs to be exchanged with *mailer*. The second thing to do is to delete the <*multipart-config*> part as it is not needed to handle file uploads in this servlet. After that the servlet *mailer* uses the code inside the *mailer.jsp* file and can be accessed by the URL pattern */admin/mailer*. It makes sense to straightaway do the same configuration for the *unsubscriber* servlet. The *mailer* servlet configuration can be copied and pasted directly below. Now the only thing to do is change each word *mailer* to *unsubscriber*.

5.2.2 Sending a Newsletter

Two files are required to be able to automatically send newsletters to customers who are willing to receive them. The first one is a JSP file responsible for choosing the products displayed in the mail and selecting the customers addresses. The second one is a servlet which creates the email that is sent to the customers based on the form data that is forwarded to it.

newsletter.jsp

This file dynamically prints a form to choose the products for the email. Therefore, the form consists of a checkbox for each product stored in the database. Each checkbox field has the value of a different product name. To communicate the value of the checkboxes to the user, the names are displayed besides the checkbox. If a checkbox field is selected when the form is submitted its value is forwarded to the servlet. In the servlet the sent values can then be requested by the parameter choice. The code in listing 35 is responsible for printing a list item in the form for each product that is stored in the database.

```
while (rs.next()) {
    out.println('<br>');
    out.println('<li><label for="choice">' + rs.getString("name") + '</label>');
    out.println('<input type="checkbox" name="choice" value="' +
rs.getString("name") + '"></li>');
}
```

Listing 35: newsletter.jsp - while loop to print all products inside the form

Nonetheless, the script also selects all customers who are willing to receive an email. By calling the method *next()* on the returned ResultSet in combination with a while loop the variable *count* is increased by one for each iteration. Thus, the variable count can be used to know how many customers are willing to receive an email. However, this information is solely used to inform the user of the number of receivers. The method by which the number of receivers is determined is shown in listing 36.

```
var qry = "SELECT * FROM customer WHERE receives_mail = '1';";
var rs = stmt.executeQuery(qry); // check how many people are subscribed

var count = 0;
while (rs.next()) {
  count++;
}
```

Listing 36: newsletter.jsp - figure out the number of email receivers

mailer.jsp

Once the form in newsletter.jsp got submitted, the form data is transmitted to the servlet *mailer*. This servlet then uses the code inside the mailer.jsp to process the data and send the emails. First to get the names of the products that should be present in the email, the method *getParamaterValues()* is used to retrieve all values that have been forwarded under the name *choice*. Listing 37 displays the line of code that is responsible for this request.

```
var choices = request.getParameterValues("choice");
```

Listing 37: mailer.jsp - getParamaterValues()

The next new part in this is the definition of a variable *choice* as an empty string. After that a for loop is used to iterate through the array *choices.* For each iteration the element of the array (in this case the product names) gets added to the *choice* variable. The resulting *choice* is then a sting that contains the list of product names that need to be sent to the customers by email. The products in *choice* are surrounded by single

quotes and separated by commas. The code responsible for this process is shown in listing 38.

```
var choice = '';
    for (var i = 0; i < choices.length; i++) {
        // append all product names to a string
        choice += "'" + choices[i] + "',";
    }
```

Listing 38: mailer.jsp – for loop to save product names in a string

Next the customers willing to receive emails are retrieved from the database by an SQL SELECT statement. An email is created for each customer by iterating over the returned ResultSet with a while. The emails are sent to the email addresses entered in the *username* field of the database table customer. Inside this while loop important attributes like the receiver address, the sender address and the subject are set. Furthermore, the content of the email gets generated by using HTML. A link to the online shop and a link to unsubscribe from those emails are added to the variable text. Also the products that should be visible in the email are added to this variable in form of HTML. Later this variable is set as the content of the email. To accomplish this, the method *setContent()* is called on an instance of the Java class *MimeMessage*. This instance, like several other necessary objects, gets created at the beginning of each iteration of the while loop. The *MimeMessage* class enables to use other formats than ASCII for the text. Therefore, HTML can be used to generate the content of the email (Oracle, n.d.-e).

Figure 23 presents a screenshot of the email received by the MailHog application under port 8025. To be able to analyze the emails that way, they need to be sent to port 1025.

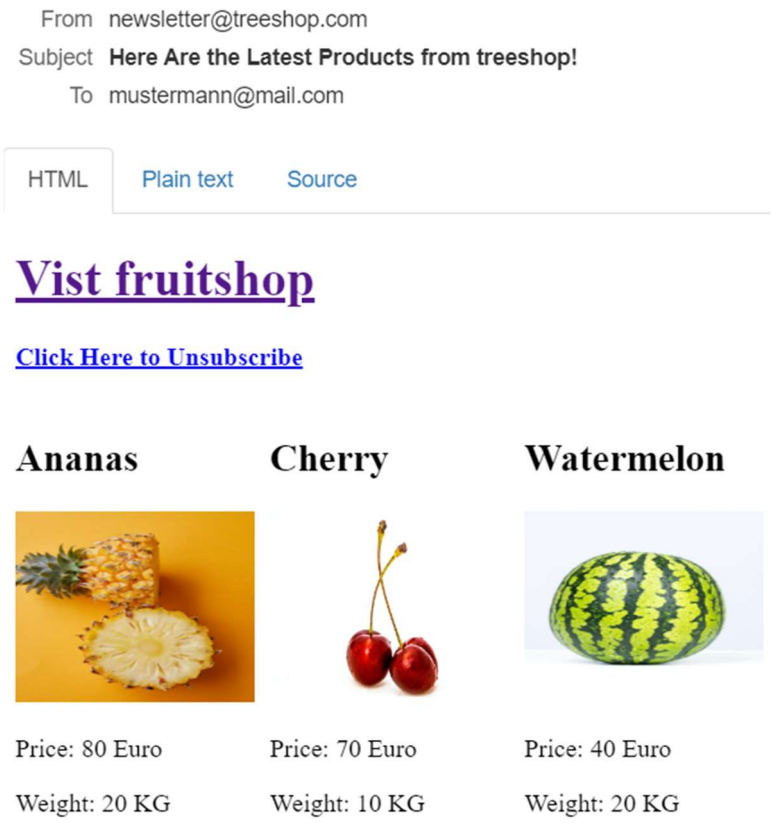

Figure 23: Newsletter email received in MailHog

5.2.3 Unsubscribing the Newsletter

It is necessary to offer a way how recipients of the email can communicate that they do not want to receive any further emails. Therefore, a link to unsubscribe is placed inside every newsletter email.

unsubscribe.jsp

When a user clicks the unsubscribe link in the email the file unsubscribe.jsp opens. Once opened, it displays a form which request the user to confirm that the newsletter subscription should be terminated. On submit the form data forwarded to the servlet uploader contains the email address of the customer. This is necessary for the servlet to know which customer should be updated in the database.

unsubscriber.jsp

The servlet unsubscriber uses the code inside the file unsubscriber.jsp. The servlet is responsible for updating the *receives_mail* column in the table customer. This can be done because the username is sent to the servlet via a parameter named *unsub*.

6 Conclusion and Future Work

Using Nashorn for the development of JSP is a beginner friendly way to start with web application development. It is a powerful yet easy way to build web applications from the client- to the server-side. During the process of developing such a web application one will learn a lot of tremendously important skills for a future career in this field. Developing an online shop from the bottom up will deliver entirely new insights into what is required for providing a fully functional web shop.

However, the process of developing such applications can be very time consuming and exhausting. During this process developers will sooner or later look at hurdles which will seem nearly impossible to overcome. Though, many of the problems had to be dealt within this thesis and are documented in this paper. Therefore, this work presents a good starting point for anyone who would like to develop JSPs with the Nashorn script engine. Unfortunately, in contrast to Node.js, many required back-end development capabilities are not implemented. It would be great if a deeply committed software engineer or group of engineers develop a new JavaScript scripting engine. However, the usage of the tag library jsr-223.tld compensates for this drawback. Furthermore, this paper presents an alternative approach for server-side development with JavaScript. Though, it is not the aim of the paper to do a comparison between this approach and the more popular framework Node.js. This comparison is left for future research.

Bibliography

Apache Software Foundation. (n.d.-a). *Apache Tomcat®—Which Version Do I Want?* Retrieved December 30, 2022, from https://tomcat.apache.org/whichversion.html

Apache Software Foundation. (n.d.-b). *Apache Tomcat 9 (9.0.71)—JNDI Resources How-To*. Retrieved January 19, 2023, from https://tomcat.apache.org/tomcat-9.0-doc/jndi-resources-howto.html

Apache Software Foundation. (n.d.-c). *Apache Tomcat 10 (10.0.27)—Manager App How-To*. Retrieved January 10, 2023, from https://tomcat.apache.org/tomcat-10.0-doc/manager-howto.html#Configuring_Manager_Application_Access

Arias, D. (2019, September 30). *How to Hash Passwords: One-Way Road to Enhanced Security*. Auth0 - Blog. https://auth0.com/blog/hashing-passwords-one-way-road-to-security/

ASM. (n.d.). Retrieved January 10, 2023, from https://asm.ow2.io/

baeldung. (2018, May 19). *Uploading Files with Servlets and JSP | Baeldung*. https://www.baeldung.com/upload-file-servlet

baeldung. (2022, May 21). *Difference Between Statement and PreparedStatement | Baeldung*. https://www.baeldung.com/java-statement-preparedstatement

BalusC. (2016, August 2). *Answer to "Cookies created in JSP page are not available in Servlet, only the JSESSIONID cookie is available."* Stack Overflow. https://stackoverflow.com/a/38714985/19175093

Bandara, S. (2018, February 11). Nashron in a Nutshell. *Medium*. https://technospace.medium.com/nashron-in-a-nutshell-b804352cf3e0

BrainStation. (n.d.). *What Is Web Development? (2023 Guide)*. BrainStation®. Retrieved January 11, 2023, from https://brainstation.io/career-guides/what-is-web-development

Broda, S. (n.d.). *Using MailHog for local email testing*. Kirby CMS. Retrieved January 27, 2023, from https://getkirby.com/docs/cookbook/forms/using-mailhog-for-email-testing

Computer Hope. (2018, May 21). *What is an Environment Variable?* Computer Hope. https://www.computerhope.com/jargon/e/envivari.htm

Concatenated Primary Key(Database Table Definition). (n.d.). Retrieved January 27, 2023, from https://www.relationaldbdesign.com/database-analysis/module2/concatenated-primary-keys.php

Coursera. (2022, November 1). *What Does a Back-End Developer Do?* Coursera. https://www.coursera.org/articles/back-end-developer

DigiCert, Inc. (n.d.). *What is an SSL Certificate?* | *DigiCert*. Retrieved January 20, 2023, from https://www.digicert.com/what-is-an-ssl-certificate

Educative, Inc. (n.d.). *GET vs. POST*. Educative: Interactive Courses for Software Developers. Retrieved January 21, 2023, from https://www.educative.io/answers/get-vs-post

erickson. (2011, July 26). *Answer to "How can bcrypt have built-in salts?"* Stack Overflow. https://stackoverflow.com/a/6833165/19175093

Fadatare, R. (n.d.). *@MultipartConfig Annotation Example*. Retrieved January 27, 2023, from https://www.javaguides.net/2019/02/multipartconfig-annotation-example.html

Fireart Studio. (2019, December 6). *Is Node.js Still Relevant For Your Startup in 2023?* Fireart Studio. https://fireart.studio/blog/why-node-js-is-still-a-good-choice-for-your-startup-in-2020/

Flatscher, R. G. (2021, September 21). *BSF4ooRexx—Browse /Sandbox/rgf/taglibs/ga at SourceForge.net*. BSF4ooRexx. https://sourceforge.net/projects/bsf4oorexx/files/Sandbox/rgf/taglibs/ga/

Flynn, M. (2013, March 6). *Answer to "Tomcat keeps resetting my tomcat-users.xml file."* Stack Overflow. https://stackoverflow.com/a/15255286/19175093

GeeksforGeeks. (2017, November 6). SQL | DDL, DQL, DML, DCL and TCL Commands. *GeeksforGeeks*. https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/

GeeksforGeeks. (2022, January 7). Top 10 Programming Languages to Learn in 2022. *GeeksforGeeks*. https://www.geeksforgeeks.org/top-10-programming-languages-to-learn-in-2022/

GitHub Inc. (n.d.). *The top programming languages*. The State of the Octoverse. Retrieved January 29, 2023, from https://octoverse.github.com/2022/top-programming-languages

gittysatyam. (2021, October 20). Difference between Session and Cookies. *GeeksforGeeks*. https://www.geeksforgeeks.org/difference-between-session-and-cookies/

How to Connect to SQLite via JDBC. (n.d.). Retrieved January 20, 2023, from https://razorsql.com/articles/sqlite_jdbc_connect.html

How To Download & Install SQLite Tools. (n.d.). *SQLite Tutorial*. Retrieved January 19, 2023, from https://www.sqlitetutorial.net/download-install-sqlite/

How to Download and Install OpenJDK 11 on Windows 10 PC for Aleph. (2020, January 3). Ex Libris Knowledge Center. https://knowledge.exlibrisgroup.com/Aleph/Knowledge_Articles/How_to_Download_and_Install_OpenJDK_11_on_Windows_10_PC_for_Aleph

International Business Machines Corporation (IBM). (2021, March 16). *IBM Documentation*. https://prod.ibmdocs-production-dal-6099123ce774e592a519d7c33db8265e-0000.us-south.containers.appdomain.cloud/docs/en/rad-fws/9.6.1?topic=SSRTLW_9.6.1/com.ibm.etools.pagedesigner.doc/topics/ccusttaglib.htm

Jakarta Server Pages Team. (n.d.). *Jakarta Server Pages*. Retrieved January 12, 2023, from https://jakarta.ee/specifications/pages/3.0/jakarta-server-pages-spec-3.0.html#what-is-a-jsp-page-2

Java DriverManager—Javatpoint. (n.d.). Www.Javatpoint.Com. Retrieved January 24, 2023, from https://www.javatpoint.com/DriverManager-class

JDBC Tutorial | What is Java Database Connectivity(JDBC)—Javatpoint. (n.d.). Www.Javatpoint.Com. Retrieved January 19, 2023, from https://www.javatpoint.com/java-jdbc

Joe. (2014, September 11). *What is servlet mapping?* Javapapers. https://javapapers.com/servlet/what-is-servlet-mapping/

Lux, D.-J. (2021). *An Introduction to Web Application Development – Combining Jakarta Server Pages with Programs Written in Scripting Languages*. Vienna University of Economics and Business Administration. Retrieved January 10, 2023, from https://wi.wu.ac.at/rgf/diplomarbeiten/BakkStuff/2021/202102_Lux_IntroductionToWebApplicationDevelopment.pdf

Mihn, N. H. (2019, August 6). *How to configure session timeout in Tomcat*. https://www.codejava.net/servers/tomcat/how-to-configure-session-timeout-in-tomcat

Miller, D. (n.d.). *BCrypt (jBCrypt 0.4 API)*. Retrieved January 26, 2023, from https://www.java-doc.io/doc/org.mindrot/jbcrypt/0.4/org/mindrot/jbcrypt/BCrypt.html

MIME-Type/Übersicht – SELFHTML-Wiki. (2022, April 12). https://wiki.self-html.org/wiki/MIME-Type/%C3%9Cbersicht

Mozilla Foundation. (2022a, December 5). *Sending form data—Learn web development | MDN*. https://developer.mozilla.org/en-US/docs/Learn/Forms/Sending_and_retrieving_form_data

Mozilla Foundation. (2022b, December 13). *Date—JavaScript | MDN*. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date

Mozilla Foundation. (2022c, December 13). *Date.prototype.getHours()—JavaScript | MDN*. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/getHours

Mozilla Foundation. (2023a, January 13). *HTTPS - MDN Web Docs Glossary: Definitions of Web-related terms | MDN*. https://developer.mozilla.org/en-US/docs/Glossary/HTTPS

Mozilla Foundation. (2023b, January 23). *Using HTTP cookies—HTTP | MDN*. https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies

Nashorn Engine. (2022, April 23). https://github.com/openjdk/nashorn (Original work published 2020)

no.good.at.coding. (2011, April 1). *Answer to "Why set a JSP page session = 'false' directive?"* Stack Overflow. https://stackoverflow.com/a/5516893/19175093

Olawanle, J. (2022, June 17). *JavaScript Get Current Date – Today's Date in JS*. FreeCodeCamp.Org. https://www.freecodecamp.org/news/javascript-get-current-date-todays-date-in-js/

Oracle. (n.d.-a). *Connection (Java Platform SE 7)*. Retrieved January 24, 2023, from https://docs.oracle.com/javase/7/docs/api/java/sql/Connection.html

Oracle. (n.d.-b). *Explicitly closing Statements, ResultSets, and Connections* [Concept]. Retrieved January 26, 2023, from https://docs.oracle.com/javadb/10.8.3.0/devguide/cdevconcepts839085.html

Oracle. (n.d.-c). *Invalidating a Session (Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications)*. Retrieved January 26, 2023, from https://docs.oracle.com/cd/E19146-01/819-2634/abxdj/index.html

Oracle. (n.d.-d). *Java Platform, Standard Edition Nashorn User's Guide*. Retrieved January 13, 2023, from https://docs.oracle.com/javase/9/nashorn/nashorn-java-api.htm#JSNUG115

Oracle. (n.d.-e). *MimeMessage (Java EE 6)*. Retrieved January 28, 2023, from https://docs.oracle.com/javaee/6/api/javax/mail/internet/MimeMessage.html

Oracle. (n.d.-f). *Processing SQL Statements with JDBC*. Retrieved January 24, 2023, from https://docs.oracle.com/javase/tutorial/jdbc/basics/processingsqlstatements.html#establishing_connections

Oracle. (n.d.-g). *The getParts and getPart Methods—The Java EE 6 Tutorial*. Retrieved January 27, 2023, from https://docs.oracle.com/javaee/6/tutorial/doc/gmhba.html

Oracle. (n.d.-h). *The @MultipartConfig Annotation—The Java EE 6 Tutorial*. Retrieved January 27, 2023, from https://docs.oracle.com/javaee/6/tutorial/doc/gmhal.html

Selzer, M. (2020, April 28). *Salt and Hash Passwords with bcrypt*. https://heynode.com/blog/2020-04/salt-and-hash-passwords-bcrypt/

Shah, A. (2019a, January 6). *Step by Step Guide to Setup and Install Apache Tomcat Server in Eclipse Development Environment (IDE) • Crunchify*. Crunchify. https://crunchify.com/step-by-step-guide-to-setup-and-install-apache-tomcat-server-in-eclipse-development-environment-ide/

Shah, A. (2019b, August 4). *Tomcat starts but Home Page does NOT open on browser with URL http://localhost:8080 • Crunchify*. Crunchify. https://crunchify.com/tomcat-starts-but-home-page-does-not-open-on-browser-with-url-http-localhost8080/

Sheldon, R., & Denman, J. (n.d.). *What is the Node.js (Node) runtime environment?—TechTarget Definition*. WhatIs.Com. Retrieved January 29, 2023, from https://www.techtarget.com/whatis/definition/Nodejs

Simmons, L. (2022, January 7). *The Difference Between Front-End vs. Back-End | ComputerScience.org*. https://www.computerscience.org/bootcamps/resources/frontend-vs-backend/

SQLite: Documentation. (n.d.). Retrieved January 19, 2023, from https://www.sqlite.org/src/doc/trunk/ext/userauth/user-auth.txt

SQLite Download Page. (n.d.). Retrieved January 19, 2023, from https://www.sqlite.org/download.html

Szegedi, A. (2022, October 14). *Answer to "How to add the Nashorn module to Tomcat 10."* Stack Overflow. https://stackoverflow.com/a/74069391/19175093

Tandetnik, I. (n.d.). *Answer to "BOOLEAN DEFAULT VALUE."* Sqlite-Users. Retrieved December 29, 2022, from https://sqlite-users.sqlite.narkive.com/zpH7xG5x/boolean-default-value

The Owasp Foundation. (n.d.). *Password Storage—OWASP Cheat Sheet Series*. Retrieved January 21, 2023, from https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Tyson, M. (2022, September 9). *What is JSP? Introduction to Jakarta Server Pages*. InfoWorld. https://www.infoworld.com/article/3336161/what-is-jsp-introduction-to-javaserver-pages.html

Vukotic, A., & Goodwill, J. (2011). Introduction to Apache Tomcat 7. In A. Vukotic & J. Goodwill, *Apache Tomcat 7* (pp. 1–15). Apress. https://doi.org/10.1007/978-1-4302-3724-2_1

W3Schools. (n.d.-a). *HTML body tag*. Retrieved January 12, 2023, from https://www.w3schools.com/tags/tag_body.asp

W3Schools. (n.d.-b). *HTML doctype declaration*. Retrieved January 12, 2023, from https://www.w3schools.com/tags/tag_doctype.asp

W3Schools. (n.d.-c). *HTML form enctype Attribute*. Retrieved January 27, 2023, from https://www.w3schools.com/tags/att_form_enctype.asp

W3Schools. (n.d.-d). *HTML Form—Javatpoint*. Www.Javatpoint.Com. Retrieved January 18, 2023, from https://www.javatpoint.com/html-form

W3Schools. (n.d.-e). *HTML head tag*. Retrieved January 12, 2023, from https://www.w3schools.com/tags/tag_head.asp

W3Schools. (n.d.-f). *HTML header Tag*. Retrieved January 12, 2023, from https://www.w3schools.com/tags/tag_header.asp

W3Schools. (n.d.-g). *HTML html tag*. Retrieved January 12, 2023, from https://www.w3schools.com/tags/tag_html.asp

W3Schools. (n.d.-h). *HTML title tag*. Retrieved January 12, 2023, from https://www.w3schools.com/tags/tag_title.asp

W3Schools. (n.d.-i). *HTTP Methods GET vs POST*. Retrieved December 30, 2022, from https://www.w3schools.com/tags/ref_httpmethods.asp

W3Schools. (n.d.-j). *SQL SELECT Statement*. Retrieved January 24, 2023, from https://www.w3schools.com/sql/sql_select.asp

Wikimedia Foundation. (2022a). Jakarta Servlet. In *Wikipedia*. https://de.wikipedia.org/w/index.php?title=Jakarta_Servlet&oldid=222892008

Wikimedia Foundation. (2022b). Nashorn (JavaScript engine). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Nashorn_(JavaScript_engine)&oldid=1094201862

Wikimedia Foundation. (2022c). SQLite. In *Wikipedia*. https://de.wikipedia.org/w/index.php?title=SQLite&oldid=227994308

Wikimedia Foundation. (2022d). Bcrypt. In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Bcrypt&oldid=1130147534

Wikimedia Foundation. (2023). Entity–relationship model. In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Entity%E2%80%93relationship_model&oldid=1134142899

yaminitalisetty. (2021, December 24). Servlet—SendRedirect() Method with Example. *GeeksforGeeks*. https://www.geeksforgeeks.org/servlet-sendredirect-method-with-example/

Zhang, L. (2022, September 14). *What's the Salt Length Used by Auth0*. Auth0 Community. https://community.auth0.com/t/whats-the-salt-length-used-by-auth0/90617

Zhukov, A. (2013, November 1). *Answer to "SQLite3 serial type wasn't incremented."* Stack Overflow. https://stackoverflow.com/a/19726143/19175093

7 Appendix

This section contains the codes of all applications developed and in addition to that showcases the necessary commands to create the database.

7.1 Table Creation and Value Insertion

The following paragraphs discuss the creation of the database tables as well as the insertion of the needed values.

Creation of the tables

This section provides pictures of the necessary SQLite instructions used to create the tables.

```
sqlite> create table fruit(
   ...> fruit_id integer primary key autoincrement NOT NULL,
   ...> name varchar(40) NOT NULL,
   ...> price varchar (20) NOT NULL,
   ...> weight varchar (20),
   ...> picture varchar (100));
```

Figure 24: Creation SQLite table fruit

In order to have incrementing integers regarding the fruit_ids SQLite provides the method *autoincrement* (Zhukov, 2013).

```
sqlite> create table customer(
   ...> customer_id integer primary key autoincrement,
   ...> username varchar (40) NOT NULL,
   ...> password varchar (100) NOT NULL,
   ...> receives_mail BOOLEAN default 0);
sqlite>
```

Figure 25: Creation SQLite table customer

Unfortunately, SQLite does not have a dedicated boolean type, thus the number *0* for *FALSE* and *1* for *TRUE* needs to be used instead (Tandetnik, n.d.).

```
sqlite> create table shopping_cart(
   ...> fruit_id integer REFERENCES fruit ON UPDATE CASCADE ON DELETE CASCADE,
   ...> customer_id integer REFERENCES customer ON UPDATE CASCADE ON DELETE CASCADE,
   ...> quantity integer,
   ...> PRIMARY KEY (fruit_id, customer_id)
   ...> );
```

Figure 26: Creation SQLite table shopping_cart

In the table *shopping_cart* the primary key used is composited of *fruit_id* and *customer_id*. Those keys are both foreign keys as they are primary keys of other tables (*Concatenated Primary Key(Database Table Definition)*, n.d.).

Inserting values

This section provides the necessary commands to insert data into the previously created tables.

```
sqlite> insert into fruit (name, price, weight, picture) values
   ...> ('Ananas', 80, 20, '/files/ananas.jpg'),
   ...> ('Orange', 50, 15, '/files/orange.jpg'),
   ...> ('Cherry', 70, 10, '/files/cherry.jpg'),
   ...> ('Banana', 100, 25, '/files/banana.jpg'),
   ...> ('Guave', 90, 10, '/files/guave.jpg'),
   ...> ('Watermelon', 40, 20, '/files/watermelon.jpg');
```

Figure 27: Value insertion table fruit

The values of the table *customer* were inserted using the built application to sign up. However, figure 28 displays how the table looks like after some customers have been created.

```
sqlite> select * from customer;
37|buggy@gmy.at|$2a$12$CpYF.x.XRMGRLs28dYBEoua9XUzVHbLrsSfyi1Ags0yYwwpIJPvmG|0
38|player@yahu.com|$2a$12$PeogW1ip6FW25AOtT86/fOjGy4umMvpEdU81yz94mR/xDYjcZYD0C|0
39|bigboy@gmoil.wu|$2a$12$dUZMUU1c6CNbTitfGqC.VeePgeoYB7yQ69ueScoNLjxt.nKRKiMh.|0
```

Figure 28: Customer table after insertion of example users

Equally to *customer*, no values were inserted by hand into the table *shopping_cart*. Though an example of how the table looks like after some values have been inserted is visible in figure 29.

```
sqlite> select * from shopping_cart;
1|38|5
3|38|7
4|38|4
```

Figure 29: shopping_cart table

The first column here is the fruit_id. By that it can be determined which product has been added. The second one is the id of the customer and therefore the row can be allocated to the correct user. The last column is indicating the quantity.

7.2 “Hello, world” Project

This dynamic web project showcases the development of some basic nutshell examples. The examples reach from basic “hello, world” applications, to more advanced projects. For example the use of cookies to generate personalized content gets examined.

7.2.1 index.html

This file acts as a port for the applications developed. From here the different programs can be accessed via links.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>helloworld</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
	<h1>Helloworld Web Applications</h1>
	<h3>Welcome! <br> This is thought to be the harbor for the helloworld projects
listed below.</h3>
	<h3>The projects which you can visited on their links start very simple and
increase in complexity</h3>
	<a href="helloworld.jsp">Go to helloworld.jsp</a><br>
	<a href="helloworld_ext.jsp">Go to helloworld_ext.jsp</a> <br>
	<a href="lastvisit.jsp">Go to lastvisit.jsp</a> <br>
	<a href="greeting.jsp">Go to greeting.jsp</a><br>
	<a href="greeting_ext.jsp">Go to greeting_ext.jsp</a><br>
</body>
</html>

<!--
    ----------------------- Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at
```

```
            http://www.apache.org/licenses/LICENSE-2.0

        Unless required by applicable law or agreed to in writing, software
        distributed under the License is distributed on an "AS IS" BASIS,
        WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
        See the License for the specific language governing permissions and
        limitations under the License.
    ------------------------------------------------------------------------
-->
```

Listing 39: /helloworld/index.jsp

7.2.2 helloworld.jsp

The following file shows how to save a string in a variable as well as how to print it to the HTML document.

```
<%@ page session="false" pageEncoding="UTF-8" contentType="text/html;
charset=UTF_8" %>
<%@ taglib uri="./WEB-INF/script-jsr223.tld" prefix="s" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Hello, World</title>
</head>
<body>

<s:script type="javascript">
      var greeting = "Hello, world! (Sent from Nashorn)";
      print("<div>" + greeting + "</div>")
</s:script>

</body>
</html>

<%--
    ------------------------ Apache Version 2.0 license ------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    ------------------------------------------------------------------------
--%>
```

Listing 40: /helloworld/helloworld.jsp

7.2.3 helloworld_ext.jsp

This file extends the previous helloworld.jsp by using a script which dynamically generates HTML code that displays the current time. To do so the method Date() is used inside an expression tag.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8"%>
<%@ taglib uri="./WEB-INF/script-jsr223.tld" prefix="s" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Hello, world</title>

</head>
<body>
<header>
<s:script type="javascript">
var greeting = "Hello, world! (Sent from Nashorn)";
print('<h1>' + greeting + '</h1>');
</s:script>
</header>
<body>

<p>The time right now: <s:expr type="javascript">Date()</s:expr></p>

</body>
</html>

<%--
    ----------------------- Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    ------------------------------------------------------------------------------
--%>
```

Listing 41: /helloworld/helloworld_ext.jsp

7.2.4 lastvisit.jsp

This program shows how cookies can be used to create a dynamic HTML page which is sent to the client.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8"%>
<%@ taglib uri="./WEB-INF/script-jsr223.tld" prefix="s" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>last visit</title>
</head>
<body>

<s:script type="javascript" throwException="true">
//reqeuest and response procedures
//first do the request and response because response won't work if something was
already printed

//request
var lastVisit;
var allCookies = request.getCookies();

//response to add the current time
var today = new Date();
var time = today.getHours() + ":" + today.getMinutes() + ":" + today.getSeconds();
timeString = String(time);
var Cookie = Java.type("jakarta.servlet.http.Cookie");
var newCookie = new Cookie("lastVisit", timeString);
newCookie.setMaxAge(60 * 60 * 24);
newCookie.setPath("/");
response.addCookie(newCookie);
</s:script>


<s:script type="javascript">
//Print the last visit or if it is the first visit

if (allCookies != null) {
	for (var i = 0; i < allCookies.length; i++) {
		var c = allCookies[i];
		if (c.getName() == 'lastVisit') {
			lastVisit = c.getValue();
			break;
		}
	}
}

if (lastVisit == null) {
	print("This is your first Visit!");
} else {
	print("Your last visit was at " + lastVisit);
}
</s:script>

</body>
</html>

<%--
    ------------------------ Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck
```

```
        Licensed under the Apache License, Version 2.0 (the "License");
        you may not use this file except in compliance with the License.
        You may obtain a copy of the License at

            http://www.apache.org/licenses/LICENSE-2.0

        Unless required by applicable law or agreed to in writing, software
        distributed under the License is distributed on an "AS IS" BASIS,
        WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
        See the License for the specific language governing permissions and
        limitations under the License.
    ------------------------------------------------------------------------
--%>
```

Listing 42: /helloworld/lastvisit.jsp

7.2.5 greeting.jsp

This example web application uses the cookie technology to individually greet a visitor once a form with a username was entered and sent to the server.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8"%>
<%@ taglib uri="./WEB-INF/script-jsr223.tld" prefix="s" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>

<!-- Request the Cookies and get the Value of the Cookie username -->
<s:script type="javascript">
var username;
var allCookies = request.getCookies();

if (allCookies != null) {
      for (var i = 0; i < allCookies.length; i++) {
            var c = allCookies[i];
            if (c.getName() == 'username') {
                  username = c.getValue();
            }
      }
}
</s:script>


<!-- Printing either the input field or the 'greeting' -->
<s:script type="javascript">
if (username == null) {
      out.println('<p>Hello what is your name?</p>');
      out.println('<form>' +
                        '<label for="username">Username:</label>' +
                        '<input type="text" name="username" required>' +
                        '<input type="submit" value="Ok">' +
                  '</form>');
} else {
```

```
        out.println('<p>Welcome back, ' + username + '!</p>')
}
</s:script>

<!-- creating cookie and adding it to the response -->
<s:script type="javascript">
if (request.getParameter('username') != null) {
        var uname = request.getParameter('username');
        var Cookie = Java.type("jakarta.servlet.http.Cookie");
        var newCookie = new Cookie("username", uname);
        newCookie.setMaxAge(60 * 60 * 24);
        newCookie.setPath("/");
        response.addCookie( newCookie );
        response.sendRedirect(request.getRequestURI())
}
</s:script>

</body>
</html>

<%--
    ------------------------ Apache Version 2.0 license ------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    ----------------------------------------------------------------------------
--%>
```

Listing 43: /helloworld/greeting.jsp

7.2.6 greeting_ext.jsp

This program extends the former greeting.jsp file by the functionality to be able to log out. Furthermore, it gets demonstrated how scripts can be outsourced to other files and then accessed by the JSP. Therefore, the functionality to add and remove the cookie is not inside this file.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8"%>
<%@ taglib uri="./WEB-INF/script-jsr223.tld" prefix="s" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>helloworld</title>
</head>
```

```
<body>


<s:script type="javascript">
var username;
var allCookies = request.getCookies();

if (allCookies != null) {
	for (var i = 0; i < allCookies.length; i++) {
		var c = allCookies[i];
		if (c.getName() == 'username') {
			username = c.getValue();
		}
	}
}

if (username == null) {
	out.println('<p>Hello what is your name?</p>');
	out.println('<form>' +
				'<label for="username">Username:</label>' +
				'<input type="text" name="username" required>' +
				'<input type="submit" value="Ok">' +
			'</form>');
} else {
	out.println('<p>Welcome back, ' + username + '!</p>')
	out.println('<form>' +
				'<input type="hidden" name="logoutButton" value="1">' +
				'<input type="submit" value="Logout">' +
			'</form>');
}
</s:script>


<s:script type="javascript" src="code/logout.js" cacheSrc="false" />

</body>
</html>

<%--
    ------------------------ Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    ------------------------------------------------------------------------------
--%>
```

Listing 44: /helloworld/greeting_ext.jsp

7.2.7 code/logout.js

This JavaScript file is accessed by the prior shown *greeting_ext.jsp* and is responsible for setting and deleting the cookies.

```
if (request.getParameter('username') != null) {
	var uname = request.getParameter('username');
	var Cookie = Java.type('jakarta.servlet.http.Cookie');
	var newCookie = new Cookie('username', uname);
	newCookie.setMaxAge(60 * 60 * 24);
	newCookie.setPath('/');
	response.addCookie(newCookie);
	response.sendRedirect(request.getRequestURI());
}

if (request.getParameter('logoutButton') != null) {
	var Cookie = Java.type('jakarta.servlet.http.Cookie');
	var removerCookie = new Cookie('username', '');
	removerCookie.setPath('/');
	removerCookie.setMaxAge(0);
	response.addCookie(removerCookie);
	response.sendRedirect(request.getRequestURI());
}

/*
    ------------------------ Apache Version 2.0 license -------------------------
	Copyright 2023 Simon Besenbäck

	Licensed under the Apache License, Version 2.0 (the "License");
	you may not use this file except in compliance with the License.
	You may obtain a copy of the License at

	    http://www.apache.org/licenses/LICENSE-2.0

	Unless required by applicable law or agreed to in writing, software
	distributed under the License is distributed on an "AS IS" BASIS,
	WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
	See the License for the specific language governing permissions and
	limitations under the License.
    ------------------------------------------------------------------------------
*/
```

Listing 45: /helloworld/code/logout.js

7.3 E-Commerce Example

This dynamic web project presents one of many possible ways to develop an online shop. Therefore, the project contains of several applications like sign-up, login and a shopping cart.

7.3.1 index.jsp

This page is the main JSP file of the online shop. However, the visible content is dynamically created in external script files.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib uri="./WEB-INF/script-jsr223.tld" prefix="s" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>fruitshop</title>
<link rel="stylesheet" href="css/style.css">
</head>
<header>
<h1>Welcome to our fruitshop!</h1>
<div class="userheader">
	<s:script type="javascript" src="code/userheader.js" cacheSrc="false" throwException="true" />
	</div>
</header>
<body>
	<s:script type="javascript" src="code/mainpage.js" cacheSrc="false" />
</body>
</html>

<%--
    ----------------------- Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
```

```
        See the License for the specific language governing permissions and
        limitations under the License.
    ----------------------------------------------------------------------------
 --%>
```

Listing 46: /fruitshop/index.jsp

7.3.2 style.css

This file is responsible for the style of the online shop.

```
@charset "UTF-8";

header {
        background-color: #F1F1F1;
    padding: 12px;
    box-shadow: -2px 0px 6px 0px rgb(0, 0, 0, 0.2);
    z-index: 1;
}

body {
        margin: 0;
        padding: 0;
        display: flex;
        flex-direction: column;
        background-color: #F9F9F9;
}

h2 {
        margin-block-start: 0;
}

.userheader {
        display: flex;
        justify-content: space-between;
}

.userheader-child {
        display: flex;
    flex-direction: row-reverse;
    gap: 20px;
}

.user_container {
        display: flex;
    flex-direction: column;
}

.submit-btn {
        cursor: pointer;
        padding: 10px;
    text-align: center;
    border-radius: 5px;
    background-color: lightgrey;
    font-weight: bold;
    text-decoration: none;
    color: black;
```

```
    margin-bottom: 14px;
    width: fit-content;
    border: none
}

.submit-btn:hover {
      background-color: rgb(199, 199, 199);
}

.warning {
      color: red;
}

.success {
      color: green;
}

.link-btn {
      padding: 10px;
    text-align: center;
    border-radius: 5px;
    background-color: lightgrey;
    font-weight: bold;
    text-decoration: none;
    color: black;
    margin-bottom: 14px;
    width: fit-content;
    margin-right: 10px;
    height: fit-content;
}

.link-btn:hover {
      background-color: grey;
}

.products-grid {
      display: flex;
      flex-wrap: wrap;
      gap: 40px;
      justify-content: center;
      margin-top: 40px;
      margin-bottom: 40px
}

.product-container {
      background-color: #F1F1F1;
      padding: 42px;
      border-radius: 8px;
}

.product-container img {
      object-fit: cover;
}

.login-form-headline {
      font-weight: bold;
      font-size: 18px;
}
```

```css
.shopping-cart-ctn {
      display: flex;
    flex-direction: column;
}

.cart-product-ctn img {
      height: 200px;
      width: 300px;
      object-fit: cover;
}

.cart-product-ctn {
      padding: 20px 0 20px 80px;
      display: flex;
    flex-direction: column;
}

.credentials-div {
      padding: 20px;
}

.sum-container {
      display: flex;
      flex-direction: column;
      align-items: center;
      padding: 20px 0;
      border-top: solid 3px grey;
}

.sum-container h4 {
      margin-block-start: 0;
}

.margin-left {
      margin-left: 10px;
}

.newsletter-ctn {
      padding: 20px;
}

@media (max-width: 700px) {
      .cart-product-ctn {
            padding: 20px;
            display: flex;
      flex-direction: column;
      }
}

/*
    ------------------------ Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0
```

```
       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    ------------------------------------------------------------------------------
*/
```

Listing 47: /fruitshop/css/style.css

7.3.3 mainpage.js

This JavaScript file dynamically generates the content that is visible in the body of the online shops index file. This means the products are displayed as well as the functionality to add the products to the shopping cart.

```
var session = request.getSession();

//session.invalidate();

// Import the Connection class from the java.sql package
var DriverManager = Java.type('java.sql.DriverManager');


// Create a connection to the SQLite database
var conn = DriverManager.getConnection("jdbc:sqlite:C://sqlite/shop.db");

// Execute a SELECT query to retrieve all data from table fruit
var stmt = conn.createStatement();
var rs = stmt.executeQuery('SELECT * FROM fruit;');

out.println('<div class="products-grid">');
while (rs.next()) {
      templateProductItem(rs.getString("name"), rs.getString("price"),
      rs.getString("weight"), rs.getString("picture"), rs.getString("fruit_id"));
}
out.println('</div>');

rs.close;
stmt.close;

var selection = request.getParameter('selection');
var quantity = request.getParameter('quantity');
quantity = parseInt(quantity);


////NOT LOGGED IN USER////

// Adjust cart for a user that is not logged in
if (session.getAttribute("logged") == null) {

  // Check if choice and quantity are not null
  if (selection != null && quantity != null) {
    // Create a new cart if it doesn't exist
    if (session.getAttribute("shopping_cart") == null) {
      var cartArray = new Array(100); //creates JavaScript array object
      session.setAttribute("shopping_cart", cartArray);
```

```
    }
    var shopping_cart = session.getAttribute("shopping_cart");
    // Add a new product to the cart or update the quantity of an existing
    //product
    if (shopping_cart[selection] == null) {
      shopping_cart[selection] = quantity;
    } else {
      shopping_cart[selection] = shopping_cart[selection] + quantity;
    }
    // as session is stored on the server it is not required
    //to use setAttribute again, cahnges are persistent
    // Close the connection and refresh the page
    conn.close();
    response.sendRedirect(request.getRequestURI());
  }
}



////LOGGED IN USER////


// Adjust cart for a logged in user (SAVED IN THE DATABASE)
if (session.getAttribute("logged") != null) {

  // Check if selection and quantity are not null
  if (selection != null && quantity != null) {
    // Check if the product already exists in the shopping cart
    var qry = "SELECT quantity from shopping_cart where customer_id = ? " +
    " and fruit_id = ?";
    var prepstmt = conn.prepareStatement(qry);
    prepstmt.setInt(1, session.getAttribute("logged"));
    prepstmt.setInt(2, selection);
    var rs = prepstmt.executeQuery();
    var cartQuantity = 0;
    while (rs.next()) {
      cartQuantity = rs.getString("quantity");
    }

    cartQuantity = parseInt(cartQuantity);

    rs.close();
    prepstmt.close();

    // Update the shopping cart
    qry = "INSERT INTO shopping_cart (fruit_id, customer_id, quantity) " +
    " VALUES (?,?,?) ON CONFLICT (fruit_id, customer_id) " +
    "DO UPDATE SET quantity = ?";
      prepstmt = conn.prepareStatement(qry);
      prepstmt.setInt(1, selection);
      prepstmt.setInt(2, session.getAttribute('logged'));
      prepstmt.setInt(3, quantity);
      prepstmt.setInt(4, cartQuantity + quantity);
      prepstmt.executeUpdate();
      prepstmt.close();
    conn.close();
```

```
    // Refresh the page
    response.sendRedirect(request.getRequestURI());
  }
}
conn.close();



//// HTML TEMPLATE ////
function templateProductItem(name, price, weight, picture, fruit_id) {
      out.println('<div class="product-container">' +
                  '<h2>' + name + '</h2>' +
                 '<img src="' + picture + '" height="300px" width="300px" />' +
                      '<p>Price: ' + price + ' Euro</p>' +
                 '<p>Weight: ' + weight + 'Kg</p>' +
                  '<form name="selection" method="post">' +
                 '<input type="hidden" name="selection" value="' +
                 fruit_id + '">' +
                   '<select name="quantity">' +
                        '<option value="1">1</option>' +
                        '<option value="2">2</option>' +
                        '<option value="3">3</option>' +
                              '<option value="4">4</option>' +
                              '<option value="5">5</option>' +
                    '</select>' +
             '<input type="submit" style="cursor: pointer;" value="Buy">' +
                   '</form>' +
                   '</div>');
}

/*
    ------------------------ Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    ------------------------------------------------------------------------------
*/
```

Listing 48: /fruitshop/code/mainpage.js

7.3.4 userheader.js

This JavaScript code is generating the content displayed in the header of the main page. It creates links to several applications like login and sign-up. Furthermore, the number of products in the cart is shown next the shopping cart link.

```
var session = request.getSession();
```

```
var shopping_cart;
var shopping_cart_quantity;

//// HEADER FOR A USER NOT LOGGED IN ////
if (session.getAttribute('logged') == null) {
	out.println('<div class="userheader-child">');
	out.println('<a href="productlist.jsp" class="link-btn">Productlist</a>');
	out.println(templateLoginButton());
	out.println(templateSignUpButton());
	out.println('</div>');

	shopping_cart_quantity = 0;
	if (session.getAttribute('shopping_cart') != null) {
		shopping_cart = session.getAttribute('shopping_cart');
		//get the number of products hel in the shopping cart

		// Iterate through the shopping cart array
		for (var i = 0; i < shopping_cart.length; i++) {
			// Check if the element at the current index is not null
			if (shopping_cart[i] != null) {
			// Add the element's value to the shopping cart quantity
			shopping_cart_quantity += shopping_cart[i];
			}
		}
	}

}


//// HEADER FOR A LOGGED IN USER ///
if (session.getAttribute('logged') != null) {
	// Import the DriverManager class from the java.sql package
	var DriverManager = Java.type('java.sql.DriverManager');

	// Create a connection to the SQLite database
	var conn = DriverManager.getConnection("jdbc:sqlite:C://sqlite/shop.db");

	var qry ="SELECT username FROM customer WHERE customer_id = ?;";
	var prepstmt = conn.prepareStatement(qry);
	prepstmt.setInt(1,session.getAttribute("logged"));
	var rs = prepstmt.executeQuery();

	out.println('<div class="userheader-child">');
	out.println('<a href="productlist.jsp" class="link-btn">Productlist</a>');
	out.println('<div class="user_container">');
	while (rs.next()) {
	    out.println('Hello, ' + rs.getString("username"));
	}
	out.println(templateLogoutButton());
	out.println('</div>');
	out.println('</div>');
	rs.close();
	prepstmt.close();

	var qry = "SELECT SUM(quantity) as sum FROM shopping_cart " +
	"WHERE customer_id = ?";
	var prepstmt = conn.prepareStatement(qry);
	prepstmt.setInt(1, session.getAttribute("logged"));
	var rs = prepstmt.executeQuery();
```

```
        while (rs.next()) {
                shopping_cart_quantity = rs.getInt("sum");
        }
        rs.close();
        prepstmt.close();
        conn.close();

        if (shopping_cart_quantity == null) {
                shopping_cart_quantity = 0;
        }
}

//// CHECKOUT BUTTON, DISPLAY NUMBER OF ITEMS ////

if (shopping_cart_quantity == 0) {
        shopping_cart_quantity == 'None';
}

out.println(templateShoppingCart());


////HTML TEMPLATES////

//login button
function templateLoginButton() {
        return '<a href="login.jsp" class="link-btn">Login</a>'
}

//signup button
function templateSignUpButton() {
        return '<a href="signup.jsp" class="link-btn">Sign Up</a>'
}

//logout button
function templateLogoutButton() {
        return '<form action="logout.jsp" method="post" style="float:left;">' +
                        '<input type="submit" class="submit-btn" value="Logout"' +
                        'class="button">' +
                        '</form>';
}

//goto shoppingCart
function templateShoppingCart() { //maybe change to link
        return '<div class="shopping-cart-userheader" style="float:right;">' +
                        '<span>Items in Cart: ' + shopping_cart_quantity + '</span>' +
                          '<form action="shopping_cart.jsp" method="post">' +
                              '<input type="submit" class="submit-btn"' +
                              'value="View Cart" class="button">' +
                        '</form>' +
                '</div>';
}

/*
    ----------------------- Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
```

```
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    ------------------------------------------------------------------------
*/
```

Listing 49: /fruitshop/code/userheader.js

7.3.5 productlist.jsp

This JavaScript code displays a list of all products which are stored in the database. The price is displayed next to the name of the product.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="WEB-INF/script-jsr223.tld" prefix="s" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Fruitshop</title>
<link rel="stylesheet" href="css/style.css">
</head>
<header>

<h1>Productlist</h1>

<a href="index.jsp" class="link-btn" style="float: left">Back to Main Page</a>

</header>
<body>
<s:script type="javascript">
// Import the DriverManager class from the java.sql package
var DriverManager = Java.type('java.sql.DriverManager');

// Create a connection to the SQLite database
var conn = DriverManager.getConnection("jdbc:sqlite:C://sqlite/shop.db");

// Execute a SELECT query to retrieve all data from table fruit
var stmt = conn.createStatement();
var rs = stmt.executeQuery('SELECT * FROM fruit;');

out.println('<ul>');
while (rs.next()) { //iterate through all rows in the table
      out.println('<li>' + rs.getString('name') + ': ' + rs.getString('price') +
    '€</li>');
};
out.println('</ul>')
</s:script>

</body>
```

```
</html>

<%--
    ----------------------- Apache Version 2.0 license -----------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    -------------------------------------------------------------------------
 --%>
```

Listing 50: /fruitshop/productlist.jsp

7.3.6 signup.jsp

The code inside this JSP is responsible to link to the external JavaScript file which handles the registration. Furthermore, the form in which a visitor needs to enter the credentials to sign up is displayed.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="WEB-INF/script-jsr223.tld" prefix="s" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<link rel="stylesheet" href="css/style.css">
<title>Fruitshop</title>
</head>
<header>

<h1>Please create a new Account if you do not have one</h1>


<a href="index.jsp" class="link-btn">Back to Main Page</a>


</header>
<body>

<div class="credentials-div">
      <s:script type="javascript" src="code/create_user.js"
      cacheSrc="false" throwException="true"/>

      <!-- Stays on the same Page (no action) -->
      <form method="post" style="margin-top:20px">
            <label for="username">Please enter your E-Mail:</label><br>
            <input type="email" name="username" required><br>
```

```
            <label for="pwd1">Please enter a safe Password:</label><br>
            <input type="password" name="pwd1" required><br>
            <label for="pwd2">Please repeat your chosen password:</label><br>
            <input type="password" name="pwd2" required><br>
            <label for="newsletter">Please check if you want to receive
            our newsletter and promotional emails:</label><br>
            <input type="checkbox" name="newsletter" value="1"><br>
            <input type="submit" style="cursor: pointer;" value="Create Account">
        </form>
</div>

</body>
</html>

<%--
    ----------------------- Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    ------------------------------------------------------------------------------
 --%>
```

Listing 51: /fruitshop/signup.jsp

7.3.7 create_user.js

This JavaScript code has the purpose to create a new user. Therefor, it is checked if a user with the same username is already existing. If not, the credentials entered in the form are inserted into the database table *customer*. However, one of the most important parts of this script is to use a hashing algorithm to safely store the password. The algorithm used in this case is Bcrypt.

```
username = request.getParameter("username");
pwd1 = request.getParameter("pwd1");
pwd2 = request.getParameter("pwd2");

if (pwd1 != pwd2) {
      out.println('<p class="warning">Your entered Passwords do not match, ' +
      'please try again!</p>');
}

if (username !== null && pwd1 !== null && pwd1 === pwd2) {
    insertIntoDB();
}
```

```
// code written in a function to be able to exit with return
function insertIntoDB() {
      // Import the Connection class from the java.sql package
      var Connection = Java.type("java.sql.Connection");
      var DriverManager = Java.type('java.sql.DriverManager');
      var Statement = Java.type('java.sql.Statement');

      // Create a connection to the SQLite database
      var conn = DriverManager.getConnection("jdbc:sqlite:C://sqlite/shop.db");

      var prepstmt = conn.prepareStatement("SELECT EXISTS (SELECT 1 FROM " +
      "customer WHERE username = ?)"); //returns 1 if row exists 0 if not
      prepstmt.setString(1, username);
      var rs = prepstmt.executeQuery();  // check if user already exists
    while (rs.next()) {
        if (rs.getBoolean(1)) {
            out.print('<p class="warning">User already exists!</p>');
            out.println(templateLoginButton());
            rs.close();
            prepstmt.close();
            conn.close();
            return;  // stop the program
      }
    }
    rs.close();
    prepstmt.close();

      var bcrypt = Java.type("org.mindrot.jbcrypt.BCrypt");

      // Generate a new salt, cost of the later hashpw will be 12
      var salt = bcrypt.gensalt(12);

      // Hash the password with the salt
      var hashedPassword = bcrypt.hashpw(pwd1, salt);

    prepstmt = conn.prepareStatement("INSERT INTO customer " +
    "(username, password) VALUES (?,?)");
    prepstmt.setString(1, username);
    prepstmt.setString(2, hashedPassword);
    prepstmt.executeUpdate();  // add new user to database
    prepstmt.close();

    if (request.getParameter("newsletter") == 1) {
        prepstmt = conn.prepareStatement("UPDATE customer SET " +
        "receives_mail=1 WHERE username=?");
        prepstmt.setString(1, username);
        prepstmt.executeUpdate();  // sign the user up for e-mails
        prepstmt.close();
    }
    conn.close();

    out.println('<p class="success">Account successfully created!</p>');
    out.println(templateLoginButton());
}

//signup button
function templateLoginButton() {
      return '<a href="login.jsp" class="link-btn">Login</a>'
}
```

```
/*
    ----------------------- Apache Version 2.0 license -----------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    --------------------------------------------------------------------------
*/
```

Listing 52: /fruitshop/code/create_user.js

7.3.8 login.jsp

This JSP file displays a form to log in for customers of the shop. However, to handle the login process it accesses the external script login.js.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="WEB-INF/script-jsr223.tld" prefix="s" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<link rel="stylesheet" href="css/style.css">
<title>Fruitshop</title>
</head>
<header>

<h1>Login to your account!</h1>

<a href="index.jsp" class="link-btn">Back to Main Page</a>

<a href="signup.jsp" class="link-btn">Sign Up</a>

</header>
<body>

<div class="credentials-div">
      <s:script type="javascript" src="code/login.js" cacheSrc="false"
      throwException="true" />

      <span class="login-form-headline">Enter your credentials:</span>
      <form method="post" style="margin-top: 12px">
            <label for="username">E-mail:</label><br>
            <input type="email" name="username" required><br>
            <label for="pwd">Password:</label><br>
            <input type="password" name="pwd" required><br>
            <input type="submit" style="cursor: pointer;" value="Submit">
```

```
      </form>
</div>

</body>
</html>

<%--
    ----------------------- Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    -------------------------------------------------------------------------
 --%>
```

Listing 53: /fruitshop/login.jsp

7.3.9 login.js

The code in this file is responsible for the process of checking if the credentials entered are matching with those stored in the database. If this is the case the user is logged in and the *customer_id* is stored in the session. Therefore, other applications in the online shop can determine who is currently logged in. Furthermore, the script transfers the products in the shopping cart of the session to the shopping cart in the database.

```
var session = request.getSession();

var username = request.getParameter("username");
var pwd = request.getParameter("pwd");

// Import the Connection class from the java.sql package
var DriverManager = Java.type('java.sql.DriverManager');

// Create a connection to the SQLite database
var conn = DriverManager.getConnection("jdbc:sqlite:C://sqlite/shop.db");

if (username != null && pwd != null) {
//else the code is executed each time you load the page
      //search for customer in database (by username)
      var prepstmt = conn.prepareStatement("SELECT password, customer_id " +
      "FROM customer WHERE username=?;");
      prepstmt.setString(1, username);
      var rs = prepstmt.executeQuery();

      if (rs.next()) {
            var id = rs.getString("customer_id");
```

```
        var hashedPassword = rs.getString("password");
} else {
        templateWrongCredentials(); //if username is not in database
}
rs.close();
prepstmt.close();

var bcrypt = Java.type("org.mindrot.jbcrypt.BCrypt");
//check if password is correct
if (bcrypt.checkpw(pwd, hashedPassword)) {
  //store the login status is the session
  session.setAttribute('logged', id)

  // Transfer session shopping cart,
  //only if shopping cart exists in session
  if (session.getAttribute('shopping_cart') != null) {
    var shopping_cart = session.getAttribute('shopping_cart');
    var user_id = session.getAttribute('logged');
    var quantity;
    var item;

 //iterate through the shopping_cart that was stored in the session
    for (var selection in shopping_cart) {
      quantity = shopping_cart[selection];
      quantity = parseInt(quantity);
      item = selection;

      // Create the prepared statement and
      //set the parameters to check if prdouct is already in cart
      var qry = "SELECT quantity FROM shopping_cart " +
      "WHERE customer_id = ? AND fruit_id = ?";
      prepstmt = conn.prepareStatement(qry);
      prepstmt.setInt(1, id);
      prepstmt.setInt(2, item);

        // Execute the query and get the result set
        rs = prepstmt.executeQuery();

        var shopping_cart_quantity = 0;
        while (rs.next()) { //can only be one time
          shopping_cart_quantity = rs.getString('quantity');
            shopping_cart_quantity = parseInt(shopping_cart_quantity);
        }
        rs.close();
        prepstmt.close();

        // Create the prepared statement and set the parameters
        var qry = "INSERT INTO shopping_cart " +
        "(customer_id, fruit_id, quantity) VALUES (?,?,?) " +
        "ON CONFLICT (customer_id,fruit_id) DO UPDATE SET quantity = ?";
        prepstmt = conn.prepareStatement(qry);
        prepstmt.setInt(1, user_id);
        prepstmt.setInt(2, item);
        prepstmt.setInt(3, quantity);
        prepstmt.setInt(4, quantity + shopping_cart_quantity);

        // Execute the update
        prepstmt.executeUpdate();
      }
```

```
            conn.close();
            session.removeAttribute('shopping_cart')
            //shopping_cart is now in database therefore not needed in session
        }
        response.sendRedirect('index.jsp')
    } else {
            templateWrongCredentials(); //if passwords do not match
            rs.close();
            prepstmt.close();
            conn.close();
    }
}

function templateWrongCredentials() {
    out.println('<p class="warning">Wrong credentials, please try again!</p>')
}

/*
    ------------------------ Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    ------------------------------------------------------------------------------
*/
```

Listing 54: /fruitshop/code/login.js

7.3.10 logout.jsp

This JSP file gets executed when a logged in customer, clicks on the logout button in the main page. Once a user gets redirected to this page the current session is cleared by calling the method *invalidate()*.

```
<%@ page session="true" pageEncoding="UTF-8"
contentType="text/html; charset=UTF-8" %>
<%@ taglib uri="WEB-INF/script-jsr223.tld" prefix="s" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<link rel="stylesheet" href="css/style.css">
<title>treeshop | signup</title>
</head>
<header>

<h1>You have been successfully logged out</h1>
```

```
<a href="index.jsp" class="link-btn">Back to Main Page</a>

<a href="signup.jsp" class="link-btn">Sign Up</a>

<a href="login.jsp" class="link-btn">Login</a>

</header>
<body>

<s:script type="javascript">
var session = request.getSession();

//clear session
session.invalidate()
</s:script>

</body>
</html>

<%--
    ----------------------- Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    ------------------------------------------------------------------------------
 --%>
```

Listing 55: /fruitshop/logout.jsp

7.3.11 shopping_cart.jsp

This JSP file is responsible for displaying the products inside the shopping cart. However, the content on this page needs to be dynamically generated. That is why the only code inside the body tag accesses an external script to produce this content.

```
<%@ page language="java" contentType="text/html;
charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib uri="WEB-INF/script-jsr223.tld" prefix="s" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<link rel="stylesheet" href="css/style.css">
<title>Fruitshop</title>
</head>
<header>
```

```
<h1>Your Shopping Cart!</h1>

<a href="index.jsp" class="link-btn">Back to Main Page</a>

</header>
<body>

<s:script type="javascript" src="code/shopping_cart.js"
cacheSrc="false" throwException="true" />

</body>
</html>

<%--
    ------------------------ Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    ------------------------------------------------------------------------------
 --%>
```

Listing 56: /fruitshop/shopping_cart.jsp

7.3.12 shopping_cart.js

The code inside this JavaScript file is accessed in shopping_cart.jsp to display the products that are currently in the shopping cart. The products are either retrieved from the database or from the session depending on the login status of the customer. Furthermore, the file is responsible for the functionality to increase or decrease the quantity of a product and to be able to completely remove a product from the shopping cart.

```
var session = request.getSession();
var conn;


// Shopping cart page if user is not loggged in
//but has values in the shopping cart
if (session.getAttribute('logged') == null &&
       session.getAttribute('shopping_cart') != null) {
       createConnectionToDatabase();

       var shopping_cart = session.getAttribute('shopping_cart');
       var totalPrice = 0;

       var i = 0;
```

```
        out.println('<div class="shopping-cart-ctn">')
        for (var selection in shopping_cart) {
                var quantity = shopping_cart[selection];
                var quantity = parseInt(quantity);
                itemID = selection;

                var qry = "SELECT * FROM fruit WHERE fruit_id=" + itemID + ";";
                var stmt = conn.createStatement();
                var rs = stmt.executeQuery(qry);

                var bgColor = getBackgroundColor(i);

                while (rs.next()) {
                        totalPrice = totalPrice + (rs.getInt('price') * quantity);
                        templateProductContainer(rs.getString('name'),
                        rs.getString('picture'), rs.getString('weight'),
                        rs.getString('price'), quantity, itemID, bgColor);
                }
                i++;
                rs.close();
                stmt.close();
        }

        templateSumContainer(totalPrice);

                //Functionality of the buttons add, reduce and delete
        if (request.getParameter("actn") == "+") {
                var fruit_id = request.getParameter("fruit_id");
                // increase quantity by 1
                var quantity = parseInt(request.getParameter("quantity")) + 1;
                shopping_cart[fruit_id] = quantity;
                response.sendRedirect(request.getRequestURI());
        }

        if (request.getParameter("actn") == "-") {
          var fruit_id = request.getParameter("fruit_id");
          // reduce quantity by 1
          var quantity = parseInt(request.getParameter("quantity")) - 1;
          shopping_cart[fruit_id] = quantity;
          if (quantity <= 0) {
                // delete product from cart if quantity goes below 1
            delete shopping_cart[id];
          }
          response.sendRedirect(request.getRequestURI());
        }

        if (request.getParameter("actn") == "del") {
          var fruit_id = request.getParameter("fruit_id");
          delete shopping_cart[fruit_id]; // delete product from cart
          response.sendRedirect(request.getRequestURI());
        }
}


//Shopping_cart page if user is logged in
if (session.getAttribute('logged') != null) {
        createConnectionToDatabase();

        var totalprice = 0;
```

```
   var qry = "SELECT * FROM shopping_cart INNER JOIN fruit USING(fruit_id) " +
                  "WHERE customer_id = ?;";
var prepstmt = conn.prepareStatement(qry);
prepstmt.setInt(1, session.getAttribute("logged"));
var rs = prepstmt.executeQuery();    // get data for all products in cart

   var i = 0;
out.println('<div class="shopping-cart-ctn">');
while (rs.next()) {
    totalprice = totalprice + (rs.getInt("price") * rs.getInt("quantity"));
    var bgColor = getBackgroundColor(i);
    templateProductContainer(rs.getString("name"),
    rs.getString("picture"), rs.getString("weight"),
    rs.getInt("price"), rs.getInt("quantity"), rs.getString("fruit_id"),
    bgColor);
   i++;
}
rs.close();
prepstmt.close();
out.println('</div>');

templateSumContainer(totalprice);


   ////Functionality of the buttons add, reduce and delete

    // increase quantity in database cart
   if (request.getParameter("actn") == "+") {
       var fruit_id = request.getParameter("fruit_id");
       var quantity = request.getParameter("quantity");
       quantity = parseInt(quantity);
       quantity = quantity + 1;

       var qry = "UPDATE shopping_cart SET quantity = ? " +
       "WHERE customer_id = ? AND fruit_id = ?;";
       var prepstmt = conn.prepareStatement(qry);
       prepstmt.setInt(1, quantity);
       prepstmt.setInt(2, session.getAttribute("logged"));
       prepstmt.setInt(3, fruit_id);
       prepstmt.executeUpdate();
       prepstmt.close();
       conn.close();

       response.sendRedirect(request.getRequestURI());  // refresh page
   }

   if (request.getParameter("actn") == "-") {
       var fruit_id = request.getParameter("fruit_id");
       var quantity = request.getParameter("quantity") - 1;
       // delete product from cart if quantity goes below 1
       if (quantity <= 0) {
           var qry = "DELETE FROM shopping_cart WHERE customer_id = ? " +
                     "AND AND fruit_id = ?;";
           var prepstmt = conn.prepareStatement(qry);
           prepstmt.setInt(1, session.getAttribute("logged"));
           prepstmt.setInt(2, fruit_id);
           prepstmt.executeUpdate();
```

```
            prepstmt.close();
        } else { // reduce quantity in database cart
            var qry = "UPDATE shopping_cart SET quantity = ? " +
                      "WHERE customer_id = ? AND fruit_id = ?;";
            var prepstmt = conn.prepareStatement(qry);
            prepstmt.setInt(1, quantity);
            prepstmt.setInt(2, session.getAttribute("logged"));
            prepstmt.setInt(3, fruit_id);
            prepstmt.executeUpdate();
            prepstmt.close();
        }
        conn.close();
        response.sendRedirect(request.getRequestURI());  // refresh page
    }

    // delete product from cart
    if (request.getParameter("actn") == "del") {
        var fruit_id = request.getParameter("fruit_id");

        var qry = "DELETE FROM shopping_cart WHERE customer_id = ? " +
                  "AND fruit_id = ?;";
        var prepstmt = conn.prepareStatement(qry);
        prepstmt.setInt(1, session.getAttribute("logged"));
        prepstmt.setInt(2, fruit_id);
        prepstmt.executeUpdate();
        prepstmt.close();
        conn.close();

        response.sendRedirect(request.getRequestURI());  // refresh page
    }
    conn.close();
}



function getBackgroundColor(i) {
    if ((i % 2) == 0) {
        var backgroundColor = '#F9F9F9';
    } else {
        var backgroundColor ='#F1F1F1';
    }
    return backgroundColor;
}



//// CONNECT TO DATABASE ////

function createConnectionToDatabase() {
    // Import the Connection class from the java.sql package
    var DriverManager = Java.type('java.sql.DriverManager');

    // Create a connection to the SQLite database
    conn = DriverManager.getConnection("jdbc:sqlite:C://sqlite/shop.db");
}


//// HTML TEMPLATES ////
```

```
function templateProductContainer(name, img, weight, price,
quantity, itemID, bgColor) {
  out.println('<div class="cart-product-ctn" style="background-color:' +
               bgColor + '">' +
              '<h2>' + name + '</h2>' +
              '<div style="display:flex; gap:40px;">' +
              '<img src="' + img + '"/>' +
                   '<div>' +
                         '<div>' +
                               '<p>Weight: ' + weight + ' Kg</p>' +
                               '<p>Price per item: ' + price + ' Euro</p>' +
                               '<p>Quantity ordered: ' + quantity + '</p>' +
                               '<p>Subtotal: ' + price * quantity + ' Euro</p>' +
                         '</div>' +
                         '<form method="post">' +
                               '<input type="hidden" name="fruit_id" value="' +
                               itemID + '">' +
                               '<input type="hidden" name="quantity" value="' +
                               quantity + '">' +
                               '<input type="submit" style="cursor: pointer;" ' +
                               'name="actn" value="+">' +
                               '<input type="submit" style="cursor: pointer;" ' +
                               'name="actn" value="-">' +
                               '<input type="submit" style="cursor: pointer;" ' +
                               'name="actn" value="del">' +
                         '</form>' +
                   '</div>' +
             '</div>' +
             '</div>');
}

function templateSumContainer(totalPrice) {
      out.println('<div class="sum-container">' +
                    '<h4>Your total is: ' + totalPrice + ' Euro</h4>' +
                    '<form action="checkout.jsp" method="POST">' +
                        '<input type="submit" style="cursor: pointer;" ' +
                        'value="Checkout" class="button">' +
                        '</form>' +
                  '</div>');
}

/*
    ------------------------ Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    ------------------------------------------------------------------------------
*/
```

Listing 57: /fruitshop/code/shopping_cart.js

7.4 Advanced Examples

The applications in this part are more advanced than the examples discussed until now. The programs are developed to ease the work of admins of the shop. Therefore, the first program that has been developed can be used to create new products. However, the products require an image, thus a file upload needs to be implemented. The second example is an application where admins can easily send advertisement emails to customers. Though what both programs have in common is that the rely on the servlet technology.

7.4.1 index.html

The index.html files task is to link to the advanced applications. The file is automatically opened by default once the admin folder is opened in the browser. Important here is that all files regarding the advanced examples are located inside the *admin* folder. Thus, requesting files outside this folder requires two dots in front of the path (e.g. *../css/style.css*)

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<link rel="stylesheet" href="../css/style.css">
<title>fruitshop_admin</title>
</head>
<header>

<h1>Fruitshop Administration Page</h1>

<a href="/fruitshop" class="link-btn">Main Page</a>

</header>
<body>
<br>
<a class="margin-left" href="addproducts.html">Add new products</a><br>
<a class="margin-left" href="newsletter.jsp">Send promotional E-Mails</a>

</body>
</html>

<!--
    ------------------------ Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0
```

```
       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    ----------------------------------------------------------------------------
 -->
```

Listing 58: /fruitshop/admin/index.html

7.4.2 addproducts.html

Admins can enter the specifications of a new product in the form displayed by this program. The data entered will be sent to the servlet uploader and inserted in the database. However, as the servlet mailer has been configured to use the code inside mailer.jsp, the action attribute can redirect to the servlet uploader (no *.jsp* needed). Furthermore, to enable uploading images the attribute *enctype="multipart/form-data"* is necessary.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<link rel="stylesheet" href="../css/style.css">
<title>fruitshop_admin</title>
</head>
<header>

<h1>Create a New Product</h1>

<a href="/fruitshop/admin" class="link-btn">Admin Starting Page</a>

</header>
<body>

<form action="uploader"  style="margin: 10px"
enctype="multipart/form-data" method="post">
  <label for="name">Name:</label><br>
  <input type="text" name="name" required><br><br>
  <label for="weight">Weight:</label><br>
  <input type="text" name="weight" required><br><br>
  <label for="price">Price:</label><br>
  <input type="number" name="price" required><br><br>
  <input type="file" name="file" required>
  <input type="submit" value="Create Product">
</form>

</body>
</html>


<!--
    ----------------------- Apache Version 2.0 license -----------------------
```

```
        Copyright 2023 Simon Besenbäck

        Licensed under the Apache License, Version 2.0 (the "License");
        you may not use this file except in compliance with the License.
        You may obtain a copy of the License at

            http://www.apache.org/licenses/LICENSE-2.0

        Unless required by applicable law or agreed to in writing, software
        distributed under the License is distributed on an "AS IS" BASIS,
        WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
        See the License for the specific language governing permissions and
        limitations under the License.
    ------------------------------------------------------------------------
 -->
```

Listing 59: /fruitshop/admin/addproducts.html

7.4.3 uploader.jsp

The servlet uploader uses the code written in uploader.jsp. The servlets task is to upload the image to the database. This is done by the instruction *request.getPart("file").write(filename)*. Furthermore, the specifications of the newly created product are inserted in the database.

```
<%@ page session="false" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="/WEB-INF/script-jsr223.tld" prefix="s" %>

<s:script type="javascript">
var session = request.getSession();

var name = request.getParameter('name');
var weight = request.getParameter('weight');
var price = request.getParameter('price');

// Import the Connection class from the java.sql package
var DriverManager = Java.type('java.sql.DriverManager');

// Create a connection to the SQLite database
var conn = DriverManager.getConnection("jdbc:sqlite:C://sqlite/shop.db");


if (checkIfProductExists()) {
       var filename = name + ".jpg";
       var location = "/files/" + filename;
       request.getPart("file").write(filename);

       var prepstmt = conn.prepareStatement("INSERT INTO fruit " +
       "(name, price, weight, picture) VALUES (?,?,?,?)");
       prepstmt.setString(1, name);
       prepstmt.setString(2, price);
       prepstmt.setString(3, weight);
       prepstmt.setString(4, location);
       prepstmt.executeUpdate();
       out.println(openHTML());
```

```
        out.println('<p class="success margin-left">Product has been ' +
        'created successfully!</p>');
        out.println(closeHTML());
        prepstmt.close();
        conn.close();
}

function checkIfProductExists() {
        var prepstmt = conn.prepareStatement("SELECT * FROM fruit WHERE name = ?");

        prepstmt.setString(1, name);
        var rs = prepstmt.executeQuery();

        while (rs.next()) {
                out.println(openHTML());
        out.println('<p class="warning margin-left">Entry already exists! ' +
          'Please try again</p>');
        out.println(closeHTML());
        rs.close();
        prepstmt.close();
        conn.close();
                return false; // Product exists
        }
        rs.close();
        prepstmt.close();
        return true; //Product does not exist
}



//// HTML TEMPLATES ////

function openHTML() {
        return '<!DOCTYPE html>' +
                  '<html>' +
                '<head>' +
                '<meta charset="UTF-8" />' +
                '<link rel="stylesheet" href="../css/style.css">' +
                '<title>fruitshop_admin</title>' +
                '<header>' +
                        '<h1>Status of Product Creation</h1>' +

                        '<a href="/fruitshop/admin" class="link-btn">' +
                        'Admin Starting Page</a>' +
                        '<a href="/fruitshop" class="link-btn">Main Page</a>' +

                '</header>' +
                '<body>'
}

function closeHTML() {
        return '</body>' +
                  '</html>'
}
</s:script>



<%--
```

```
    ------------------------ Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    ------------------------------------------------------------------------------
 --%>
```

Listing 60: /fruitshop/admin/code/uploader.jsp

7.4.4 newsletter.jsp

To send newsletter emails to a customer the user can visit the newsletter.jsp page. Here a form will be displayed where one or more products can be selected and then are sent in the email. By submitting the form the data and the user are sent to the *mailer* servlet.

```
<%@ page session="false" pageEncoding="UTF-8"
contentType="text/html; charset=UTF-8" %>
<%@ taglib uri="/WEB-INF/script-jsr223.tld" prefix="s" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<link rel="stylesheet" href="../css/style.css">
<title>fruitshop_admin</title>
</head>
<header>

<h1>Create a Newsletter</h1>

<a href="/fruitshop/admin" class="link-btn">Admin Starting Page</a>

</header>
<body>

<s:script type="javascript">
var DriverManager = Java.type('java.sql.DriverManager');

// Create a connection to the SQLite database
var conn = DriverManager.getConnection("jdbc:sqlite:C://sqlite/shop.db");

var stmt = conn.createStatement();
var qry = "SELECT * FROM customer WHERE receives_mail = '1';";
var rs = stmt.executeQuery(qry); // check how many people are subscribed

var count = 0;
```

```
while (rs.next()) {
    count++;
}
rs.close();
stmt.close();

stmt = conn.createStatement();
qry = "SELECT name FROM fruit;";
rs = stmt.executeQuery(qry); // get the names of all available products

out.println('<div class="newsletter-ctn">')
out.println('<form action="mailer" method="post">');
while (rs.next()) {
    out.println('<br>');
    out.println('<li><label for="choice">' + rs.getString("name") + '</label>');
    out.println('<input type="checkbox" name="choice" value="' +
    rs.getString("name") + '"></li>');
}
rs.close();
stmt.close();
conn.close();
out.println('<input type="submit" style="margin-top: 20px;" ' +
'value="Send Newsletter to ' + count + ' Receivers">');
out.println('</form>');
out.println('</div>');
</s:script>

</body>
</html>

<%--
    ----------------------- Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    ------------------------------------------------------------------------------
 --%>
```

Listing 61: /fruitshop/admin/newsletter.jsp

7.4.5 mailer.jsp

The code inside this JSP gets executed when the user submits the form present in newsletter.jsp. This is because the servlet *mailer* uses the code inside this mailer.jsp file. The program is responsible for creating one email for each customer who is willing to receive an email. The content of the email is written in HTML. Therefore, the product

can be displayed in the email. The MailHog application was used to be able to analyze the sent emails. The emails are sent to port 1025. This is because MailHog's SMTP server runs under this port. If MailHog is up and running a developer can look at the sent emails under the URL localhost:8025.

```
<%@ page session="false" pageEncoding="UTF-8" contentType="text/html; charset=UTF-
8" %>
<%@ taglib uri="/WEB-INF/script-jsr223.tld" prefix="s" %>

<s:script type="javascript">
var choices = request.getParameterValues("choice");

// make sure at least one product is selected
if (choices !== null) {
    var DriverManager = Java.type('java.sql.DriverManager');

    // Create a connection to the SQLite database
    var conn = DriverManager.getConnection("jdbc:sqlite:C://sqlite/shop.db");

    var choice = '';
    for (var i = 0; i < choices.length; i++) {
        // append all product names to a string
        choice += "'" + choices[i] + "',";
    }
    // remove the string's last comma
    choice = choice.substring(0, choice.length - 1);

    var stmt1 = conn.createStatement();
    var qry1 = "SELECT * FROM customer WHERE receives_mail = 1;";
    // select all customers who wish to receive the newsletter
    var customers = stmt1.executeQuery(qry1);
    var emailCount = 0;

    while (customers.next()) {
      var props = new (Java.type("java.util.Properties"))();
      var session = Java.type("jakarta.mail.Session").getInstance(props);
      var msg = new (Java.type("jakarta.mail.internet.MimeMessage"))(session);

      var sender = new (Java.type("jakarta.mail.internet.InternetAddress")
      )("newsletter@treeshop.com");
      msg.setFrom(sender);

      var receiverAddress = customers.getString("username");
      var receiver = new (Java.type("jakarta.mail.internet.InternetAddress")
       )(receiverAddress);
      var type = Java.type("jakarta.mail.Message$RecipientType").TO;
      msg.addRecipient(type, receiver);


      msg.setSubject("Here Are the Latest Products from treeshop!");

      var stmt2 = conn.createStatement();
      var qry2 = "SELECT * FROM fruit WHERE name IN (" + choice + ")";
      var products = stmt2.executeQuery(qry2);

      var i = 0;
```

```
	var productHTML = [];

	while (products.next()) {
		var line1 = '<div style="float: left; margin-right: 10px;">';
		var line2 = '<h2>' + products.getString("name") + '</h2>';
		var line3 = '<img src="http://localhost:8080' +
		products.getString("picture") + '" height="120" width="150" />';
		var line4 = '<p>Price: ' + products.getString("price") +
		' Euro</p>';
		var line5 = '<p>Weight: ' + products.getString("weight") +
		' KG</p>';
		var line6 = '</div>';

		productHTML.push(line1 + line2 + line3 + line4 + line5 + line6);
		i++;
		}
	products.close();
	stmt2.close();

	var text = '<html><head><meta charset="UTF-8" /></head><header>';
	text += '<h1><a href="http://localhost:8080/fruitshop">' +
	'Vist fruitshop</a></h1>';
	text += '<h4><a href="http://localhost:8080/fruitshop/admin/' +
	'unsubscribe.jsp?unsub=' + receiverAddress +
	'">Click Here to Unsubscribe</a></h4>';

	for (var count = 0; count < i; count++) {
		//i is one more then the index in the array
		text += productHTML[count];
	}

	text += '</body></html>';
	msg.setContent(text, "text/html");

	var transport = session.getTransport("smtp");
	transport.connect("localhost", 1025, "username", "pw");
	transport.sendMessage(msg, msg.getRecipients(type));

	emailCount++;
	}
	customers.close();
	stmt1.close();

	out.println(openHTML());
	out.println('<p class="success">Sending emails has been successfull!</p>');
	out.println(closeHTML());
} else {
	out.println(openHTML());
	out.println('<p class="warning">At least one product ' +
	'needs to be chosen!</p>');
	out.println(closeHTML());
}



//// HTML TEMPLATES ////

function openHTML() {
```

```
      return '<!DOCTYPE html>' +
                '<html>' +
               '<head>' +
               '<meta charset="UTF-8" />' +
               '<link rel="stylesheet" href="../css/style.css">' +
               '<title>fruitshop_admin</title>' +
               '<header>' +
                     '<h1>Sending a Newsletter</h1>' +

                     '<a href="/fruitshop/admin" class="link-btn">' +
                     'Admin Starting Page</a>' +
                     '<a href="/fruitshop" class="link-btn">Main Page</a>' +

               '</header>' +
               '<body>'
}

function closeHTML() {
      return '</body>' +
                '</html>'
}

</s:script>

<%--
    ----------------------- Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    ------------------------------------------------------------------------------
 --%>
```

Listing 62: /fruitshop/admin/code/mailer.jsp

7.4.6 unsubscribe.jsp

The *unsubscribe* link to get to this page is placed inside the newsletter emails. The purpose of this file is to ask the customer for confirmation. The confirmation is done via a form that redirects the user to the *unsubscriber* servlet and is responsible for forwarding the username of the customer.

```
<%@ page session="false" pageEncoding="UTF-8"
contentType="text/html; charset=UTF-8" %>
<%@ taglib uri="/WEB-INF/script-jsr223.tld" prefix="s" %>
<!DOCTYPE html>
<html>
```

```
<head>
<meta charset="UTF-8" />
<link rel="stylesheet" href="../css/style.css">
<title>Fruitshop</title>
</head>
<header>

<h1>Unsubscribe from our Newsletter:</h1>

<a href="/fruitshop/admin" class="link-btn">Admin Starting Page</a>

</header>
<body>

<s:script type="javascript" throwException="true">

var email = request.getParameter("unsub");

out.println('<p class="margin-left warning">' +
'Are you sure you want to stop receiving e-mails at: ' + email + '</p>');

out.println('<form class="margin-left" action="unsubscriber" method="post">' +
                        '<input type="hidden" name ="unsub" value="' + email +
                        '"><br>' +
                        '<input type="submit" value="Unsubscribe">' +
                    '</form>');
</s:script>

</body>
</html>

<%--
    ----------------------- Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0

       Unless required by applicable law or agreed to in writing, software
       distributed under the License is distributed on an "AS IS" BASIS,
       WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       See the License for the specific language governing permissions and
       limitations under the License.
    ------------------------------------------------------------------------------
 --%>
```

Listing 63: /fruitshop/admin/unsubscribe.jsp

7.4.7 unsubscriber.jsp

The code in this JSP file is accessed by the servlet named *unsubscriber*. The purpose of this servlet is to update the column *receives_mail* in the table customer. The updated value needs to be *0* as this is equivalent to false.

```
<%@ page session="false" pageEncoding="UTF-8"
contentType="text/html; charset=UTF-8" %>
<%@ taglib uri="/WEB-INF/script-jsr223.tld" prefix="s" %>

<s:script type="javascript">
 var DriverManager = Java.type('java.sql.DriverManager');

// Create a connection to the SQLite database
var conn = DriverManager.getConnection("jdbc:sqlite:C://sqlite/shop.db");

email = request.getParameter("unsub");
prepstmt = conn.prepareStatement("UPDATE customer SET receives_mail=0 " +
"WHERE username=?");
prepstmt.setString(1,email);
prepstmt.executeUpdate();
prepstmt.close();
conn.close();

out.println(openHTML());
out.println('<p class="margin-left">' + email +
' has been successfully unsubscribed!</p>');
out.println(closeHTML());

//// HTML TEMPLATES ////

function openHTML() {
       return '<!DOCTYPE html>' +
                 '<html>' +
              '<head>' +
              '<meta charset="UTF-8" />' +
              '<link rel="stylesheet" href="../css/style.css">' +
              '<title>fruitshop_admin</title>' +
              '<header>' +
                    '<h1>Subscription Status:</h1>' +

                    '<a href="/fruitshop/admin" class="link-btn">' +
                      'Admin Starting Page</a>' +
                    '<a href="/fruitshop" class="link-btn">Main Page</a>' +

              '</header>' +
              '<body>'
}

function closeHTML() {
       return '</body>' +
                 '</html>'
}
</s:script>

<%--
    ----------------------- Apache Version 2.0 license -------------------------
       Copyright 2023 Simon Besenbäck

       Licensed under the Apache License, Version 2.0 (the "License");
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at

           http://www.apache.org/licenses/LICENSE-2.0
```

```
      Unless required by applicable law or agreed to in writing, software
      distributed under the License is distributed on an "AS IS" BASIS,
      WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
      See the License for the specific language governing permissions and
      limitations under the License.
    ------------------------------------------------------------------------
--%>
```

Listing 64: /fruitshop/admin/code/unsubscriber.jsp