WIRTSCHAFTSUNIVERSITÄT WIEN
Vienna University of Economics and Business

Bachelor Thesis

Titel of Bachelor Thesis (english)	An Introduction to Web Application Development - Combining Jakarta Server Pages With Programs Written in Scripting Languages
Titel of Bachelor Thesis (german)	Einführung in die Entwicklung von Web Anwendungen
Author (last name, first name):	Lux Dimitry-Janos
Student ID number:	01146180
Degree program:	Bachelor of Science (WU), BSc (WU)
Examiner (degree, first name, last name):	ao. Univ. Prof. Mag. Dr. Rony G. Flatscher

I hereby declare that:

1. I have written this Bachelor 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 Thesis has not been previously presented as an examination paper in this or any other form in Austria or abroad.
3. This Bachelor 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 thesis was written together with

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

Date

Unterschrift

WIRTSCHAFTSUNIVERSITÄT WIEN
Vienna University of Economics and Business

An Introduction to Web Application Development – Combining Jakarta Server Pages With Programs Written in Scripting Languages

Bachelor Thesis by Dimitry-J. Lux

Supervised by ao. Univ. Prof. Mag. Dr. Rony G. Flatscher

Abstract: This thesis aims to communicate all knowledge necessary to enable the reader to develop web applications quickly and efficiently. To achieve this goal, three key tools are used: The Open Object Rexx scripting language, the Bean Scripting Framework for Open Object Rexx and the Apache Tomcat Software. Tag libraries are used to combine these components. After discussing the main technological components, nutshell examples with increasing complexity are used to guide the reader.

Methodology: This thesis commences with a brief literature review of the main technologies used. Given the nature of the topic, most information has been gathered from documentations, tutorials as well as selected scientific papers. These core components were then utilized to create nutshell examples, demonstrating possible implementations.

1. Introduction

The World Wide Web was invented by Sir Tim Berners-Lee at the European Organization for Nuclear Research. In 1990 he created the first web client and server, as well as the specifications for Uniform Resource Identifiers (URI), the Hypertext Transfer Protocol (HTTP) and the Hypertext Markup Language (HTML) [1]. With HTTP being a stateless protocol [2, p. 1], in the beginning web pages were simple, static sources of information. Today users of the internet are used to performing complex tasks all from within their web browser.

While the task of developing such web applications might daunting at first, the tools used in this thesis will allow beginners to quickly create web applications of their own. The Apache Tomcat Software (Tomcat) offers the infrastructure and tools to deploy said web applications, mainly the Jakarta Server Pages (JSP) technology. By interlacing HTML code with the human-oriented [3] Open Object Rexx (ooRexx) programming language, results can be achieved with minimal prior knowledge. The Bean Scripting Framework for ooRexx (BSF4ooRexx) includes Tag Libraries (taglibs), used to accomplish this. In addition, BSF4ooRexx allows the usage of the countless internal and external Java classes all from within ooRexx. The reader will be able to use these tools for web development, with the only limiting factor being one's imagination.

This thesis was written with the ooRexx programming language in mind. While the development of web applications is being covered from the very beginning, basic knowledge of ooRexx and HTML is recommended. Nonetheless, the components used for web application development and deployment support a multitude of existing scripting languages.

To begin with, `2. Technologies` will discuss the underlying technologies of the internet and web applications in theory. From `3. Apache Tomcat` onwards, nutshell examples are used to practically demonstrate the concepts discussed. Put together, these examples will form a functioning shopping web site. In case the reader would like to use any of this code, be it in full or a fragment, the author encourages such usage, in hope it will help.

For a collection of hyperlinks to all the software necessary, please refer to the appendix: `9.1.1. Software required` to begin.

2. Technologies

A Java web application is built upon a Java Runtime Environment provided by a web server and a combination of components such as JSP's, servlets, JavaBeans, and static pages like HTML [4, Sec. 1.2.].

This section will introduce the core technologies used. First the programming languages to create the programs are discussed, followed by the infrastructure and technologies that enable them to be accessed over the internet. Should the reader be familiar with these topics, he might wish to jump directly to: `3. Apache Tomcat`

2.1. System Programming Languages and Scripting Programming Languages

In 1998 Ousterhout predicted that: "*scripting languages will handle many of the programming tasks in the next century better than system programming languages*" [5, p. 23].

"*System programming languages were designed for building data structures and algorithms from scratch, starting from the most primitive computer elements such as words of memory. In contrast, scripting languages are designed for gluing: They assume the existence of a set of powerful components and are intended primarily for connecting components*" [5, p. 23].

System programming languages were designed to abstract from assembly languages to make the development process faster. While assembly languages statements correspond directly to machine instructions, system languages require a compiler which translates the source code into binary instructions. Scripting languages abstract even further, with power and ease of use in mind [5, pp. 23-24].

Scripting languages use interpreters instead of compilers. The translation does not happen all at once, but instruction by instruction [6]. This allows a quicker development process without compile times. Additionally, programmers are more flexible since the applications are programmed at runtime [5, p. 26]. In contrast to system languages, scripting languages are usually kept in source form [7].

"*Scripting languages are higher level than system programming languages in the sense that a single statement does more work on average*" [5, p. 26].

While system programming languages are generally strongly typed, scripting languages do not share this trait. Typing refers to variables being declared a particular type such as integer or string. A strongly typed language offers performance gains, since the compiler only needs to load specific instructions. While potential errors are detected during compile time, errors in scripting languages occur when a value is used. Scripting languages are generally typeless; variables can freely switch data types. This results in the interchangeability of code and data, easing the process of combining different components. Overall, strongly typed languages are more restrictive and less flexible, yet more performant. [5, pp. 24-27].

With the increase of computing power, the performance difference is negligible in most situations. In case of an application where performance is crucial, a system programming language might be the better choice though. This is particularly the case for programs that are slow to change. On the other hand, scripting languages are particularly useful for programs implementing a Graphical user interface, connect through the internet or utility component frameworks like Java Beans [5, pp. 27-28].

Real life enterprise systems are usually made up of many programs working together, like web servers, database servers and billing shipping and receiver software. System administration, web applications and document processing are areas where the application of scripting languages is preferred [7].

In conclusion, scripting programming languages are a perfect match for the development of web applications.

2.2. Java

Even though, the Java system programming language is not being directly used for the creation of the example web applications, it is still used as an underlying component throughout this work. Therefore, a basic understanding of its architecture is required.

The main feature of the Java programming language is its architecture-neutral approach. Instead of machine code its compiler creates so called bytecode that runs on the Java Virtual Machine [8, Ch. 4]. The Java platform is software-only and runs on top of hardware or software environments. In addition to the Java Virtual Machine, the Java platform also includes the Java Application Programming Interface (API) [9].

"*The API is a large collection of ready-made software components that provide many useful capabilities. It is grouped into libraries of related classes and interfaces; these libraries are known as packages*" [9].

After source code is written and saved with the `.java` extension it needs to be compiled into a `.class` file by the `javac` compiler. A `.class` file contains bytecodes which then gets read and executed by the Java Virtual Machine [9].

The Java Language Classes, `java.lang`, contain the base types and are always imported into a compilation unit. They include the fundamental classes, such as object or the so-called wrapper classes for primitive types like Booleans. The complete Java system also includes the libraries: `java.io`, `java.util` and `java.awt` [8, Ch. 9]. These core libraries enable a huge variety of features, such as network communications, security management or file handling [10].

In addition, external libraries are created by other people to extend the functionality of Java even further. These reusable bits and pieces can be used to add missing functionality or help a programmer write less code and therefore save time. Beginners can use them to create programs including features they would not be able to create themselves. A library consists of a bundle of packages, which hold Java classes and interface definitions [10].

The libraries also include the application programming interface (API) documentation. Javadoc reads the comments in the library's code and uses

them to create this API documentation, holding reference information to ease usage. Libraries usually come usually packaged as `.jar` files [10].

Java Archives use the `.jar` extension and are based on `.zip` files. They are used to package multiple files together, compressing them to decrease size. While they usually hold multiple `.class` files, whole applications can be packaged in the same way, also including pictures and audio [11].

2.3. Java and Scripting Languages

Generally, code written in a scripting language can be compiled into Java bytecode, enabling its execution on the Java Virtual Machine. However, this approach results in the loss of benefits that Java offers.

2.3.1. JSR 223

The Java Specification Request (JSR) 223 was released at the end of the year 2006 [12]. It enables the embedding of scripts in Java applications and the access of Java objects from within scripts. A script written in compliance with JSR 223 can access the entire standard Java library. Equally important, a Java application written with JSR 223 in mind, allows the embedding of scripts without the need to specify a scripting language [13, Sec. 1].

"A program specification describes the results that a program is expected to produce -- its primary purpose is to be understood not executed. Specifications provide the foundation for programming methodology" [14].

The Java Scripting API is defined by JSR 223. Its classes and interfaces can be found in the `javax.scipt` package. It contains the `ScriptEngineManager` class, which discovers script engines through the `.jar` file service discovery mechanism. After discovery, a `ScriptEngine` object gets instantiated to perform the interpretation [13, Sec. 2]. The `ScriptEngine's eval` method is used to execute a script that has been given as input parameter; afterwards, the value that gets returned from the execution of the script is returned [15].

The `java.script` API has been included in the Java Standard Edition since version 6 [16].

2.3.2. Bean Scripting Framework

"*JavaBeans are classes that encapsulate many objects into a single object (the bean)*" [17].

The Bean Scripting Framework resulted from a research project of Sanjiva Weerawarana at IBM in 1999. Its goal was to access JavaBeans from scripting language environments. The project continued as an open-source project at IBM before it was donated to the Apache Software Foundation at version 2.3 [18].

"*Bean Scripting Framework (BSF) is a set of Java classes which provides scripting language support within Java applications, and access to Java objects and methods from scripting languages*" [19].

The framework's two main classes are the `BSFManager` and the `BSFEngine` [19]. The `BSFManager` class gets instantiated when an application decides to run a script. It is then used to register beans, load script engines, and run scripts. Furthermore, the `BSFManager` registers all available script engines, loads, and unloads them. Each Java Virtual Machine can run multiple `BSFManagers` but each `BSFManager` can only load one engine per language [20]. The `BSFEngine` abstracts a scripting language's capabilities and allows generic handling of script execution and object registration within the execution context of the scripting language [19].

Releases under the newer version 3.x use the JSR 223 API. The Open Object Rexx programming language is supported with its own BSF engine: BSF4ooRexx [21].

2.4. Open Object Rexx

All the nutshell examples accompanying this thesis have been created using the ooRexx scripting language. Even if the reader is not familiar with the language, its easily understandable syntax might help with the development of web applications in another language.

Initially developed in 1979 by Mike F. Cowlishaw, the Rexx programming language aimed to make the programming of IBM mainframes easier to

understand and more human centric. After gaining popularity in the industry, the language evolved in 1997, implementing object-oriented features, in the IBM product Object Rex. In 2004 the source code was given to the Rexx Language Association, which released the first open-source version of the language, called Open Object Rexx. This powerful yet extensible language is available for all major operating systems [22, pp. iii-v].

2.5. Bean Scripting Framework for Open Object Rexx

In 2001, the Bean Scripting Framework for Rexx was first introduced at the 12th International Rexx Symposium by Rony G. Flatscher. In this first iteration, based on a seminar paper by Peter Kalender, the Rexx and Object Rexx interpreters were incorporated into the BSF framework. Henceforth, it was possible for Java programs to invoke and cooperate with Rexx and Open Rexx programs [23, p. 5]. Two years later, a further improved version, with the ability to start Java from Rexx programs, was presented, allowing Java to be used as a Rexx function library [24, p. 5]. In 2009, the first BSF400Rexx version was released, implementing new features made possible by native ooRexx APIs introduced with ooRexx 4.0 [25, p. 4].

One of BSF400Rexx's main achievements is to camouflage Java, allowing the ooRexx user to utilize Java class objects without requiring extensive Java knowledge. The user can send messages to those so called "proxy classes", which will be forwarded to the Java object that they represent. This is achieved by the Bean Scripting Framework supporting module `BSF.cls`. It constructs an object-oriented interface to the Java Runtime Environment, enabling access to features like Java arrays [26, pp. 13-18].

2.6. Hypertext Transfer Protocol

Before a client and a server can start communicating, they need to agree on common rules for data transmission and the information's structure. These rules are established in form of a protocol [27].

The most popular protocols of the Internet, the Hypertext Transfer Protocol, is an asymmetric, stateless pull protocol, running on the application layer. The client sends a request and gets a response from the server. This request is most often based on a Uniform Resource Locator, which the browser converts to a request [28].

"A URL (Uniform Resource Locator) is used to uniquely identify a resource over the web. URL has the following syntax: protocol://hostname:port/path-and-file-name" [28].

It typically runs over a TCP/IP connection, but also allows other reliable transport methods. Given its stateless nature, requests are not connected, and are not aware of previous communications. The negotiation of data type and representation systems are independent from the way the data is transferred [28].

In addition to the `get` request method, `post` is used to send data to the webserver, while `delete` requests its deletion [28]. A typical HTTP request can be seen in the image below, after the request line, headers inform the receiver about what type of files can be received and includes other information, like the message's length.

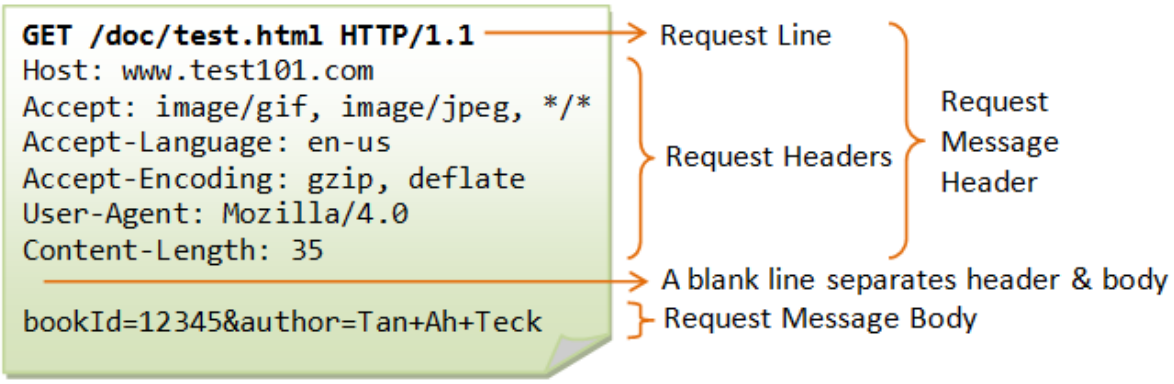

Figure 1: HTTP Request [28]

After the request is sent, the server replies with a response message. As can be seen in the figure below, instead of a request line it begins with a status code. For example, the Code 404 means that the requested resource cannot be found. While the response headers include additional information, the requested content can be found in the message body.

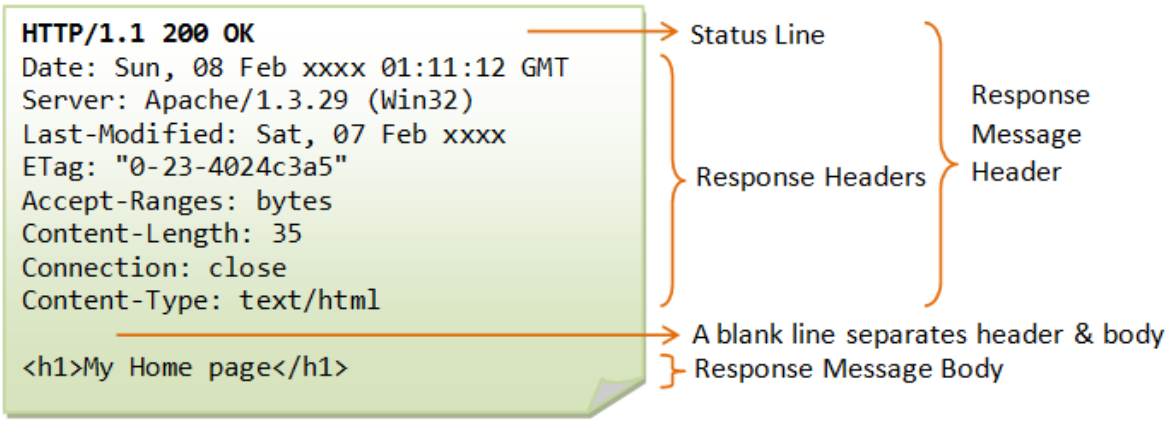

Figure 2: HTTP Response [28]

2.7. Hypertext Markup Language

While first intended to describe scientific documents, the Hypertext Markup Language soon became the core markup language of the World Wide Web [29, Sec. 1.1.].

HTML is used to define the structure of a web page. A tree of elements, such as `<head>`, `<body>`, or `<p>` is used to describe how a document is to be displayed [30]. Each element consists of a starting and end tag, with the content displayed in between: `<>Text</>`. Additionally, elements can have attributes placed in their start tag: `<form action="put">` [29, Sec. 1.9.].

It is important to note that: "*HTML documents represent a media-independent description of interactive content. HTML documents might be rendered to a screen, or through a speech synthesizer, or on a braille display.*" [29, Sec. 1.9.].

When a web browser parses such a document, it transforms it into a Document Object Model (DOM) tree and stores it in memory. While this representation of a web page is static in nature, scripts can be used to manipulate it [29, Sec. 1.9.].

The presentation of such a document can be altered using Cascading Style Sheets (CSS): *"The CSS1 language is human readable and writable, and expresses style in common desktop publishing terminology"* [31]. CSS rules can be either applied by an external file, or within the HTML document itself, in form of inline styling.

2.8. Jakarta Servlets

A servlet is a program running inside a web server that creates a customized or dynamic response for each incoming HTTP request. For example, the user might have filled in a form on a web page. On submission, a web page will be tailormade according to the input parameters. Another example would be the creation of a user-specific webpage, using data from a database or another application. Furthermore, time sensitive information, such as stock quotes, might need to be requested. As a result, the response is not the same for each request, but changes dynamically [32, Sec. 1.].

"Java servlet is the foundation of the Java server-side technology, JSP (JavaServer Pages), JSF (JavaServer Faces), Struts, Spring, Hibernate, and others, are extensions of the servlet technology" [32, Sec. 1.]. They share the advantages of the mature Java programming language: server- and platform-independency, reusability, portability, and high performance [33].

The server-side Java stack below shows how the servlet's components are related to each other. The Java Virtual machine runs on top of the operating system and serves as an environment for the Java Server. A Java application can then use the facilities of the server as well as of the Servlet API [34].

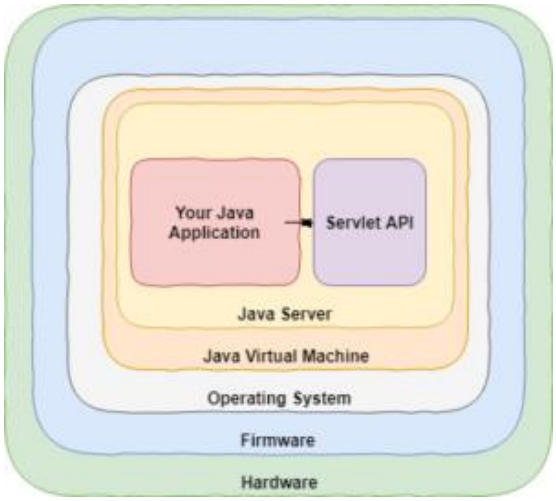

Figure 3:Server-Side Java Stack [34]

A Java Server like Apache Tomcat provides the necessary run time environment for the servlet. Additionally, it handles all networking services and resources necessary, while also managing the life cycle of servlets [35]. It decodes and formats MIME-based requests and formats MIME-based responses [36, Sec. 1.2.]. The Multipurpose Internet Mail Extension (MIME)

is used to specify the media type, subtype and encoding of data that has been received in the body of a message. For example, types like application, image or audio are defined [37, Sec. 1.].

A single process instance handles all request, increasing performance and saving memory. Security is enhanced by the servlet being part of the web server, and therefore inheriting its security measures [35]. The web server can implement security restrictions based on the Java permission architecture for the servlet's environment. Jakarta Servlets of the Version 5.0 support HTTP/1.1 and HTTP/2.0. It is important to note that the server can modify HTTP requests before and after the processing of the Servlet to allow caching [36, Sec. 1.2.].

When creating a servlet, one can either extend the `jakarta.servlet.GenericServlet` interface or the `jakarta.servlet.http.HttpServlet` interface. The more specific `HttpServlet` includes methods supporting the HTTP protocol like `doGet`, which handles HTTP GET requests [38]. After being instantiated the servlet first gets created with the `init` method, then handles all client calls with the `service` method, and finally retires with the `destroy` method [39]. By default, only a single instance is created for each servlet declaration [36, Sec. 2.2.].

2.9. Jakarta Server Pages

"*A JSP page is a text-based document that describes how to process a request to create a response*" [4, Sec. Overview].

Jakarta Server Pages are closely related to Servlets and are built based on their specification [40]. The most common application of JSP pages is in the form of HTML and XML content. They enable the usage of web applications, servlet contexts, sessions, requests, and responses [4, Sec. Overview].

"*From a coding perspective, the most obvious difference between them is that with servlets you write Java code and then embed client-side markup (like HTML) into that code, whereas with JSP you start with the client-side script*

or markup, then embed JSP tags to connect your page to the Java backend" [40].

While HTTP is the default protocol for requests and responses, other protocols are also acceptable, if the container supports them [4, Sec. 1.1.].

Before being requested by clients during the request phase, JSP's need to be translated by the container, creating a servlet class. This class is called the JSP implementation class. This translation is performed once per page and can take place at deployment time or once requested. After being instantiated at request time, this class creates responses for incoming requests. [4, Sec. Overview]. In short, after the translate phase is concluded the JSP will be indistinguishable from any other servlet.

JSP's can contain fragments written in a scripting language, which are referred to as scriptlets [41].

Most significantly, JSP's functionality can be extended by Tag Libraries. They introduce custom actions, to be used manually or by Java development tools. The tag handler is an instance of a Java class, corresponding to a tag and implementing its functionality [42]. The collection of Java classes that makes up a custom Tag Library can be packaged in a `.jar` file.

The Tag Library Descriptor (TLD) is used to describe the Tag Library in form of an XML file and uses the extension `.tld`. It allows JSP containers to interpret pages using a tag library. Additionally, a `TagLibraryValidator` class can be used to check whether a JSP page is valid according to a set of expected constraints [4, Sec. 7.3.].

While Java is the default scripting language used in JSP, other languages can be added using two different methods. Declaring them in the page directives at the beginning of a page has the disadvantage that some containers only support Java. Utilizing a tag library is beneficial since this method is portable between containers. All JSP containers must support the tag extension mechanism. Additionally, this approach allows using multiple different scripting languages on the same page [43, p. 34].

2.10. Apache Tomcat

"*Apache Tomcat is a long-lived, open source Java servlet container that implements several core Java enterprise specs, namely the Java Servlet, JavaServer Pages (JSP), and WebSockets APIs*" [44].

In 1997 the American software developer James Duncan Davidson started to work as an engineer for JavaSoft, which at the time was a part of Sun Microsystems. While working on the Java Web Server he created a reference implementation for the Java Servlet specification, called the Java Servlet Web Development Kit [45]. In 1999 the project was donated to the Apache Software Foundation and was thereafter called Tomcat. In 2005 Tomcat became a top-level Apache project to be managed by itself [46]. Tomcat refers to multiple components working together, mainly Catalina, Jasper and Coyote.

"*Catalina provides Tomcat's actual implementation of the servlet specification; when you start up your Tomcat server, you're actually starting Catalina*" [47]. This is also the reason why Tomcat's main directory is often referred to as `CATALINA_HOME` [48]. The Java class `Catalina` not only provides the servlet container's main functionality, but is also responsible for its configuration, security, and logging [47].

The Jasper JSP engine is used to implement the Jakarta Server Pages specification. It compiles JSP's into Java code to be used by Catalina as servlets. It can detect changes in a JSP at runtime and compile them immediately [49]. Therefore, changes by the user will be immediately realized.

The Coyote HTTP/1.1 Connector enables Tomcat to work as a stand-alone web server. It gets instantiated and listens for connections on a specified TCP port number [50]. Without this class, a dedicated HTTP server like the Apache HTTP Server would be required. As a matter of fact, Tomcat can be connected to an Apache HTTP server by means of a connection module [51].

Since Tomcat is often also referred to as a web server, it is important to establish the difference between a webserver and a web container. A web server is used to store and deliver web pages to requesting clients. To

accomplish this, the HTTP protocol is used, which also allows the transmission of information from clients [52]. JSPs and Servlets are referred to as web components, they are placed in the web container and can then be used in an environment provided by the web container [4, Sec. 1.1.1.]. Therefore, every web container can be referred to as a web server, while not every web server is a web container. Furthermore, a web container utilizing the Jakarta Servlet API can be referred to as a servlet container.

2.11. Open-Source Software

Most of the software mentioned throughout this thesis is open source, meaning that its source code is freely available for anyone to inspect, modify and enhance. This approach not only allows control and security but also the creation of a community [53].

"*Open source projects, products, or initiatives embrace and celebrate principles of open exchange, collaborative participation, rapid prototyping, transparency, meritocracy, and community-oriented development*" [53].

Open-source software is generally released under a license. Copyleft licenses, like the GNU General Public License, allow the creation of derivative works, but require them to be using the same license. In contrast, permissive licenses allow the user to freely use, modify and redistribute the software [54]. The Apache license is an example for a license that is permissive.

2.11.1 The Apache Foundation

The Apache Software Foundation is a not-for-profit corporation established in 1999. An all-volunteer board oversees over 350 open-source projects and supports them with a framework for intellectual property and financial contributions [55].

"The mission of the Apache Software Foundation (ASF) is to provide software for the public good" [55].

The Apache Tomcat Software is released under the Apache License version 2.0 [56]. At the time of writing, in January 2021, the current version of the

Apache License is 2.0, which was approved in 2004. Most importantly, software released under this license can be reproduced and used to create derivates without paying royalties. To offer legal protection, the license is revoked should an entity file a lawsuit claiming patent infringement by the creator or any contributor [57]. All the examples created for this thesis are licensed under the Apache License 2.0. Therefore, should the reader deem them useful, free usage is encouraged.

2.11.2 The Eclipse Foundation, Jakarta Namespace

The Eclipse Foundation is a not-for-profit organization with over 275 members, focusing on open-source projects [58]. It was initially created by IBM in the year 2001 [59].

"The Eclipse Foundation provides our global community of individuals and organizations with a mature, scalable, and business-friendly environment for open-source software collaboration and innovation" [58].

Until 2019, the Jakarta Servlets and the Jakarta Server Pages were officially known as Java Servlets and Java Server Pages. This name change goes hand in hand with the rebranding of Java Enterprise Edition (Java EE) to Jakarta Enterprise Edition (Jakarta EE), the set of specifications under which the Java Servlet specification was originally released. In the year 2017 the then owner of Java Enterprise Edition, Oracle, decided to donate it to the open-source Eclipse Foundation. On September 10, 2019 the first version under a new name was released, Jakarta EE8. The release changed all the included specifications names, including the Java Servlet, which was referred to as Jakarta Servlet from this point onward [60].

While the Standard Edition of the Java Platform enables the development and deployment of desktop and server applications, its Enterprise Edition is focused on large, multi-tiered enterprise applications [61]. "*The focus of the Jakarta EE platform is not to bundle a bunch of unrelated APIs. The purpose of Jakarta EE is to ensure that a variety of useful enterprise APIs work in harmony*" [62].

TomEE is a version of Tomcat, that builds on the standard edition, by adding all the Jakarta EE API's. Only a single of those API's is required for

the web applications accompanying this thesis, namely Jakarta Mail. To simplify, it has been added to the standard Tomcat version as a standalone `.jar` file. Alternatively, TomEE could have been used instead [62].

Tomcat 10 was the first version to implement this change, which is reflected in the naming of all primary packages [63]. To give an example, the cookie class needs to be referred to as `javax.servlet.http.Cookie` in version 9 and `javax.servlet.http.Cookie` in version 10.

This namespace change is directly related to this thesis, since at the time of writing, in January 2021, the Apache Tomcat Software Version 10 was still in its Beta phase. Nonetheless, to future proof this work, it is fully based on Tomcat 10. Should the reader prefer the stable version 9, most components should be near identical. Otherwise, the only difference is that, instead of the examples found directly in the zip archive, the ones in the directory `ZIP_ARCHIVE/javax_for_Tomcat09` need to be used instead. They are identical, except for the above-mentioned name changes. For more information on the usage of Tomcat 9 please refer to the appendix: **`9.3. Tomcat 9`**

2.12. Putting it All Together

The Apache Tomcat Software is used to provide the necessary infrastructure and to make web applications available. It handles all incoming HTTP requests and outgoing HTTP responses. The conventions and structure of Java web applications is used to create web applications, with JSPs forming the core technology. The JSP contains custom tags, holding code written in a scripting language, implementing the web application's functionality. These tags originate from one of two tag libraries programmed by Rony G. Flatscher for BSF4ooRexx, using either the BSF or the JSR 223 framework.

Scripts that are invoked (evaluated) this way are supplied with the implicit objects normally available to a standard Java scriptlet inside a JSP. Most notably `request`, `response` and `out`. [43, p. 35].

Even though many scripting languages can be used this way, ooRexx has been selected as the scripting language of choice. The necessary script

engine, `RexxScriptEngine`, is made available by BSF4ooRexx, enabling BSF and JSR 223 support [64, p. 5]. Additionally, BSF4ooRexx enables the inclusion of countless external Java libraries.

Hence, the script runs on the server, generating dynamic content based on the request sent by the user. This technique is also referred to as server-side scripting. Even though the client is not required to support the scripting language used, increased latency might be a disadvantage for some applications [65]. After processing, the user receives a standard HTML document that gets rendered by a web browser.

3. Apache Tomcat Fundamentals

The following chapter introduces the Apache Tomcat Software, communicating all knowledge necessary to run the complementary example web applications included with this thesis. At this stage, ooRexx, BSF4ooRexx and Tomcat should be installed. Download links for the first two components can be found in the appendix: `9.1.1. Software required to` begin, as well as detailed installation instructions for Tomcat: `9.2. Tomcat Installation Guide`

3.1. TOMCAT_HOME

The author uses `TOMCAT_HOME` to describe Tomcat's main directory. The figure below shows its contents.

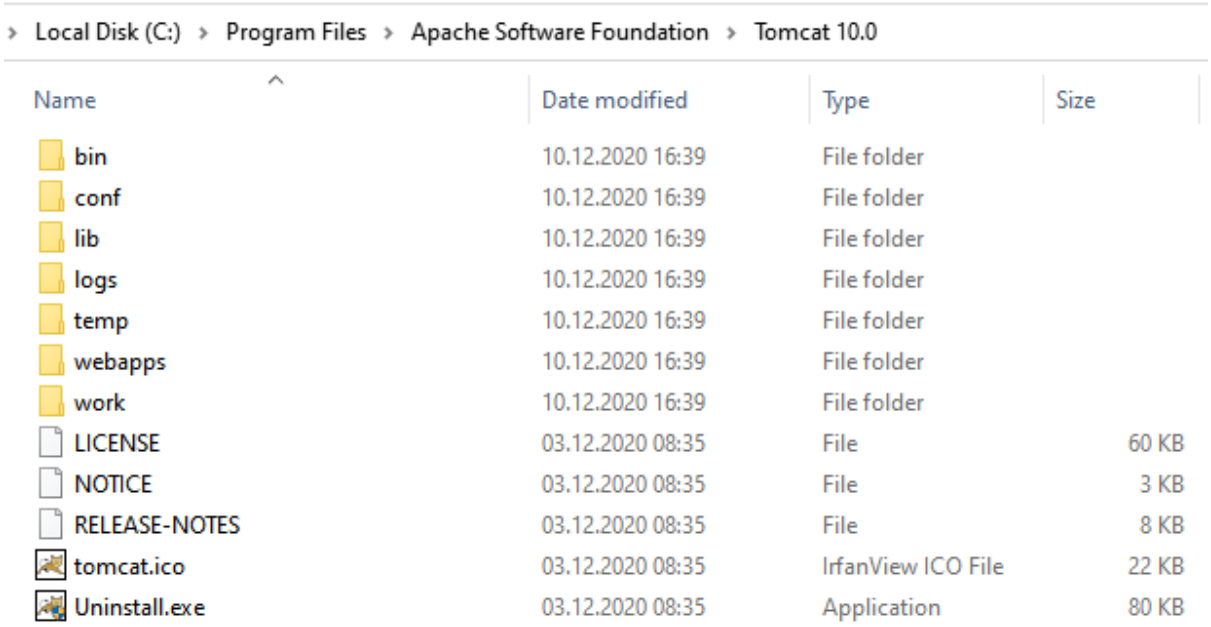

Figure 4: TOMCAT_HOME Directory

`TOMCAT_HOME\bin` contains scripts in the form of `.bat` files. Mainly, `startup.bat` and `shutdown.bat` which can be used to start and stop the server.

`TOMCAT_HOME\conf` holds multiple files used to configure the software's properties. The `server.xml` file is used to change the initial server configuration on startup, for example it points to external static resources. The file `web.xml` is used to deploy and configure web applications. The files in this folder serve as a default, for certain parameters to be overwritten by a `web.xml` file specific to a web application if needed [66].

`TOMCAT_HOME\lib` contains `.jar` files that are shared among all web applications. The files placed in the main directory's lib folder are accessible not only to all web applications but also to the Tomcat application itself. Files that are necessary for the functioning of the software, like `catalina.jar` and `jasper.jar` come preinstalled [67].

`TOMCAT_HOME\logs` contains log-files, useful for debugging and testing self-written web applications. Particularly, for each day the server is run, a file called `tomcat10-stderr.yyyy-mm-dd.log` is created, containing error messages. This file is particularly useful to detect the cause of exceptions. Also, it might prove useful to regularly delete old log files to quickly find relevant entries. To make deletion possible the Tomcat server needs to be shut down. For convenience, all files within the log folder can be deleted if

no longer needed, since the Tomcat Software will automatically recreate all necessary files on the next startup.

`TOMCAT_HOME\webapps` is the directory where all web applications can be found. Depending on the installation parameters chosen, this folder might already come shipped with default applications.

`TOMCAT_HOME\work` is used by Tomcat for intermediate files during runtime. For example, once a JSP is compiled, the result is placed here [68].

3.2. Deploying Web Applications

"Deploying your application means putting it on a Web server so that it can be used either through the Internet or an intranet" [69].

To begin with, two demo web application shipped with this work needs to be deployed and made accessible. There are two ways to accomplish this, either the web app is deployed exploded or in the form of a web archive file.

Web Application Archives use the `.war` file extension and contain all necessary files for a web project. Everything that is needed such JSPs, scripts and the configuration files are contained in a single file. They are quite like `.jar` files and can be create from the command line with the jar tool included in the Java Development Kit. For example, the command `jar -cvf projectname.war *` will create a web archive from all the files contained in a directory [70].

The usage of `.war` files is especially convenient because they use the `.zip` format [71]. Instead of using the command line, it is also possible to create a simple `.zip` archive and giving it the `.war` file extension. To view its contents, `.war` files can also be unpacked by a compression software, for example with 7-zip.

Lesson Learned: When web applications are shipped as .war files, all required files are expected to be included in the archive, special attention needs to be given to .jar files.

After placing a `.war` file in `TOMCAT_HOME\webapps`, the Software will automatically unpack the files in a new folder of the same name on startup.

Afterwards, all files can be conveniently viewed. Once a `.war` file is unpacked it is considered as exploded.

When web applications are developed, they are usually deployed exploded. A folder in the webapps directory is created and the files inside are modified without the need to compress them into a single file.

Furthermore, all web servers compliant with Jakarta EE, handle web applications the same way, allowing identical `.war` files to be used with different Java webservers, like IBM WebSphere. They all handle the `.war` files as an independent application, using its main directory as a virtual root. Therefore, any concepts used for web application development with Apache Tomcat can be directly transferred to other Java web servers [72].

While `.war` files contain `.jar` files, `.ear` files contain multiple `.war` files. This format used by the Jakarta EE platform to create application packages [73].

At this point the reader is encouraged to copy the `.war` files `helloworld.war` and `treeshop.war` to `TOMCAT_HOME\webapps`. While the web application called `helloworld` will be the subject of the next sections, `treeshop` will be discussed at a later stage. The files can be found directly in complementary archive.

3.4. Starting Tomcat

There are multiple ways to start the Apache Tomcat software. The previously mentioned scripts `startup.bat` and `shutdown.bat` `TOMCAT_HOME\bin` exist for all platforms but might have the file extension `.sh` instead on different operating systems.

On the Microsoft Windows (Windows) operating system the Apache Commons Daemon Service Manager, which creates a taskbar icon, can be run from the start menu entry `Monitor Tomcat`. It can be found in the folder `Apache Tomcat 10.0 Tomcat10`. It offers a convenient way to configure, start and stop the server.

Figure 5: Apache Commons Daemon Service Manager

The reason why Tomcat cannot be started by conventional methods lies in its nature as a service. On the Microsoft Windows operating system, Windows services run in their own Windows session and are used for applications that require long-running functionality, without interfering with other users on the same machine. Additionally, they allow a different security context [74].

Therefore, yet another way to control the status of the Apache Tomcat 10 can be accessed by typing `services.msc` in the Windows Powershell or command prompt. A list of all services will be displayed. By right-clicking on `Apache Tomcat 10.0 Tomcat10`, the server can be started and stopped.

Once running, Tomcat can be reached from the URL: `http://localhost:8080/`

Localhost is a top-level domain, referring to the current computer and is interchangeable with the Internet Protocol address (IP address) `127.0.0.1`. The number 127 at the beginning of the address triggers a so-called loopback; the request is not forwarded to the internet but handled by the local computer instead. This feature is mainly used by administrators for testing purposes [75].

Ports are interfaces on a computer to which other devices can connect for communication purposes. The ports are numbered starting from 0 to 65535. Ports numbered 0 to 1023 are also called well-known ports, which are reserved for common services like the HTTP protocol, which has port 80 assigned to it [76]. During the Tomcat installation process, the HTTP connector port got assigned to 8080. Here Tomcat's Coyote component is listening for incoming requests.

In case the reader has defined a different port during installation, the URL needs to be changed accordingly. Differences concerning the usage of Tomcat 9 are in name only.

3.5. Tomcat Manager

If it has been installed, the Tomcat Manager App can be accessed from: `http://localhost:8080/manager/html`

When started, the application asks for the username and password given during the installation.

Among other features, this web application gives an overview of all installed web applications, allows them to be deployed, undeployed and reloaded, all without necessitating a restart [77]. This is particularly useful for production environments where multiple people work together.

4. Introducing web applications `/helloworld`

Thus, after introducing the fundamentals of working with Tomcat, the web pages contained in the application `helloworld` will introduce the reader to web application development.

At this point, the file `heloworld.war`, should have been placed in TOMCAT_HOME\webapps. After restarting the software, the web archive's contents are exploded automatically. The folder structure directly influences the path from which pages are accessed: Files in the directory TOMCAT_HOME\webapps\helloworld, can be accessed from the URL: `http://localhost:8080/helloworld`

The main directory of `helloworld`, found in the directory `TOMCAT_HOME\webapps\helloworld` is referred to as the application's context path [71]. To allow generalization, this thesis uses the path `WEBAPP\` to refer to this directory.

4.1. Web Application Architecture

Some elements are common to all Java based web applications. The directory `WEBAPP\WEB-INF` contains all resources necessary to run the application. Typically it holds `.jars`, `.tlds` and the `web.xml` file. Notably, these resources are not made accessible to a web user [78].

The web.xml file contains the Web Application Deployment Descriptor. It is used by the JSP container to gather general configuration information [4, Sec. 3.1.]. The main web.xml file can be found in TOMCAT_HOME\conf, while the version specific to a web application is located at WEBAPP\WEB-INF. The latter is used in case deviating or additional configuration parameters are required. [79]. For example, it holds information used to name and describe the web application in the Tomcat Manager application. At a later stage, this file will be used to add configuration parameters.

To minimize potential errors and to showcase the interchangeability, both Tag Library Descriptor files for the JSR 223 (script.jsfr223.tld) as well as the BSF (script-bsf.tld) tag library were placed in WEBAPP\WEB-INF folder.

The directory WEBAPP\WEB-INF\lib contains Java .class files in .jar archives. Like the web.xml file, the contained libraries are specific to the web application.

For the web applications shipped with this thesis to function, two Java Archives are always needed. First, the file jakarta.ScriptTagLibs.jar holds the actual BSF and JSR 223 tag libraries. The bsf4ooRexx-v641-20201124-bin.jar includes the Bean Scripting Framework, the bridge between Java and ooRexx.

The leaves the question, whether to place the classes necessary for a web application in TOMCAT_HOME\lib or WEBAPP\WEB-INF\lib. For the application helloworld, the author has chosen to package all necessary .jar files in the WEBAPP\WEB-INF\lib directory.

Generally, the benefit of not requiring the user to modify his Tomcat installation outweighs the redundancy of having multiple identical .jar files. The result are web applications that run simply after being placed in the webapp folder. Nonetheless, other factors complicating this issue will be discussed later.

4.2. Introducing Jakarta Server Pages /helloworld/helloworld.jsp

The listing below shows the contents of the document helloworld.jsp. At first glance, the Jakarta Server Page is almost identical to a standard HTML

page. By interweaving static and dynamic content the JSP gets transformed into a Rexx Server Page.

```
<%@ 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>
<header>

<s:script type="rexx">
USE ARG request, response, out

greeting = "Hello, world! (Sent from Open Object Rexx)"
out~println(greeting)
</s:script>

</header>
</html>
```

Listing 1: helloworld.jsp

4.2.1. JSP Directives

All JSPs share a common set of characteristics and begin with the so-called directives, containing messages to the JSP container. All directives follow the syntax: `<%@ directive {attr="value"}*%>`. The three existing directive types are `page`, `taglib` and `include` [4, Sec. 1.10.].

The `page` directive is used to communicate page dependent properties to the JSP container. It can occur multiple times and at any position in the document, except for the `pageEncoding` and `contentType` attributes, which are expected to appear at the beginning. Attributes are limited to a single instance, except for `import` and `pageEncoding` [4, Sec. 1.10.1.].

To begin with, the `session` attribute of the `page` directive is used to identify a user across multiple requests. The identification is made possible by means of a cookie, or alternatively by rewriting the URL [80]. No specific session cookie is created for this simple web page with the only goal of displaying information.

The `pageEncoding` attribute determines the encoding of the JSP itself, while the `contentType` attribute defines the `MIME` type and character encoding of

the response. Additionally, the character encoding can be defined by the `CHARSET` attribute [4, Sec. 1.10.1.].

Encoding is particularly important for webpages since they might contain text in many different languages. Characters on computers are stored as bytes which need to be mapped to characters using a specific code. The characters in this context are grouped into character sets. Many different character sets exist for different purposes and languages; for the use case of creating a web page, the Unicode `UTF-8` is recommended. `UTF-8` includes a multitude of characters, for almost any possible situation, making it unnecessary to switch or convert between encodings throughout a project [81]. Furthermore, it ensures maximum compatibility with different languages. If the `pageEncoding` is not explicitly declared, `ISO-8859-1` will be used instead [4, Sec. 4.1.1.].

In the second line, the `taglib` directive declares that a tag library is used to extend the page's functionality. The `uri` attribute points to the Tag Library Descriptors exact location in the directory `WEBAPP\WEB-INF`. The declared `prefix` attribute is used to indicate the usage of one of the library's custom actions throughout the document [4, Sec. 1.10.2.].

The `include` directive is used to insert text, data, or code of a specified resource at JSP translation time [4, Sec. 1.10.3.]. In this example the directive has been omitted.

4.2.2. JSP Content

The `<!DOCTYPE html>` declaration informs the browser about the nature of the document and that its author is following standard practices. Having the `doctype` declaration at the beginning of a web page is good practice and a sign of quality [82]. Similarly, it is a good idea to declare the charset as `UTF-8` once again. The previously `charset` attribute declared in the directives is sent in the HTTP response header. Should the server configuration change or the page gets saved locally, the HTTP response header would be missing [83]. For this situation, the charset gets declared again. Even though, `UTF-8` is the standard charset that will be applied to any

HTML5 page in case it is omitted, the web user's browser's behavior is not guaranteed, especially if an older browser is used [84].

Afterwards, scripting code is used to display a message in the document's header. The dynamic content starts with the previously declared taglib prefix `s`. After the double colon, the tag `script` indicates the start of a script. The attribute `type` defines the scripting language used, in this case Open Object Rexx [85].

Alternatively, many other scripting languages could be used instead. For example, the addition of the file `jython.jar` would allow the insertion of code written in the Python programming language in place of ooRexx [86, p. 19].

First, the objects `request`, `response` and `out` are fetched. These objects are part of the implicit objects, nine of which are created by the JSP engine during translation phase [87]. Invoking scripts by means of the tag libraries developed by Rony G. Flatscher, supplies the implicit objects automatically, merely requiring them to be fetched. With ooRexx, this is done with the instruction `USE ARG` [85].

The `requests` object provides data the client has transmitted when initially requesting the page, usually it originates from forms. The `response` object modifies or delays the response that is sent back. The third fetched object `out`, is responsible for writing content to the HTML page the user receives. Furthermore, it enables the formatting of messages [35].

After fetching the implicit objects, the script defines a greeting string and stores it in the variable `greeting`. The `out` object refers to an instance of the Java class `JspWriter`. Next, the `println` method is used to print the characters and terminating the line afterwards [88]. As a result, the greeting previously defined will be displayed in the HTML page header. The closing tags conclude this first script. Since no HTML tags have been given, the `printl` method prints the sentence in verbatim without any formatting applied.

The following figure showcases the HTML document the user receives when requesting `helloworld.jsp`.

Figure 6: hellworld.jsp in Web Browser

As can be seen, the resulting page looks like a standard web page, the content generated by the script is indistinguishable from the static HTML parts.

```
1
2
3  <!DOCTYPE html>
4  <html>
5  <head>
6  <meta charset="UTF-8" />
7  <title>hello, world</title>
8  </head>
9  <header>
10
11 Hello, world! (Sent from Open Object Rexx)
12
13
14 </header>
15 </html>
16
```

Listing 2: helloworld.jsp HTML Source Code

Lesson Learned: For URLs, upper- and lower-casing matters, it needs to reflect the JSP document's exact name.

From this point onwards, the contents of `WEBAPP\WEB-INF`, as well as the page directives and HTML code up to the header can be copied and reused as standard building blocks. The next example page builds on the first.

4.3. BSF Taglib, Styling, Expressions `/helloworld/helloworld_ext.jsp`

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

<s:script type="rexx">
```

```
USE ARG request, response, out

greeting = "Hello, world! (Sent from Open Object Rexx)"
SAY '<h1>'greeting'</h1>'
</s:script>

</header>
<body>

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

<s:script type="rexx">
USE ARG request, response, out

SAY '<p style="color:blue" !important>'pp("This paragraph is made possible by the
BSF taglib")'</p>'

/* Note: when using the BSF framework we need to require BSF.CLS in each script
and expression,
        if we need to access its public routines or public classes */
::REQUIRES "BSF.CLS"    -- make sure the Java bridge is there
</s:script>

</body>
</html>
```

Listing 3: helloworld_ext.jsp

Compared to the first page, this document's head includes the `link` tag. It is used to extend the document with additional resources. The `rel` attribute, standing for relation, is communicating the nature of the linked resource [89]. A stylesheet of the type `.css`, which can be found by the URL: `css/shop.css` has been added. This path is relative, referring to the location of the page. Therefore, the file is found in: `helloworld/css/treeshop.css`. It is also possible to utilize an absolute path, by giving a full URL. However, relative paths are best practice, should the domain or computer change, all paths would need to be changed otherwise [90].

Moving on, the first script block has been both simplified and extended at the same time. The simplification is achieved using the ooRexx `SAY` instruction which, thanks to the used tag library, results in the same result as the `println` method. Additionally, the line demonstrates how HTML tags can be used to format outputs from scripting languages. First the tag gets printed in single quotation marks, which are closed to insert the previously defined variable greeting. Afterwards, the end tag is printed. This approach allows weaving together outputs and HTML tags, to generate dynamic

content. The result, that gets sent to the end user will look like simple, static HTML code, leaving no trace of ever containing scripting code.

```
 9 </head>
10 <header>
11
12 <h1>Hello, world! (Sent from Open Object Rexx)</h1>
13
14
15 </header>
16 <body>
```

Listing 4: helloworld_ext.jsp HTML Code

Lesson Learned: This approach requires extra care when using quotation marks. Since double quotation marks are needed to specify the attribute values, the `SAY` command uses single quotes to enclose the message.

While for other scripting languages the `out` object in combination with the `println` method is a universal way to write HTML content to the JSP, the demo pages will use the SAY instruction from this point onwards instead.

In addition to `script`, the `expr` tag can be used to fetch the result of an expression defined in a scripting language [85]. In the example the body of the HTML file includes a paragraph after the script has been concluded. It is used to demonstrate how the expression tag is used to intertwine the output of the ooRexx `time()` function with standard HTML code. The function returns a timestamp which is consecutively displayed on the web page. The output reflects the point in time, at which the page was originally generated. Since HTML is static by nature, expressions allow quick enhancements with dynamic content.

Next, the scripting tag is used once again. The first thing to note is the style attribute, indicating a CSS rule by means of inline styling.

The rule consists of the declaration `color:blue`. When looking at the external stylesheet `shop.css`, the selector for a paragraph, `p`, already exists. For inline styling, the selector can be omitted. Overall, the CSS 2.1 version has over 90 properties, allowing in-depth customization of a web page, including fonts, tables, and backgrounds [91, Sec. 2.1.].

The web page `helloworld_ext.jsp` therefore has two different style sources which might conflict each other. A cascading order is used to determine the applicable value, which also gave the stylesheet its name. Since the

declaration in the HTML document is declared `!important` it takes precedence.

Lesson Learned: When creating and changing style elements in an external .css file, the changes might not be immediately reflected on the actual web page. The reason being, that the web browser will usually cache front end resources. The hard refresh feature might prove useful by clearing the cache and reloading all resources. For example, a hard fresh can be performed by the button combination `CTRL-Shift-R` when using Firefox on Windows [92].

In contrast to the first example, this one uses the BSF tag library. Most importantly, when using this taglib, `::REQUIRES "BSF.CLS"` needs to be included at the end of a script. Without its addition, public routines, and classes of the BSF400Rexx framework will not be functional. When using the JSR 223 taglib the directive can be omitted. The `BSF.CLS` which is part of BSF400Rexx is used to define public routines and classes. For example, public routines offer functions such as the creation of Java Arrays, while public classes such as `BSF_PROXY` enable sending ooRexx Messages to Java proxy objects [93]. Therefore, to avoid inexplicable errors or missing content, it is good practice to always include this directive when using the BSF taglib.

For the usage with ooRexx, the choice of taglib hardly matters. Nonetheless, both the BSF and the JSR 223 tag library exist to ensure maximum compatibility and the ability to run programs without making any changes. Programming languages like Groovy internally prefer the BSF framework, while older languages might not offer JSR 223 at all.

When looking at the text that is printed, the BSF400rexx public function `pp` is used to place the text inside square brackets. Otherwise, all the difference in look and feel stem from the applied stylesheet.

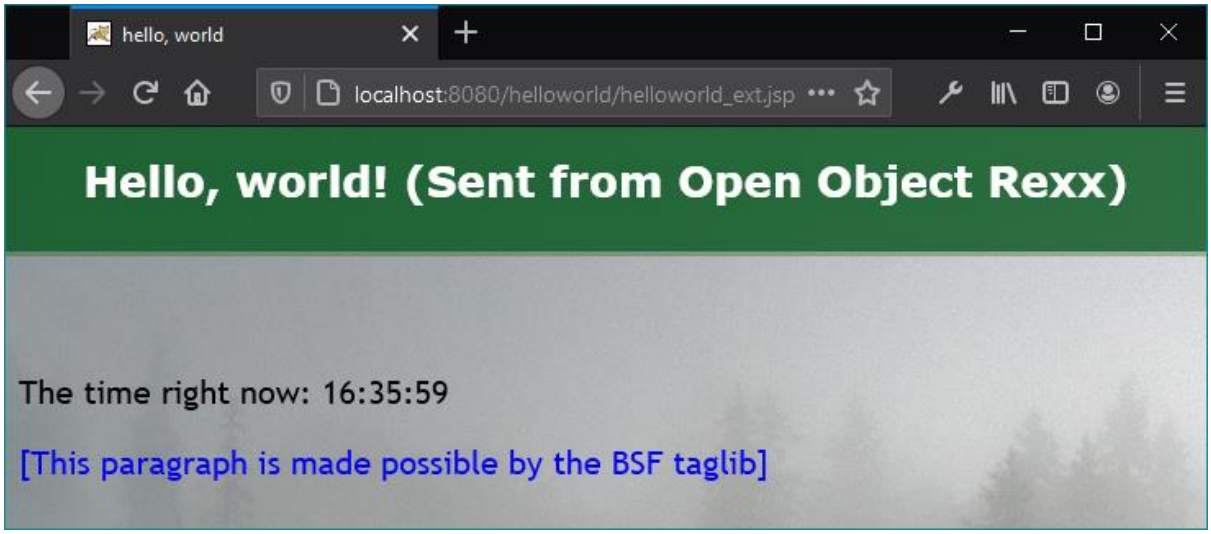

Figure 7: helloworld_ext.jsp in Web Browser

4.4. Welcome Files `/helloworld/index.html`

This is a good point to mention welcome files. If no specific page is requested, tomcat redirects the user to a welcome file. For example, the URL: `http://localhost:8080/treeshop/helloworld` opens the welcome page for the `helloworld` web application. A welcome file can be configured for the main directory, as well as all subfolders. The server will first look for a welcome file declared in the web application's `web.xml` file. It will check for the existence in the following order: `index.html` → `index.htm` → `index.jsp` [94].

4.5. An Introduction to Cookies `/helloworld/lastvisit.jsp`

In previous parts of this thesis the HTTP protocol and its stateless nature have been discussed. Cookies are a solution to the problem that different HTTP requests are considered separate from each other. Most website features that are taken for granted, like shopping carts, are enabled by cookies.

To maintain state, the server sends information in the `Set-Cookie` HTTP response header. When the user contacts the same server at a later point, the previously received cookie data gets sent in the `Cookie` HTTP request header. By changing the path and domain attributes, the scope of a cookie can be altered. By default, the cookie only gets sent to exact path from

where the web page has been requested from. Each cookie is represented by a cookie-pair consisting of `cookie-name` and `cookie-value`. Should the user agent receive a cookie with the same `cookie-name`, `domain-value` and `path-value` as an existing cookie, the stored data gets replaced with the newly received values [95, Sec. 8.6.].

Furthermore, cookies include a `Max-Age` attribute; after the stated number of seconds has passed, the cookie gets deleted. Similarly, cookies can include an `Expires` attribute, which indicates the time and date at which the cookie expires. Should the cookie have both the `Max-Age` and the `Expires` attribute, the `Max-Age` attribute takes precedence. The cookie's `Domain` attribute indicates the hosts it gets transmitted to [95, Sec. 8.6.].

While the first parts of the page `lastvisit.jsp` are identical to the previous example, the document's body contains code utilizing cookies.

```
<s:script type="rexx">
USE ARG request, response, out

lastVisit = .nil
allCookies = request~getCookies()

IF allCookies \= .nil THEN DO singleCookie OVER allCookies -- iterate over cookies
   IF singleCookie~name == "lastVisit" THEN lastVisit = singleCookie~value
END

IF lastVisit == .nil THEN SAY '<p>This is your first visit!</p>'
                     ELSE SAY '<p>Your last visit was at: 'lastVisit'</p>'

/* Create/Overwrite cookie with the current time */
cookie = .bsf~new("jakarta.servlet.http.Cookie","lastVisit",time())
cookie~setMaxAge(60*60*24) -- the cookie will expire after 1 day
response~addCookie(cookie)
</s:script>
```

Listing 5: lastvisit.jsp

The method `getCookies` is used to gather all cookies that are included in the request. The method results in an array of all transmitted cookies or `.nil` in case no cookies exist [96]. This array is assigned the variable `allCookies`.

If cookies are present, a `DO OVER` loop iterates over all the cookies contained in the newly created array. It is necessary to first check for the existence of cookies, since a `.nil` value for `allCookies` would result in an error. If a cookie with the name `lastVisit` is present, its value will be assigned to a variable of the same name. In this case, short message about the users last

visit will be displayed, otherwise he will be informed that this is his first visit.

Afterwards, a cookie is created by utilizing the class `Jakarta.servlet.http.Cookie`. A name and value are given with the constructor [97]. In the final step the method `setMaxAge` is used to define the cookie's expiration date. Since this value is given in seconds, a small mathematical operation is used to define a maximum age of one days. After the cookie is created, the `addCookie` method is used to add the cookie named `lastVisit` to the response that gets sent to the client. As value, the built in Open Object Rexx function `time()` is used to store a timestamp corresponding to the exact moment in time when the code is executed [98]. For all future visits, this timestamp will be displayed and afterwards updated. Unless the visits are further apart than one day, after which the cookie will be automatically deleted.

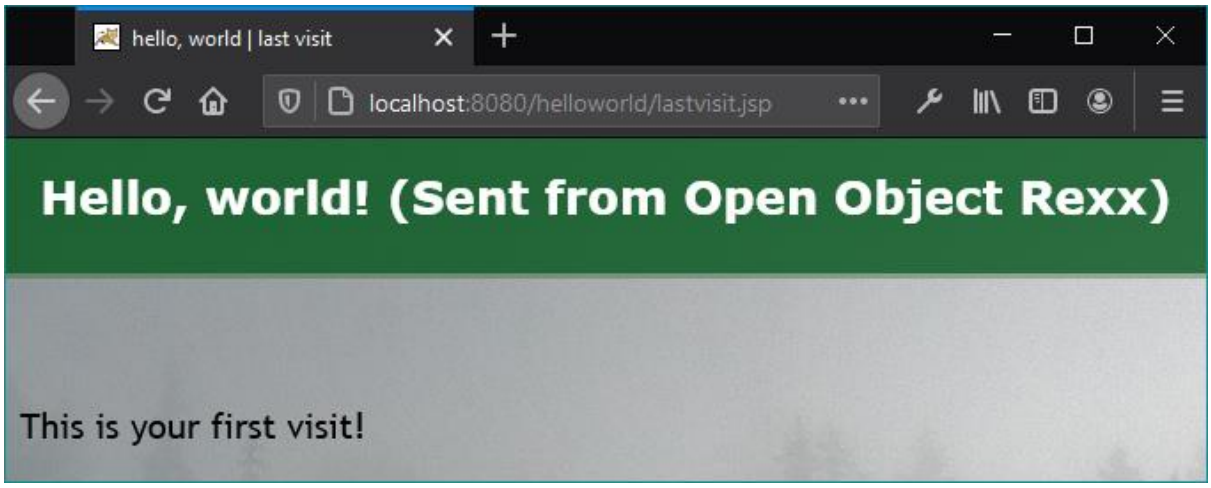

Figure 8: lastvisit.jsp in Web Browser First Visit

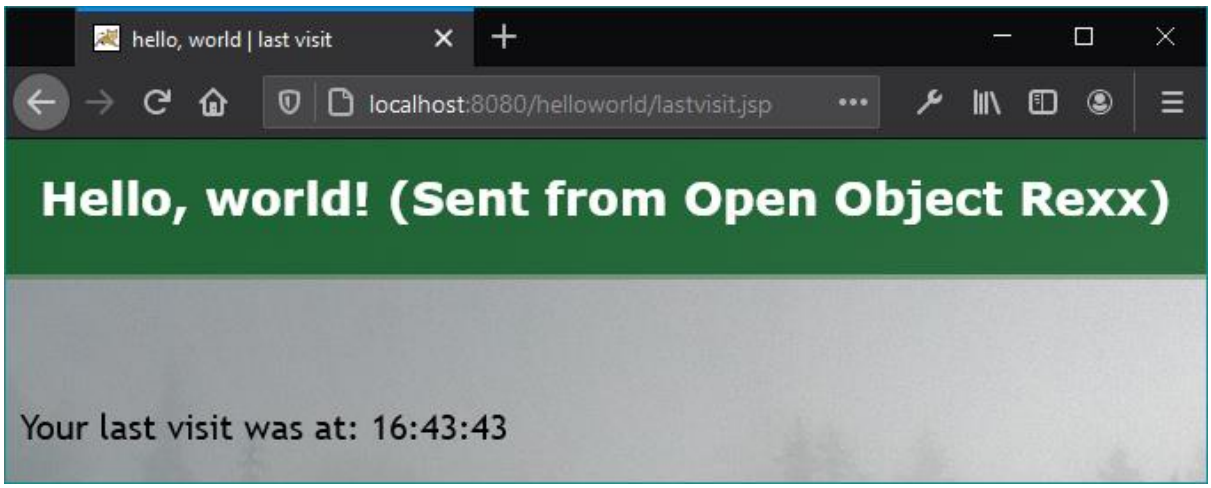

Figure 9: lastvisit.jsp in Web Browser Consecutive Visit

Although simple, this third nutshell example might prove useful; it becomes clear how cookies are created, transmitted, and accessed.

4.6. Combining User Input and Cookies `/helloworld/greeting.jsp`

The next example shows how information provided by the user can be stored and reused with a cookie. A visitor is asked for his name, for the web page to be able to personally greet him in the future. Again, the relevant code can be found in a scipt located in the document's body.

```
<s:script type="rexx">
USE ARG request, response, out

allCookies = request~getCookies()

username = .nil    -- set as nonexistant to begin
IF allCookies \= .nil THEN DO singleCookie OVER allCookies -- iterate over cookies
   IF singleCookie~name == "username" THEN username = singleCookie~value
END

IF username == .nil THEN DO
   SAY '<p>Hello, what is your name?</p>'
   SAY '<form>'
      SAY '<label for="username">Username:</label>'
      SAY '<input type="text" name="username" required>'
      SAY '<input type="submit" value="Ok">'
   SAY '</form>'
END
ELSE DO
   SAY '<p>Welcome back, 'username'</p>'
END

IF request~getParameter("username") \= .nil THEN DO
   cookie =
.bsf~new("jakarta.servlet.http.Cookie","username",request~getParameter("username")
)
   cookie~setMaxAge(60*60*24) --The cookie will expire after 1 day
   response~addCookie(cookie)
   response~sendRedirect(request~getRequestURI())  -- refresh page
END
</s:script>
```

Listing 6: greeting.jsp

After fetching the implicit objects and any cookies attached to the request, the script first checks whether a cookie called `username` is existent or not. Should none exist, a form is generated asking the user to input his name. Since no submission method is declared, the default method `get` is used, appending the data to the URL. The action attribute is used to define the

processing agent [99, Sec. 17.13.]. To clarify, here the web page is given, to which submitted the data is sent. Since this is a single page application, the form data should be sent to the page itself. For HTML5 it is sufficient to simply omit the action attribute to achieve this. HTML4 on the other hand, requires a value for the action attribute to function [100].

Lesson Learned: In HTML, elements like `input type="text"` should always be accompanied by a label. This will help users who use screen readers or have trouble clicking on small fields [101].

To avoiding unexpected behavior and potential malfunctions it is important to design a web page in a way to make common mistakes impossible. One such scenario would be a user not entering his name at all, for example by prematurely submitting the form. This scenario would result in an awkward greeting message, without any name given. Avoiding this can be done without any additionally code in a simple but elegant way. The `required` attribute can be given for input types such as `text`, `url`, `email` or `password`, only allowing forms to be submitted if the field has been filled [102].

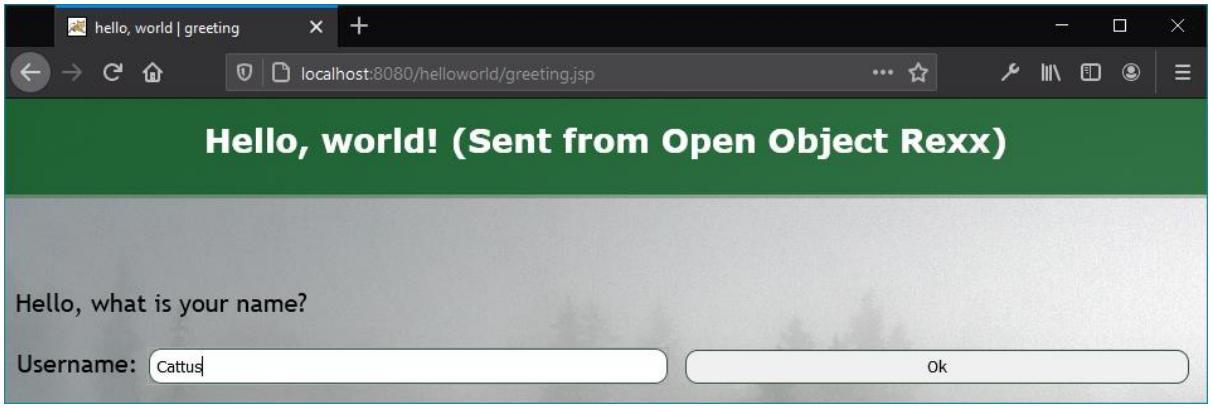

Figure 10: greeting.jsp in Web Browser First Visit

Like any other program, all code is executed top to bottom. The last block of code is activated, should the user submit the previously generated form. Once the form is submitted the user is redirected to the same page and the values entered are contained in the request. The method `getParameter` is used to fetch this information. This name for the parameter has been previously declared inside the form using the `name` attribute.

Only if the page is accessed by means of a form submission, the request will contain the parameter `username`. In this case, the `IF` loop at the end of the program is activated and a cookie is created. Finally, the `sendRedirect` method of the response object is used to refresh the page. This is accomplished by fetching the current page address with the `getRequestURI` method. Since the request now contains a cookie called `username`, the form and cookie creation are skipped, and the personalized greeting displayed instead.

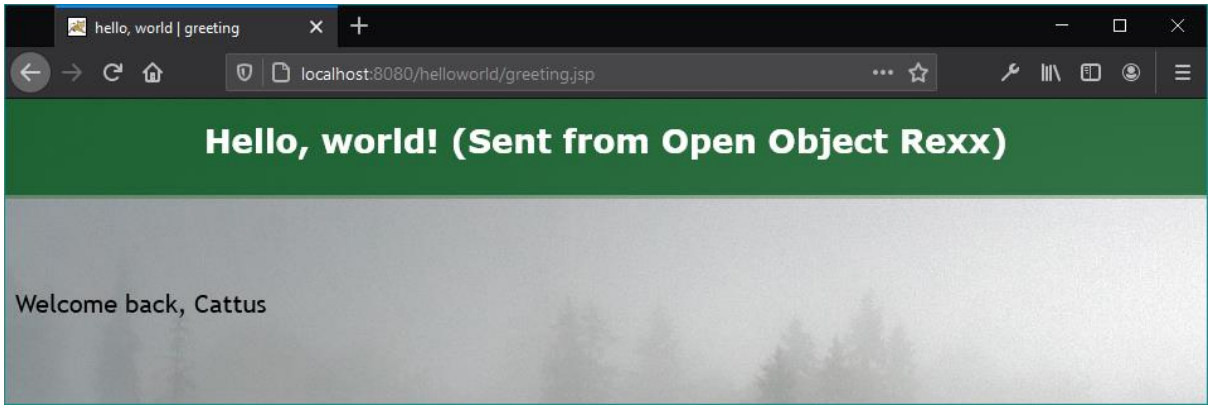

Figure 11: greeting.jsp in Web Browser Consecutive Visit

4.7. Deleting Cookies, External Scripts `/helloworld/greeting_ext.jsp`

The next example page `greeting_ext.jsp` builds on the previous one, adding a logout button to demonstrate how the previously entered name can be removed again.

At first glance the page looks almost identical, yet the structure has been improved. Depending on the existence of a cookie containing a username, either a login form or a logout button is displayed. The HTML code to generate those components is stored in a `::RESOURCE` directive, which can be found at the bottom of the script.

```
::RESOURCE logoutButton
<form>
   <input type="hidden" name="logoutButton" value="1">
    <input type="submit" value="Logout">
</form>
::END
```

Listing 7: greeting_ex.jspl ::RESOURCE logoutButton

A `::RESOURCE` contains an unlimited number of strings up until the `::END` directive, which are stored in a `Stringtable`. The name given to the resource

serves as an index, in this case, `logoutButton`, for which all the lines given are stored in an array. Afterwards the entries are fetched with the environment symbol `.RESOURCES` and the given name `~logoutButton` [103, p. 3]. This feature of ooRexx can prove particularly useful for reusing HTML building blocks. Another benefit of this feature is the ability to effortlessly write HTML code without quotation marks.

Additionally, this example highlights how an input of the type `hidden` is used to attach data to a request. After the button is clicked, the request will include the parameter `logoutButton` with the value 1 attached. Essentially, information about the user's previous actions are transmitted, enabling state without utilizing cookies.

With this newest addition the web page has four possible behaviors, depending on the information attached to the request. In case a cookie containing a username is present a personalized greeting is displayed, otherwise the user will be asked to input a name.

Should the request indicate that the user has just filled the form or wishes to logout, a cookie needs to be either created or deleted. While the first two behaviors are programmed directly in the JSP, the latter are implemented by an external script.

The following line of code adds the external script to the body of the document:

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

Listing 8: greeting_ext.jsp src Attribute

The linked script `logout.rex` is stored in the directory `helloworld\code`. The script tag's `src` attribute allows the addition of an external file, containing scripting code in the specified language. Like the addition of a `.css` file, the path is relative. Additionally, the `cacheSrc` attribute is set to `true`. If a web page is still under development, it is highly recommended to set this attribute to `false`, preventing the file from being cached and instead rereading it each time. If the attribute is omitted, it is set to `true` by default. This results in the caching of the file, necessitating a full server restart for changes to be reflected on the JSP [85].

Additionally, multiple optional attributes for the `script` and `expr` tags are available, mainly for the purpose of debugging. The sample web application that comes included with the tag libraries, `Jakarta.demorexx` (see download links at the beginning), contains examples showcasing these features.

```
USE ARG request, response, out

IF request~getParameter("username") \= .nil THEN DO
   cookie =
.bsf~new("jakarta.servlet.http.Cookie","username",request~getParameter("username")
)
   cookie~setMaxAge(60*60*24) -- the cookie will expire after 1 day
   response~addCookie(cookie)
   response~sendRedirect(request~getRequestURI()) -- refresh page
END

IF request~getParameter("logoutButton") \= .nil THEN DO
   removerCookie = .bsf~new("jakarta.servlet.http.Cookie","username","")  --
overwrite existing cookie
   removerCookie~setMaxAge(0) --The cookie will expire immediately
   response~addCookie(removerCookie)
   response~sendRedirect(request~getRequestURI()) -- refresh page
END

::REQUIRES "BSF.CLS"     -- make sure the Java bridge is there
```

Listing 9: logout.rex

Just for any other script, the implicit objects are made available to the external script. The first `IF` loop contains the previously used code to store a username in a cookie. Should the request contain any value for the parameter `logoutButton`, the second `IF` loop is activated. Since there is no specific method to delete cookies, instead a new cookie with the same name and an empty value is created to set its `maxAge` to zero [104]. By adding this cookie to the response, the existing cookie is being replaced and the page gets refreshed afterwards. The `maxAge` attribute results in the cookie being deleted after zero seconds have passed. The user is then presented with the previous page that contains the form to fill in a username.

While the page would still function, should the tag referring to an external script be at a different position, it has been placed at the beginning of the body on purpose. First, placing it inside the document's head might seem intuitive, but could easily result in it being overlooked. Afterall, the first lines of web applications, might consist of copy and pasted building blocks, making modifications inconvenient. Since the page is executed like any other program, from top to bottom, placing it at the end of the body might

result in unnecessary loading times. If any of the two `IF` loops are activated the page is refreshed, rendering the creation of other parts useless.

5. Database Connection

More sophisticated web applications require access to a persistent data source and the ability to freely add, delete and modify information. The separation of data and functionality offers a high flexibility and the chance to improve and update components separately. The ability to easily backup critical data is also a requirement for most operations. The following chapter describes the components necessary to connect a web application to a database. Afterwards, the configuration steps required are demonstrated.

5.1. Java Database Connectivity

In general, the reader may use any database management system of his choice, the only requirement being the availability of the Java Database Connectivity (JDBC) API.

JDBC is used to connect to a database, issue queries and commands, and to handle result sets. It can be implemented for both client-side and server-side connections. In a first layer, the Java application communicates with the JDBC manager through the JDBC API. Afterwards, in a second layer, the JDBC manager communicates with the database driver [105].

Each time a user connects to the database, resources are committed to creating, maintaining, and closing the connection. To allow a high number of users simultaneous and responsive access, the connections can be pooled and reuses by means of connection pooling. Instead of closing and reopening connections for every request, the connections are cached and consecutively reused. For example, each PostgreSQL connection can take up to 1.3 megabytes in memory, multiplied by the number of connections, this number can easily skyrocket [106].

"It lets your database scale effectively as the data stored there and the number of clients accessing it grow. Traffic is never constant, so pooling can better manage traffic peaks without causing outages." [106].

Nonetheless, connection pooling can result in problems, if handled incorrectly. A so-called database connection pool leak can occur if a web application does not explicitly close objects related to the database connection, resulting in those resources being unavailable and a failure of the data connection [107].

5.2. Java Naming and Directory Interface

In many cases applications utilize different services, provided by different components. For the given use case, a web application needs to find a database. The Java Naming and Directory Interface (JNDI) allows for different components to find each other.

Especially for distributed system, naming services are of great importance. Innovations like powerful microprocessors, high-speed computer networks and the miniaturization of computer systems have made distributed systems a possibility. Multiple autonomous computing elements are working together, while appearing as a single coherent system to the user [108, pp. 967-968]. For example, it might be plausible for the web application and the database to be running on different machines.

Names are used to refer to an entity, which can be practically anything, for example it could be a host or a file. Those entities can be then used to perform operations on them. Each entity has one or multiple access points, which are a special kind of entity. Their name is called an address. For example, a host, running a webserver is an entity whose access point is a combination of IP address and port. Since addresses are usually not readable in a human friendly way and might change over time names are preferred [109].

Not only offers JNDI a single location for programs to find resources, but it also provides a common interface to existing naming services. Additional to naming, JNDI also offers directory service, which manages the storage and distribution of shared information [110].

6. E-Commerce Example `/treeshop`

From this point onward, all examples will be based on a fictional company, selling trees to be planted in the name of a buyer. A webpage has been created to sell their products, including the ability for users to login and access a shopping cart. Furthermore, administrators of the web site can add new products and send promotional e-mails to customers. All content will be dynamically created according to entries in a database.

The data structure is kept minimalistic on purpose, only containing three tables with basic data. One to hold the products, another for the customers and a cart to connect them, realizing a many-to-many relationship. The following Entity-relationship model is representative for the necessary database entries:

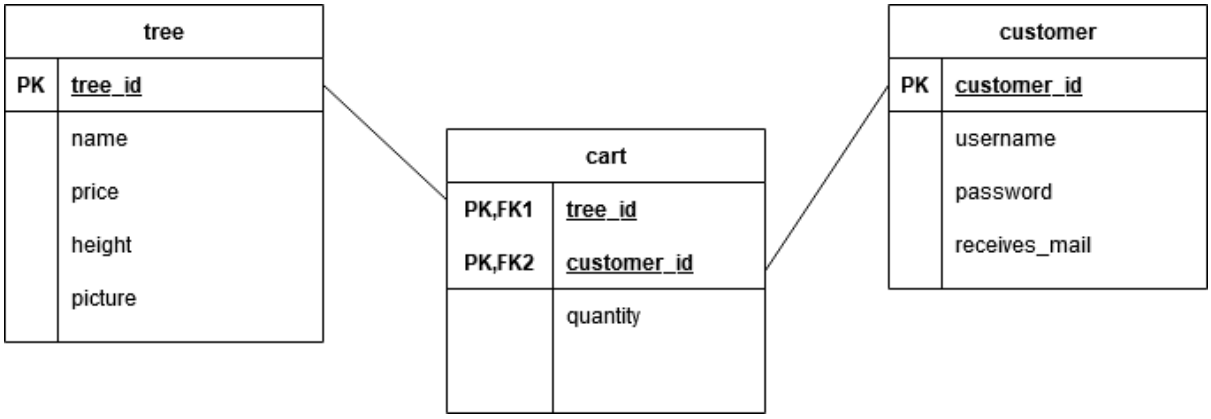

Figure 12: Entity-relationship Model Database

6.1. Setup

Highly Recommended: All setup steps are summarized to be viewed and copied from the URL: `http://localhost:8080/helloworld/support/`

For the examples to function the user is required to perform three configuration steps. First, Tomcat's configuration needs to be changed to enable the server to serve static files, like pictures. Then, a database management system needs to be installed and set up. Finally, two `.jar` files need to be copied to `TOMCAT_HOME\lib`.

6.1.1. Serving Static Content

In general, files can be served directly by a web application using the `DefaultServlet`. But, since web applications are often deployed from `.war` files, any changes made, would require redeployment [111]. Additionally, between redeployments, files might get lost.

Since the shopping website is intended to keep functioning, even if new products are dynamically added, this approach would not work. Later examples will introduce a way to add new products, including pictures, to the database. These pictures are to be stored in a directory outside of the web application, with their path stored in the database.

To enable Tomcat to serve them and any other static content like stylesheets or HTML pages, some extra configuration steps are necessary.

The file `server.xml` is can be found in `TOMCAT_HOME/conf`. The `<host>` element can be found at the bottom of the document and needs to be extended with the following line: `<Context docBase="C:\Program Files\Apache Software Foundation\Tomcat 10.0\files\" path="/files" />`

```
        <Context docBase="C:\Program Files\Apache Software Foundation\Tomcat
10.0\files\" path="/files" />
      </Host>
    </Engine>
  </Service>
</Server>
```

Listing 10: server.xml Context

`Docbase` is used to indicates the directory from which static files are read from. For this purpose, a new folder called `files` needs to be created in the main Apache Tomcat directory. Generally, the direct path to any folder on the computer running Tomcat can be given. `TOMCAT_HOME` has been chosen, since it is assumed that all readers have a folder with the exact or at least similar path.

Furthermore, the `path` attribute is used to define the URL that files will be made accessible from. After the changes have been saved, any files placed in the files folder will be accessible from the URL: `http://localhost:8080/files/Maple.jpg` [111]. This path is also used by the following examples to load pictures from.

The directory of the complementary archive `ZIP_ARCHIVE\supportfiles` contains folder named `files` that has already been set up with six sample product pictures. For the page to be properly displayed it is recommended to copy this folder to `TOMCAT_HOME`.

6.1.2. Database Configuration

For the web application to function, the database needs to hold three tables, containing specific columns. The appendix contains detailed instruction starting from downloading the software to adding six example products: `9.4.` PostgreSQL

6.1.3. Tomcat's Handling of `.jar` Files

By default, Tomcat creates four class loaders, while ignoring the `CLASSPATH` environment that is used by standard Java environments. The loading of classes also slightly differs from what is standard practice for Java, where classes are in a parent-child relationship to each other [112].

In the default configuration of Tomcat, the class loader on top of the hierarchy is called `Bootstrap` and loads classes provided by the Java Virtual Machine and the extensions directory of the Java Runtime Environment. Next, the `WebappX` class loader makes classes available to a specific web application. To accomplish this, it looks for classes located in the directories `WEBAPP\WEB-INF\lib` and `WEBAPP\WEB-INF\classes`. Next the `System` class loader loads classes required to initialize Tomcat as well as classes for logging and the Apache Commons Daemon project. Only then, the `Common` class loader loads classes from `TOMCAT_HOME\lib` [112].

Systemwide, the BSF4ooRexx library is only loaded once. Placing it in multiple `WEBAPP\WEB-INF\lib` directories will result in it being requested multiple times and the scripting content not working. Only the first application using it will function normally. Therefore, the `bsf4ooRexx-v641-20201217-bin.jar` needs to be placed in the `TOMCAT_HOME\lib` directory.

Additionally, it is recommended to place all JDBC related `.jar` files in the `TOMCAT_HOME\lib` directory as well. This is due to a broken service provider

mechanism, which is supposed to enable the drivers to announce themselves without specific registration. Tomcat's JRE Memory Leak Prevention Listener fixes this issue by loading all drivers on server startup. If the `.jar` file is placed inside the web application though, the listener will not be able to find the driver. Instead, it will be loaded by the first web application requiring it. This approach can lead to various errors and unexpected behavior [107].

To summarize the files `postgresql-42.2.18.jar` and `bsf4ooRexx-v641-20201217-bin.jar` need to be copied from the `ZIP_ARCHIVE/supportfiles` directory to TOMCAT_HOME\lib. Additionally, should the reader wish to keep using the `helloworld` application, it is recommended to delete the file `bsf4ooRexx-v641-20201217-bin.jar` from `helloworld\WEB-INF\lib`. It has originally been included to provide an easier introduction.

6.2. Reading Data `/treeshop/productlist.jsp`

For database access to function, the webapp's `WEBAPP\META-INF` directory needs to contain a file called `context.xml`. This context is used to specify additionally required configuration information. While entering the data source solely in this file is sufficient, it is recommended to also define the resource in the previously mentioned `web.xml` file, mainly to document a web application's resource requirements [113]. Since these files are specific to the web application, it is already configured accordingly, requiring no action from the reader.

```
<Context>
<Resource name="jdbc/postgres" auth="Container"
          type="javax.sql.DataSource" driverClassName="org.postgresql.Driver"
          url="jdbc:postgresql://127.0.0.1:5432/shop"
          username="cattus" password="tomtom12" maxTotal="100" maxIdle="10"
        maxWaitMillis="-1" removeAbandonedOnBorrow="true"
removeAbandonedTimeout="60" />
</Context>
```

Listing 11: context.xml

First, the name to be used by JNDI and attributes relating to the driver are specified. The `url` attribute is used to point to the database server's IP and database name. For the following example, the default values of a PostgreSQL database are used. Should the reader prefer a different

database management system the entries need to be adjusted accordingly. It is also necessary to specify the username and password of a previously created user. The web application will use the given credentials to login and perform operations within the database. It is beneficial to create a unique user, since only the minimum rights need to be assigned and actions taken by the application can be quickly identified.

Lesson Learned: When working with databases, the configuration is important. Should the given user not have the necessary permissions, nothing will work. Additionally, the problem's source cannot be easily identified using Tomcat's logs. Should inexplicable problems occur, it is therefore recommended to apply the debug method shown in the appendix: **`9.5. Debug Code`**

Additionally, to mitigate any possible database connection pool leaks the two attributes `removeAbandonedOnBorrow` and `removeAbandonedTimeout` are added. After a database connection has been left idle for the specified amount of time, it is terminated automatically [107].

To begin with, `productlist.jsp` gives a quick overview of the products currently listed in the database.

```
<s:script type="rexx">
cntxt=.bsf~new("javax.naming.InitialContext")
ds = cntxt~lookup("java:/comp/env/jdbc/postgres")
con = ds~getConnection()  -- connect to database

stmt = con~createStatement()
qry = "SELECT * FROM tree;"
rs = stmt~executeQuery(qry) -- retrieve all data from the table

/* Create a list using product names and prices */
SAY '<ul>'
DO WHILE rs~next() -- iterate through all the rows stored in the table
    SAY '<li>'rs~getstring("name")':' rs~getstring("price")'€</li>' -- for each
row, fetch data
END
SAY '</ul>'

/* Close resultSet, statement and connection */
rs~close()
stmt~close()
con~close()
</s:script>
```

Listing 12: productlist.jsp

First, the `InitialContext` class gets instantiated. Since this class is already included in the Java Standard Edition, no additional class files are needed. A context represents a set of bindings that all share the same naming convention. The created object gives access to the most basic methods, like naming or looking up an object [114]. The use of Tomcat further simplifies the configuration since it provides a JNDI `InitialContext` implementation instance that gets configured for each web application during its initial deployment. Resources are placed in the JNDI namespace under `java:comp/env` [113]. No further JNDI configuration is necessary and the database can be accessed effortlessly from the previously defined path, using the `lookup` method.

After the `context` has been configured the `getConnection` method is used to establish a connection. Once again it is recommended to use the shown three lines of codes as standard building blocks for future web applications.

The `executeQuery` method of the `Statement` Interface uses a statement a SQL statement, as input parameter and returns a `ResultSet` object. This object contains the resulting data of the query which in turn is used to generate HTML code to display information to the user [115].

The `ResultSet` Interface represent the data of the query in form of a table. The data is navigated by a cursor, which is initially at a position before the first row. The `next` method is used to advance the cursor along the table's rows. By default, the type is set as `TYPE_FORWARD_ONLY`, meaning that it is not possible to go backwards and that the object is not sensitive to changes to the underlying data. The `ResultSet` offers a multitude of methods to access the desired data, for example the `getString` Method can be used to retrieve data from a column by name [116]. A `DO` loop iterates through entries of the result set, each representing a row in the output of the database query.

This example's query resulted in a row for each tree, for which its name and price are fetched and displayed. Afterwards, the cursor of the `ResultSet` is advanced to the next row, until none are left.

For good practice and to avoid errors, the `ResultSet`, `Connection`, `Statement` and the later discussed `preparedStatement` should always be explicitly

closed. Especially with connection pools it is uncertain at what time `statements` and `preparedStatements` are otherwise closed [117].

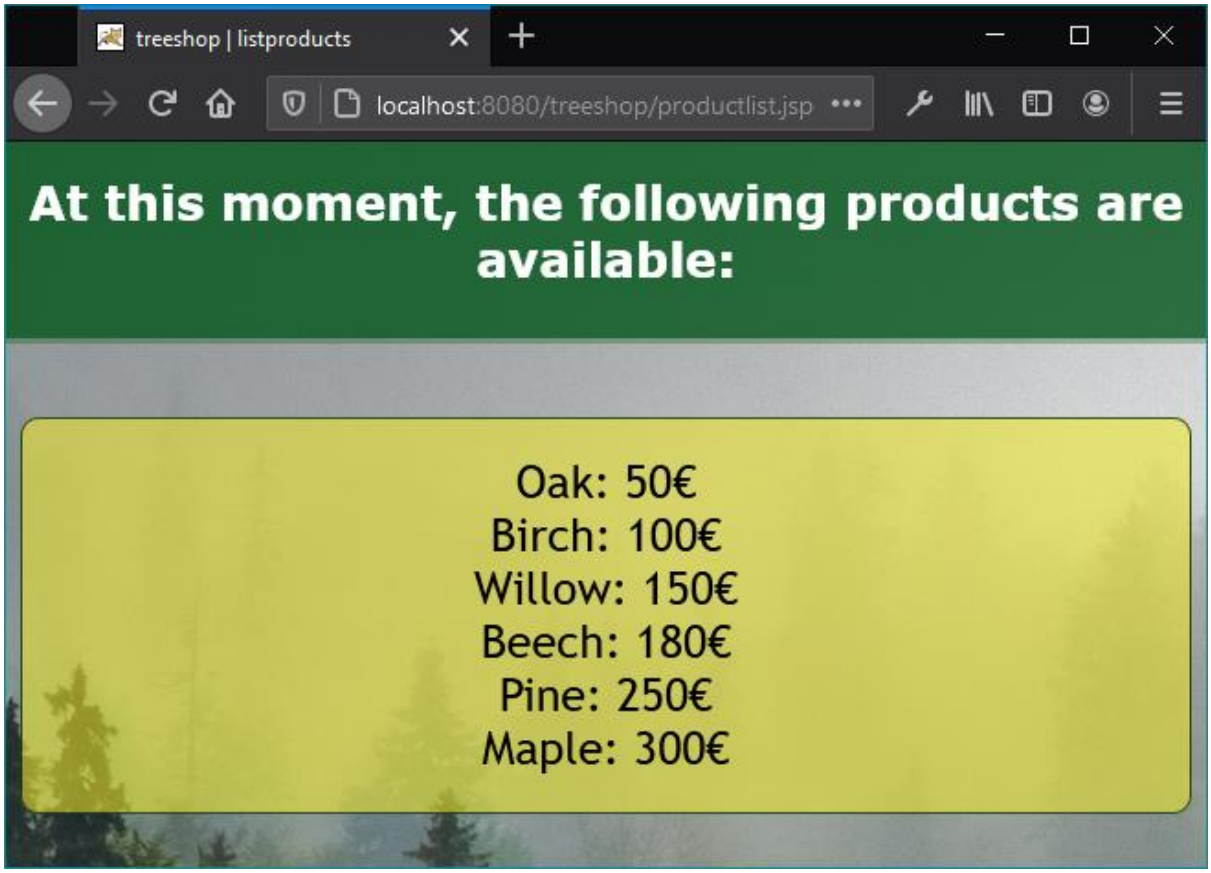

Figure 13: productlist.jsp in Web Browser

6.3. Writing Data `/treeshop/signup.jsp`

Most dynamic web applications not only make data available, but also allow the user to interact and provide new data. As a minimum, almost all modern websites allow users to create an account.

While the writing of new data is relatively straightforward and quite like the previously shown solution, the storage of user provided data requires the consideration of additional security aspects. Not only does the user's data need to be stored safely, but the web application itself needs to be protected from unwanted manipulation by ill-intentioned actors.

The page `signup.jsp` starts like any other and continues to display a form for a user to enter a username in form of an e-mail address and a password, which needs to be repeated. Once again, all three fields have been set as `required`, not allowing the user to proceed without filling them first. This

is particularly important, since form fields left blank might result in erroneous database entries.

Additionally, a checkbox can be ticket, for users who wish to receive promotional e-mails. The automatic generation of said e-mails will be implemented in a later example. At this point the user's consent is requested to flag the newly created account accordingly. Even though, according to the European Union's General Data Protection Regulation, direct marketing E-mails about products or services can be sent to existing customers, any other promotional E-mails require prior consent [118]. Generally, it is good practice to only send e-mails to users who explicitly wish to receive them, not only to avoid complaints, but also to build a positive brand image. Consent should be given in the form of a clear, affirmative action; therefore, the checkbox needs to be explicitly clicked on and the corresponding label is not formulated ambiguously [119].

After the form data has been transmitted to the web server, the external script `createuser.rex` is activated and only progresses if the user has given the same password twice.

6.3.1. `get` and `post` Methods

The first important difference can be observed in the form using the `post` method, instead of the default, `get`.

`post` signals the webserver that data is being sent and attaches it to the body of the message. In contrast, the data transmitted by a `get` request is appended to the URL and therefore easily visible. Not only might it be concerning for the user to see his potentially sensitive data such as passwords in plain text, `get` requests are usually cached by web browsers and might additionally appear in their history. In conclusion, it is good practice to default to `post`, especially when dealing with forms of this nature [120].

6.3.2. Securely Storing Passwords

It is imperative not to store users' passwords as plain text. Vulnerabilities previously unknown or other security risks might result in a compromised database. Passwords are especially sensitive since users might use the same password for multiple websites [121].

The suggested solution to safe password storage is the application of a cryptographic hash function. They take an input (preimage) and generate a unique cryptographic fingerprint (digest) for it. Each fingerprint is unique to the input and irreversible making it impossible to backtrack to the original input [122].

"A hash function is a function that deterministically maps an arbitrarily large input space into a fixed output space" [122].

Since some users might use similar or identical standard or basic passwords, that might result in the same fingerprints, the concept of salting is introduced. Otherwise, an attacker in possession of all the stored hash values, who managed to guess one of them correct, might would have access to all the user accounts using the same password. Before the cryptographic hash function is performed a unique, randomly created string is attached to the user's password. [123].

When it comes to implementation, the Open Web Application Security Project (OWASP) suggests to strictly using third party libraries implementing the necessary algorithms. While Java itself offers cryptographic functionality and the creation of a message digest, there is too much room for error when creating a custom solution. For example, the widely used SHA-256 algorithm is simply too fast. OWASP recommends the Bcrypt hashing algorithm as the default choice [123].

The Blowfish encryption algorithm, developed by Provos and Mazières, allows users to increase the verification time to adjust it to increasing processor speeds, by modifying the cost value. It is based on their Eksblowfish Algorithm and offers a possible salt space so large, that it makes the precomputation of hash values based on common passwords incredibly difficult, since the required storage would be enormous [124, pp. 6-11]. The goal should be finding a balance between performance impact

and security, tailored to the CPU speeds of the current day and age. It is also worthy to note, that should the algorithm be too taxing, an attacker might be able to perform a denial-of-service attack on the webserver [123].

Most curiously, the implementation of the Bcrypt not only makes the process of password storage much more secure, but also extremely simple. Damien Miller offers the functionality of Bcrypt in form of a convenient java library called `jBcrypt`, which comes included with the `treeshop` web application. Although already included the latest version of which can be downloaded from: `https://www.mindrot.org/projects/jBCrypt/`

```
bcrypt=.bsf~new("org.mindrot.jbcrypt.BCrypt")
fingerprint = bcrypt~hashpw(pw1,bcrypt~gensalt(12)) -- create a hash value that is
safe to store
```

Listing 13: createuser.rex jBcrypt

The method `hashpw` takes the user's input and a salt value to output the hash value in string format. Since the library also offers a secure method to create the salt value, the method `gensalt` is used. Most curiously, this method takes the previously discussed work factor as input. Even though `jBcrypt-0.4` uses a work factor of 10 as default, OWASP recommends raising it to 12 [123].

6.3.3. SQL Injection

Before showcasing how the username and hashed password are stored, injection flaws need to be discussed. OWASP identifies injection as the number one web application security risk. A hostile user might send untrusted data as part of a command or a query, to trick the interpreter to execute unintended commands or accessing data without authorization [125].

Su and Wasserman find web applications being susceptible to a large class of malicious attacks know as command injection attacks: *"This is because queries are constructed dynamically in an ad hoc manner through low-level string manipulations. This is ad hoc because databases interpret query strings as structured, meaningful commands, while web applications often view query strings simply as unstructured sequences of characters"* [126, p. 1].

For example, on an unprotected database anybody could enter `username' OR '15' = '15` in a form prompting for a username and leave the password field blank. Should the web application forward this request directly to the database resulting in the SQL query: `SELECT * FROM username WHERE user = 'username' OR '15' = 15' AND password = '';`. As a result, the output would include all data available for the specified user.

There is a quite simple solution to this problem, instead of the previously used `statement` interface the extended version `PreparedStatement` is used. When an SQL query is executed, it first gets parsed and compiled. Afterwards, the data acquisition path is planned and optimized. In the final step the query is executed, and the result gets returned. In comparison to the normal `Statement`, which goes through the four steps when the query is executed, `PreparedStatement` performs the first three steps when the statement is created and only performs the last step during execution. Not only does this increase the speed of database access and allow other features like batch processing, but all special characters are automatically escaped [127]. Escaping special characters results in them being treated as regular parts of a string [128]. The above-mentioned exploit is therefore not possible since the user input is strictly treated as a normal string with no power to change the database query. In conclusion, when dealing with database queries based on user input, the minimum-security measure suggested is using the `PreparedStatement` interface.

```
prepstmt = con~prepareStatement("INSERT INTO customer (username, password) VALUES
(?,?)")
prepstmt~setString(1,username)
prepstmt~setString(2,fingerprint)
prepstmt~executeUpdate() -- add new user to database
prepstmt~close()
```

Listing 14: createuser.rex prepareStatement

The parts of the query where user input is used are omitted and instead filled with question marks. In the next lines the `setString` method is used to replace the question marks with the user input.

6.3.4. Hypertext Transfer Protocol Secure

Above all, the most important security measures necessary to facilitate a secure web application is the use of the Hypertext transfer protocol secure

(HTTPS) in place of regular HTTP. The Transport Layer Security (TLS) protocol, which was formerly known as Secure Sockets Layer (SSL) protocol is used to achieve this by means of an asymmetric public key infrastructure. All information sent with the HTTP protocol is sent as plaintext and therefore extremely vulnerable [129].

To prove one's identity it is necessary to obtain a certificate for a domain from a certificate authority. Since this work focusses on the implementation of a Tomcat server on a local network, the HTTPS protocol is not further discussed. Should the reader decide to make his web application accessible over the internet, the author encourages the use of HTTPS as an absolute necessity.

Since this work's focus is not the security aspect, the used measures are suggested as a bare minimum, with encouragement to invest additional time in research for real-world use cases.

6.4. Creating a Dynamic Web Page, Sessions `/treeshop/index.jsp`

After establishing how database access works, now the focus shifts to the main page of the `treeshop` web application.

The main difference compared to previously shown Jakarta Server Pages is the usage of sessions. While sessions utilize cookie technology, they are more advanced and require the server to store data for each user. In contrast to cookies, the user only stores and transmits her session id, which the server uses to access data corresponding to it. For web development, using sessions is quite like using cookies, the only difference being slightly different methods used.

Since the data is stored by the server and does not be transmitted, the usage of sessions is more secure. Compared to cookies, which have a maximum size of 4 kilobytes, sessions can hold up to 128 megabytes each. To summarize, sessions and cookies both store user related data, the first on the server, the latter in the web browser [130].

The Apache Tomcat container uses the HTTP Session interface to create a session id for each user and stores it in a cookie called `JSESSIONID`, which

then gets sent with each request. If cookies are disabled, the URl is rewritten [131]. Consecutively objects related to a session id can be bound to it, as well as general information about the session accessed or manipulated. For example, the time of session creation can be requested with the method `getCreationTime` [132]. To enable sessions, the `session` attribute of the `page` directive needs to be set to `true`.

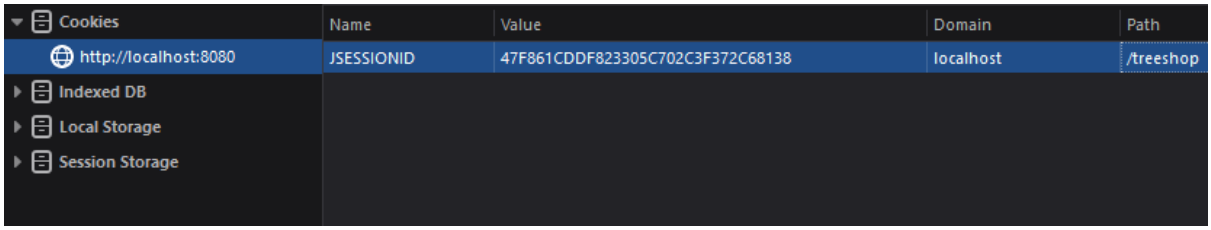

Figure 14: JSESSIONID Cookie

Since the default timeout value for Tomcat sessions is only thirty minutes, the `web.xml` file needs to be adjusted to extend this duration. By changing the `session-timeout` attribute of the `session-config` keyword the lifetime can be easily extended [133]. Once again, the `web.xml` specific to the web application is edited, therefore the reader is not required to change it for the `treeshop` web application. The value has been set to 60*24 minutes, as a result a guest's data will be deleted after one day.

The main page of `treeshop` is created by the file `index.jsp`, two ooRexx scripts are used to build its components:

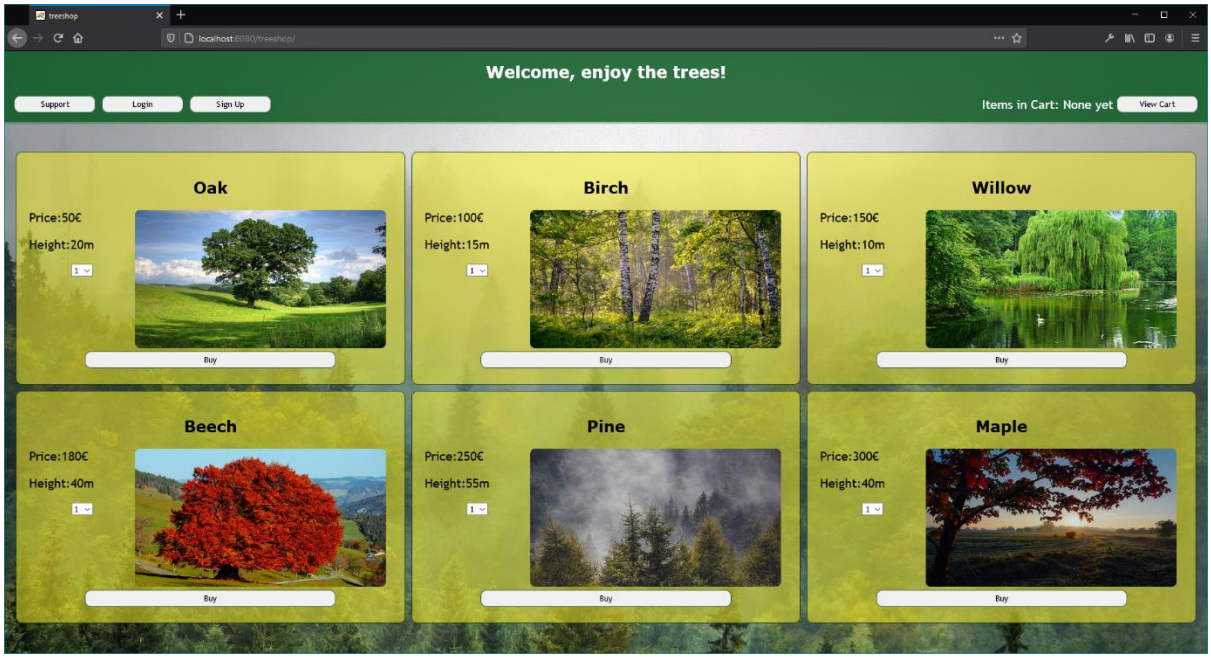

Figure 15: treeshop Main Page in Web Browser

6.4.1. mainpage.rex

The body of the shopping websites main page is created by the external script `mainpage.rex`. After the implicit objects, request, response and out are fetched, the method `getSession` is used to get access to data related to the session. The `sessionid`, which has either been sent as a cookie or is created on first visit is used to identify the user.

To begin with, the contents of the main page are built by querying all entries for the table tree, which contains all available products and information related to them. The `ROUTINE createProduct` uses this data to create a box for each product, displaying related information and enabling the user to put a specified quantity into her basket.

```
::ROUTINE createProduct
PARSE ARG name, picture, price, height, tree_id

SAY '<div class="grid-item">'
    SAY '<h2>'name'</h2>'
    SAY '<img src="'picture'" height="100" width="100" />'
    SAY '<p>Price:'price'€</p>'
    SAY '<p>Height:'height'm</p>'

    SAY '<form name="choice" method="post">'
        SAY '<input type="hidden" name="choice" value="'tree_id'">'
        SAY '<select name="quantity">'
            SAY '<option value="1">1</option>'
            SAY '<option value="2">2</option>'
            SAY '<option value="3">3</option>'
            SAY '<option value="4">4</option>'
            SAY '<option value="5">5</option>'
        SAY '</select>'
        SAY '<input type="submit" value="Buy">'
    SAY '</form>'
SAY '</div>'
```

Listing 15: mainpage.rex ::ROUTINE createProduct

The attribute `src` of the `<img>` tag specifies the URL of an image. The database entries for the images all look the same way: `/files/Imagename.jpg`. The slash at the beginning of the path indicates a relative URL, referring to the current page. Therefore, the web page loads the image from: `http://localhost:8080/files/Maple.jpg` The main advantage of this approach is that, should the domain change, the web application will still work as intended [134]. Another benefit is the opportunity to easily modify the page. In case the pictures need to be

loaded from another web page, the only thing that needs to change is the URL in the database.

The script contains the necessary code for two approaches of handling the shopping cart. Depending on the session containing the attributed `logged`, the quantity chosen in the form at the bottom of each box is processed differently.

A guest user's shopping cart is stored in a simple java array, consisting of integers for both the index and the corresponding element. The index refers to a product id, while the element specifies the amount in the shopping basket. The id is used to identify an item in the database. The array is stored in the session, enabling it to scale in the future. Special attention is given to products already present in the cart, instead of overwriting the quantity it needs to be updated.

```
IF session~getAttribute("cart") == .nil THEN DO
    cartArray = .bsf~bsf.createJavaArray("java.lang.String",100)   -- create a new
cart if it doesn't exist
    session~setAttribute("cart",cartArray)
END

cart = session~getAttribute("cart")
IF cart[choice] == .nil THEN DO
    cart[choice] = quantity -- add a new product to the cart
    session~setAttribute("cart",cart)
END
ELSE DO
    cart[choice] = cart[choice] + quantity  -- update the quantity of an existing
product
    session~setAttribute("cart",cart)
END
```

Listing 16: mainpage.rex cartArray

Should the user be logged in, her shopping cart is stored in the database instead, using a `preparedStatement`.

```
qry ="INSERT INTO cart (customer_id, tree_id, quantity) VALUES (?,?,?) ON CONFLICT
(tree_id,customer_id) DO UPDATE SET quantity = ?;"
prepstmt = con~prepareStatement(qry)
prepstmt~setInt(1,session~getAttribute("logged"))
prepstmt~setInt(2,choice)
prepstmt~setInt(3,quantity)
prepstmt~setInt(4,quantity + cartquantity)
prepstmt~executeUpdate()    -- update shopping cart
prepstmt~close()
con~close()
```

Listing 17: mainpage.rex Edit Table cart

When looking at the query, the amount selected gets inserted in the table `cart` as a combination of `customer_id` and `tree_id`, realizing the many-to-many relationship. Since the combination of values are defined as unique in the database, should they be duplicated the database will give an error. By using `ON CONFLICT`, should this situation occur the values will be update instead, combining the new quantity chosen with the one previously stored.

6.4.2. `userheader.rex`

This script, creating the header on top of multiple web pages, demonstrates how conveniently external scripts can be reused. It enhances the header with the current number of products in the shopping cart as well as buttons to login or logout, depending on the status stored in the session.

6.5. Logging In `/treeshop/login.jsp`

The purpose of the page `login.jsp` is to take the users credentials and check their validity. Afterwards, the login state is stored in the session. Once again, the form input is processed by an external script called, `login.rex`.

The script first uses a database query to determine the existence of the given username. In case it does not, a label is used to jump to the same block of code that is used to display a message for a wrongly entered password. The reason being, that otherwise a third party would be able to find out if an e-mail address is registered on the website.

Afterwards, jBcrypt's `checkpw` method uses the entered password and the hash stored in the database to authenticate the user. On success, `TRUE` is returned and the attribute `logged` is added to the session, using the `userid` as value, serving as an identifier. This attribute is used by multiple pages to determine the login status.

```
bcrypt=.bsf~new("org.mindrot.jbcrypt.BCrypt")
IF bcrypt~checkpw(pw, ha) THEN DO   -- only proceeds if password is correct
    session~setAttribute("logged",id)   -- store login status in session
```

Listing 18: login.rex checkpw

The use of a session instead of cookies is highly beneficial to the website's overall security. Were the user information directly transmitted with a cookie, a third party could easily replicate a cookie with a particular user's id to gain access. If a session token is transmitted instead, a unique value is generated each time the user logs in. Furthermore, sessions expire in a shorter time span [135].

It is a common scenario for a user to browse and add products to the cart as a guest. Only on checkout, login is required. It is important that the shopping cart is not lost during this process. Therefore, all the products stored in the cart array need to be moved to the database. Additionally, should any product be already present in the database basket, instead of overwriting it, it should be updated.

6.6. Invalidating a Session `/treeshop/logout.jsp`

It is essential to also give the user opportunity to log out again. To achieve this, the page `logout.jsp` is being made accessible by logout buttons throughout the web site. Once the page is accessed, the method `invalidate` is used to clear the session and all its associated parameters.

```
session~invalidate()    -- clear session
```

Listing 19: logout.jsp invalidate

6.7. Accessing the Shopping Cart `/treeshop/shoppingcart.jsp`

When the user clicks on the shopping cart button, found in the page's header, he gets redirected to the page `shoppingcart.jsp`. It allows to review all items stored in the cart, as well as adding, removing, or fully deleting products. The main functionality is held in the script `shoppingcart.rex`, additionally `userheader.rex` is reused to generate a dynamic header.

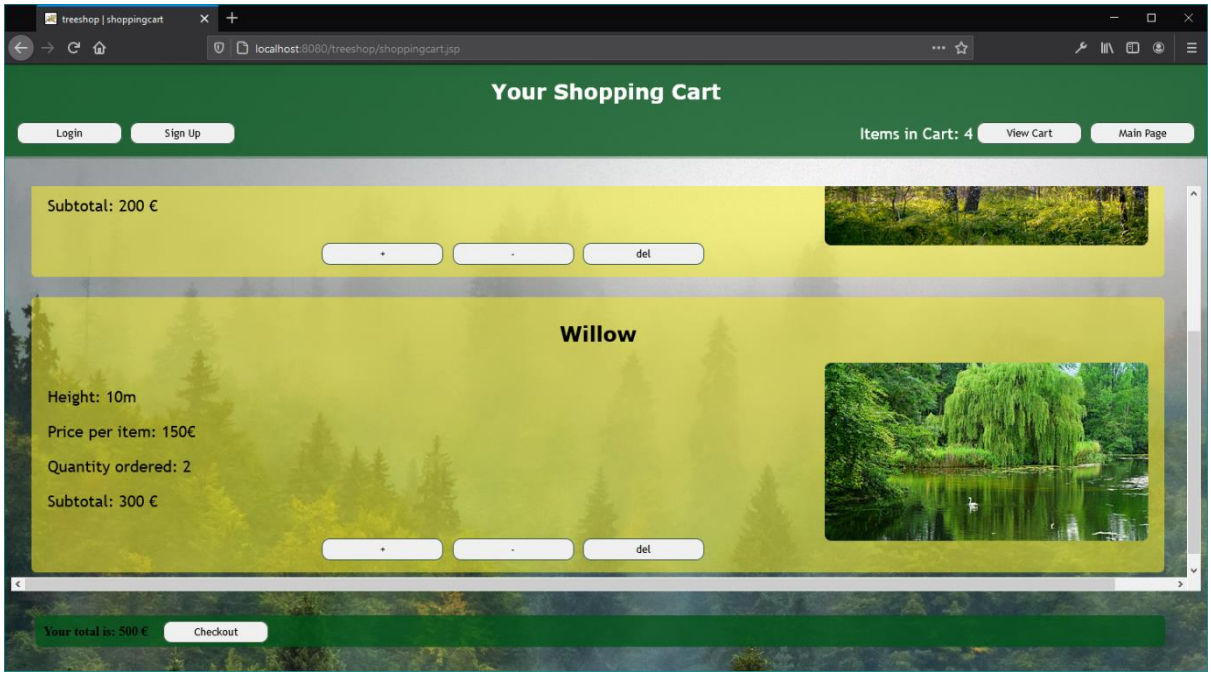

Figure 16: shoppingcart.jsp in Web Browser

Just like the main page, the script is split into two parts, which get executed based on the user's logins status.

```
cart = session~getAttribute("cart")
totalprice = 0
cartsupp = cart~supplier()
SAY '<div id="cartcontainer">'
DO WHILE cartsupp~available()    -- iterate over cart array
    qry = "SELECT * FROM tree WHERE tree_id="cartsupp~index";"
    stmt = con~createStatement()
    rs = stmt~executeQuery(qry) -- get data for product in cart

    DO WHILE rs~next()
        totalprice += rs~getString("price") * cartsupp~item
        CALL createProduct rs~getString("name"), rs~getString("picture"),
rs~getString("height"), rs~getString("price"), cartsupp~item,
rs~getString("price"), cartsupp~index
    END
    rs~close()
    stmt~close()
    cartsupp~next
END
SAY '</div>'
con~close()

CALL printtotal totalprice
```

Listing 20: shoppingcart.rex Create Guest Cart

A supplier is created if an array is stored in the session, representing a guest's shopping cart. Each iteration yields an index indicating the item id and its corresponding item representing the items quantity. Additionally,

the total price of all items gets updated during each iteration, to be printed at the bottom of the page.

The routine `createProduct` then uses these values to create a box containing information on the various products. Additionally, three buttons, `+`, `-`, and `delete` are created to manipulate the cart's contents.

Three `IF` blocks correspond to the previously generated buttons. The minus button necessitates to consider a situation, where the quantity reaches zero. In this case, the index value needs to be set to `.nil`, indicating the product to be nonexistent. The same logic applies to the delete button where the value is set to `.nil` straight away.

```
IF request~getParameter("actn") == "-" THEN DO
    id = request~getParameter("tree_id")
    qty = request~getParameter("qty") - 1
    cart[id] = qty  -- reduce quantity by 1
    IF qty <= 0 THEN cart[id] = .nil    -- delete product from cart if reduction
goes beyond 1
    session~setAttribute("cart", cart)
    response~sendRedirect(request~getRequestURI())  -- refresh page
END
```

Listing 21: shoppingcart.rex Minus Button

Should the user be logged in, the created page is similar in concept, except for the data being accessed from the database instead of the array.

6.8. Concluding the Purchase Process `/treeshop/checkout.jsp`

The final page `checkout.jsp` simply removes all the entries in the cart table that are related to the logged in user, simulating a concluded purchase process. A guest is required to login instead. In a real-world use case, the user would be asked for payment and shipping details instead.

7. Advanced examples `/treeshop/admin`

This is a good time to think about design decisions. For most examples until now, most of the required code is stored directly in the JSP or an ooRexx script. While this approach has advantages, like all the code being in one place and the ability to conveniently update it, the nested `IF` commands make it hard to work with and create redundantly executed lines of code.

It might be beneficial to separate the request actions, implementing logic, from the generation of content. This would also result in increased efficiency, since less unnecessary elements need to be processed and loaded. Nonetheless, for a beginner working with web applications, the shown approach is a fantastic way to quickly develop working web pages. The conclusory web examples in the subfolder `admin` of the `treeshop` web application will be used to show a slightly different approach.

While the contents of `application/WEB-INF` are shared across all directories, each subdirectory can be assigned its own resources and a unique welcome page: `http://localhost:8080/treeshop/admin`

As can be seen by the URL above, the folder structure directly influences the path to access pages. Also, special attention needs to be given to shared resources like stylesheets.

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

Listing 22: Link Resource for Subdirectory

The two leading dots indicate for the resource to be found in its parent directory. Therefore, the linked folder `css` is not found in the subdirectory `admin`, but one directory up. This enables pages found in subfolders to all use the same stylesheet.

7.1. Uploading Files `/treeshop/admin/addproducts.html`

Since entering data into a database can be time consuming, the page `addproducts.html` offers the functionality to enter new products to be sold on the main shop page. Since all of `treeshop's` pages are dynamically created using the database, newly added products will appear immediately. Images are an essential part of modern web pages. This example facilitates their upload, to be seamlessly integrated, named and stored.

On request, the page `addproducts.html` displays a set of fields corresponding to the database's columns. To enable uploading files, a form needs to be given the attribute `enctype` with the value `multipart/form-data`.

```
<form action="uploader" method="post" enctype="multipart/form-data" id="mailform">
```

Listing 23: addproducts.html Upload Form

An `enctype` defines the document's encoding type, with `multipart/formdata` allowing file uploads. Usually, it is not necessary to specify a form's encoding, with file uploads being the exception [136].

In contrast to previous examples, no external `.rex` script is used to process the data, instead the form points to the servlet `uploader`. For a JSP to be used this way, the `web.xml` file specific to the web application needs to be edited. Should the reader have copied the nutshell examples, no further modifications are necessary.

```
<servlet>
  <servlet-name>uploader</servlet-name>
  <jsp-file>/admin/code/uploader.jsp</jsp-file>
  <multipart-config>
    <location>C:\Program Files\Apache Software Foundation\Tomcat
10.0\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 24: web.xml Uploader Servlet Configuration

The file `uploader.jsp` is configured as a servlet named `uploader`. This enables further configuration, otherwise only available to Java servlets. For example, the load order, initialization attributes and security roles can be configured [137]. Since the JSPs content is written in the ooRexx language, a fully functional Rexx Servlet has been created. Now, the servlet's `MultipartConfig` can be easily modified.

To begin with the location for temporary files is specified. The proper location and filename for the file will be chosen inside the script. A temporary location is necessary since the file is first written as a temporary file and only afterwards processed to be stored permanently [138]. The `maxRequestSize` and `maxFileSize` Elements are used to set a limit for the size of both the file and the request in bytes [139]. Here the maximum size has been set to the equal of 10 megabytes.

Additionally, the servlet is registered in the `servlet-mapping`. This map is used by the container to resolve request [137]. Henceforth, the servlet is accessible from the path `/uploader`. This configuration allows the JSP to

directly process the request generated by the form on the page `addproducts.html`. Since the request gets directly sent to the servlet, no more `IF` loops looking for request parameters are required.

7.1.1. Upload Servlet `/upload`

The servlet functions like any other JSP, the only difference being the omission of any HTML references. After the JSP declarations, the scripting content immediately starts. Since all form fields have been set as `required`, the requests will always contain standard form fields as well as an uploaded file. The value of the fields can be easily accessed like a standard form using the `getParameter` method. Since no duplicate products are allowed, the script first checks the database for any entries with an identical name. Should an entry with the same name exist, a warning is displayed to the user. Whenever content needs to be displayed, the resources `leadin` and `leadout` are used to create a proper HTML page.

Before the new product can be added to the database, the uploaded file needs to be processed.

```
filename = name || ".jpg"
location = "/files/" || filename
request~getPart("file")~write(filename)  -- permanently write file to temporary
location
```

Listing 25: uploader.jsp File Processing

The product's name is given in the form and will be used together with the `.jpg` file extension to name the file. As has been previously discussed, the file is placed in the `files` folder, which has been defined as a source for static content. The string `location` gets stored in the database and is later used to by the `img` tag to access the picture from its relative path.

The `getPart` method is used to fetch a specific part from the request. The file has been given the name `file` in the form, which gets used to fetch it. At this point the file is stored in the temporary location, defined in the `MultiPartConfig`, found in the `web.xml` file. The method `write` is used to write the file to the disk with the previously given filename. Since no specific path is given, it gets permanently stored in the temporary location [140].

After writing the file to disc, a confirmation page is generated for the user. The resources `leadin` and `leadout` are used again to create a proper HTML page.

Should multiple files be contained in the request, the method `getParts` can be used to get a collection of all Parts, to be then iterated over [141].

After all fields are entered and an image is uploaded, the newly created product will immediately be visible on all of the application's web pages.

7.2. Common Gateway Interface

Not to forget, the Common Gateway Interface (CGI) offers the possibility to directly execute a script on the web server, generating a response for each request. Just like with JSPs, output methods can be used to create HTML pages. Tomcat allows the usage of CGI scripts by registering them like any other servlet. Since every request leads to the creation of a new process on the server, this approach can lead to significant performance problems in high traffic situations. Compared to Jakarta Servlets, which use Java, CGI scripts are dependent of the server's operating system, interpreters, and compilers. The requests of a CGI script leads to its direct execution from the command line [142, pp. 13-17].

This results in the programs being run outside of the Java Virtual machine, bypassing the Java Security Manager. Given all these limitations, CGI scripts are commonly used during development [143]. While this work focuses on the use of JSPs, this method is still briefly mentioned, since it offers an alternative way to directly execute scripts.

7.3. Sending E-Mails `/treeshop/admin/sendnewsletter.jsp`

The final example page demonstrates how a web application can be used to send e-mails. Since the database already includes product details and the customer's e-mail addresses, all information necessary to create a newsletter promoting current products exists.

The script found in the body of `sendnewsletter.jsp` creates a list of all products in the database and allows them to be selected by a checkbox. The

JSP `mailer.jsp` is then used to create and send e-mail messages. Therefore, just like in the previous example, the servlet called mailer needs to be registered in the `web.xml` file.

For this web application to function the `.jar` files containing Jakarta Mail and Jakarta Activation are required. The demo web application already includes both of them.

Jakarta Mail can be downloaded from `https://eclipse-ee4j.github.io/mail/` This API was previously known as JavaMail and offers functionality for Java applications to implement e-mail functionality, such as sending and reading e-mails [144].

The second prerequisite, Jakarta Activation can be downloaded from `https://eclipse-ee4j.github.io/jaf/` According to the Eclipse Foundation, it is used to: *"determine the type of an arbitrary piece of data; encapsulate access to it; discover the operations available on it; and instantiate the appropriate bean to perform the operation(s)"* [145].

Lesson Learned: The correct file `jakarta.activation-2.0.0.jar` should not be mixed up with `jakarta.activation-api-2.0.0.jar`. Furthermore, version 2.0.0 of Jakarta Mail requires Java Activation to be using the Jakarta namespace, therefore Version 2.0.0 should be used for both.

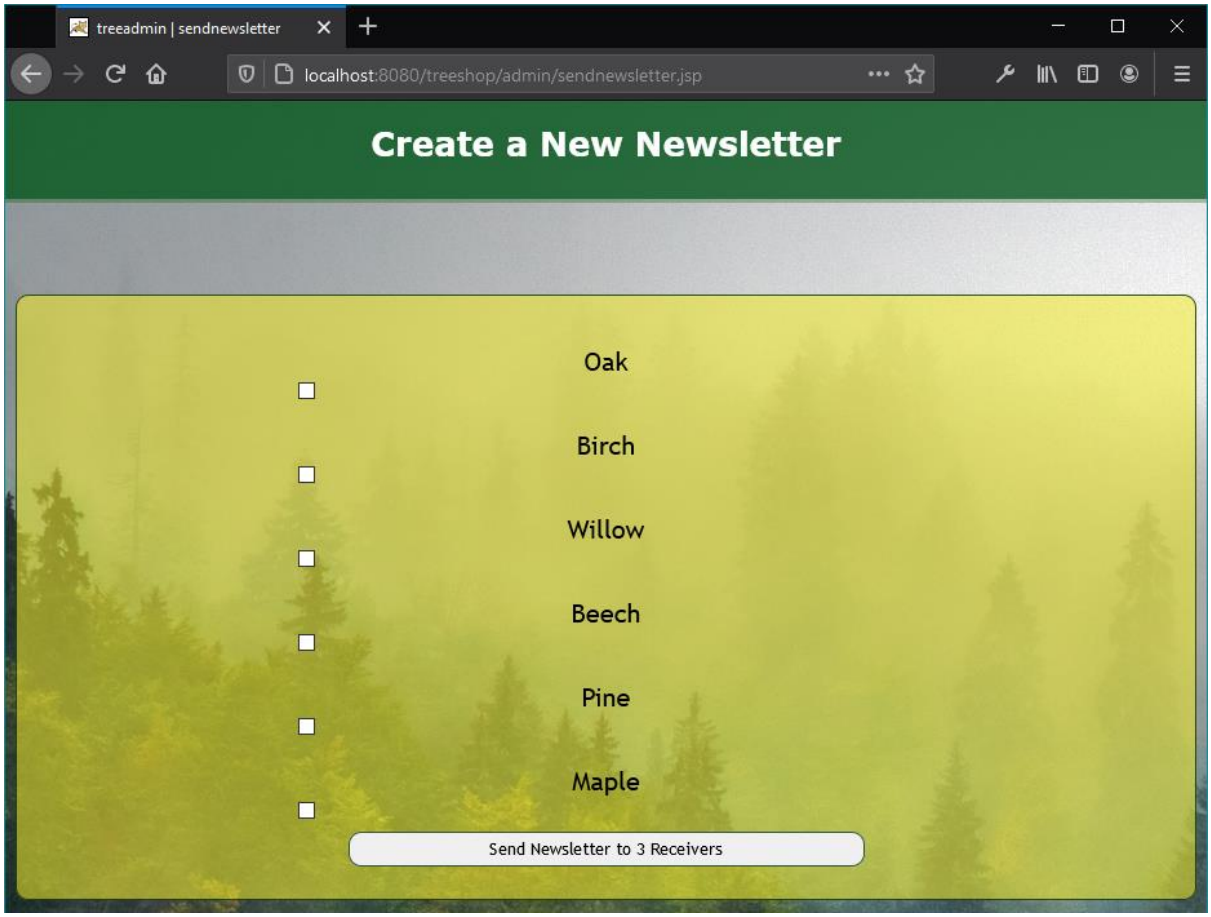

Figure 17: createnewsletter.jsp in Web Browser

The page `sendnewsletter.jsp` generates a form with a checkbox for each product in the database. All the form's checkboxes have the name `choice` and the corresponding product id as value. The page also uses a database query to count the number of recipients to dynamically display how many e-mails will be sent in the submit button.

The servlet `mailer` first makes sure that at least one product has been selected, otherwise an empty e-mail would be sent. Since `choice` most likely has multiple values, instead of `getParameter` the method `getParameterValues` is needed. It will fetch all values and store them in an iterable string array [146].

```
choices = request~getParameterValues("choice")
choice = ""
DO productname OVER choices -- append all product names to a string
    choice = choice || "'" || productname || "'"  || ","
END
choice = choice~delStr(choice~length)   -- remove the string's last comma
```

Listing 26: mailer.jsp Choicearray

To create the message, first the variable `choice` is defined as an empty string. The previously fetched string array is iterated over, to create a list of product names, which is usable in a database query. For each iteration, a new product is added until none are left. The existing string (for the first iteration it is empty) is extended with the product name, enclosed in single quotation marks and a comma. After the iteration is concluded, the last comma is removed to create a functioning database query. This is achieved by the `delStr` Method.

Delete String removes the character at the given position. By giving the length of the whole string as attribute, the last character is deleted [98].

Concerning the receivers, when a user signs up for the web page, her e-mail address is given. Therefore, a list of receivers can be easily generated. Each database entry referring to a user contains a Boolean used to determine whether the user wishes to receive emails.

It might be a mistake to simply set all the customers as receivers for a single e-mail. Afterall, each of them would see the whole list of customers in the recipient fields. Since this information should not be exposed, the script sends a separate e-mail to each customer.

```
stmt1 = con~createStatement()
qry1 = "SELECT * FROM customer WHERE receives_mail = 't';"
rs1 = stmt1~executeQuery(qry1)  -- select all customers who wish to receive the
newsletter

emailcount = 0
DO WHILE rs1~next() -- create an e-mail for each customer and send it
```

Listing 27: mailer.jsp Select Receivers

In consequence, the database is quered for all e-mail entries that are signed up for the mailing list. The variable `mailcount` meanwhile keeps track of the number of e-mails sent. The created `resultSet` is then used to create and send a personal e-mail for each iteration. Since the `resultSet` of this first query will contain another, the variables `stmt`, `qry` and `rs` have been numbered accordingly. Should the same variables be used for both, they might overwrite each other and cause problems.

```
props = .bsf~new("java.util.Properties")
session = bsf.loadclass("jakarta.mail.Session")~getInstance(props)
msg = .bsf~new("jakarta.mail.internet.MimeMessage",session)

sender = .bsf~new("jakarta.mail.internet.InternetAddress",
```

```
"newsletter@treeshop.com")
msg~setFrom(sender)

receiveraddress =  rs1~getString("username")
receiver = .bsf~new("jakarta.mail.internet.InternetAddress",receiveraddress)
type = bsf.loadclass("jakarta.mail.Message$RecipientType")
msg~addRecipient(type~to,receiver)

msg~setSubject("Here Are the Latest Products From treeshop!")
```

Listing 28: mailer.jsp Create Message

To begin with a Jakarta Mail session needs to be created. To instantiate the corresponding class a set of Java properties is necessary. The class `java.util.Properties` creates a persistent set of properties where a key corresponds to a property, both of which are strings [147]. For the example given, no special properties are needed, they need to be defined either way, since properties are needed to create a session instance.

For example, should the reader utilize a server requiring authentication, the property `mail.smtp.auth` would be set as `true` with the command `setProperty("mail.smtp.auth","true")` [148, p. 51].

Afterwards a session instance is created. The session is used as a bridge to the Jakarta Mail API, handling configuration and authentication. Using this session, the message to be sent is created. More specifically, the subclass `Mimemessage` is used, which allows the use of different mimetypes and headers [149].

Now, sender and receiver are added to the newly created message; the addresses need to be defined using the `Internetaddress` class.

While defining the sender is straightforward, the receiver additionally requires the recipient type to be set. Afterall, the receiver can be of the type `TO`, `CC` or `BCC` [150]. For this example, a simple `TO` receiver is used, the address being made available by the database. To conclude, the subject is added.

The last piece missing is the message's body. To create it a second database query is nested into the first.

```
stmt2 = con~createStatement()
qry2 = "SELECT * FROM tree WHERE name in ("choice");"
rs2 = stmt2~executeQuery(qry2)  -- get data for all products that have been
selected

i = 0
DO WHILE rs2~next   -- create html code for each product
```

```
    line1 = '<div style = "float: left; margin-right: 10px;">'
    line2 =  '<h2>' rs2~getstring("name") '</h2>'
    line3 = '<img src="http://localhost:8080' || rs2~getString("picture") '"
heigth="120" width="150" />'
    line4 = '<p>Price:' rs2~getstring("price") 'Euro</p>'
    line5 = '<p>Height:' rs2~getstring("height") 'Metres</p>'
    line6 = '</div>'

    i += 1
    product.i = line1 || line2 || line3 || line4 || line5 || line6  -- append all
lines of html code
END
rs2~close()
stmt2~close()

text = '<html><head><meta charset="UTF-8" /></head><header>'  -- create proper
html leadin
text =  text || '<h1><a href="http://localhost:8080/treeshop/shop">Vist
treeshop</a></h1>'
text =  text || '<h4><a
href="http://localhost:8080/treeshop/admin/unsubscribe.jsp?unsub=' ||
receiveraddress '">Click Here to Unsubscribe</a></h4>'

DO count = 1 TO i   -- append all products
    text = text || product.count
End

text = text || '</body></html>' -- append proper html leadout

msg~setContent(text,"text/html")
```

Listing 29: mailer.jsp Create Message Content

The database is queried for all the product data related to the choice made on the previous page. The main problem of this approach is that the whole message needs to be contained in a single string. For this reason, and to easy formatting and the insertion of pictures, the message is created by HTML code.

A HTML segment for each product is created, to be later added to the main message. For each product, six lines of HTML code create a `<div>` section. Additionally, each iteration increases the index value `i` by one. The six lines are then appended together to form a single line and stored under the variable `product.i`, where `i` is used to index them accordingly. This gives the script the flexibility needed to adapt to a varying number of products.

After all the `<div>` sections are created the main e-mail text commences with the tags to properly define an HTML document. Afterwards a headline, linking to the shop's main page is generated, followed by a link to unsubscribe. The feature to unsubscribe from all newsletters will be

discussed in the next section. Next, all the previously generated products are added to the string, followed by the HTML closing tags.

The `setContent` method is then used to add this string as the message's body, finalizing the message.

7.3.1. Sending and Receiving E-mails with MailHog `/mailer`

The Jakarta Mail API is platform and protocol independent and can therefore be used on any operation system and with most e-mail service providers allowing IMAP, POP3 or SMTP protocol access [144]. Therefore, once this application has been properly tested, it can be linked with an e-mail server to send messages into the real world.

During the first phases of testing, usually many e-mails needs to be sent. To simplify this process the author suggests the use of an open-source tool called MailHog. The appendix includes instructions to set up a local testing SMTP server on the localhost: **`9.6. MailHog`**. The process is incredibly easy and can be done within minutes.

Since this testing tool accepts any combinations of username and password, placeholder values for them will be used. By default, MailHog uses the port 1025 on the local host. Therefore, these configuration parameters will be used for this example. Should the reader wish to use a different e-mail server, adjustments are necessary.

```
transport = session~getTransport("smtp")
transport~connect("localhost",1025,"username","pw")
transport~sendMessage(msg,msg~getRecipients(type~to))    -- send message using a
test server

emailcount += 1
```

Listing 30: mailer.jsp Send Message

The session's `transport` object is used to send the e-mail. The connect method establishes a connection to the server, using the attributes host, port, username, and password. Finally, the `sendMessage` method takes the message that has been previously created and its recipients as inputs to send the e-mail.

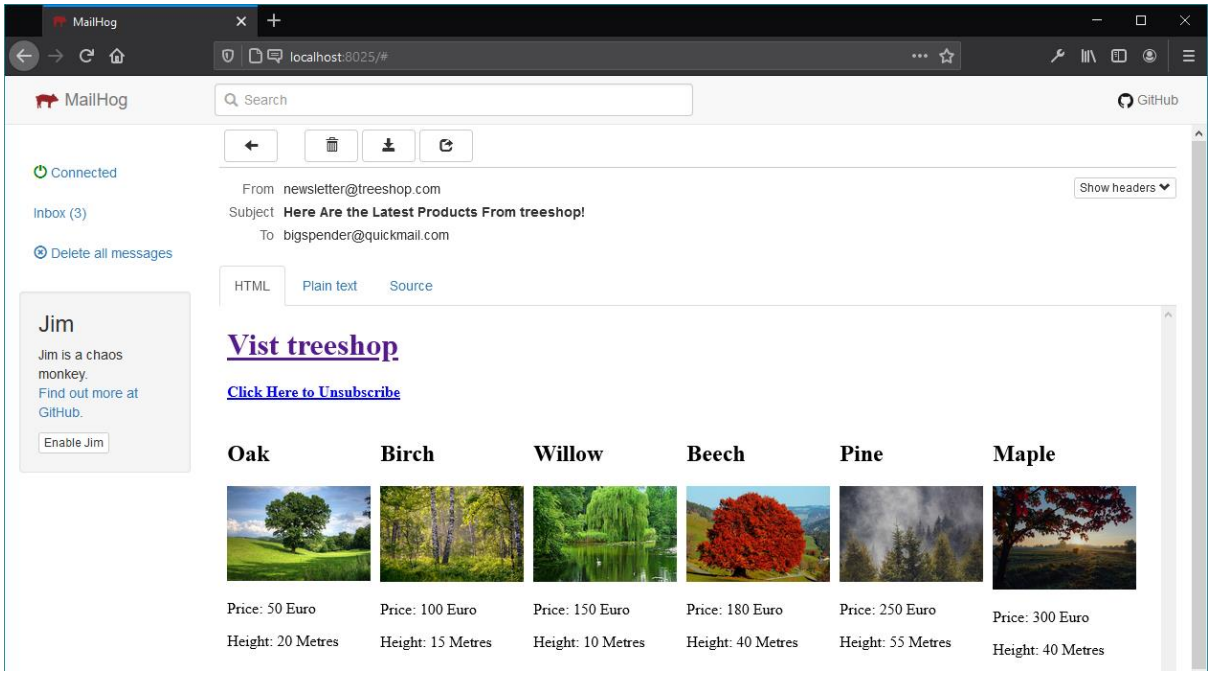

Figure 18: Newsletter in Web Browser

LESSON LEARNED: The pictures in the e-mail are only visible if the Tomcat server is up and running.

7.4. Unsubscribing from E-mails `/treeshop/admin/unsubscribe.jsp`

The European Union's General Data Protection Regulation requires that users can object to receiving direct marketing e-mails at any time and that companies then must stop using their data immediately [118]. To implement this regulation and help users to easily unsubscribe from unwanted e-mails, each email includes an unsubscribe link, which gets dynamically created, requiring the receiver to simply click on the link.

Essentially this provided link includes a `get` request, rewriting the URL to include the e-mail address of the receiver: `localhost:8080/treeshop/admin/unsubscribe.jsp?unsub=bigspender@quickmail.com`

Figure 19: unsubscribe.jsp In Web Browser

The script in the document's body fetches the e-mail address included in the request by using its assigned parameter `unsub`. Instead of immediately unsubscribing, the user is asked if he is certain and displayed a button to confirm. Upon clicking it, the user's e-mail address is forwarded to the servlet `remover.jsp`. Please note that to function the servlet needs to be registered in the `web.xml` file, just like in the previous examples. The servlet simply changes the column `receives_mail` for the corresponding user to `FALSE`. Upon success, a confirmation message will be displayed.

8. Conclusion

After working with web applications extensively, the author will never look at web pages in the same way. It is astonishing how technologies that were created over twenty years ago are still used today, to create the modern world wide web we take for granted. Furthermore, the author hopes to inspire readers to create their own web applications. By using the building blocks introduced in this thesis, in combination with countless available external Java libraries, even a beginner will be able to quickly turn ideas into reality.

Three different approaches have been introduced: Adding scripts directly to a JSP, linking external `.rex` files containing the logic, and configuring a JSP with ooRexx content as a servlet. While each of the approaches has its advantages and disadvantages, they offer great insights into web development and enable adaptation to a given situation.

For more sophisticated web applications, involving a team of developers, the intermingling of programming logic and design components on a single JSP will prove problematic. Those projects will use the model-view-controller design pattern, implemented by a framework like Apache Struts. Nonetheless, the methods suggested in this work allow a single individual to create dynamic web pages in record time.

9. Appendix

9.1. Prerequisites

This section not only contains a collection of hyperlinks to all the required software but can also be used as a checklist. This work was finished in the beginning of the year 2021 and reflects the current development stage. For future use, the download locations might change, and the software will most likely by updated. The author recommends downloading the latest versions available.

9.1.1. Software required to begin

- Nutshell examples: `http://wi.wu.ac.at/rgf/diplomarbeiten/`
  - An archive containing the demo applications should come included with this work. In case it is missing, please search for this thesis in the collection provided

- ooRexx: `https://sourceforge.net/projects/oorexx/`
  - As a minimum, Version 5.0.0 needs to be used
  - The latest Beta version is recommended

- OpenJDK: `https://bell-sw.com/pages/downloads/`
  - Any other Java implementation will also work
  - `Liberica Full JDK` should be chosen for maximum compatibility
  - Needs to match the ooRexx version installed, a 64-bit ooRexx installation requires a 64-bit version of Java, whereas a 32-Bit version requires a matching 32-Bit installation

- BSF4ooRexx: `https://sourceforge.net/projects/bsf4oorexx/`

- The file `bsf4ooRexx-v641-20201217-bin.jar` can be found in the downloaded archive, or once installed, in the installation directory of BSF4ooRexx
- Also recommended from this source, but optional:
    - IntelliJ IDEA plugin, enabling text highlighting for ooRexx

- Additionally, the Tag Libraries need to be downloaded: `https://sourceforge.net/projects/bsf4oorexx/files/Sandbox/rgf/taglibs/`
- Also recommended from this source, but optional:
    - `demoRexx` web application

❖ Apache Tomcat 10 (Beta status in January 2021): `https://tomcat.apache.org/download-10.cgi`

❖ As an Alternative: Apache Tomcat 9 (Stable status in January 2021): `https://tomcat.apache.org/download-90.cgi`

9.1.2. Software required for advanced examples:

❖ PostgreSQL: `https://www.postgresql.org/download/`

❖ PostgreSQL JDBC Driver: `https://jdbc.postgresql.org/`

❖ jBcrypt: `https://www.mindrot.org/projects/jBCrypt/`

❖ Jakarta Mail: `https://eclipse-ee4j.github.io/mail/`

❖ Jakarta Activation: `https://eclipse-ee4j.github.io/jaf/`

- MailHog: https://github.com/mailhog/MailHog

9.2. Tomcat Installation Guide

The following part will show a step-by-step installation guide for the Apache Tomcat Software version 10.0.0 on the Windows 10 Operating system. Before beginning, as a minimum Java needs to be installed.

In case the reader prefers the stable Tomcat Version 9, the installation process is identical.

Before beginning the installation, it is recommended to inquire about the current development status and stable releases: https://tomcat.apache.org/whichversion.html

The apache-tomcat-10.0.0.exe can be downloaded from the webpage: https://tomcat.apache.org/download-10.cgi by clicking the link `32-bit/64-bit Windows Service Installer (pgp, sha512)`.

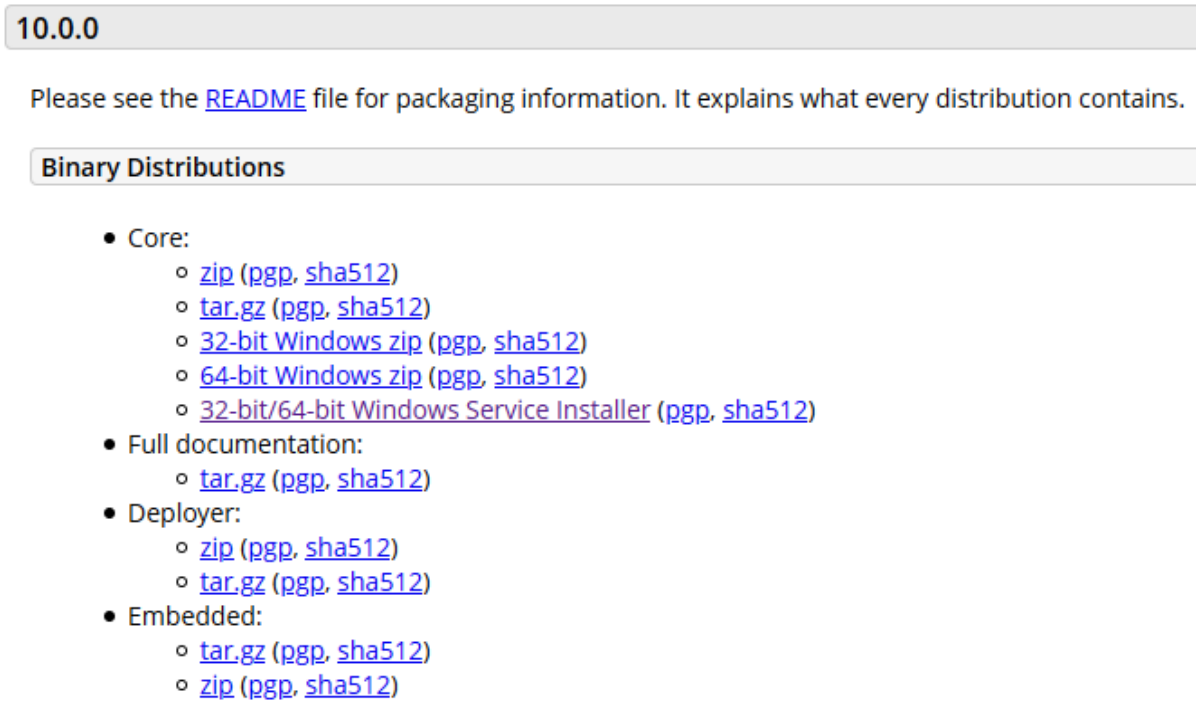

Figure 20: Tomcat 10 Download Page

After downloading and executing the file `apache-tomcat-10.0.0.exe` one is greeted with the following window. It is worth to note that under most

Windows 10 configurations, upon running the exe file, one is greeted with a popup from Windows User Account Control, where it is necessary to grant the app permission to make changes on the device to proceed with the installation. The installer being still labeled as Apache Tomcat 9 is most likely a result of the software still being in Beta status.

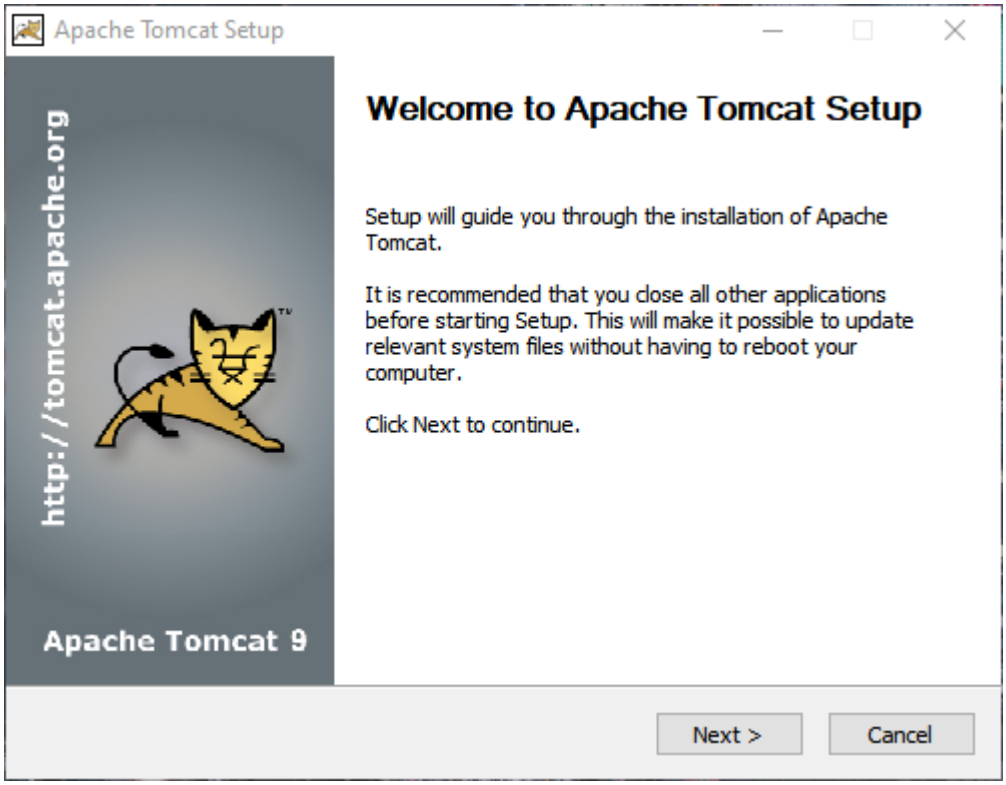

Figure 21: Welcome to Apache Tomcat Setup

After clicking the `Next` button, the License Agreement can be reviewed and needs to be accepted by clicking on "I Agree".

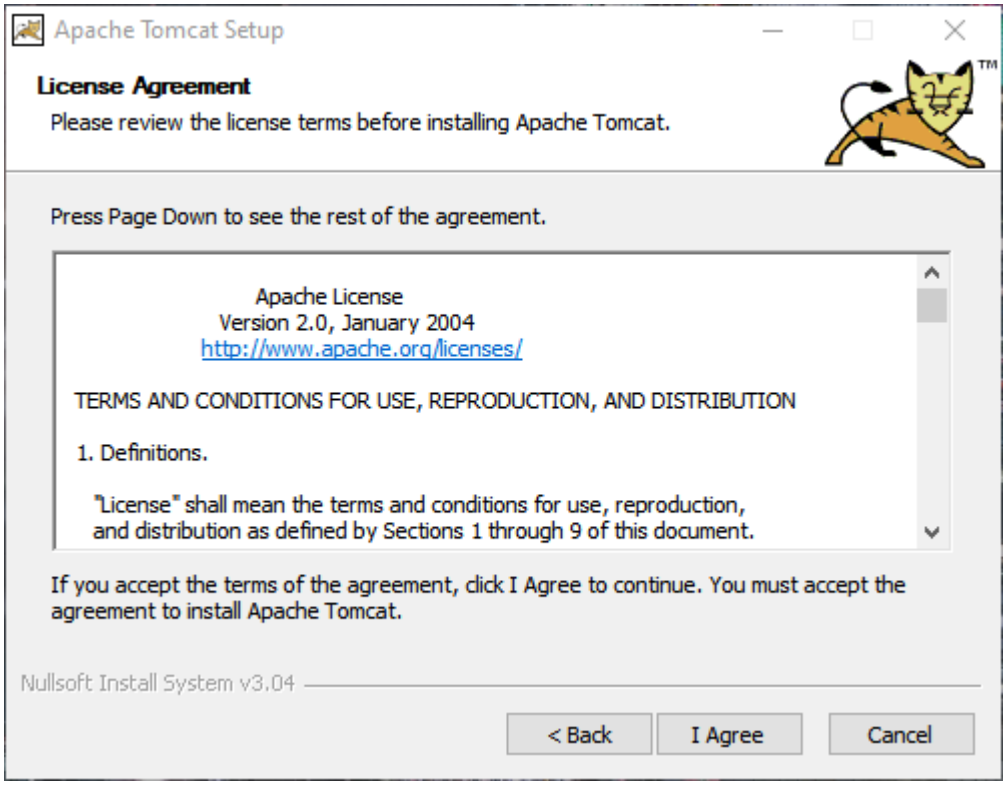

Figure 22: Tomcat 10 Setup License Agreement

The following window allows the customization of components to be installed. It is recommended to select a `Full` installation.

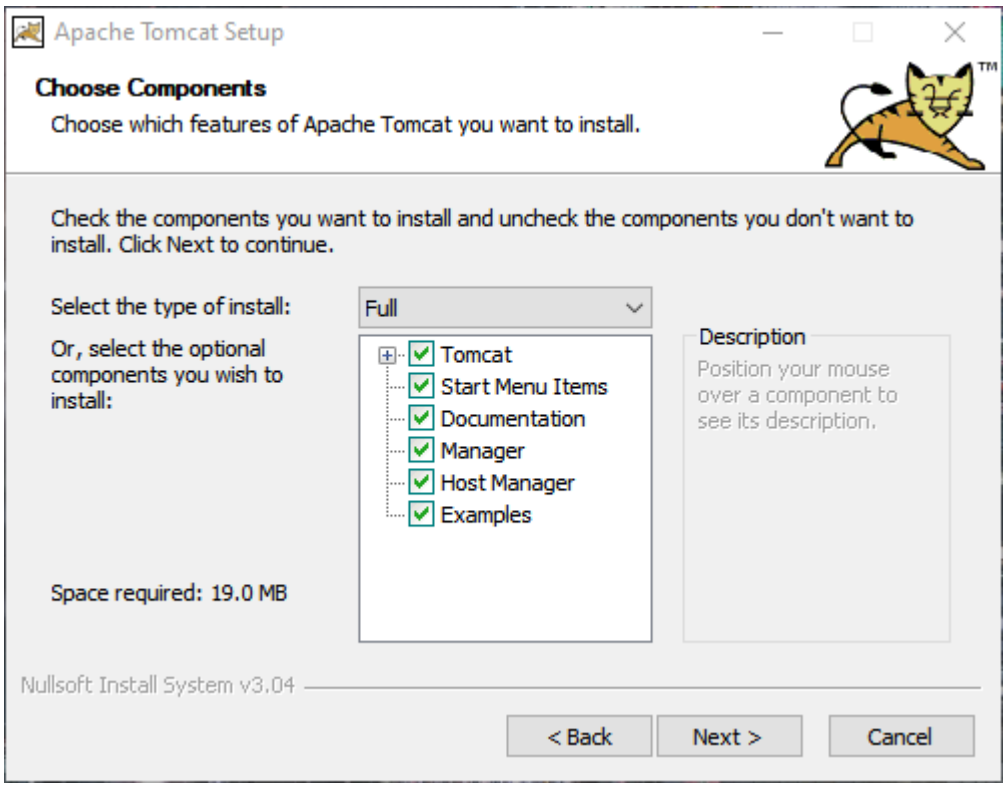

Figure 23: Tomcat 10 Setup Choose Components

The entries `Start Menu Items`, `Documentation` and `Examples` are self-explanatory and might prove useful. The `Manager` entry is used to create a web application, accessible from */manager*, with various functions like listing all currently deployed web applications and its users [77]. The Host manager is used for creating multiple websites on a single server [151].

After choosing the components, the next window for the installations allows further configuration.

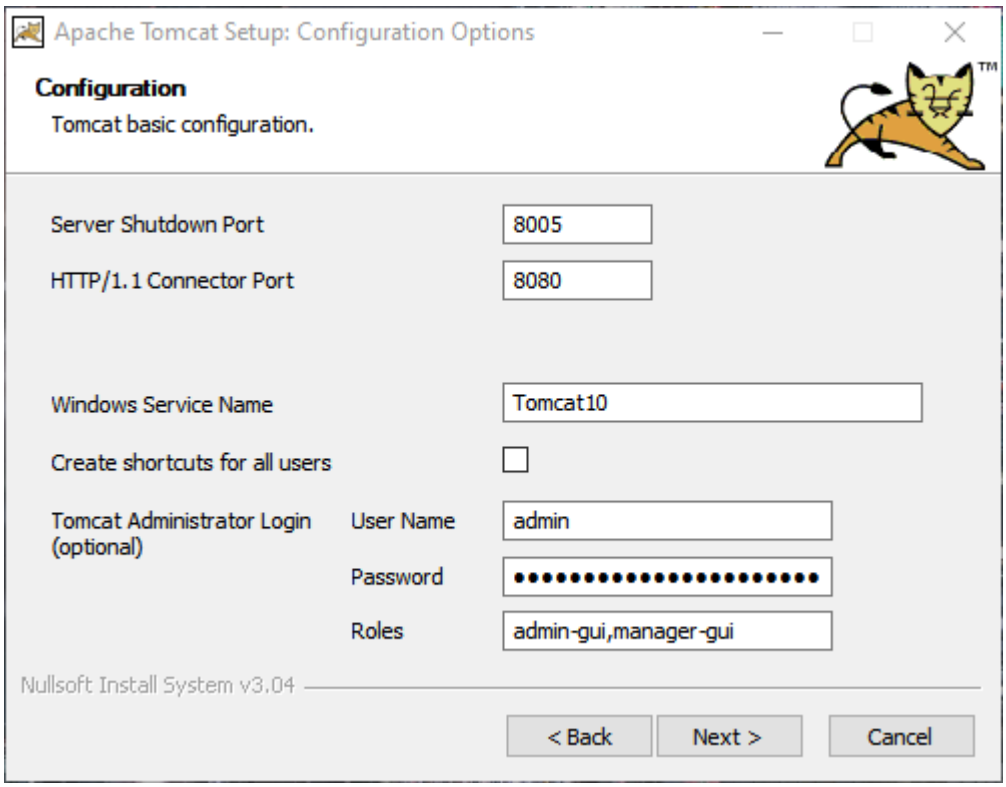

Figure 24: Tomcat 10 Setup Configuration

Most importantly the Server Shutdown Port is, by default, set as disabled, port -1. Here it is highly recommended to choose another available port like 8005, the default shutdown port. This port is used for the server to wait for a shutdown command. It is not recommended to disable the port while running and stopping the server with the standard shell scripts. When using the Apache Commons Daemon from the taskbar, disabling it is an option [152]. Setting the Server Shutdown ports allows the user to use both the Apache Commons Daemon as well as the standard shell scripts. This approach is taken to counteract potential errors. The main part of this work describes in detail how Tomcat is used.

Additionally, it is recommended to pick a username and password, since this is the most convenient way to set them. These credentials are used in the manager web application.

The next part of the installation asks for the path of the Installed Java Runtime Environment, which is usually automatically detected.

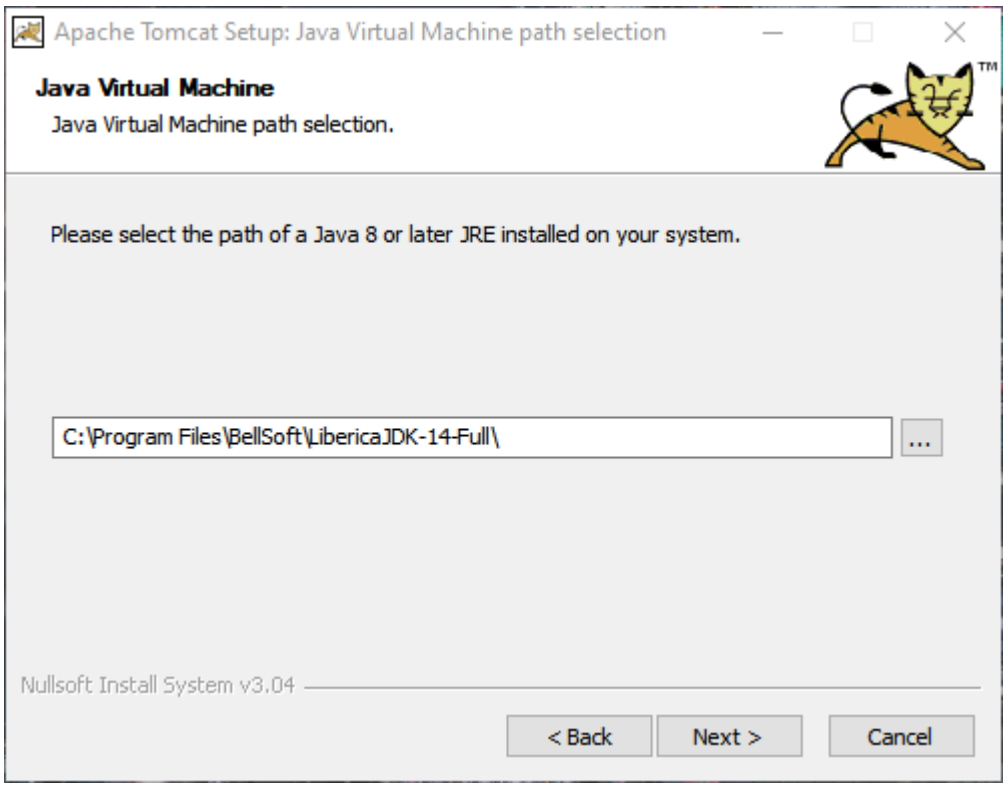

Figure 25: Tomcat 10 Setup Java Virtual Machine

Afterwards the installation path is chosen, in the figure below the default path is left unchanged.

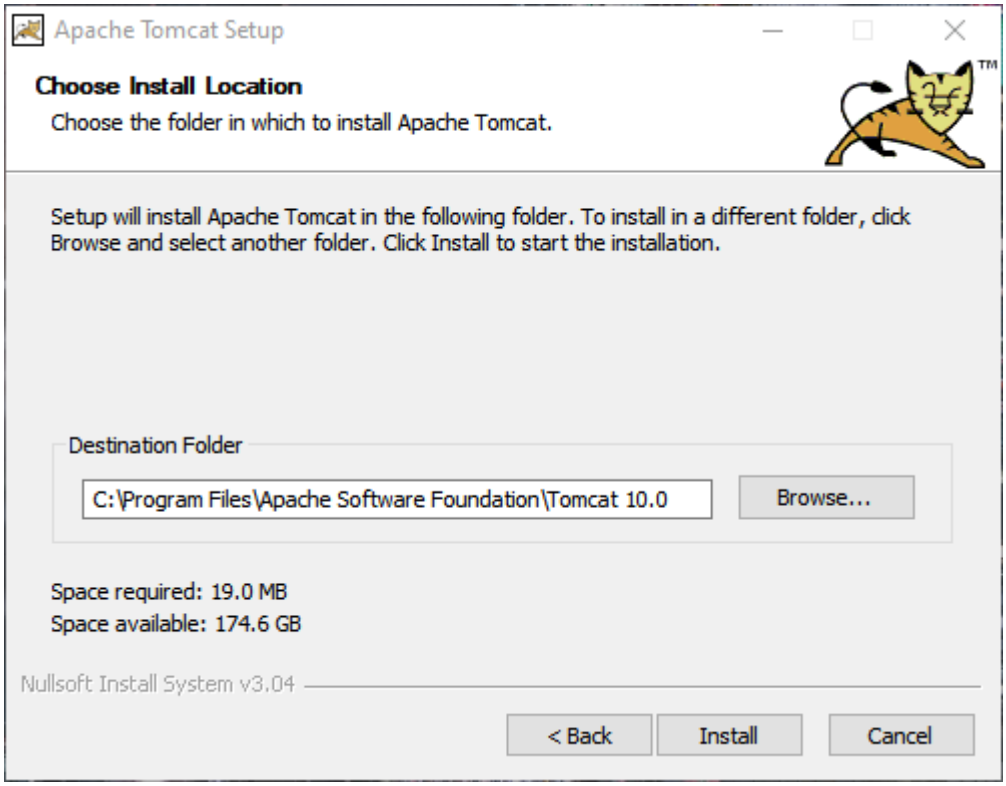

Figure 26: Tomcat 10 Setup Choose Install Location

After confirming the installation path, the software gets installed and the user is greeted with the following window. If it is desired to immediately start, the checkbox to Run Apache Tomcat can be picked. Additionally, the Readme file can be reviewed.

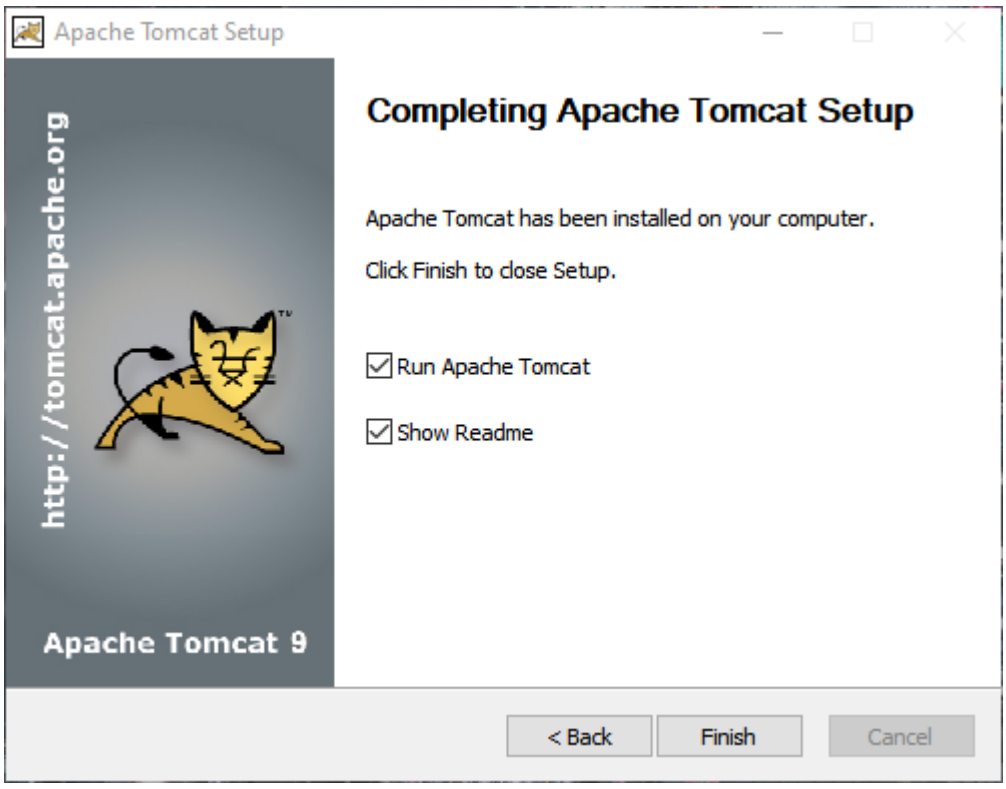

Figure 27: Completing Apache Tomcat Setup

9.3. Tomcat 9

At the time of writing in January 2021, the Apache Tomcat 10 version was still in a Beta status. Should the reader prefer the stable Version 9, small changes are necessary. The reason being, that Tomcat 10 uses the Jakarta namespace, while Tomcat 9 uses the JavaX namespace.

Instead of using the web applications `helloworld.war` and `treeshop.war`, modified versions can be found in the zip archive included with this work, more precisely in the directory `ZIP_ARCHIVE/javax_for_tomcat9`.

The main difference is the tag library used, since it is already part of the web applications, no further actions are required for the reader.

Therefore, should the reader start creating her own applications and prefer using Tomcat 9, it is instrumental to use the file `javax.ScriptTagLibs.jar` instead of `Jakarta.ScriptTagLibs.jar`.

Other than the taglib used, the main difference can be observed in the naming of certain classes, for the examples used, the only difference is related to the creation of cookies.

For example, while in the Jakarta version a cookie is created by using the class `jakarta.servlet.http.Cookie`, Tomcat 9 uses `javax.servlet.http.Cookie` instead.

This name change is not universal, for example both versions still use `javax.naming.InitialContext` to refer to the `InitialContext` class. Furthermore, since the latest version of Jakarta Mail is added as an external library, both Tomcat 9 and Tomcat 10 use the Jakarta namespace to send e-mails.

As a result, it is a good idea to keep this name change in mind, especially when encountering inexplicable errors relating to classes not being found.

9.4. PostgreSQL

This section will be used to show all necessary steps to install and setup a PostgreSQL database management system, enabling the full functionality of the nutshell examples shown.

9.4.1. Installation

To begin with the latest installer for the operating system of choice can be downloaded from: `https://www.postgresql.org/download/` At the time of writing the current version was 13.1-1. After downloading and executing the file `postgresql-13.1-1-windows-x64.exe` one is greeted with the following screen on the Microsoft Windows 10 operating system. Usually, it is necessary to allow the application to make changes in a User Account Control popup warning.

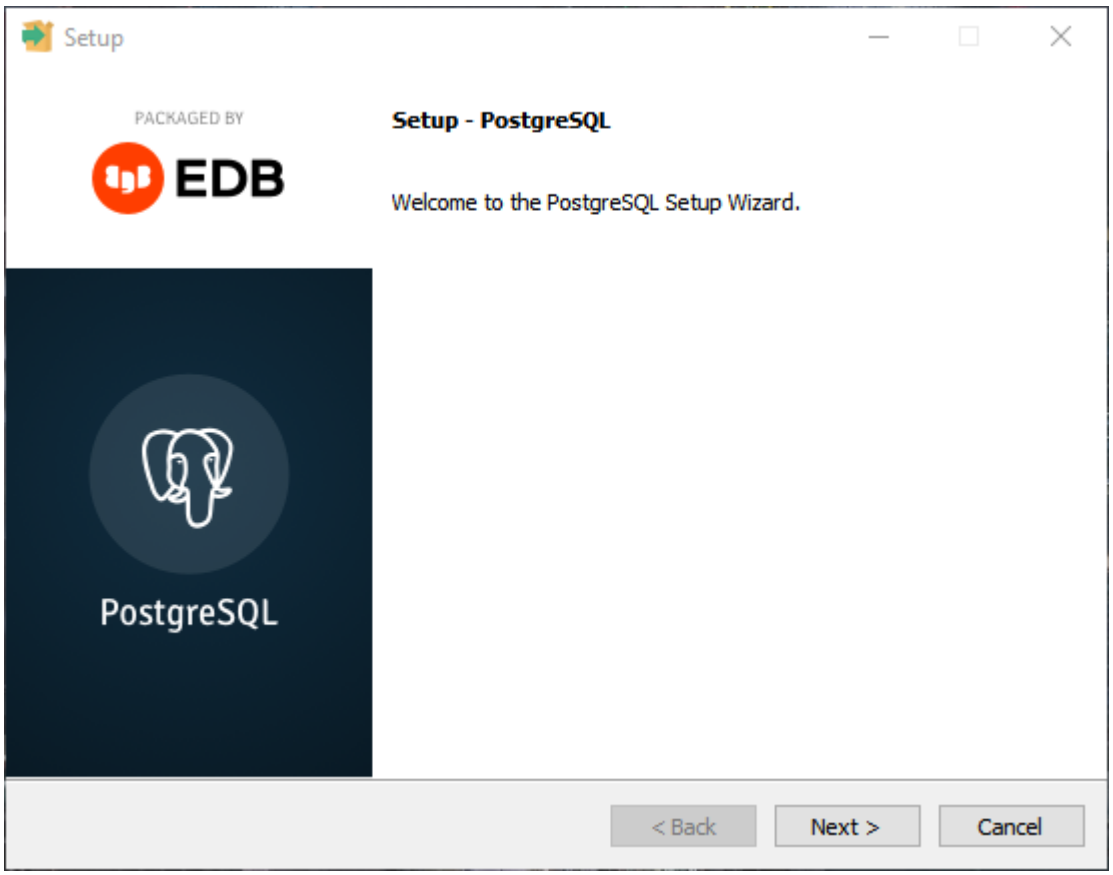

Figure 28: PostgreSQL Setup

After hitting the next button, the installation directory is selected, the default directory will work fine for most installations on a private machine.

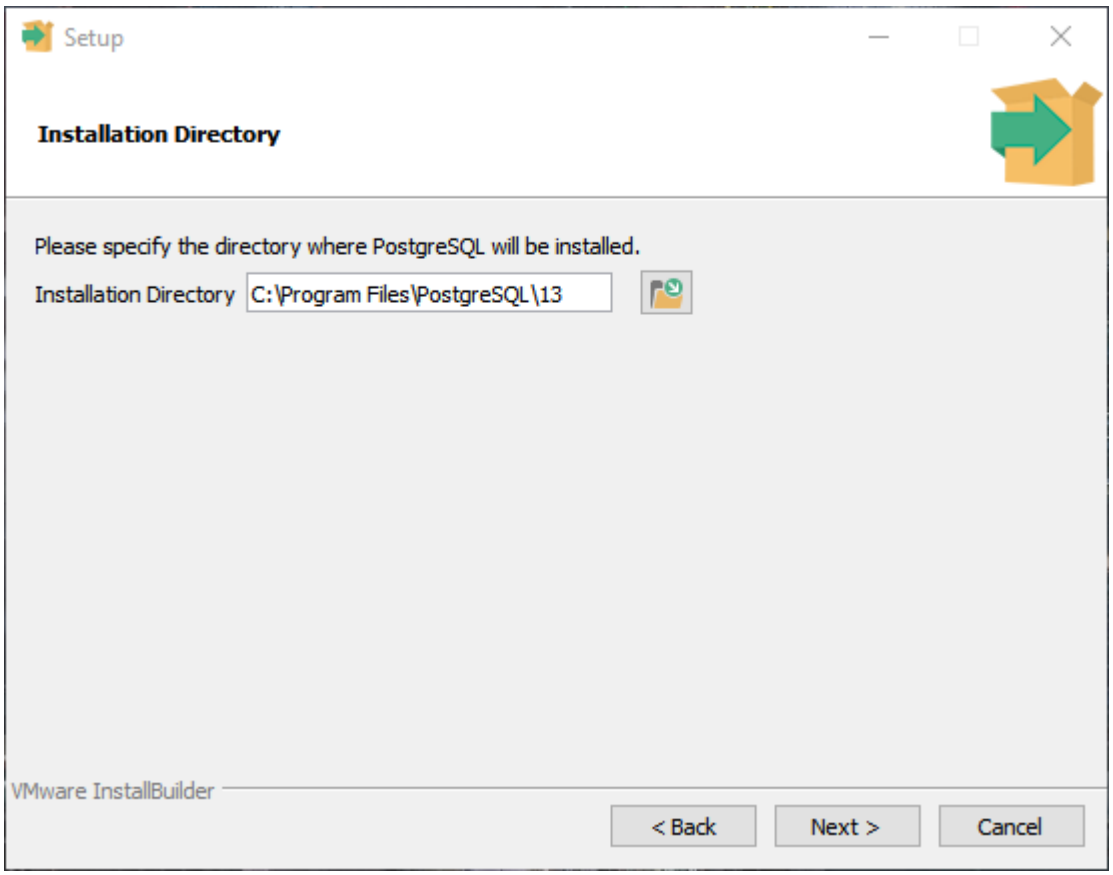

Figure 29: PostgreSQL Setup Installation Directory

Afterwards, the components to be installed are be chosen. For the use case described, the only mandatory option is PostgreSQL Server.

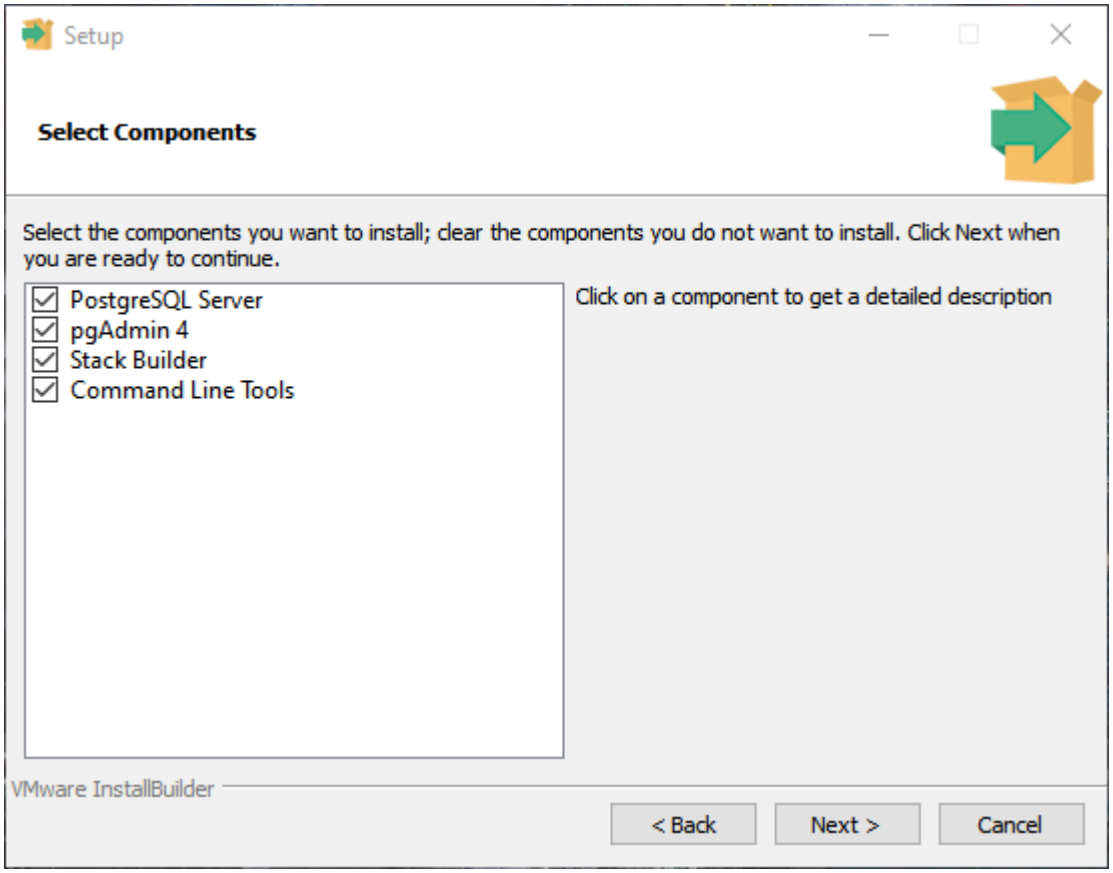

Figure 30: PostgreSQL Setup Select Components

The next window asks for a directory to store the actual data that gets managed in the database management system. Per default, a directory within the default installation directory is chosen. Again, for use cases on a private machine this default option will work fine.

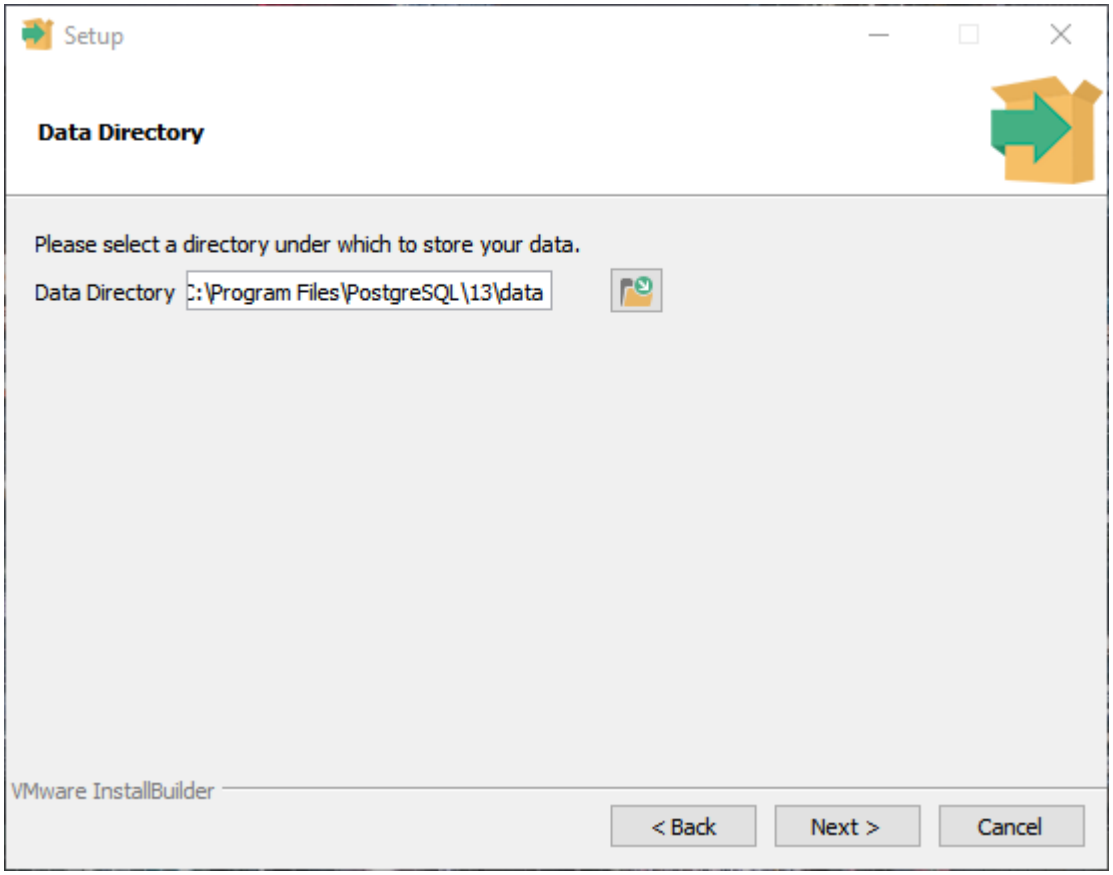

Figure 31: PostgreSQL Setup Data Directory

The next step requires choosing a password for the database superuser account `postgres`. Should the database include sensitive data, it is necessary to choose a strong password, since this account has all possible permissions. For Apache Tomcat, a separate account will be added later, therefore this superuser password is usually not repeated frequently.

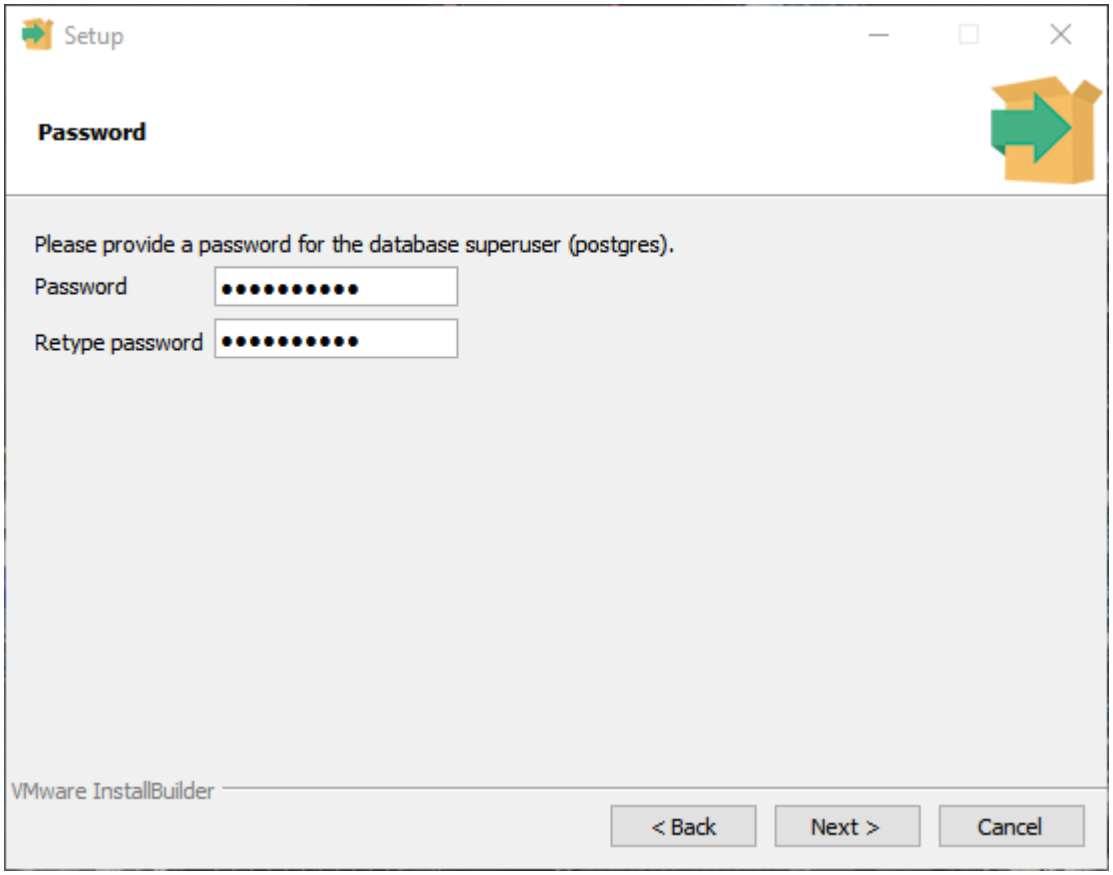

Figure 32: PostgreSQL Setup Password

In the next step the port, under which the database server is accessible is chosen. Once again, for the use on a private machine the default port 5432 does not need to be changed and is used for the demo examples.

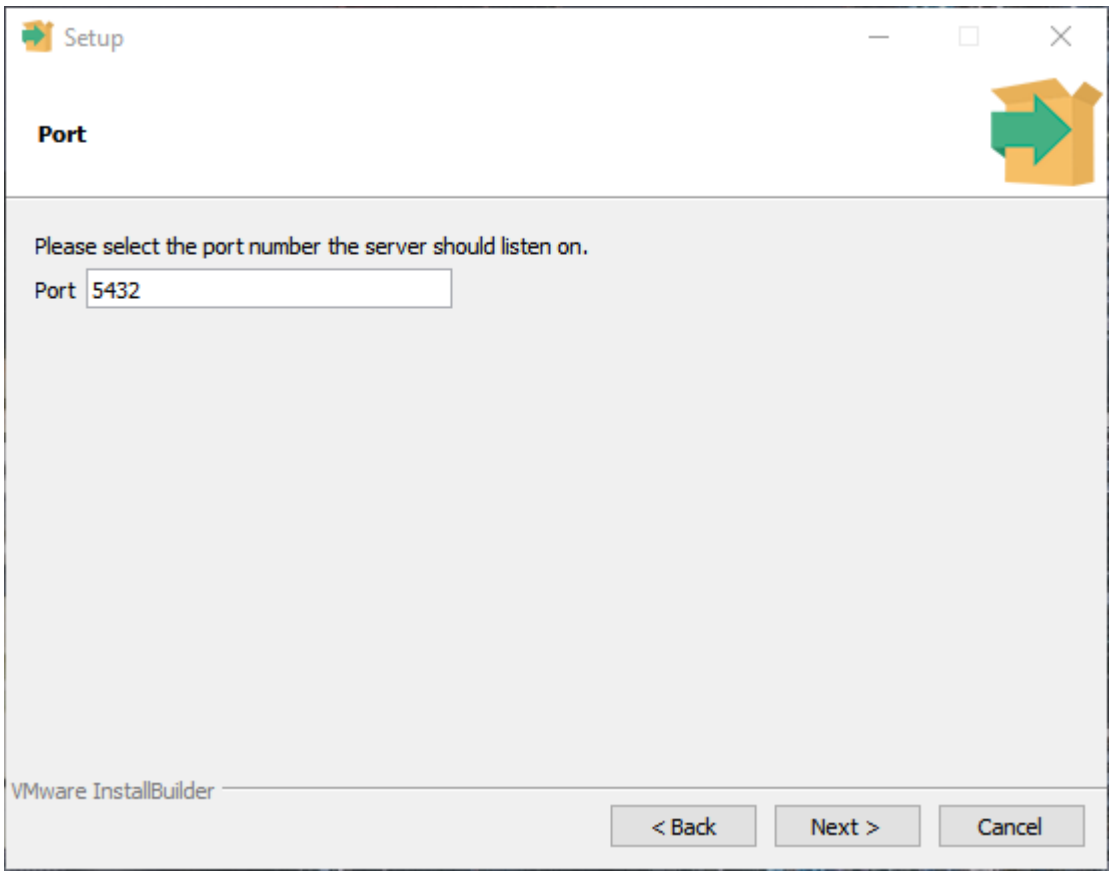

Figure 33: PostgreSQL Setup Port

In the next step the locale to be used is chosen. This setting affects the language, alphabets, number formatting used in the database cluster. By choosing the default locale, the locale of the operating system is used [153].

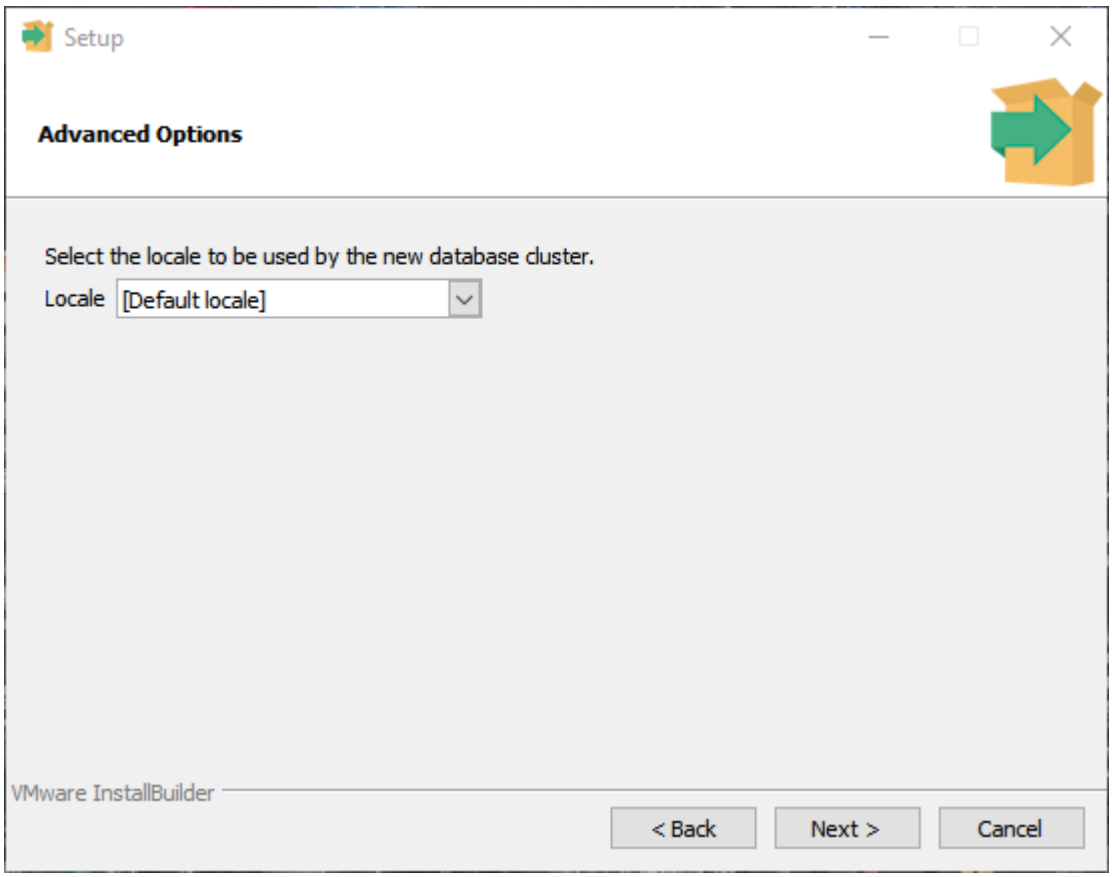

Figure 34: PostgreSQL Setup Advanced Options

The next window summarizes all chosen options. Figure 30 depicts a setup where all installation components are chosen, and the default options have been selected.

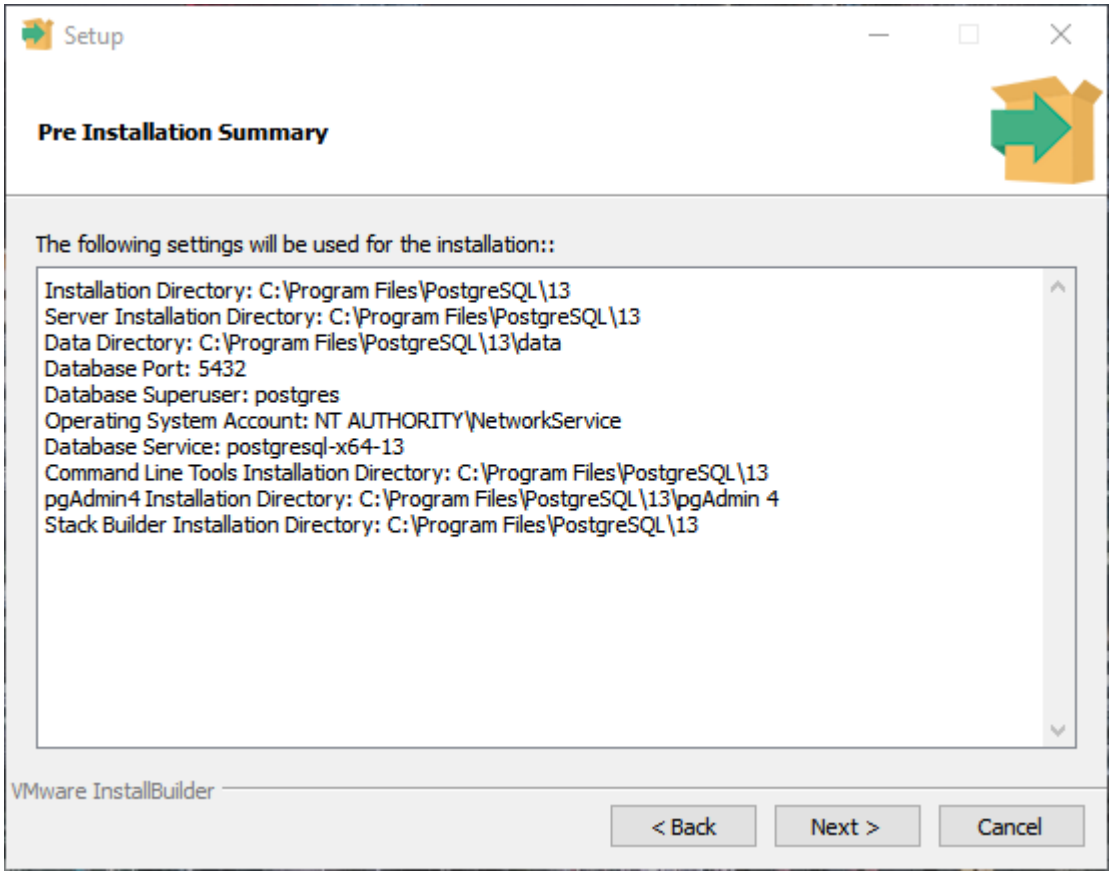

Figure 35: PostgreSQL Pre Installation Summary

The next window informs the user that the setup is ready to start the installation.

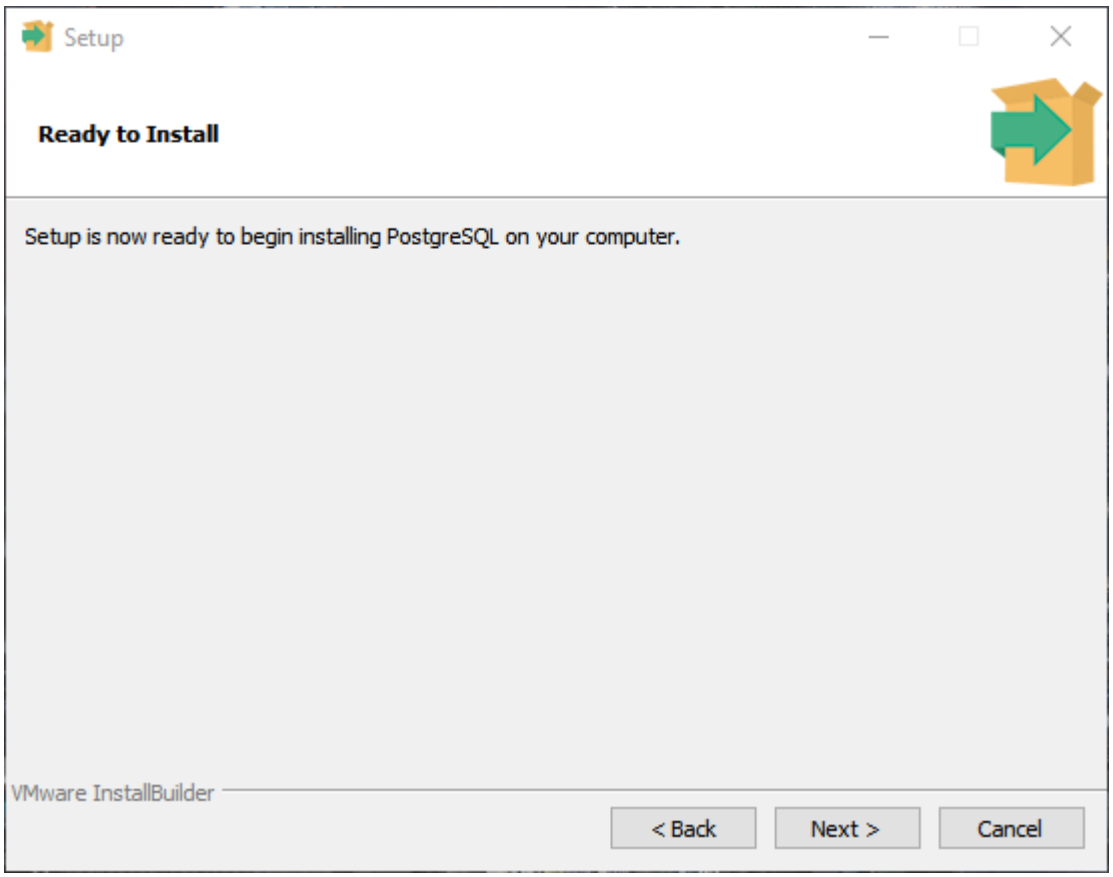

Figure 36: PostgreSQL Setup Ready to Install

After the installation process has been concluded the setup is finished. The Stack Builder feature is not relevant for the use case described.

Figure 37: Completing the PostgreSQL Setup Wizard

Even though the database comes with a graphic administration interface called `pgAdmin 4`, this tutorial uses a command line interface to work with the database. To use PostgreSQL from within the Windows PowerShell or the Command prompt, it is necessary to add the database management system to the Environment Variables.

To accomplish this, first the `About your PC` window must be opened on a computer running Microsoft Windows 10. This window can be easily found by typing `about` in the Windows Search. After clicking on Advanced system settings on the right a smaller window with system properties is opened.

After clicking on `Environment Variables,` a new window is opened. By selecting the Variable `Path` on the lower half under System Variables and clicking on the `Edit` button, the environment variables can be accessed.

By clicking on `New` and then `Browse` the `bin` folder from within the PostgreSQL installation directory needs to be chosen: `C:\Program Files\PostgreSQL\13\bin.` Afterwards the choice is confirmed by clicking

on the OK button. After restarting all currently open Windows Powershell or command line windows the PostgreSQL database management system can be accessed with the command psql.

Like Apache Tomcat, the PostgreSQL server which allows access to the databases is started and stopped by its corresponding service. By typing services.msc in the command line or the Windows Powershell, all Windows Services are listed. By right-clicking on the entry postresql-x64-13 the server can be started or stopped. Since its default Startup Type is automatic, should a web application require access to the database and the service is not currently running it will be started automatically.

9.4.2. Setting up a Database for treeshop

As an alternative to this guide, a web page included in both web applications, that come included with this thesis offers an easy way to quickly copy and paste all commands. It can be accessed from: http://localhost:8080/helloworld/support

After the installation has been accomplished, now a separate database for the Apache Tomcat server is created. To begin with the command: psql postgres postgres, allows access to the database management system. Psql is the environmental variable used to communicate to the command line that PostgreSQL is to be addressed. The following variable postgres signals the database management that the default database postgres is to be accessed. The repetition of the variable postgres signals the database management system to access the default database postgres as the superuser postgres. Afterwards the password for the superuser account, which has been defined during the installation process needs to be typed in.

```
PS C:\> psql postgres postgres
Password for user postgres:
psql (13.1)
Type "help" for help.

postgres=#
```

Listing 31: PostgreSQL Start Database Management System

Using this superuser account now a new database is created, using the command: `CREATE DATABASE shop;` Since the database is mainly used for an online shop web application it is called `shop`.

```
postgres=# CREATE DATABASE shop;
CREATE DATABASE
postgres=#
```

Listing 32: PostgreSLQ Create Database shop

Afterwards a connection to the newly created database is established.

```
postgres=# \connect shop;
You are now connected to database "shop" as user "postgres".
```

Listing 33: PostgreSQL Connect to Database Shop

Now, three tables for the web application shop be created.

```
shop=# CREATE TABLE tree (
shop(# tree_id serial PRIMARY KEY,
shop(# name varchar (40) NOT NULL,
shop(# price varchar (20) NOT NULL,
shop(# height varchar (20),
shop(# picture varchar (100)
shop(# );
CREATE TABLE
```

Listing 34: PostgreSQL Create Table tree

```
shop=# CREATE TABLE customer (
shop(# customer_id serial PRIMARY KEY,
shop(# username varchar (40) NOT NULL,
shop(# password varchar (100) NOT NULL,
shop(# receives_mail boolean DEFAULT FALSE
shop(# );
CREATE TABLE
```

Listing 35: PostgreSQL Create Table customer

```
shop=# CREATE TABLE cart (
shop(# tree_id integer REFERENCES tree ON UPDATE CASCADE ON DELETE CASCADE,
shop(# customer_id integer REFERENCES customer ON UPDATE CASCADE ON DELETE CASCADE,
shop(# quantity integer,
shop(# PRIMARY KEY (tree_id, customer_id)
shop(# );
CREATE TABLE
```

Listing 36: PostgreSQL Create Table cart

Each user and tree both have a unique ID assigned to them. This is done automatically by the sequence object available in PostgreSQL. It assigns a unique identifier for each row of the table [154].

In the next step a unique user for Apache Tomcat is created.

```
shop=# CREATE USER cattus WITH ENCRYPTED PASSWORD 'tomtom12';
CREATE ROLE
```

Listing 37: PostgreSQL Create User cattus

Since no schema has been defined, all previously created tables are assigned to the schema `public`. The following commands grants the newly created user the necessary rights to operate on the tables.

```
shop=# GRANT ALL ON ALL TABLES IN SCHEMA public TO cattus;
GRANT
```

Listing 38: PostgreSQL Grant Rights to cattus

Finally, special permissions need to be given, so that the new user may work on tables using sequences. Even though sequences look like fields, they are single-row tables that require explicit permission to perform functions on them. Each time a new row is added, a function is performed to auto increment the sequence number, which is stored as `bigint` [155].

```
shop=# GRANT ALL ON SEQUENCE tree_tree_id_seq TO cattus;
GRANT
```

Listing 39: PostgreSQL Grant Sequence Rights tree_id

The same operation needs to be repeated for the customer table.

```
shop=# GRANT ALL ON SEQUENCE customer_customer_id_seq TO cattus;
GRANT
```

Listing 40: PostgreSQL Grant Sequence Rights customer_id

In the next step six example products are added to the table tree.

```
shop=# INSERT INTO tree (name, price, height, picture) VALUES
shop-# ('Oak', 50, 20, '/files/Oak.jpg'),
shop-# ('Birch', 100, 15, '/files/Birch.jpg'),
shop-# ('Willow', 150, 10, '/files/Willow.jpg'),
shop-# ('Beech', 180, 40, '/files/Beech.jpg'),
shop-# ('Pine', 250, 55, '/files/Pine.jpg'),
shop-# ('Maple', 300, 40, '/files/Maple.jpg');
INSERT 0 6
```

Listing 41: PostgreSQL Insert tree

Afterwards, three example accounts are created. All entries follow the same password convention: the password for `bigspender@quickmail.com` is bigspender.

```
shop=# INSERT INTO customer (username, password, receives_mail) VALUES
shop-# ('bigspender@quickmail.com', '$2a$12$21bWzhN.SG22WomYuO.w5uh0djvgsZPC5YZMasE.010.Zteeww8Eq', TRUE),
shop-# ('smallspender@slowmail.com', '$2a$12$gA49Tgpzh9KL3aKTonV8nuxvzyURj4xQ31eQWDNfZ5uM/0HmPGMpW', TRUE),
shop-# ('mediumspender@sendbypigeon.com', '$2a$12$xr4fQfB1VWqGfKChfUSVl.x2VrCc../FaXGPSAP173Tgm3cUsfy.i', TRUE);
INSERT 0 3
```

Listing 42: PostgreSQL Insert customer

This concludes the setup process for the database.

9.5. Debug Code

The following lines of code, written by Rony G. Flatscher can be placed at the top and bottom of a script, creating a detailed list of exceptions that occurred. This is particularly useful to determine problems related to database operations.

```
SIGNAL ON SYNTAX

/* CODE */

RETURN

SYNTAX:                    -- label to jump to, if syntax condition gets raised
above
   co=condition("object")  -- get condition object
       -- get Java exception chain as a Rexx string, insert "<br>" after LF
("0a"x)
   strChain=ppJavaExceptionChain(co)~changeStr("0a"x, "0a"x "<br>")
   .error~say(strChain)    -- write to error stream
   say strChain            -- write to output stream (generates as HTML text)
   raise propagate         -- propagate exception (recreate exception in caller)
::REQUIRES "BSF.CLS" --enable Java support
```

Listing 43: Debug Code

9.6. MailHog

The MailHog software can be found on the following web page: https://github.com/mailhog/MailHog By clicking on Releases, the latest version can be downloaded. Both a 64-Bit and a 32-Bit version are available.

On the Microsoft Windows operation system, once downloaded, the file `MailHog_windows_amd64.exe` / `MailHog_windows_amd32.exe` will open a command prompt window on execution. As long as this window remains open, the software is running.

After configuring Jakarta Mail to send e-mails with the SMTP server on the localhost and port 1025, all incoming and outgoing e-mails will be processed by MailHog. The username and password choice does not matter. The program creates an inbox, which can be accessed by a web browser, using the URL: `http://localhost:8025/`

This convenient interface allows to view e-mails from the receiver's perspective. Given how easily this e-mail testing environment is set up, the author highly recommends.

Glossary

API	Application Programming Interface
ASF	Apache Software Foundation
BSF	Bean Scripting Framework
BSF4ooRexx	Bean Scripting Framework for Open Object Rexx
CSS	Cascading Style Sheet
CGI	Common Gateway Interface
DOM	Document Object Model
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP address	Internet Protocol address
Jakarta EE	Jakarta Enterprise Edition
JAR	Java Archive
Java EE	Java Enterprise Edition
JDBC	Java Database Connectivity
JDK	Java Development Kit
JNDI	Java Naming and Directory Interface
JRE	Java Runtime Environment
JSP	Jakarta Server Pages
JSR	Java Specification Request
MIME	Multipurpose Internet Mail Extensions
ooRexx	Open Object Rexx
OWASP	Open Web Application Security Project
SSL	Secure Socket Layer

SMTP	Simple Mail Transmission Protocol
SQL	Structured Query Language
Taglib	Tag Library
TLD	Tag Library Descriptor
TLS	Transport Layer Security
Tomcat	Apache Tomcat Software
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WAR	Web Application Archive
Webapp	Web Application
Windows	Microsoft Windows

Figures

Listings

References

[1] World Wide Web Consortium, "Tim Berners-Lee," World Wide Web Consortium, 16 July 2020. [Online]. Available: https://www.w3.org/People/Berners-Lee/. [Accessed 10 September 2020].

[2] R. Fielding, J. Gettys, M. J., F. H., L. Masinter, P. Leach and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," Internet Engineering Task Force, June 19999. [Online]. Available: https://tools.ietf.org/html/rfc2616. [Accessed 10 September 2020].

[3] Rexx Language Association, " About Open Object Rexx," Rexx Language Association, [Online]. Available: https://www.oorexx.org/about.html. [Accessed 26 December 2020].

[4] Jakarta Server Pages Team, "Jakarta Server Pages Specification, Version 3.0," Eclipse Foundation, 21 October 2020. [Online]. Available: https://jakarta.ee/specifications/pages/3.0/jakarta-server-pages-spec-3.0.html. [Accessed 10 January 2021].

[5] J. Ousterhout, "Scripting: Higher-Level Programming for the 21st Century," *IEEE Compute,* vol. 31, no. 03, pp. 23-30, 1998.

[6] R. Sedgewick and K. Wayne, "8.2 Compilers, Interpreters, and Emulators," Princeton University, 24 October 2006. [Online]. Available: https://introcs.cs.princeton.edu/java/82compiler/. [Accessed 22 September 2020].

[7] R. Toal, "Scripting Languages," Loyola Marymount University, [Online]. Available: https://cs.lmu.edu/~ray/notes/scriptinglangs/. [Accessed 22 September 2020].

[8] J. Gosling and M. Henry, "The Java Language Environment," Oracle, May 1996. [Online]. Available: https://www.oracle.com/java/technologies/language-environment.html. [Accessed 23 September 2020].

[9] Oracle, "About the Java Technology," Oracle, [Online]. Available: https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html. [Accessed 23 September 2020].

[10] C. Hermansen, "Using external libraries in Java," Opensource.com, 11 February 2020. [Online]. Available: https://opensource.com/article/20/2/external-libraries-java. [Accessed 26 December 2020].

[11] GeeksforGeeks, "Jar files in Java," GeeksforGeeks, 26 May 2017. [Online]. Available: https://www.geeksforgeeks.org/jar-files-java/. [Accessed 26 December 2020].

[12] Oracle, "JSR 223: Scripting for the JavaTM Platform," Oracle, [Online]. Available: https://jcp.org/en/jsr/detail?id=223. [Accessed 26 December 2020].

[13] Oracle, "Java Platform, Standard Edition Java Scripting Programmer's Guide," Oracle, [Online]. Available: https://docs.oracle.com/javase/10/scripting/toc.htm. [Accessed 24 September 2020].

[14] A. Fleck, "Prologue on Program Specification," University of Iowa, [Online]. Available: http://homepage.divms.uiowa.edu/~fleck/spec.html. [Accessed 06 September 2020].

[15] Oracle, "Interface ScriptEngine," Oracle, [Online]. Available: https://docs.oracle.com/javase/10/docs/api/javax/script/ScriptEngine.html. [Accessed 24 September 2020].

[16] J. O'Conner, "Scripting for the Java Platform," Oracle, July 2006. [Online]. Available: https://www.oracle.com/technical-resources/articles/javase/scripting.html. [Accessed 04 January 2021].

[17] GeeksforGeeks, "JavaBean class in Java," GeeksforGeeks, 14 September 2017. [Online]. Available: https://www.geeksforgeeks.org/javabean-class-java/. [Accessed 25 September 2020].

[18] Apache Software Foundation, "BSF FAQ," Apache Software Foundation, 17 October 2011. [Online]. Available: https://commons.apache.org/proper/commons-bsf/faq.html. [Accessed 25 September 2020].

[19] Apache Software Foundation, "BSF Manual," Apache Software Foundation, 17 October 2011. [Online]. Available: https://commons.apache.org/proper/commons-bsf/manual.html. [Accessed 25 September 2020].

[20] S. Weerawarana, M. J. Duftler, S. Ruby, O. Gruber, D. Schwarz and R. G. Flatscher, "Class BSFManager," 13 September 2008. [Online]. Available: http://wi.wu.ac.at:8002/rgf/rexx/bsf4rexx/current/docs/docs.apache.bsf/org/apache/bsf/BSFManager.html. [Accessed 25 September 2020].

[21] Apache Software Foundation, "BSF About," Apache Software Foundation, 2011 October 2011. [Online]. Available: https://commons.apache.org/proper/commons-bsf/index.html. [Accessed 25 September 2020].

[22] R. G. Flatscher, Introduction to REXX and ooRexx, Vienna: Facultas, 2013.

[23] R. G. Flatscher, "Java Bean Scripting With Rexx," in *Proceedings of the „12th International Rexx Symposium“*, Raleigh, North Carolina, USA, 2001.

[24] R. G. Flatscher, "The Augsburg Version of BSF4Rexx," in *Proceedings of the „14th International Rexx Symposium"*, Raleigh, NorthCarolina, USA, 2003.

[25] R. G. Flatscher, "The 2019 Edition of BSF4ooRexx," in *Proceedings of the "2019 International RexxLA Symposium"*, Hursley, Great Britain, 2019.

[26] R. G. Flatscher, "Camouflaging Java as Object REXX," in *Proceedings of the "2004 International Rexx Symposium"*, Research Triangle Park, North Carolina, 2004.

[27] The Editors of Encyclopaedia Britannica, " Protocol," Encyclopaedia Britannica, 31 August 2018. [Online]. Available: https://www.britannica.com/technology/protocol-computer-science. [Accessed 08 September 2020].

[28] H.-C. Chua, "HTTP (HyperText Transfer Protocol)," Nanyang Technological University, 20 October 2009. [Online]. Available: https://personal.ntu.edu.sg/ehchua/programming/webprogramming/HTTP_Basics.html. [Accessed 07 January 2021].

[29] World Wide Web Consortium, "HTML 5.2," World Wide Web Consortium, 14 December 2017. [Online]. Available: https://www.w3.org/TR/html52/. [Accessed 27 December 2020].

[30] w3schools, "HTML Introduction," w3schools, [Online]. Available: https://www.w3schools.com/html/html_intro.asp. [Accessed 27 December 2020].

[31] H. W. Lie and B. Bos, "Cascading Style Sheets, level 1," World Wide Web Consortium, 17 December 1996. [Online]. Available: https://www.w3.org/TR/REC-CSS1-961217. [Accessed 15 December 2020].

[32] H.-C. Chua, "Java Server-Side Programming," Nanyang Technological University, October 2012. [Online]. Available: https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaServlets.html. [Accessed 01 September 2020].

[33] Oracle, "Java Servlet Technology Overview," Oracle, [Online]. Available: https://www.oracle.com/java/technologies/servlet-technology.html. [Accessed 02 September 2020].

[34] M. Tyson, "What are Java servlets? Request handling for Java web applications," InfoWorld, 17 October 2018. [Online]. Available: https://www.infoworld.com/article/3313114/what-is-a-java-servlet-request-handling-for-java-web-applications.html. [Accessed 02 September 2020].

[35] A. Singh, "Introduction to Java Servlets," GeeksforGeeks, 23 October 2019. [Online]. Available: https://www.geeksforgeeks.org/introduction-java-servlets/. [Accessed 02 September 2020].

[36] Jakarta Servlet Team, "Jakarta Servlet Specification, Version 5.0," Eclipse Foundation, 07 September 2020. [Online]. Available: https://jakarta.ee/specifications/servlet/5.0/jakarta-servlet-spec-5.0.html. [Accessed 10 January 2021].

[37] N. Freed and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies," Internet Engineering Task Force, November 1996. [Online]. Available: https://tools.ietf.org/html/rfc2045. [Accessed 06 September 2020].

[38] Eclipse Foundation, "Class HttpServlet," Eclipse Foundation, [Online]. Available: https://jakarta.ee/specifications/platform/9/apidocs/jakarta/servlet/http/HttpServlet.html. [Accessed 02 September 2020].

[39] Eclipse Foundation, "Interface Servlet," Eclipse Foundation, [Online]. Available: https://jakarta.ee/specifications/platform/9/apidocs/jakarta/servlet/servlet. [Accessed 02 September 2020].

[40] M. Tyson, "What is JSP? Introduction to JavaServer Pages," InfoWorld, 29 January 2019. [Online]. Available: https://www.infoworld.com/article/3336161/what-is-jsp-introduction-to-javaserver-pages.html. [Accessed 02 September 2020].

[41] Oracle, "JSP Scriptlets," Oracle, [Online]. Available: https://docs.oracle.com/javaee/5/tutorial/doc/bnaou.html. [Accessed 28 December 2020].

[42] Oracle, "JSP Tag Libraries," Oracle, [Online]. Available: https://docs.oracle.com/cd/B14099_19/web.1012/b14014/taglibs.htm#i1012403. [Accessed 28 December 2020].

[43] S. Ryabenkiy, *Java Web Scripting and Apache Tomcat,* Vienna: Vienna University of Economics and Business, 2010.

[44] M. Tyson, "What is Tomcat? The original Java servlet container," InfoWorld, 19 December 2019. [Online]. Available: https://www.infoworld.com/article/3510460/what-is-apache-tomcat-the-original-java-servlet-container.html. [Accessed 28 December 2020].

[45] TEDBlog, "James Duncan Davidson," TEDBlog, [Online]. Available: https://blog.ted.com/author/duncandavidson/. [Accessed 01 September 2020].

[46] Apache Software Foundation, "The Tomcat Story," Apache Software Foundation, [Online]. Available: https://tomcat.apache.org/heritage.html. [Accessed 01 September 2020].

[47] MuleSoft, "Meet Tomcat Catalina," MuleSoft, [Online]. Available: https://www.mulesoft.com/tcat/tomcat-catalina. [Accessed 28 December 2020].

[48] Apache Software Foundation, "Introduction," Apache Software Foundation, 03 December 2020. [Online]. Available: https://tomcat.apache.org/tomcat-10.0-doc/introduction.html. [Accessed 28 December 2020].

[49] Wikipedians, "Apache Tomcat," Wikipedia, 13 December 2020. [Online]. Available: https://en.wikipedia.org/wiki/Apache_Tomcat. [Accessed 28 December 2020].

[50] Apache Software Foundation, "The Coyote HTTP/1.1 Connector," Apache Software Foundation, [Online]. Available: https://tomcat.apache.org/tomcat-4.1-doc/config/coyote.html. [Accessed 28 December 2020].

[51] Apache Software Foundation, "Apache HTTP Server HowTo," Apache Software Foundation, 09 March 2020. [Online]. Available: https://tomcat.apache.org/connectors-doc/webserver_howto/apache.html. [Accessed 28 December 2020].

[52] P. Manh, "The different between Web server, Web container and Application server," GitHub, 01 April 2020. [Online]. Available: https://ducmanhphan.github.io/2020-04-01-The-difference-between-web-server-web-container-application-server/. [Accessed 02 September 2020].

[53] Opensource.com, "What is open source?," Opensource.com, [Online]. Available: https://opensource.com/resources/what-open-source. [Accessed 28 December 2020].

[54] Opensource.org, "Frequently Answered Questions," Opensource.org, [Online]. Available: https://opensource.org/faq. [Accessed 28 December 2020].

[55] Apache Software Foundation, "What is the ASF?," Apache Software Foundation, [Online]. Available: https://www.apache.org/foundation/. [Accessed 01 September 2020].

[56] Apache Software Foundation, "Apache Tomcat," Apache Software Foundation, [Online]. Available: https://tomcat.apache.org/. [Accessed 01 September 2020].

[57] Apache Software Foundation, "Apache License, Version 2.0," Apache Software Foundation, [Online]. Available: https://www.apache.org/licenses/LICENSE-2.0. [Accessed 01 September 2020].

[58] Eclipse Foundation, "About the Eclipse Foundation," Eclipse Foundation, [Online]. Available: https://www.eclipse.org/org/. [Accessed 06 September 2020].

[59] Eclipse Foundation, "Explore Our Members," Eclipse Foundation, [Online]. Available: https://www.eclipse.org/membership/exploreMembership.php. [Accessed 06 September 2020].

[60] A. Tijms, "Transition from Java EE to Jakarta EE," Oracle, 27 February 2020. [Online]. Available: https://blogs.oracle.com/javamagazine/transition-from-java-ee-to-jakarta-ee. [Accessed 02 September 2020].

[61] Oracle, "Java Documentation," Oracle, [Online]. Available: https://docs.oracle.com/en/java/index.html. [Accessed 06 September 2020].

[62] R. Monson-Haefel, "TomEE vs. Tomcat," Tomitribe, 05 December 2019. [Online]. Available: https://www.tomitribe.com/blog/tomee-vs-tomcat/. [Accessed 28 December 2020].

[63] Apache Software Foundation, "Tomcat 10 Software Downloads," Apache Software Foundation, [Online]. Available: https://tomcat.apache.org/download-10.cgi. [Accessed 28 December 2020].

[64] R. G. Flatscher, ""RexxScript" – Rexx Scripts Hosted and Evaluated by Java (Package javax.script)," in *Proceedings of the "The 2017 International Rexx Symposium"*, Amsterdam, The Netherlands, 2017.

[65] Cloudflare, "What do client side and server side mean? | Client side vs. server side," Cloudflare, [Online]. Available: https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/. [Accessed 06 January 2021].

[66] MuleSoft, "Tomcat Configuration - A Step By Step Guide," MuleSoft, [Online]. Available: https://www.mulesoft.com/tcat/tomcat-configuration. [Accessed 28 December 2020].

[67] Apache Software Foundation, "Application Developer's Guide," Apache Software Foundation, 03 December 2020. [Online]. Available: https://tomcat.apache.org/tomcat-10.0-doc/appdev/deployment.html. [Accessed 10 December 2020].

[68] G. Shachor, "Tomcat 3.3 User's Guide," Apache Software Foundation, [Online]. Available: https://tomcat.apache.org/tomcat-3.3-doc/tomcat-ug.html#directory_structure. [Accessed 28 December 2020].

[69] Microfocus, "Deploying and Running Your Application," Microfocus, [Online]. Available: https://supportline.microfocus.com/documentation/books/sx22sp1/pidepl.htm. [Accessed 29 December 2020].

[70] JavaTpoint, "War File," JavaTpoint, [Online]. Available: https://www.javatpoint.com/war-file. [Accessed 29 December 2020].

[71] Baeldung, "How to Deploy a WAR File to Tomcat," Baeldung, 12 February 2020. [Online]. Available: https://www.baeldung.com/tomcat-deploy-war. [Accessed 29 December 2020].

[72] Uniface, "Creating and Deploying a Web Application WAR File," Uniface, [Online]. Available: https://u.uniface.info/docs/1000/uniface/webApps/webDeployment/Prepare_your_Web_environment.htm. [Accessed 29 December 2020].

[73] FileInfo, ".EAR File Extension," FileInfo, 22 March 2019. [Online]. Available: https://fileinfo.com/extension/ear. [Accessed 29 December 2020].

[74] Microsoft, "Introduction to Windows Service Applications," Microsoft, 30 March 2017. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/windows-services/introduction-to-windows-service-applications. [Accessed 18 September 2020].

[75] A. Sharma, "What is Local Host?," GeeksforGeeks, 09 August 2019. [Online]. Available: https://www.geeksforgeeks.org/what-is-local-host/. [Accessed 13 September 2020].

[76] K. Vijay Kulkarni, "14 common network ports you should know," Red Hat, 04 October 2018. [Online]. Available: https://opensource.com/article/18/10/common-network-ports. [Accessed 17 September 2020].

[77] Apache Software Foundation, "Manager App How-To," Apache Software Foundation, 03 December 2020. [Online]. Available: https://tomcat.apache.org/tomcat-10.0-doc/manager-howto.html. [Accessed 29 December 2020].

[78] MuleSoft, "The Tomcat Web app Quick Reference Guide," MuleSoft, [Online]. Available: https://www.mulesoft.com/tcat/tomcat-webapp. [Accessed 29 December 2020].

[79] R. Nazarov, "Tomcat web.xml Configuration Example," Java Code Geeks, 18 March 2015. [Online]. Available: https://examples.javacodegeeks.com/enterprise-java/tomcat/tomcat-web-xml-configuration-example/. [Accessed 10 December 2020].

[80] Eclipse Foundation, "Interface HttpSession," Eclipse Foundation, 2019. [Online]. Available: https://jakarta.ee/specifications/servlet/4.0/apidocs/javax/servlet/http/HttpSession.html. [Accessed 21 October 2020].

[81] R. Ishida, "Character encodings for beginners," W3C, 16 April 2015. [Online]. Available: https://www.w3.org/International/questions/qa-what-is-encoding. [Accessed 21 October 2020].

[82] O. Thereaux, "Don't forget to add a doctype," W3C, 20 August 2002. [Online]. Available: https://www.w3.org/QA/Tips/Doctype. [Accessed 22 October 2020].

[83] webhint, "Use charset `utf-8`," webhint, [Online]. Available: https://webhint.io/docs/user-guide/hints/hint-meta-charset-utf-8/. [Accessed 14 December 2020].

[84] Maggie, "Why is <meta charset="utf-8"> important?," DEV, 19 October 2020. [Online]. Available: https://dev.to/maggiecodes_/why-is-lt-meta-charset-utf-8-gt-important-59hl. [Accessed 14 December 2020].

[85] R. G. Flatscher, "SourceForge BSF4ooRexx Taglibs Readme.md," 24 November 2020. [Online]. Available: https://sourceforge.net/projects/bsf4oorexx/files/Sandbox/rgf/taglibs/beta/. [Accessed 11 December 2020].

[86] N. Lengyel, *BSF4ooRexx: JSP with javax.script Languages,* Vienna, Austria: Vienna University of Economics and Business, 2020.

[87] C. Singh, "Jsp Implicit Objects," BeginnersBook, [Online]. Available: https://beginnersbook.com/2013/11/jsp-implicit-objects/. [Accessed 22 October 2020].

[88] Apache Software Foundation, "Class JspWriter," Apache Software Foundation, [Online]. Available: https://tomcat.apache.org/tomcat-7.0-doc/jspapi/javax/servlet/jsp/JspWriter.html. [Accessed 22 October 2020].

[89] W3Schools, "HTML <link> Tag," W3Schools, [Online]. Available: https://www.w3schools.com/tags/tag_link.asp. [Accessed 14 December 2020].

[90] W3Schools, "HTML File Paths," W3Schools, [Online]. Available: https://www.w3schools.com/html/html_filepaths.asp. [Accessed 14 December 2020].

[91] B. Bos, T. Çelik, I. Hickson and H. W. Lie, "Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification," 12 April 2016. [Online]. Available: https://www.w3.org/TR/CSS2/. [Accessed 15 December 2020].

[92] FileCloud, "Tech tip: How to do hard refresh in Chrome, Firefox and IE?," FileCloud, 06 March 2015. [Online]. Available: https://www.getfilecloud.com/blog/2015/03/tech-tip-how-to-do-hard-refresh-in-browsers/. [Accessed 04 January 2021].

[93] R. G. Flatscher, "External BSF4ooRexx Functions - Overview," 08 December 2010. [Online]. Available: http://wi.wu-wien.ac.at:8002/rgf/rexx/bsf4oorexx/current/additionalResources/refcardBSF4ooRexx.pdf. [Accessed 14 December 2020].

[94] Javatpoint, "welcome-file-list in web.xml," Javatpoint, [Online]. Available: https://www.javatpoint.com/welcome-file-list. [Accessed 17 December 2020].

[95] A. Barth, "HTTP State Management Mechanism," April 2011. [Online]. Available: https://tools.ietf.org/html/rfc6265. [Accessed 27 September 2020].

[96] Eclipse Foundation, "Interface HttpServletRequest," Eclipse Foundation, [Online]. Available: https://jakarta.ee/specifications/servlet/4.0/apidocs/javax/servlet/http/httpservletrequest. [Accessed 22 October 2020].

[97] Eclipse Foundation, "Class Cookie," Eclipse Foundation, [Online]. Available: https://jakarta.ee/specifications/servlet/4.0/apidocs/javax/servlet/http/Cookie.html. [Accessed 22 October 2020].

[98] W. D. Ashley, R. G. Flatscher, M. Hessling, R. McGuire, M. Miesfeld, L. Peedin, R. Tammer and J. Wolfers, "Built-in Functions," Rexx Language Association, 14 August 2009. [Online]. Available: https://www.oorexx.org/docs/rexxref/x23579.htm. [Accessed 22 October 2020].

[99] D. Ragget, A. Le Hors and I. Jacobs, "HTML 4.01 Specification," World Wide Web Consortium, 24 December 1999. [Online]. Available: https://www.w3.org/TR/html401/. [Accessed 14 December 2020].

[100] F. Bohórquez, "HTML Forms: The Action Attribute," Career Karma, 12 August 2020. [Online]. Available: https://careerkarma.com/blog/html-form-action/. [Accessed 16 December 2020].

[101] W3Schools, "HTML <label> Tag," W3Schools, [Online]. Available: https://www.w3schools.com/tags/tag_label.asp. [Accessed 04 January 2021].

[102] w3schools, "HTML <input> required Attribute," w3schools, [Online]. Available: https://www.w3schools.com/tags/att_input_required.asp. [Accessed 20 December 2020].

[103] R. G. Flatscher and G. Müller, "ooRexx 5 Yielding Swiss Army Knife Usability," in *The Proceedings of the Rexx Symposium for Developers and Users*, Hursley, Great Britain, 2019.

[104] baeldung, "Handling Cookies and a Session in a Java Servle," baeldung, 28 February 2020. [Online]. Available: https://www.baeldung.com/java-servlet-cookies-session. [Accessed 23 October 2020].

[105] M. Tyson, "What is JDBC? Introduction to Java Database Connectivity," InfoWorld, 11 April 2011. [Online]. Available: https://www.infoworld.com/article/3388036/what-is-jdbc-introduction-to-java-database-connectivity.html. [Accessed 12 November 2020].

[106] M. Aboagye, "Improve database performance with connection pooling," Stack Overflow, 14 October 2020. [Online]. Available: https://stackoverflow.blog/2020/10/14/improve-database-performance-with-connection-pooling/. [Accessed 15 November 2020].

[107] Apache Software Foundation, "JNDI Datasource How-To," Apache Software Foundation, 06 October 2020. [Online]. Available: https://tomcat.apache.org/tomcat-9.0-doc/jndi-datasource-examples-howto.html. [Accessed 12 November 2020].

[108] M. van Steen and A. S. Tanenbaum, "A brief introduction to distributed systems," *Computing,* vol. 98, no. 10, pp. 967-1009, 2016.

[109] F. T. Marchese, "Naming," Pace University Seidenberg School of CSIS, [Online]. Available: http://csis.pace.edu/~marchese/CS865/Lectures/Chap5/Chapter5.htm. [Accessed 11 November 2020].

[110] T. Sundsted, "JNDI overview, Part 2: An introduction to directory services," InfoWorld, [Online]. Available: https://www.infoworld.com/article/2076901/jndi-overview--part-2--an-introduction-to-directory-services.html. [Accessed 11 November 2020].

[111] S. Claridge, "Serving static content (including web pages) from outside of the WAR using Apache Tomcat," More Of Less, 04 April 2014. [Online]. Available: https://www.moreofless.co.uk/static-content-web-pages-images-tomcat-outside-war/. [Accessed 21 December 2020].

[112] Apache Software Foundation, "Class Loader How-To," Apache Software Foundation, 03 December 2020. [Online]. Available: https://tomcat.apache.org/tomcat-10.0-doc/class-loader-howto.html. [Accessed 10 December 2020].

[113] Apache Software Foundation, "JNDI Resources How-To," Apache Software Foundation, 06 October 2020. [Online]. Available: https://tomcat.apache.org/tomcat-9.0-doc/jndi-resources-howto.html. [Accessed 12 November 2020].

[114] T. Sundsted, "JNDI overview, Part 1: An introduction to naming services," InfoWorld, 01 January 2000. [Online]. Available: https://www.infoworld.com/article/2076888/jndi-overview--part-1--an-introduction-to-naming-services.html. [Accessed 11 November 2020].

[115] Oracle, "Interface Statement," Oracle, [Online]. Available: https://cr.openjdk.java.net/~iris/se/15/latestSpec/api/java.sql/java/sql/Statement.html. [Accessed 18 November 2020].

[116] Oracle, "Interface ResultSet," Oracle, [Online]. Available: https://cr.openjdk.java.net/~iris/se/15/latestSpec/api/java.sql/java/sql/ResultSet.html. [Accessed 18 November 2020].

[117] J. Holý and M. Mære, "JDBC: What resources you have to close and when?," DZone, 13 February 2013. [Online]. Available: https://dzone.com/articles/jdbc-what-resources-you-have. [Accessed 20 December 2020].

[118] European Union, "Data protection and online privacy," European Union, 09 March 2020. [Online]. Available: https://europa.eu/youreurope/citizens/consumers/internet-telecoms/data-protection-online-privacy/index_en.htm. [Accessed 25 December 2020].

[119] A. Beylkin, "Opt in checkboxes & consent for email marketing," Words on Marketing, [Online]. Available: https://www.amandabeylkin.com/marketing-blog/opt-in-checkboxes-consent-email-marketing/. [Accessed 25 December 2020].

[120] R. Degges, "Everything You Ever Wanted to Know About Secure HTML Forms," Twilio, 30 September 2017. [Online]. Available: https://www.twilio.com/blog/2017/09/everything-you-ever-wanted-to-know-about-secure.html-forms.html. [Accessed 18 November 2020].

[121] G. Barré, "How to store a password in a web application?," Meziantou's Blog, 17 June 2019. [Online]. Available: https://www.meziantou.net/how-to-store-a-password-in-a-web-application.htm. [Accessed 19 November 2020].

[122] H. Qureshi, "Hash Functions," Nakamoto, 29 December 2019. [Online]. Available: https://nakamoto.com/hash-functions/. [Accessed 19 November 2020].

[123] OWASP, "Password Storage Cheat Sheet," OWASP, [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#password-hashing-algorithms. [Accessed 20 November 2020].

[124] N. Provos and D. Mazière, "A Future-Adaptable Password Scheme," in *Proceedings of the FREENIX Track:1999 USENIX Annual Technical Conference*, Monterey, California, USA, 1999.

[125] OWASP, "OWASP Top Ten," OWASP, [Online]. Available: https://owasp.org/www-project-top-ten/. [Accessed 18 November 2020].

[126] Z. Su and G. Wassermann, "The Essence of Command Injection Attacks in Web Application," in *Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Charleston, South Carolina, 2006.

[127] P. Kumar, "JDBC Statement vs PreparedStatement – SQL Injection Example," JournalDev, [Online]. Available: https://www.journaldev.com/2489/jdbc-statement-vs-preparedstatement-sql-injection-example. [Accessed 18 November 2020].

[128] B. Brumm, "How to Escape Single Quotes in SQL," Database Star, 01 May 2017. [Online]. Available: https://www.databasestar.com/sql-escape-single-quote/. [Accessed 18 November 2020].

[129] Cloudflare, "What Is HTTPS?," Cloudflare, [Online]. Available: https://www.cloudflare.com/learning/ssl/what-is-https/. [Accessed 19 November 2020].

[130] Guru99, "Difference between Cookie and Session," Guru99, [Online]. Available: https://www.guru99.com/difference-between-cookie-session.html. [Accessed 02 January 2021].

[131] Pankaj, "Session Management in Java – HttpServlet, Cookies, URL Rewriting," JournalDev, [Online]. Available: https://www.journaldev.com/1907/java-session-management-servlet-httpsession-url-rewriting. [Accessed 20 December 2020].

[132] JavaTPoint, "https://www.javatpoint.com/http-session-in-session-tracking," JavaTPoint, [Online]. Available: https://www.javatpoint.com/http-session-in-session-tracking. [Accessed 20 December 2020].

[133] N. H. Minh, "How to configure session timeout in Tomcat," CodeJava, 06 August 2019. [Online]. Available: https://www.codejava.net/servers/tomcat/how-to-configure-session-timeout-in-tomcat. [Accessed 20 December 2020].

[134] W3Schools, "HTML <img> src Attribute," W3Schools, [Online]. Available: https://www.w3schools.com/tags/att_img_src.asp. [Accessed 21 December 2020].

[135] S. Kamani, "Web security essentials - Sessions and cookies," { Soham Kamani }, 08 January 2017. [Online]. Available: https://www.sohamkamani.com/blog/2017/01/08/web-security-session-cookies/. [Accessed 03 January 2021].

[136] C. Broadley, "Form Enctype HTML Code: Here's How It Specifies Form Encoding Type," HTML.com, [Online]. Available: https://html.com/attributes/form-enctype/. [Accessed 23 December 2020].

[137] Oracle, " Creating and Configuring JSPs," Oracle, [Online]. Available: https://docs.oracle.com/cd/E13222_01/wls/docs92/webapp/configurejsp.html. [Accessed 24 December 2020].

[138] Guru99, "JSP File Upload & File Download Program Examples," Guru99, [Online]. Available: https://www.guru99.com/jsp-file-upload-download.html. [Accessed 24 December 2020].

[139] Apache Software Foundation, "Annotation Type MultipartConfig," Apache Software Foundation, [Online]. Available: https://tomcat.apache.org/tomcat-10.0-doc/servletapi/jakarta/servlet/annotation/MultipartConfig.html. [Accessed 24 December 2020].

[140] Eclipse Foundation, "Uploading Files with Jakarta Servlet Technology," Eclipse Foundation, [Online]. Available: https://eclipse-ee4j.github.io/jakartaee-tutorial/servlets011.html. [Accessed 24 December 2020].

[141] N. H. Minh, "Java File Upload Example with Servlet 3.0 API," CodeJava, 27 June 2019. [Online]. Available: https://www.codejava.net/java-ee/servlet/java-file-upload-example-with-servlet-30-api. [Accessed 24 December 2020].

[142] L. Hubmaier, *Tomcat Web Server: CGI vs. Servlet,* Vienna Austria: Vienna University of Economics and Business, 2017.

[143] Apache Software Foundation, "CGI How To," Apache Software Foundation, 03 December 2020. [Online]. Available: https://tomcat.apache.org/tomcat-10.0-doc/cgi-howto.html. [Accessed 06 January 2021].

[144] Eclipse Foundation, "Jakarta Mail FAQ," Eclipse Foundation, [Online]. Available: https://eclipse-ee4j.github.io/mail/FAQ#1. [Accessed 24 December 2020].

[145] The Eclipse Foundation, "Jakarta Activation," The Eclipse Foundation, [Online]. Available: https://eclipse-ee4j.github.io/jaf/. [Accessed 24 December 2020].

[146] S. Kandula, "Example on getParameterValues() method of Servlet Request," Java4s, 28 January 2013. [Online]. Available: https://www.java4s.com/java-servlet-tutorials/example-on-getparametervalues-method-of-servlet-request/. [Accessed 24 December 2020].

[147] GeeksforGeeks, "Properties Class in Java," GeeksforGeeks, 24 November 2020. [Online]. Available: https://www.geeksforgeeks.org/java-util-properties-class-java/. [Accessed 24 December 2020].

[148] G. Mayer, *SCRIPTING THE ODF TOOLKIT IN PRACTICAL USE,* Vienna, Austria: Vienna University of Economics and Business, 2012.

[149] Tutorials Point, "JavaMail API - Core Classes," Tutorials Point, [Online]. Available: https://www.tutorialspoint.com/javamail_api/javamail_api_core_classes.htm. [Accessed 24 December 2020].

[150] Eclipse Foundation, "Uses of Class jakarta.mail.Message.RecipientType," Eclipse Foundation, [Online]. Available: https://jakarta.ee/specifications/mail/2.0/apidocs/jakarta/mail/class-use/message.recipienttype. [Accessed 24 December 2020].

[151] R. Kumar, "How to Create VirtualHost in Tomcat 9/8/7," TecAdmin, [Online]. Available: https://tecadmin.net/create-virtualhost-in-tomcat/. [Accessed 12 September 2020].

[152] Apache Software Foundation, "The Server Component," Apache Software Foundation, [Online]. Available: https://tomcat.apache.org/tomcat-10.0-doc/config/server.html. [Accessed 10 December 2020].

[153] The PostgreSQL Global Development Group, "23.1. Locale Support," The PostgreSQL Global Development Group, [Online]. Available: https://www.postgresql.org/docs/current/locale.html. [Accessed 09 December 2020].

[154] The PostgreSQL Global Development Group, "9.17. Sequence Manipulation Functions," The PostgreSQL Global Development Group, [Online]. Available: https://www.postgresql.org/docs/current/functions-sequence.html. [Accessed 18 December 2020].

[155] S. Weiss, "error handling: permission denied for sequence _id_seq…," /* Code Comments */, 20 November 2018. [Online]. Available: https://stephencharlesweiss.com/20181120-error-handling-permission-denied-for-sequence-_id_seq/. [Accessed 18 December 2020].

Images Used

Background, by Robert Balog: `https://pixabay.com/photos/landscape-nature-forest-fog-misty-975091/`

Oak, by Kevan Craft: `https://pixabay.com/photos/tree-oak-landscape-view-field-402953/`

Birch, by Анатолий Стафичук: https://pixabay.com/photos/summer-landscape-background-dawn-2913409/

Willow, by Mabel Amber: https://pixabay.com/photos/weeping-willow-pond-water-swan-4334489/

Beech, by Couleur: https://pixabay.com/photos/tree-beech-deciduous-tree-old-tree-3601155/

Pine, by Szabolcs Molnar: https://pixabay.com/photos/pine-forest-pine-trees-forest-pine-5572944/

Maple, by Free-Photos: https://pixabay.com/photos/maple-autumn-season-fall-foliage-984420/