Michael's book list
Deutsch
English
Search books by title, subtitle or a
Search
CSS Database Documentation Dot.NET HTML Java JavaScript Linux Management and Methods Other Programming languages SAP UML Video Virtualisization Web Windows
book list
review list
welcome page
Admin
Michael's blog
welcome page | Michael's blog | About

Welcome to my IT book-reviews

Discover the books I've reviewed. Get references to the printed publication or links to online reviews.

Please choose from the categories or enter a term for searching. Or you might choose from the list of reviews published on this site.

To cover parts of the server fees, I participate an Amazone partner progam. I you follow a link from a cover image to buy something at Amazon you will help to finance this site. Of course, you may buy the books where ever you like.

This site is powered by JavaServer Faces technology. Read about the implementation as well as my exhaustive JSF turorial in my blog [blog.mueller-bruehl.de].

Latest reviews on this site:

- The Art of Unit Testing
- DevOps for Developers
- IT-gestütztes Prozessmanagement im Gesundheitswesen
- Tangled Web
- Microsoft SQL Thinking

Web Development with Java and JSF

A Journey through

Java EE Technologies

whilst developing Web Applications with JavaServer Faces.

Michael Müller

Web Development with Java and JSF

A Journey through Java EE Technologies whilst developing Web Applications with JavaServer Faces.

Michael Müller

This book is for sale at http://leanpub.com/jsf

This version was published on 2018-02-12

Tweet This Book!

Please help Michael Müller by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just purchased ”Web Development with Java and JSF” by @muellermi.
https://leanpub.com/jsf

The suggested hashtag for this book is #JavaWebDevelopment.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#JavaWebDevelopment

The web is everywhere. And today's software often uses a web interface to use it from any place - via intranet within a company or via internet worldwide. Java is a well established programming language, not only for business critical applications, but for web applications too. This way, it is a good choice to use Java for the web.

JavaServer Faces is a web framework build upon the Java servlet technology which makes it easy to construct a UI from a set of reusable UI components. It provides a simple model for wiring client-generated events to server-side application code. Depending on the developers preference, the client side can be constructed of tags which hides most of that HTML stuff, or it can build up almost with HTML, flavored with some small special markers.

This book introduces into web development (not only) with Java and JSF. Based on this, complete, small to comprehensive applications are build. The goal of this book is to create a good foundation in web development and take the reader to a more sophisticated level. To pursue this goal, related JEE technologies like persistence (JPA), CDI, container based security, WebSockets, test tools and more are discussed.

For a couple of experienced developers, web development still feels a bit unusual. It's not possible to develop an application by just one programming language, controlling everything by just one program (maybe consisting of a bunch of loosely coupled tiers). Browser and application, these are apparently two distinct worlds. The browser just queries some content from a server to display it at the client, and then forgets about the connection. Thus, to continue with the application, the client must include some identifier into the next call to enable the server remembering and restoring the last state. JSF as part of the Java universe is made for this. But you need more. And for those developers, who are not familiar with HTML, CSS and other stuff, I add some introductory chapters in the appendix to help.

Enjoy!

Contents

"I bought your book "Web Development with Java and JSF" and started working through this. I am a Java developer, but also developing and maintaining web applications for Karolinska Institutet which is a big medical university in Stockholm.

Your book is very well written, with practical work and good explanation of the subject area. I want to follow the book as you develop the it further."

(Pratap Chatterjee)

"I have just purchased your book "Web Development with Java and JSF" and my initial impression is very good. I've already noticed you explaining things that aren't even mentioned in a couple other books I have on JSF."

"Once again I must compliment you. Explanations like this: "#{…} denotes the Expression Language (EL), which is used to glue parts of the user interface to the managed bean." are excellent! Your one paragraph is clear and easily understandable. Other books are terrible at explaining this simple yet vital concept. And shortly after that your "But, a picture is worth a thousand words." is excellent."

(John Wright)

The picture John mentioned shows a simplified relation between browser display, HTML page and Java code. The only drawback: Drawing such pictures takes a lot of time. But, I'm going to treat this comment as an incentive to draw more pictures.

Preliminary note

This book became first published in a very early stage. Usually, it takes six month to two years to write such a book. Publishing early allows you to gain information as soon as possible. You may have noticed, that some publishers of printed books offer early access programs too: It's possible to buy a book long time before it is printed or published as soft copy. Unlike as in general for such a program, there is no need to pay a fixed price for this book. It's up to you to decide whether you pay the recommended price and get all updates, or just pay the minimum price.

Albeit it is possible for you, to earn the book in an early stage for low fee, I hope, you would like to pay a fair price for good work. If you don't like to buy a pig in a poke, I suggest, to buy the book for a lower price, and, depending on completion, buy it again later on. By and by, I'm going to increase the minimum price, depending on the degree of completion.

I prepared some yet empty chapters into this early version, to give you an impression, what you can expect in future versions. And be assured, there will be more.

There is one special reason for me to start with a very low price: I'd like to give you the book nearly for free, if you would help me to improve this book. Since I'm not a native English speaker it would be helpful if somebody revises grammar and diction. And technical reviewers are welcome too. As a thank-you gift I'm going to acknowledge all those people, who provided significant support. To me, it was a great pleasure to help other authors.

As stated above, this book is in a very early stage. Thus, everything might be subject of change.

Michael Müller

Brühl, Germany, June 2014

One year later…

Dear reader,

Sometimes people asked me about the completeness of this book. When I started, I planned to write a book with round about 200 to 250 pages. By means of that, the book has been more than 100% complete since the end of last year.

But, whilst writing this book, I extended the scope a little here, a little there, and there are still more ideas. Already in July 2014, during a phase of massive writing, I expected about 300 pages, and just now, I hope to finish between 350 and 400. My first plan was to finish the book within one year. Now, one year is gone, and a I just said, there is even more to write. Since I write this book in my spare time, and I have to perform other obligations (and sometime the unexpected happens), I stopped to schedule an end of writing. Maybe, it will be a kind of never ending story, getting updated as JSF and Java EE evolves.

The good news, I never increased the recommended price of this book. Writing this book takes a lot of time. I'll be thrilled if people honor this effort by buying this book early. Once you purchased your copy, you may download all updates at Leanpub without additional charge. Much better than buying a second or third editing to get updates of a printed book, isn't it? ;-)

Stay tuned and enjoy!

Michael Müller

Brühl, Germany, June 2015

Cover image

The screen-shot on the cover image shows the landing page of it-rezension.de[1]. There you'll get a more or less up-to-date list of the book-reviews I published.

I wrote this application to show some features of web development with JavaServer faces. Beside JSF, this application takes advantage of the object-relational mapping (JPA) and context and dependency injection (CDI). It is one of the applications you'll find in this book, to demonstrate the practical usage of the discussed technologies.

[1]http://it-rezension.de/

Acknowledgements

To my wife Claudia and my kids:

Thank you for your patience during night-writing and other long sessions.

I love you.

A special *thank you* to my contributors.

Preface

Developing web applications with Java and JavaServer Faces had been a great pleasure (and success) to me for a couple of years, when I first wanted to write a book about JSF in late 2010. At this time, I got in touch with some German publishers. My goal was to write apx. 200-250 pages about this subject, nothing more. “No thanks, too special” had been the common answer. And: “Great. Add some more pages, e.g, another thousand, and write about the whole Java Enterprise edition. Keep JSF smaller than 200 pages.” Frustrating answers.

Thus, I started to blog about JSF in early 2011. Depending on my spare time and loosely coupled. And I became a member of the JSF (JSR 344) Expert Group. Sadly, at a late time, when almost the most work had been done. Unlike most of the other volunteers, I’ not a JSF implementer, but an expert JSF user. I hope, next round, I’m able to contribute early and more.

My tutorial about Web Development with JSF is still the most popular part of my blog (blog.mueller-bruehl.de/[2]), and I never gave up my intention to write a book about this subject. Over time, I switched from blogging in German to write about development in English mostly. And similar to the cited suggestion of this German publisher, I do not write about JSF only. That’s why I changed the title to “with Java and JSF”. This book covers a lot of related Java (EE) technologies. And for those developers, who are not familiar with HTML, CSS and other stuff, I add some introductory chapters in the appendix for free.

Your feedback is welcome. Please send me an email to feedback@webdevelopment-java.info[3].

Enjoy!

[2]http://blog.mueller-bruehl.de/

[3]mailto:feedback@webdevelopment-java.info

Contributors

All content unless other mentioned is written by me. But a couple of people provided me feedback, helped on wording, or did a technical review on *TinyCalculator* and/or *Books*. Beside the people who are mentioned below, a special thank you to John Wright, who also provided a couple of comments. And a special thank you to all the other people who provided a mostly one time feedback. All of you helped to improve this book.

Pratap Chatterjee

Pratap Chatterjee is a Software Engineer, working with enterprise application software development for over 20 years. He has worked mainly in the Telecommunication Industry in England with BT and T-Mobile UK as programmer, designer, developer and team leader with Java and Web technologies.

Currently Pratap lives in Sweden with his wife and two sons and he is working for Karolinska Institutet which is one of the world's leading medical universities. Pratap enjoys programming and in his role as a programmer developer, he has written applications which help in the publication of doctoral courses and admission of students by the University.

Pratap has also contributed in the review of technical articles and recently reviewed Grails in Action, Second Edition by Glen Smith and Peter Ledbrook.

Constantin Marian Alin

Constantin Marian Alin is a passionate Java developer focused on developing web/desktop applications using the latest Java technologies. Beside daily work and learning, in the past few years, he has written and published articles for Developer.com and DZone communities. Currently, he's focused on developing RIA/SPA applications for the GIS field by integrating the power of Java frameworks, like JavaServer Faces, PrimeFaces, AngularJS, Bootstrap, RESTful, EJB, JPA etc, with the GIS specialized software, like ArcGIS, OpenLayers, GeoServer, Google Maps etc.

Anghel Leonard

Anghel Leonard is a senior Java developer with more than 13 years of experience in Java SE, Java EE, and related frameworks. He has written and published more than 50 articles about Java technologies and more than 500 tips and tricks for many websites that are dedicated to programming. In addition, he has written the following books:

- Tehnologii XML XML în Java, Albastra
- Jboss Tools 3 Developer's Guide, Packt Publishing
- JSF 2.0 Cookbook, Packt Publishing
- JSF 2.0 Cookbook: LITE, Packt Publishing
- Pro Java 7 NIO.2, Apress
- Pro Hibernate and MongoDB, Apress
- Mastering JavaServer Faces 2.2, Packt Publishing

Currently, Anghel is developing web applications using the latest Java technologies on the market (EJB 3.0, CDI, Spring, JSF, Struts, Hibernate, and so on). Over the past two years, he's focused on developing rich Internet applications for geographic information systems.

About the Author

Michael Müller is an IT professional with more than 30 years of experience including about 25 years in the healthcare sector. During this time, he has worked in different areas, especially project and product management, consulting, and software development. He gained international knowledge not only by targeting international markets, but also by leading external teams (from Eastern Europe and India).

Michael Müller

Currently, he is the head of software development at the German DRG institute inek.org[4]. In this role, he is responsible for Web applications as well as other Java and .NET projects. Web projects are preferably built with Java technologies such as JSF with the help of supporting languages like JavaScript.

Michael is a JSF professional user and a member of the JSR 344 and JSR 372 (JSF) expert groups.

He frequently reads books and writes reviews as well as technical papers, which are mostly published in German-printed magazines and on his website at it-rezension.de[5]. Besides that, he irregular blogs about software development at blog.mueller-bruehl.de[6].

[4]http://inek.org
[5]http://it-rezension.de
[6]http://blog.mueller-bruehl.de

About this book

Web development – this is first of all a very non-concrete concept which can enclose everything what has to act with the Internet, from an easy HTML page up to a complicated application with a specialized client wired over the net to a server. In this book this concept refers quite specifically to the development of applications which are usable via browser with the undermentioned technologies. Therefore it becomes clear that it concerns not static Internet pages, but full applications which are hosted on a server and are available at any place, free Internet access assumed. This might be for example, a member's management, an information system, a communication platform or other.

To realize such an application a whole bundle of apparently (or really) different technologies are necessary. First of all, from a user's perspective, a browser is needed. This usually is third party software. A browser displays web pages provided via network by a server. A browser mainly understands markups like HTML and CSS. It might be enhanced by a client side language like JavaScript.

On the server side, we need a programming language to create our application. As the book title suggests, we use Java here. Since we don't like to re-invent the wheel, we use an application server to host the applications and to provide a stack of Java Enterprise Edition (Java EE) features. JavaServer Faces is the first choice web framework discussed in this book. Beside that, we take a short look onto alternatives.

What is an application without data storage? Thus a persistence layer as well as a database system are required. A web application communicates with the user by means of forms and dialogs. The validation of inputs should be very fast so that the users can work without having to wait for lengthy reaction from the server. Therefore a part of the application, like input validation, is shifted to the client side.

In this book the web development takes place with these main technologies:

- HTML (HyperText Markup Language) with CSS (Cascading Style Sheets) for representation in the browser
- Java as server-side programming language
- JavaServer Faces as web framework
- Java EE 7 compliant application server

- Java Persistence with SQL database
- JavaScript for client side programming
- AJAX as technology to update a page partly
- Web services for the solution of special problems
- Test frameworks and tools

In part I, we start with *TinyCalculator*. This is a very small application just to show some concepts of JSF. Following this "appetizer", some basic concepts, like view definition language (VDL) or the JSF lifecycle are introduced.

Part II focus on *Books*. This is the application shown on the cover image. It's used to show a list of reviews at it-rezension.de[7]. This application uses templating, persistence (JPA) and context and dependency injection (CDI).

The biggest application *Alumni* is subject of part III. Alumni allows to organize former classes and upcoming events. We discuss authentication and authorisation as well as other security aspects. Re-usable components, testing and much more is explained.

Part IV covers additional techniques like web services. And we take a look onto Java based alternatives to JSF. One of them, AngularJS is a JavaScript framework. But we'll use Java at the back-end.

The appendix offers you some extra chapters, like introduction into the basic concepts of HTML, CSS, JavaScript and more.

Sources of the applications are (partially) available for download at webdevelopment-java.info[8].

This book combines theoretical foundation with real applications. This denotes, beside explanation, you receive information from the practice. With so much screen-shots and source-code that you can understand the whole purely mental. It is not necessary that you sit down immediately in front of your computer and strike into the keys. Who would like, may do this naturally also. A detailed explanation follows the screenshots and listings in each case. This book is more concrete, without omitting however the theoretical foundation.

Albeit this book starts slightly a small example, it primarily addresses persons who already have programming knowledge in Java. Indeed, HTML and CSS belong in the repertoire of every web developer, but just with the entrance in this area some Java (mainly Java

[7]http://it-rezension.de
[8]http://webdevelopment-java.info

SE) developer may not have suitable knowledge. Hence, you find some accompanying information in the appendix as well as links on deepening information.

The main focus of this book is on the development of applications, not on the installation of the environment. Hence, the practical part also does not start with vast installation instructions.

Best of all you can understand of this book by means of NetBeans. You find this development environment under netbeans.org/downloads/index.html[9]. Select there either the package "Java EE" or "All" and install this according to instructions. Prerequisite is an installed Java JDK. If you do not have installed this yet, then you'll find a suitable link in the download area of NetBeans.

Conventions

Information box

This symbol is used to indicate important information.

This is a Warning

Common pitfalls or other mentionable warnings will be marked by this symbol.

Exercise

This symbol indicates a short exercise, you might perform to get some more practice.

Links

During this book you'll find a lot of useful links, pointing to further or additional information. All these links have been checked by the author at the time of writing. Since these links point

[9]http://netbeans.org/downloads/index.html

to external resources that might be changed at any time by the appropriate content owner, a link might not be valid any more or does not point to the expected information. If you discover such an invalid link, please contact the author of this book to enable a correction.

Errors and typos

All information for this book have been faithful researched or developed. However, there can’t be any guarantee of being be error-free. If you discover errors or typos, please inform the author.

Copyright

Part I: *TinyCalculator*

1. TinyCalculator

TinyCalculator is the first and simplest application discussed in this book. It's nothing more than a simple calculator for basic arithmetics. But, is is useful to show some fundamental concepts of web applications with Java and JSF.

As a kind of appetizer, and unlike the following applications, I first show you the whole application, with a little explanation only afterwards. Then, as a kind of re-start, I talk about the foundations of web applications and explaining TinyCalculator and different approaches.

In this chapter, we'll discuss

- Creating a web application from scratch
- A brief introduction into the used protocols and languages
- A brief introduction into the underlying Servlet technology
- Mapping between UI components and their server representation
- Component tree
- Bean scopes & JSF Lifecycles
- Simple UI test with selenium

1.1 Creating the application

[This section isn't available in the book sample]

1.2 TinyCalculator briefly explained

This chapter briefly explains some aspects of the TinyCalculator application.

1.2.1 Managed Bean

I assume, for you as a Java developer, the code of the managed bean had been the most familiar part of the application. In JSF, developers often talk about *Managed Beans*, and that is how NetBeans called this in its *New File* dialog. However, this term is not really accurate.

Java Bean

A Java Bean is defined as a reusable software component. Its nothing but a pure Java class with no-argument constructor and properties following a special convention. A property is a private attribute (field) which is accessed via a pair of getter and setter. These methods follow the naming convention of set*Name* and get*Name*.

Attribute, Field, Property

From the abstract perspective of object oriented programming, the state of an object is held by attributes. An attribute is a variable, which can have any modifier like private, protected, public, static and so on. The term attribute is widely used and can imply nearly everything. Although it is the correct term to describe the state of an object, in Java often other terms are used (which might called different in other Languages). Throughout this book, I will mostly use these:

A *field* is a private (or sometimes protected) instance variable.

> A *property* is a field which is exposed by a pair of getter and setter.

JavaServer Faces is designed to run both in EJB (Enterprise Java Beans) containers as well as in servlet containers. Both are called *Application Servers*. Whilst the first one contains a full or partial[1] stack of Java EE technologies (including servlet), the latter "only" serves Servlets. Thus, other Jave EE technologies like *Context and Dependency Injection* (CDI) are not available in Servlet containers.

Some widely known examples of EJB containers are GlassFish or WildFly. Examples of servlet containers are Tomcat or Jetty.

If JSF runs on a pure Servlet container, so called *Backing Beans* are managed by JSF. Such a bean will be annotated as *@ManagedBean* and that's what the term *Managed Bean* originates. Starting with Java EE 6 / JSF 2.0, developers could use CDI *Named Beans* too, which is today's recommended technology. In TinyCalculator, we use a CDI Named Bean.

A named or managed bean might be accessed by its name.

The second annotation *@RequestScoped* declares the bean's lifetime: With every request (from your browser), an instance of this class is created and destroyed by the termination of this request. A longer lifetime might be declared by *@SessionScoped* (providing one instance of the bean per user session), *@ApplicationScoped* (one instance during the application's lifetime), and more. This will be discussed further on.

In the example, we used a CDI named bean.

CDI annotation for a request scoped bean

```
import javax.inject.Named;
import javax.enterprise.context.RequestScoped;

@Named
@RequestScoped
public class TinyCalculator {...}
```

A JSF managed bean, on the other hand, would be declared like this:

[1]In fact, beside the *Full Platform*, only one profile, the *Web Profile* is defined.

JSF annotation for a request scoped bean (do not use)

```java
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;

@ManagedBean
@RequestScoped
public class TinyCalculator {...}
```

Mixing annotations

Always use the annotations for named/managed beans in conjunction with the appropriate scope annotations. E.g. both JSF annotations or both CDI annotations. In particular, a named bean with JSF scope would result in a runtime error.

Although you can't mix theses annotations within a single bean, it is possible to use both types (named and managed) beans within one application. This might be useful for migrating older applications, which used JSF managed beans: It is possible to add new features by implementing them as CDI named beans whilst keeping the existing beans. Then, over time, the existing beans might be changed to named beans too.

But why should you prefer CDI named beans over JSF managed beans? As the name suggests, JSF managed beans are developed especially for and dedicated only to JSF.

CDI (Context and Dependency Injection) is relatively new to Java EE and has made into EE 6. It allows the container to “inject” appropriate objects into your object. These objects might not be known at compile time and the dependencies will be resolved at runtime. This allows loosely coupling of objects. One essential part of this solution is a general naming concept.

Since CDI named beans might be used in different Java EE technologies, JSF itself slowly migrates to CDI, which sometimes replaces the proprietary solution. JSF 2.3 at last simply allows to inject JSF related objects into places where it had been very tricky before to access these values. The migration to CDI is mostly complete for the latest version.

Deprecate managed beans

As of JSF 2.3 (Java EE 8), JSF managed beans (@ManagedBean) became deprecated.

But, is CDI available in pure Servlet containers like Apache Tomcat? The bad news is simply no. But JSF isn't available too, until deploy the JSF library to enable it. And, the same way, you can use CDI together with a servlet container. Just provide a CDI implementation, for example Weld[2]. I case of Tomcat, there is a simpler solution: Use TomEE[3], which is Tomcat bundled with Java EE technologies, implementing the Java EE Web Profile.

Bean Passivation

If the bean's lifetime is expanded to more than one request, the server still has to manage this object, even thought the next request may take a while, or will never occur. The latter is mitigated by a session time out. But till then, the bean is alive. Where there is a lot of traffic, this memory consumption may cause problems.

To avoid memory problems, or for other reasons, depending on the implementation, the container might passivate a bean: The object is persisted somewhere, e.g. to disk, and, during the next request if needed, it is restored into memory (activated).

To enable this feature, the bean must implement the *Serializable* interface.

[2]http://repo1.maven.org/maven2/org/jboss/weld/servlet/weld-servlet/
[3]http://tomee.apache.org/index.html

Examine scopes

Scopes will be discussed later in this book. To get an initial feeling you may perform a little task:

Add a logger and parameterless constructor to the TinyCalculator class

```
private static final Logger LOGGER = Logger.getLogger("TinyCalculator");
public TinyCalculator() {
    LOGGER.log(Level.INFO, "ctor TinyCalculator");
}
```

Launch the application and watch the NetBeans console window. Perform some calculations. Close and re-open the browser and application. Perform some other calculations. Next exchange the `@RequestScoped` annotation by `@SessionScoped`, later by `@ApplicationScoped` and perform the same operations. Observe the different output.[4]

The calculation methods might look quite strange. These are not functions, returning the calculated value, but methods returning a *String* and perform the calculation by changing the status of the result variable. The simple reason, the calculation is called within the action of the appropriate button. Such an action performs a page navigation. Return an empty *String* or *null* keeps the browser "staying" on the current page. In fact, this page will be reloaded. We cover page navigation in detail later on. Using JSF 2.2 or later, such a function might be declared as void, alternatevely.

1.2.2 The page

I don't want to anticipate the detailed explanation during the next chapters. Just to point out, the page will be rendered as HTML and is sent to the browser. If you're familiar with HTML, you would have recognized only some HTML elements. JSF's former intention was to provide the programmer a kind of known interface, hiding away that HTML stuff. Because of that, JSF offers a set of its own *tags*, which are included into the page and will be replaced before sending the page to the browser. This tags are regular XML tags, assigned to appropriate namespaces.

[4]A reader asked me what `ctor` stands for. It's a widely used abbreviation for constructor. Lesson learned for me: I never shall assume that widely means known by everybody.

```
<h:outputLabel value="Param1: "/>
<h:inputText value="#{tinyCalculator.param1}"/>
```

This is a label with a value (text) of *Param1:* followed by a text element. The value of the text element is retrieved from and sent back to our bean.

#{...} denotes the *Expression Language* (EL), which is used to glue parts of the user interface to the managed bean. *tinyCalculator* refers to our bean. By default, the EL uses the bean name with a lower-case first letter. And then, with dot-notation, referring methods. In case of properties, this establishes a two way communication with the getter/setter pair. *name* refers to get*Name* and set*Name* (omitting the get and set prefixes). Thus, the text element reads and writes the property.

```
<h:commandButton value="Add" action="#{tinyCalculator.add}"/>
```

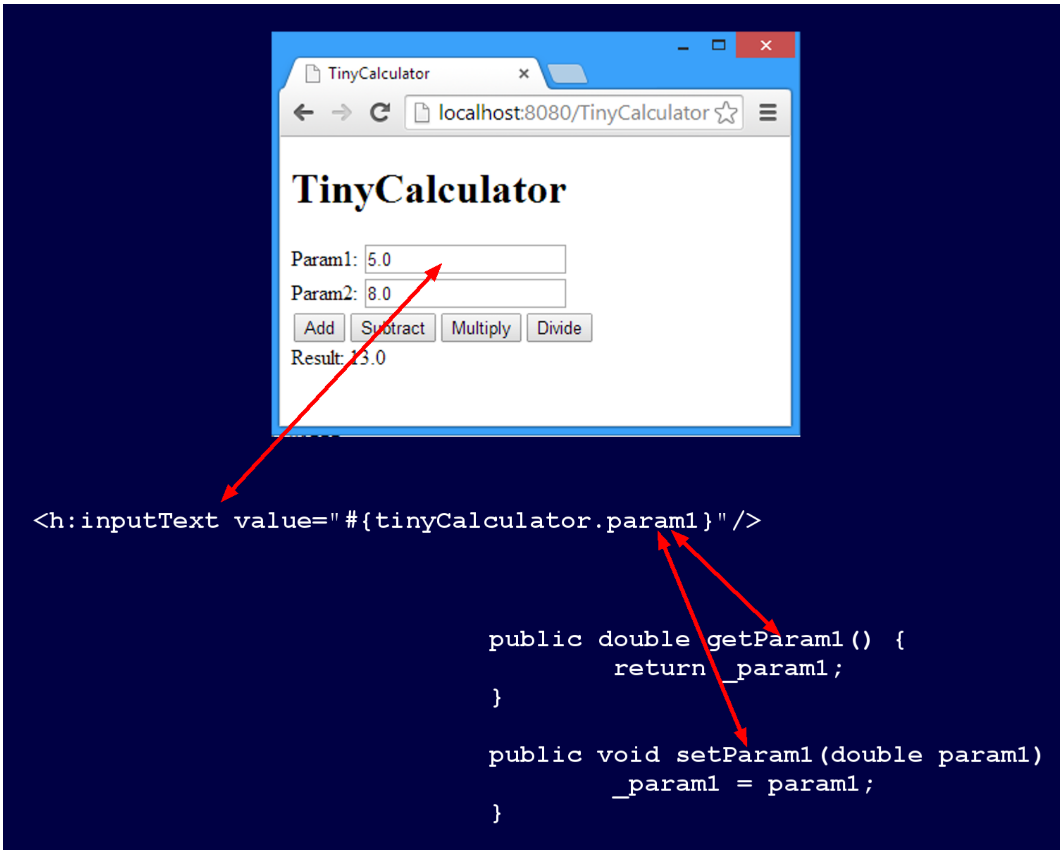
In the case of the buttons, each action defines one of the calculating methods.

HTML friendly markup

Starting with JSF 2.2, an alternative approach was introduced, reducing the JSF specific tags and using more pure HTML. This is known as *HTML friendly markup* or more often *HTML5 friendly markup*, even though it is not HTML5 specific.

1.2.3 Relationship

The rough relationship between browser view, page definition and managed bean should be understandable from the explanation. But, a picture is worth a thousand words.



```
<h:inputText value="#{tinyCalculator.param1}"/>

public double getParam1() {
        return _param1;
}

public void setParam1(double param1)
        _param1 = param1;
}
```

Rough relationship between code and view

The tag `<inputText ...>` represents a text field in the browser. Its values are bound to a getter/setter pair of the bean.

Component tree

The simplified presentation shown before is useful for an initial understanding of JSF.

But it ignores one important fact: The component tree. Whilst processing a request, all JSF components are held within a tree data structure. Depending on enhanced feature of your application, it might be useful to access or manipulate this component tree directly. The component tree will be discussed later on.

1.3 Summary

In this chapter we created a first small JSF application. For getting a quick result, this application was introduced as a step by step tutorial, followed by a brief description.

There is much more to explain, like the configuration NetBeans built for us under the hood and which we modified by simply changing the project's properties. But this chapter is just an appetizer, a quick start. Beginning with the next chapter, we'll discuss the foundations.

2. Foundations

For web development, a couple of different technologies are used. This chapter discusses the basics of some, and mentions other important technologies.

- Definition of web application
- Hypertext Transfer Protocol (HTTP)
- Hypertext Markup Language (HTML)
- Cascading Style Sheets (CSS)
- JavaScript
- Java
- Maven
- Selenium & Arquillian
- Servlets

2.1 Web Application

First of all, the term *web application*, as it is used throughout this book needs to be defined.

> **A web application is a client-server application interacting dynamically with the user via web browser.**

Thus, we are talking about distinct parts, the client, presenting the user interface, and the server, executing the main part of the application. The application logic might be organized in layers and split between client and server. Especially to get a responsive user experience, some bits are executed on client side. Unlike an application using a specially programmed client, a common web browser is used to display the presentation layer. Thus, we cannot manage every detail. Instead, we have to rely on the browser functionality, which mainly is to retrieve and display content like HTML pages and related stuff and executing script code.

A simple web server, only delivering static pages, is not an web application in the sense of this book.

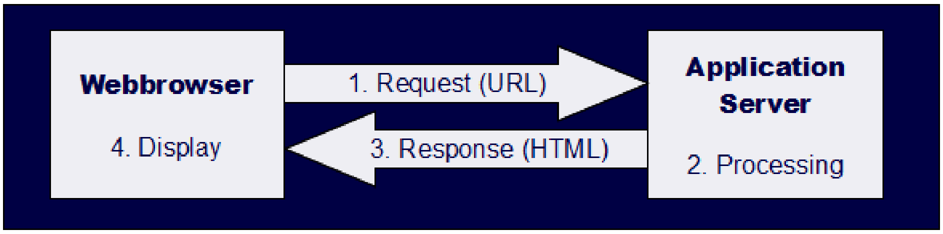
By using a standard web browser, a web application might be used on any platform for which a (modern) browser is available. A web browser is connected to a web server via HTTP. Because of that, a web application has to deal with some restrictions of this protocol.

The client is linked to the server via the internet.

2.2 HTTP

The Hypertext Transfer Protocol is a logical transport layer (*application protocol*), acting on a stack of other layers, down to the physical transport layer. Usually it is used atop TCP/IP. As the name suggests, it was primarily designed, to distribute hypertext information. As such, it is the foundation of the world wide web.

HTTP is a stateless request-response protocol. A server is listening for a request and sends back a response. Then the communication terminates.



HTTP request/response cycle

The simplified picture shows a typical request-response-cycle as triggered by the user when she wants to retrieve a page. In fact, the response is a HTTP response, composed of a message header and body, which might carry information other than HTML, e.g. JSON, XML or something else.

Any subsequent request acts independently of the prior ones. This kind of protocol affects the way a web application works with its clients. To proceed with a user session of the application, session management has to be implemented on top of HTTP. The server might save the application's state and provide some information to the client. The client sends back this information during the next request. Using to this, the server is able to restore the current state. This piece of information might be anything useful.

One extreme is to send all attributes (state) of the application to the client. In this case, there is nothing to remember on server side. Any status might be restored by the information the client provides. Such an approach keeps memory consumption on the server lean. Sending lots of information over the web has some important drawbacks. Latency will slow down the application. Another aspect is security. If the network transport isn't secured by a protocol like TLS[1], then this information might be read by unauthorized persons. Saving the whole state to the client avoids "sticky" sessions: The following request might be processed by a different server behind a load balancer.

The other extreme is to keep any information on the server and send an identifier only to the client. Such an identifier is called *session id*. Common possible ways to convey a session id are the usage of a hidden field or sending a HTTP cookie[2]. If the server holds the state, the session needs to be continued by that server unless you implement a solution to share this state information between different servers. In practice, a modern server might serve a few ten or hundred thousands sessions per day. Thus, in many scenarios such "sticky" sessions won't matter. * * *

Session hijacking

Due to the nature of the HTTP, the server needs information like a session id to determine how to proceed. If an unauthorized person (e.g. a cracker[a]) captures this information, she's able to take over the session. This is called *session hijacking*. JSF offers some additional features to secure your application.

[a]During this book, I do not use the more common word *hacker*. A hacker might act with decent intent. To indicate they act in bad face, I prefer the term *cracker* which is short for *criminal hacker*

Luckily session management is implemented by the Java environment. Within Servlet/JSF configuration, it is possible to choose between client or server state.

During a request, HTTP addresses an uniform resource identifier *URI*[3]. Additionally, it tells the server, which method to use. This is often called a *verb*. Most interesting methods for use with JSF are *GET* and *POST*.

GET queries information from the server by use of an URI. Additional parameters might be appended to the URI.

[1]http://en.wikipedia.org/wiki/Transport_Layer_Security
[2]http://en.wikipedia.org/wiki/HTTP_cookie
[3]http://en.wikipedia.org/wiki/Uniform_Resource_Identifier

Example: http://it-rezension.de/Books/books.xhtml?catId=2

This addresses the server found at it-rezension.de, and within that, the application and page Books/books.xhtml. A parameter *catId=2* is appended.

With a POST request, information is send to the server within the body of the request.

Example: A web form (HTML) contains some input fields and a submit button. Hitting this button initiates a POST request.

Within the context of Representational State Transfer (REST[4]) the methods (verbs) *PUT* and *DELETE* are important too. The idea behind REST is to assign defined actions to the verbs.

2.3 HTML

The Hypertext Markup Language is used to describe the content of a web page. Like XML, HTML is derived from the *Standard Generalized Markup Language* SGML[5].

As a Java developer, I assume, you would be familiar with XML. Like XML, HTML uses tags to structure the content of a page. Unlike XML, these tags are predefined, HTML is not extensible in the sense of XML.

Tag guessing

In one way, HTML is less restrictive than XML: Often missing closing tags will not be alerted as an error, but automatically closed. Browsers try to handle overlapping tags and more.

Some people praise this behavior. It allows writing less markup. Less work, less data to be transferred, faster[6] page loading

But, different browsers may handle missing or interlaced tags in a different way. And sometimes, guessing how to complete the missing parts, results in dangerous code. Read *The Tangled Web* [MZ2011] to inform yourself. To avoid this bunch of problems, I advise you to use XHTML, a variant of HTML re-defined on XML. This enforces well formed documents and avoids a couple of problems.

[4]http://en.wikipedia.org/wiki/Representational_state_transfer

[5]http://en.wikipedia.org/wiki/Standard_Generalized_Markup_Language

[6]Re-think this: Omitting some few characters of a page, will this really speed up page loading?

A HTML document starts with a document type, followed by exactly one HTML tag. Within that, a tag for header and body might be included. In case of XHTML, the doctype is preceded by the XML version.

Sample HTML page with form

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html>
<html>
    <head>
        <title>TinyCalculator</title>
    </head>
    <body>
        <h1>TinyCalculator</h1>
        <form>
            <div>
                <label>Param1: </label>
                <input type="text" value="0.0" />
            </div>
        </form>
    </body>
</html>
```

Further information about HTML, including a commented list of important tags is provided in the appendix.

2.4 CSS

HTML provides some little tags for styling the page, like emphasized, italics and more. The Cascading Style Sheets have been developed as a much more powerful designing tool. As a rule of thumb, you should separate layout from content. Thus, don’t use the HTML styling tags. Use CSS exclusively for the layout of a page.

By using CSS, it is possible to address an element and to assign layout information within curly braces.

Sample CSS statements

```css
h1 {
    font-size: 2em;
}

h2 {
    font-size: 1.5em;
    font-style: italic;
}
```

The small code above addresses header 1 (h1), which is set to a font size of 2em, and header 2, which is set to a smaller font and italics. Within the curly braces, a couple of layout statements might be placed.

To address an element, its tag name is used. To layout different elements with the same tag name in different styles, a couple of tag names might be used, building a path. Or a class or an id is assigned to the element and might be used for addressing. These addressing elements might be combined to define complex paths. Certain rules exists to avoid ambiguity. The layout information might differ depending on the output device or screen size. Using these features, it is for example possible, to define a *responsive webdesign*: The layout is changed and adjusted by the size of the browser window.

In Part II, Books application, we'll discuss CSS in a web application. You may open [Books]}(it-rezension.de/Books/) with different devices or simulate this by changing the size of your browser window.

As with HTML, I don't want to bore those readers, who are familiar with CSS. Readers needing an introduction to CSS can the foundations in the appendix.

2.5 JavaScript

JavaScript is the programming language of the client side. Almost every modern browser has implemented a JavaScript interpreter or a just in time compiler. Even though Microsoft tried to introduce VB Script some years ago, JavaScript is the language of choice, and it is well understood by the Microsoft browser too.

ECMAScript

JavaScript was created by Netscape and became standardized by the European Computer Manufacturer's Association. Thus, the official name is ECMAScript. JavaScript is the name of Netscape's implementation of ECMAScript, whilst Microsoft's implementation officially is called *JScript*. JavaScript is commonly used as synonym for ECMAScript, which I will use in this book.

JavaScript and Java

JavaScript is neither Java nor derived from Java! Previously it was called *LiveScript*, but became renamed soon. However, it is a full-fledged programming language. It is used for server programming, too (Node.js[7]). Java 8 comes together with a JavaScript interpreter written in Java, called *Nashorn*. It is capable of accessing Java classes, and this enables running and testing portions of Java code without the need to compile. This is especially useful when discovering new libraries or testing some new Java constructs.

And even better, Java's next version 9 includes the Java Shell (JShell) which realizes a real read-eval-print loop (REPL).

JavaScript is used to enhance client behavior or to initiate partial requests (AJAX). JSF hides JavaScript behind its AJAX tag. Sometimes it is useful to develop a bit of JavaScript code. As a Java developer, it should be no problem for you to follow the simple examples described in this book.

JavaScipt example: Showing a message

```
alert("The information has been saved");
```

Unlike Java, JavaScript is not a typed language. Thus you may assign an integer to a variable and later on replace its value by a string.

[7]http://nodejs.org/

2.6 Java

Java is the main technical foundation for programming web applications with this book. I assume my readers to be familiar with Java SE.

All web applications discussed in this book are built up using Java EE. The Java EE platform is build as a JSR (Java Specification Request, Java EE 7: JSR 342, Java EE 8: JSR 366). It is an umbrella specification describing a complete architecture built up from a mass of technologies, each defined by its own JSR. Throughout this book, most of those technologies will be introduced.

2.7 Maven

For professional Java development, a build tool is needed. There are lots of developers, who never care about their build tool, because it is configured by their favourite IDE. Other developers perfectly know their build tool and like to tune every detail.

Among others, there are two popular tools in the Java world: Apache ANT[8] which acts more imperative and Apache Maven[9] which follows a more declarative approach. On their web site, they call Maven a "*software project management and comprehension tool*". Popular IDEs, like NetBeans, have built-in support for both tools. Nevertheless, whichever tool you prefer, ANT based projects often use an IDE specific configuration, whilst Maven projects follow a more strict convention. Thus, they become mostly independent of the IDE.

To ensure most compatibility with your favourite IDE, all applications discussed in this book are built with Maven.

2.8 Selenium & Arquillian

Selenium[10] automates browsers. Beside macro-recording and replay, such an automation may be fully controlled from within a Java application, for example by a test. Doing so, enables GUI-testing of web applications.

[8]http://ant.apache.org/
[9]http://maven.apache.org/
[10]http://docs.seleniumhq.org/

Testing beans or other components, which are managed by a container can be a hassle. Arquillian[11] allows testing the interesting parts of a web application within a container. It fully integrates with test frameworks like JUnit.

Although this book does not focus on unit testing or test driven development, we'll discuss some simple test scenarios with both tools.

2.9 Servlet

A Servlet is a Java class hosted in a servlet container that dynamically processes requests and constructs reponses. This class must conform to the Java Servlet API. Like other Java EE components, it is specified by the *Java Community Process* (JCP). The servlet version included in Java EE 7 is JSR 340: Java Servlet 3.1 Specification[12] respectively JSR 369: Java Servlet 4.0[13] in case of Java EE 8. JSR is short for *Java Specification Request*.

Although a servlet may respond to any request, we're talking about responding to a HTTP request. Thus, we use servlet as synonym for HTTP servlet.

The servlet's life-cycle is maintained by the container. The web client, e. g. browser, interacts with the servlet by request/response as described in the section HTTP.

A servlet is a class extending `javax.servlet.http.HttpServlet`. In most common scenarios, at least two methods *doGet* and *doPost* will be overwritten to implement specific behaviour and send back the response.

A servlet is invoked by a client's request to a specific path (of the URI). By using the simple annotation `@WebServlet("/path")` this might be defined. Whilst discussing JSF configuration, we'll talk about configuring servlets by a configuration file. JSF itself is implemented as servlet (*FacesServlet*).

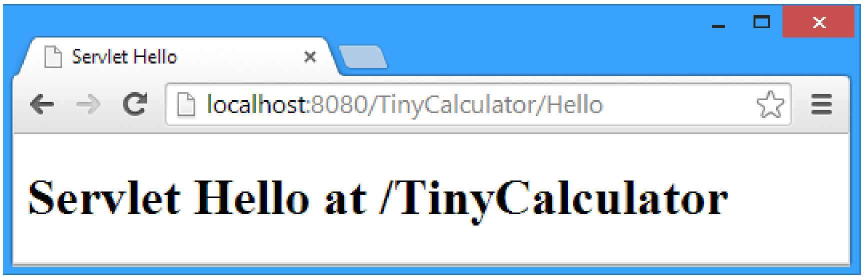
If you enhance the TinyCalculator by NetBeans' 8 *Add Servlet* wizard, and provide *Hello* as name, NetBeans generates the following code example.

[11] http://arquillian.org/
[12] https://jcp.org/en/jsr/detail?id=340
[13] https://jcp.org/en/jsr/detail?id=369

Programmatically generate a HTML page by a servlet

```java
[imports omitted]

@WebServlet(name = "Hello", urlPatterns = {"/Hello"})
public class Hello extends HttpServlet {

    /**
     * Processes requests for both HTTP <code>GET</code>
     * and <code>POST</code> methods.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            /* TODO output your page here.
               You may use following sample code. */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet Hello</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet Hello at "
                + request.getContextPath() + "</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }

    /**
     * Handles the HTTP <code>GET</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Handles the HTTP <code>POST</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    [...code omitted ...]
}
```

If you start this application and complete the URI by `/Hello`, you can verify the servlet responding to your request.



Servlet Hello at /TinyCalculator

Hello servlet in action

Within the methods `doGet` and `doPost`, we implement our code, or as in this generated skeleton, we simply call a common method if Get and Post can be handled in the same fashion. The HTML code is generated within the `processRequest` method.

Writing HTML code directly inside Java code is not a good approach for anything beyond a trivial example. One approach is to separate the HTML page from the Java code using *Java Server Pages* (JSP). The page is stored on its own with embedded JSP tags and is compiled into a servlet at a later time. Even though still active, JSP might be considered as ancestor of JSF.

2.10 Deployment

All servlets, and this includes JSF, run within a servlet container. When we launched *TinyCalculator*, NetBeans automatically started GlassFish (if not running) and deployed the application to the application server.

If we want to install an application to a production server, we may add this server to the NetBeans environment (or to your favourite IDE) and let the IDE perform the deployment to the production system. If, for some reason, it is not possible to use the IDE for deployment, you might deploy the application by yourself. This process depends on the application server you use.

In the case of GlassFish, you can either deploy at command line, using the *asadmin* command, or you can use your browser to log into the administrator console. GlassFish offers an *application* menu. This is intended to deploy, un-deploy, start an application and more. And last but not least, you may use maven to deploy the application.

2.11 Summary

This chapter introduced into the technical foundation of web applications as developed during this book: HTTP, HTML, CSS, JavaScript, Java, Maven, Selenium and Arquillian, Servlet and Deployment. Selected technologies have been briefly explained, whilst other have just been mentionend. It did not mention Java EE and especial JSF, which are not part of the foundation, but the main subject of this book.

For readers, who are not familiar with the browser afine parts of this technology stack, this book offers additional information about HTML, CSS and more within the appendix .

3. JavaServer Faces

[This chapter as well as all following isn't available in the book sample]

Sadly, it is not possible yet, to integrate a full TOC automatically into the sample book. Leanpub is working on such a feature.

A TOC of existing as well as planned content is available at the landing page of this book[1]

[1]https://leanpub.com/jsf