

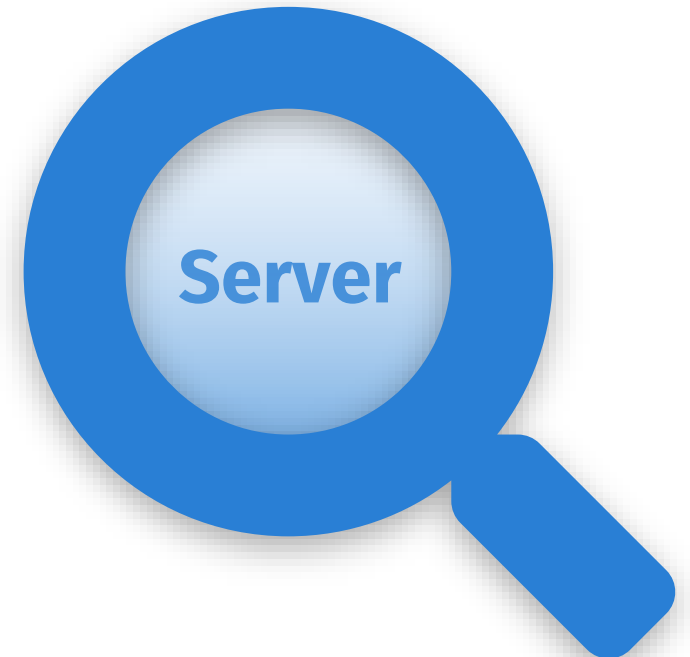
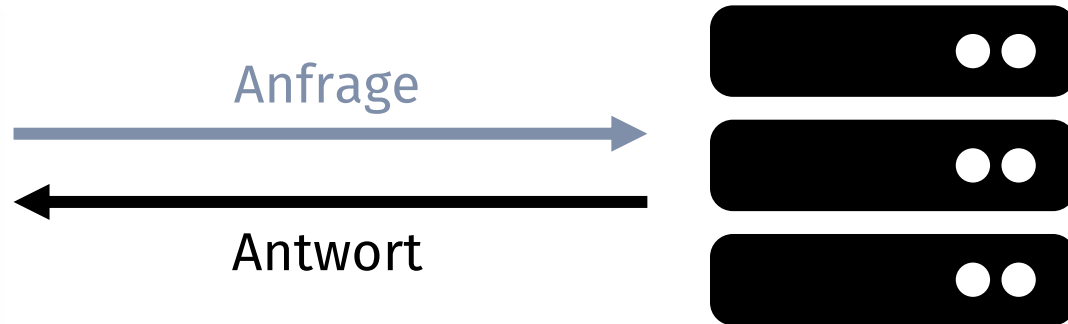
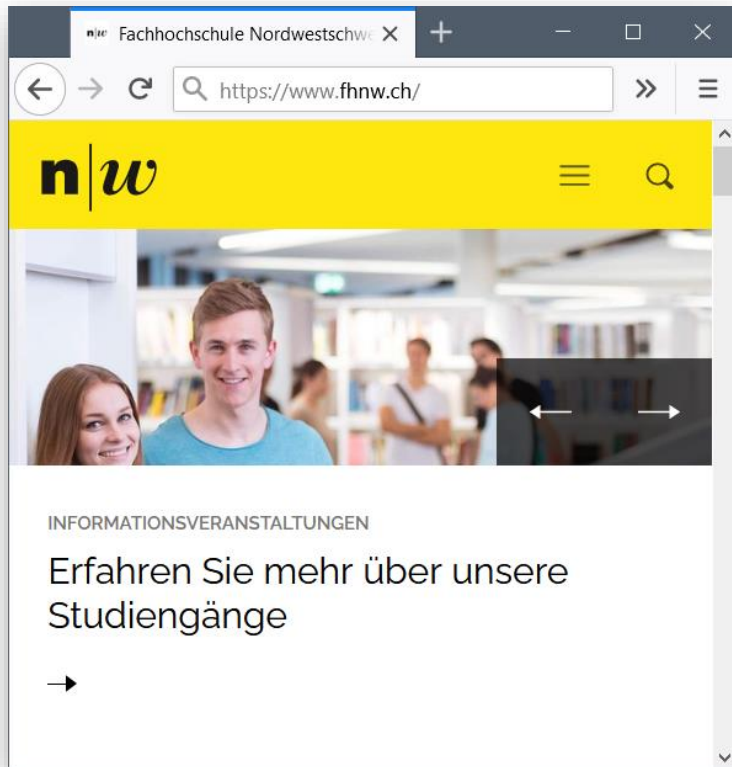
## **Web Engineering**

# **MVC mit Spring Boot**

Adrian Herzog

*(basierend auf der Arbeit von Michael Faes, Michael Heinrichs & Prof. Dierk König)*

# Das World Wide Web



Client: **HTML & CSS** ✓

# HTTP(S)

# HTTP

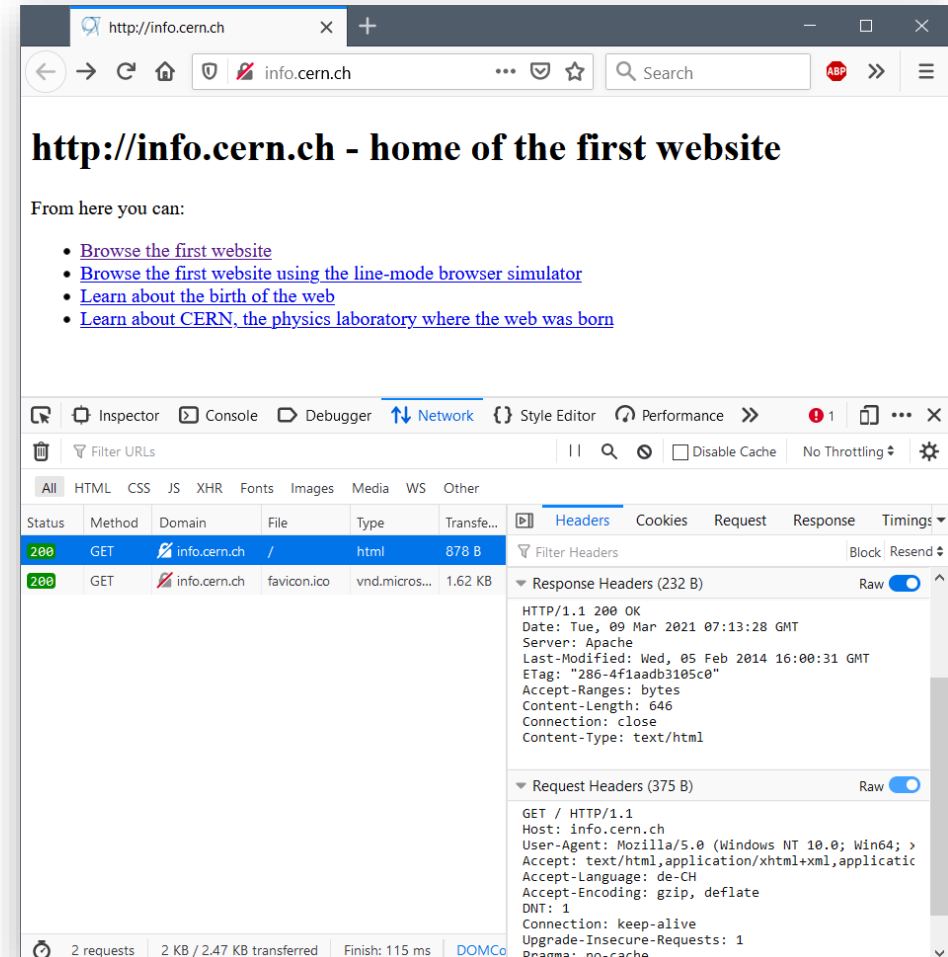
Wie sprechen Client und Server eigentlich miteinander? Auf HTTP!

```
michael@Michael-X1-Extreme: /mnt/c/Windows/system32$ telnet info.cern.ch 80
Trying 188.184.21.108...
Connected to webafs706.cern.ch.
Escape character is '^]'.
GET / HTTP/1.1
Host: info.cern.ch

HTTP/1.1 200 OK
Date: Tue, 09 Mar 2021 07:09:38 GMT
Server: Apache
Last-Modified: Wed, 05 Feb 2014 16:00:31 GMT
ETag: "286-4f1aadb3105c0"
Accept-Ranges: bytes
Content-Length: 646
Connection: close
Content-Type: text/html

<html><head></head><body><header>
<title>http://info.cern.ch</title>
</header>

<h1>http://info.cern.ch - home of the first website</h1>
<p>From here you can:</p>
<ul>
<li><a href="http://info.cern.ch/hypertext/WWW/TheProject.html">Browse the fi
rst website</a></li>
<li><a href="http://line-mode.cern.ch/www/hypertext/WWW/TheProject.html">Brow
se the first website using the line-mode browser simulator</a></li>
<li><a href="http://home.web.cern.ch/topics/birth-web">Learn about the birth
of the web</a></li>
<li><a href="http://home.web.cern.ch/about">Learn about CERN, the physics lab
oratory where the web was born</a></li>
</ul>
</body></html>
Connection closed by foreign host.
michael@Michael-X1-Extreme: /mnt/c/Windows/system32$
```



# HTTP Methoden

automatisch  
vom Browser  
verwendet



Methode	Beschreibung
<b>GET</b>	Holt eine «Ressource» vom Server, durch URL identifiziert. Server schickt Ressource in Body zurück.
<b>HEAD</b>	Wie GET, aber Server schickt nur HTTP Header ohne Body zurück. Wird verwendet, um zu prüfen, ob Inhalt geändert wurde.
<b>POST</b>	Schickt Daten an den Server zur Verarbeitung, z. B. durch das Ausfüllen eines Web-Formulars. Kann neue Ressourcen auf Server erstellen oder vorhandene modifizieren.
<b>PUT</b>	Ersetzt vorhandene Ressource unter der angegebenen URL (kann im Vergleich zu POST keine neue Ressource anlegen)
<b>PATCH</b>	Ändert die Ressource unter einer URL (partiell Update)
<b>DELETE</b>	Löscht die Ressource unter einer URL.

# Argumente in HTTP

## Zwei Möglichkeiten

### 1. In URL:

```
GET /wiki/Spezial:Search?search=Katzen&go=Artikel HTTP/1.1
```

Schlüssel	Wert
search	Katzen
go	Artikel

### 2. In Body:

```
POST /wiki/Spezial:Search HTTP/1.1
Host: de.wikipedia.org
Content-Type: application/x-www-form-urlencoded
Content-Length: 24
```

```
search=Katzen&go=Artikel
```

beliebiger  
Inhalt möglich

nur bei **POST**,  
**PUT**, **PATCH**

# HTTP-Statuscodes

Server antwortet immer mit **Statuscode**:

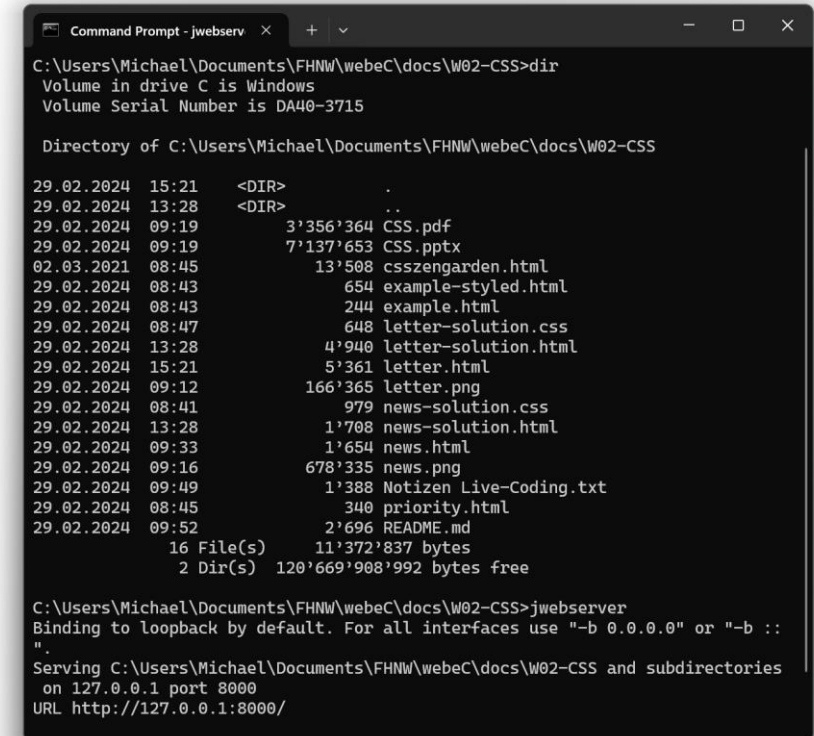
```
HTTP/1.1 200 OK
Server: Apache
Content-Length: 646
Content-Type: text/html
```

```
<!DOCTYPE HTML> ...
```

Codes	Kategorie	Beschreibung
1xx	Information	Anfrage dauert noch an...
2xx	Erfolgreich	Anfrage wurde bearbeitet, Resultat kommt zurück
3xx	Umleitung	Weitere Schritte nötig, z. B. weil Ressource verschoben wurde. Enthält <b>Location</b> -Header.
4xx	Client-Fehler	z. B. nicht-existierende Ressource ( <b>404</b> )
5xx	Server-Fehler	Verarbeitungsfehler auf dem Server

# Übung 1: Einfacher Web-Server mit Java

1. Öffne ein Terminal und wechsle in den Ordner der CSS-Übung von letzter Woche.
2. Führe den Befehl `jwebserver` aus, um einen Web-Server zu starten, der den Inhalt des aktuellen Ordners über HTTP zur Verfügung stellt.
3. Öffne im Browser die URL <http://localhost:8000> und dann die HTML Datei. Schau dir die HTTP Header der Requests in den Dev Tools an.

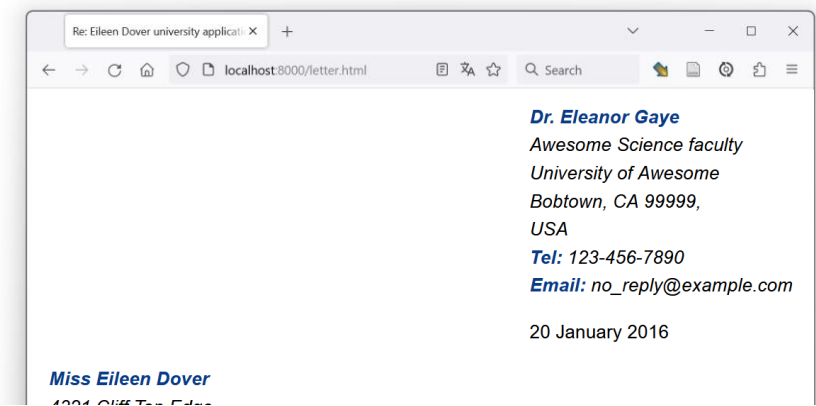


```
Command Prompt - jwebserver
C:\Users\Michael\Documents\FHNW\webeC\docs\W02-CSS>dir
Volume in drive C is Windows
Volume Serial Number is DA40-3715

Directory of C:\Users\Michael\Documents\FHNW\webeC\docs\W02-CSS

29.02.2024  15:21    <DIR>          .
29.02.2024  13:28    <DIR>          ..
29.02.2024  09:19             3'356'364  CSS.pdf
29.02.2024  09:19             7'137'653  CSS.pptx
02.03.2021  08:45             13'508    csszengarden.html
29.02.2024  08:43              654    example-styled.html
29.02.2024  08:43              244    example.html
29.02.2024  08:47              648    letter-solution.css
29.02.2024  13:28             4'940    letter-solution.html
29.02.2024  15:21             5'361    letter.html
29.02.2024  09:12            166'365    letter.png
29.02.2024  08:41              979    news-solution.css
29.02.2024  13:28             1'708    news-solution.html
29.02.2024  09:33             1'654    news.html
29.02.2024  09:16            678'335    news.png
29.02.2024  09:49             1'388    Notizen Live-Coding.txt
29.02.2024  08:45              340    priority.html
29.02.2024  09:52             2'696    README.md
               16 File(s)          11'372'837 bytes
               2 Dir(s)         120'669'908'992 bytes free

C:\Users\Michael\Documents\FHNW\webeC\docs\W02-CSS>jwebserver
Binding to loopback by default. For all interfaces use "-b 0.0.0.0" or "-b ::
".
Serving C:\Users\Michael\Documents\FHNW\webeC\docs\W02-CSS and subdirectories
on 127.0.0.1 port 8000
URL http://127.0.0.1:8000/
```





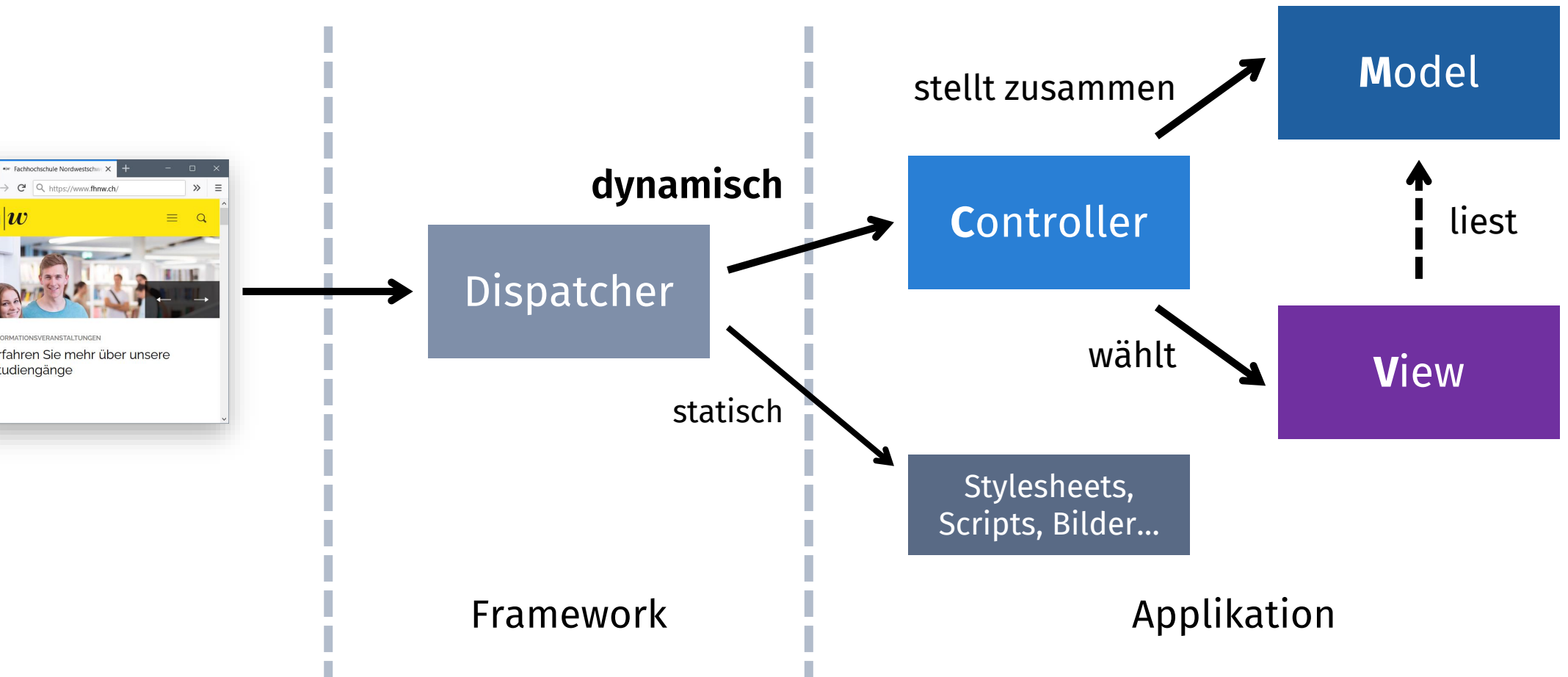
# MVC im Web



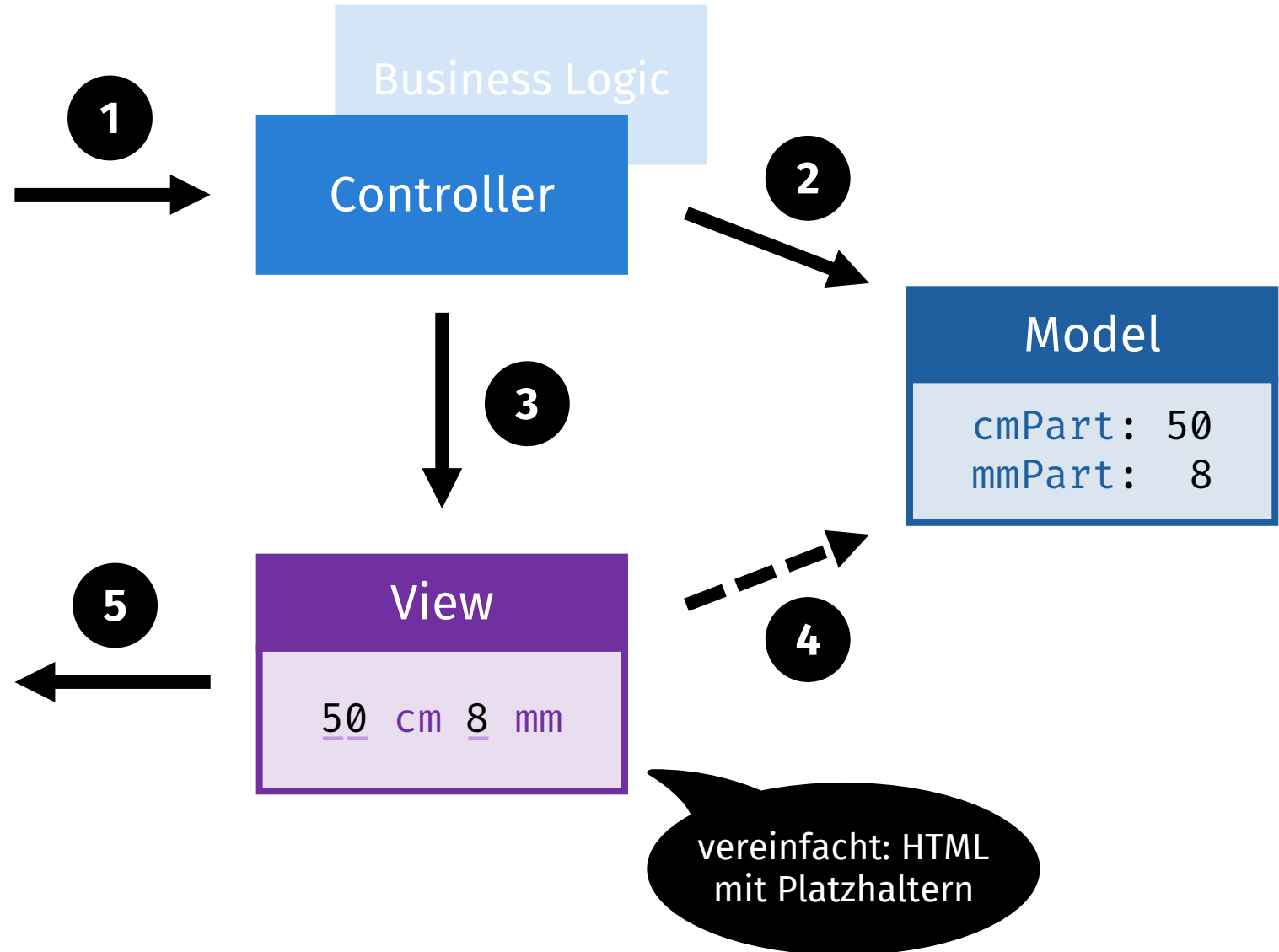
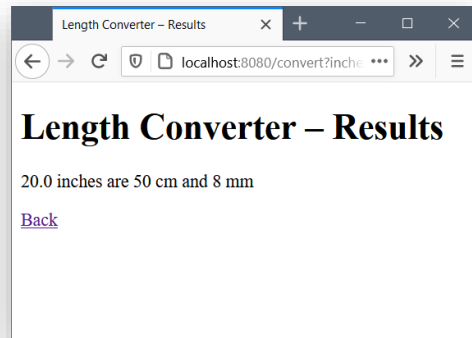
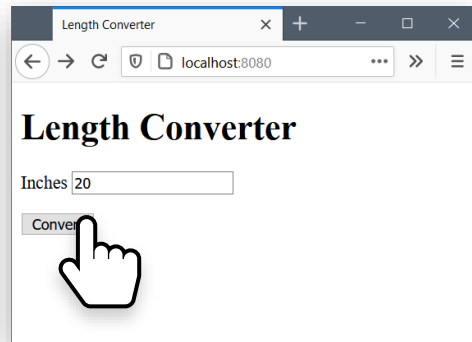
Model View  
Controller

# Web-Server

Server kann Anfrage *irgendwie* verarbeiten, solange er HTTP-Protokoll einhält. Typisch für Web-Applikationen: **statisch/dynamisch** & **MVC**



# MVC: Ablauf



# MVC: Zuständigkeiten

## Model

*Enthält die Daten*

Unabhängig von

**View** und  
**Controller**

## View

*Präsentiert die  
Daten*

Kennt das **Model**,  
aber *liest* es nur

Zuständig für User-  
Interaktion

Unabhängig von  
**Controller**

## Controller

*Empfängt Anfragen  
und ruft nötige  
Ressourcen auf*

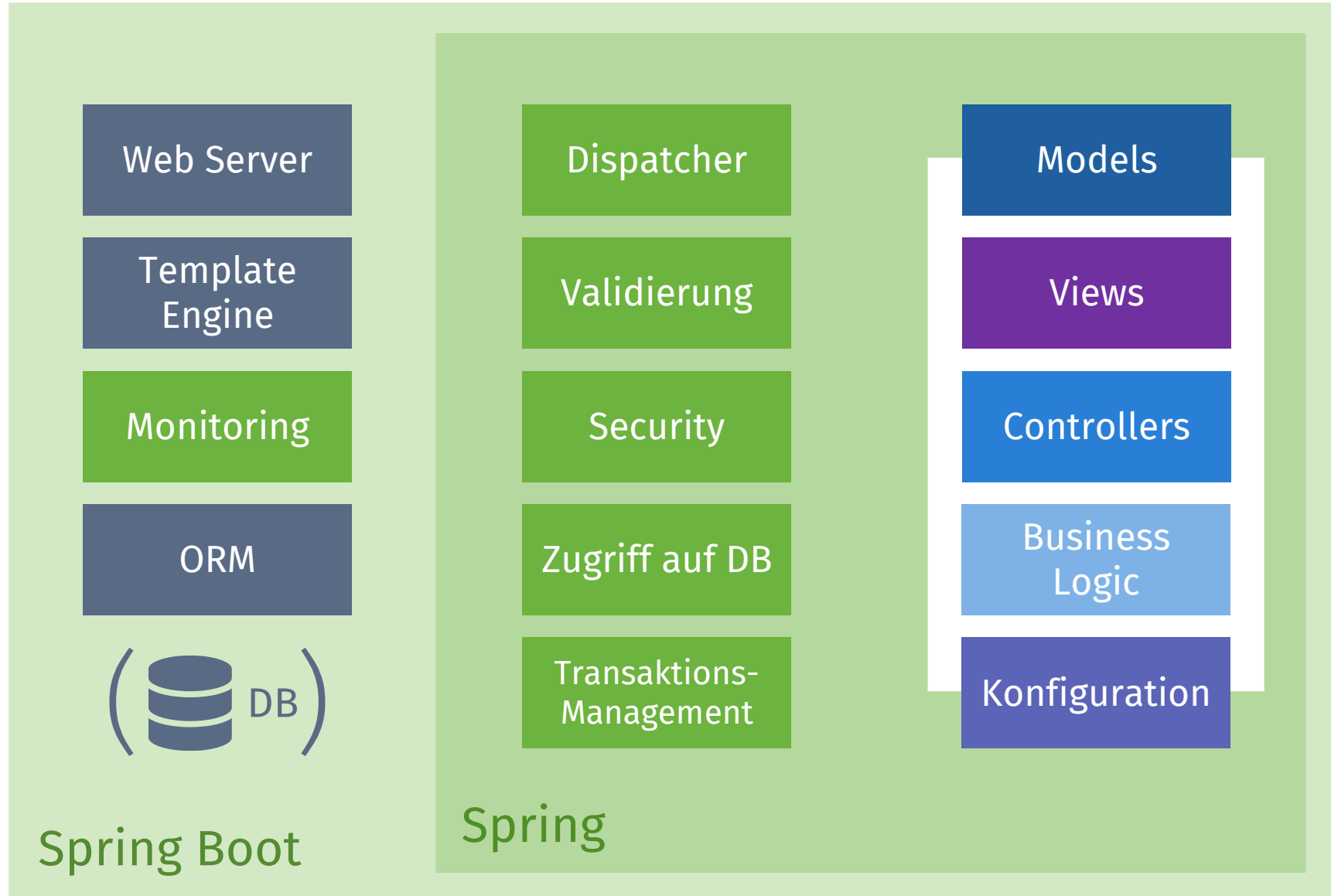
Validiert Eingaben

Stellt **Model**  
zusammen und  
wählt **View** aus

# Spring & Spring Boot



# Spring & Spring Boot



# Spring Framework

Framework für Java-Web-Apps. Grundfunktionalität:

**Web-MVC:** Eingebauter Dispatcher & einige Model-Klassen, einfache Definition der Controller & Views

***Inversion of Control*** (*Dependency Injection*): Finden, Konfigurieren und «verkabeln» von Komponenten

Viele weitere Goodies:

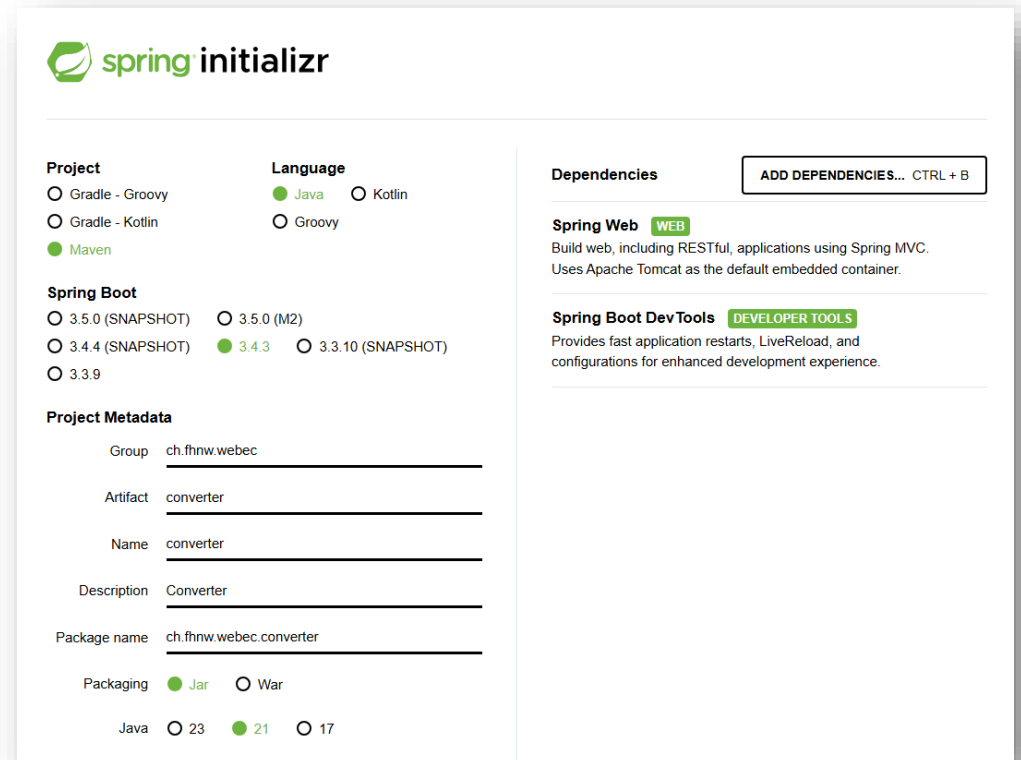
- Validierung & Typumwandlung (Woche 5)
- Einfache Konfiguration von eigenen Komponenten (Woche 5)
- Einfacher Zugriff auf Datenbank, ohne SQL (Woche 6)
- Transaktions-Management (Woche 7)
- Security (Woche 8)

# Spring Boot

## Convention over Configuration

Bietet extrem schnellen Weg, eine Web-Applikation in Java aufzusetzen.

Basiert auf Spring Framework, unterstützt diverse Frameworks & Libraries, enthält Default-Konfiguration für alle.



The screenshot shows the Spring Initializr web application generator interface. It features a clean, modern design with a white background and green accents. The interface is divided into several sections:

- Project:** Includes radio buttons for `Gradle - Groovy`, `Gradle - Kotlin`, and `Maven` (selected).
- Language:** Includes radio buttons for `Java` (selected), `Kotlin`, and `Groovy`.
- Spring Boot:** Includes radio buttons for `3.5.0 (SNAPSHOT)`, `3.5.0 (M2)`, `3.4.4 (SNAPSHOT)`, `3.4.3` (selected), `3.3.10 (SNAPSHOT)`, and `3.3.9`.
- Project Metadata:** Includes input fields for `Group` (ch.fhnw.webec), `Artifact` (converter), `Name` (converter), `Description` (Converter), and `Package name` (ch.fhnw.webec.converter).
- Packaging:** Includes radio buttons for `Jar` (selected) and `War`.
- Java:** Includes radio buttons for `23`, `21` (selected), and `17`.
- Dependencies:** Includes a button `ADD DEPENDENCIES... CTRL + B` and a section for `Spring Web` (WEB) with a description: "Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container." and a section for `Spring Boot DevTools` (DEVELOPER TOOLS) with a description: "Provides fast application restarts, LiveReload, and configurations for enhanced development experience."



# Spring Boot Magic

Warum überhaupt ein Framework? Warum MVC nicht selber machen?

## Vorteile von Spring (Boot)

- weniger aufwändig
- weniger fehleranfällig
- gute Dokumentation
- viele eingebaute Features

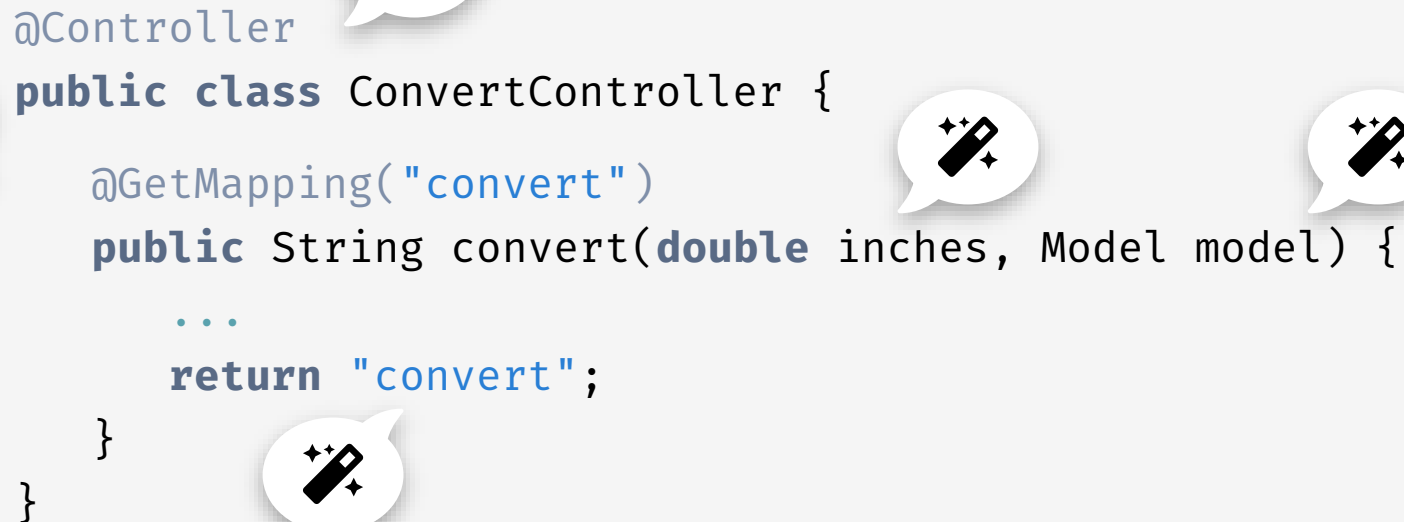
## Nachteile

- Viel Magic... ✨🔧

# Magic mit Annotationen

In Spring wird praktisch alles mit Java-Annotationen gemacht.

**Beispiel:** Controller



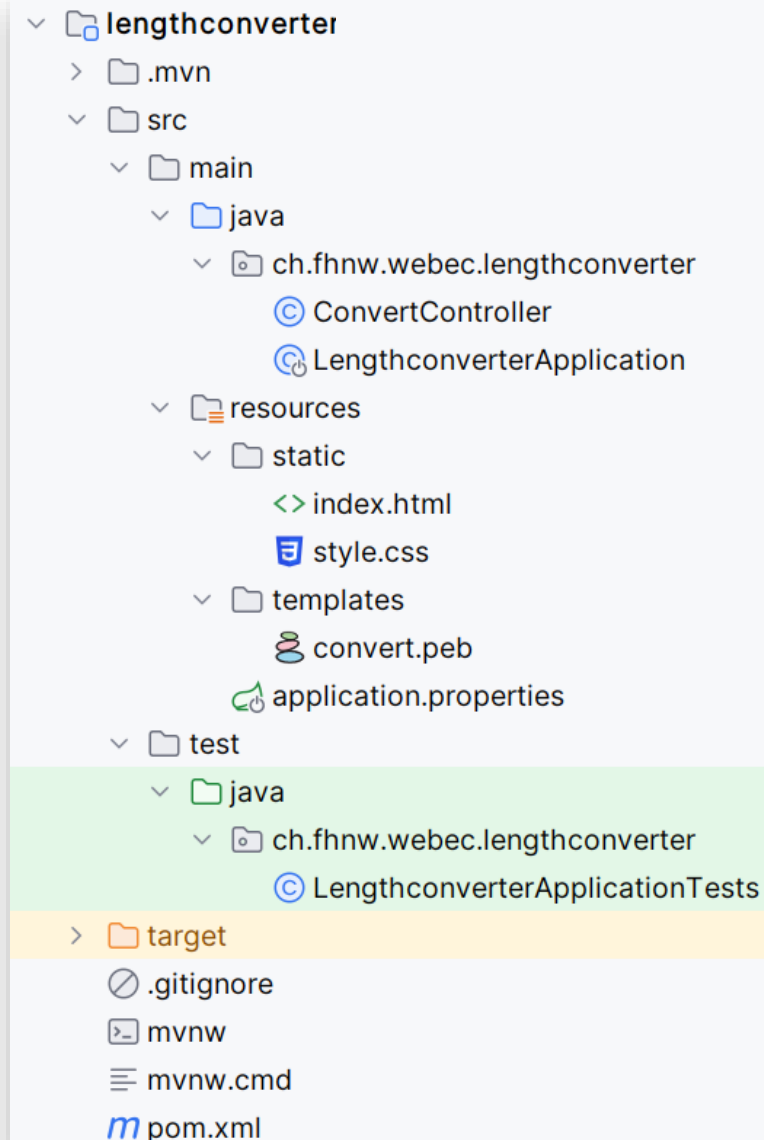
```
@Controller
public class ConvertController {

    @GetMapping("convert")
    public String convert(double inches, Model model) {
        ...
        return "convert";
    }
}
```

The code snippet is presented in a light gray box. Five callout icons, each containing a pencil and stars, are positioned around the code: one above the `@Controller` annotation, one to the left of the `public class` line, one above the `@GetMapping` annotation, one above the `convert` method name, and one below the closing curly brace of the `convert` method.

Kein weiterer (eigener) Code oder Konfiguration nötig!

# Spring Boot: Projektstruktur



Grundstruktur hier: Maven

**src/main/java:** Code für Business Logic, Controllers, ... Aufgeteilt in Packages.

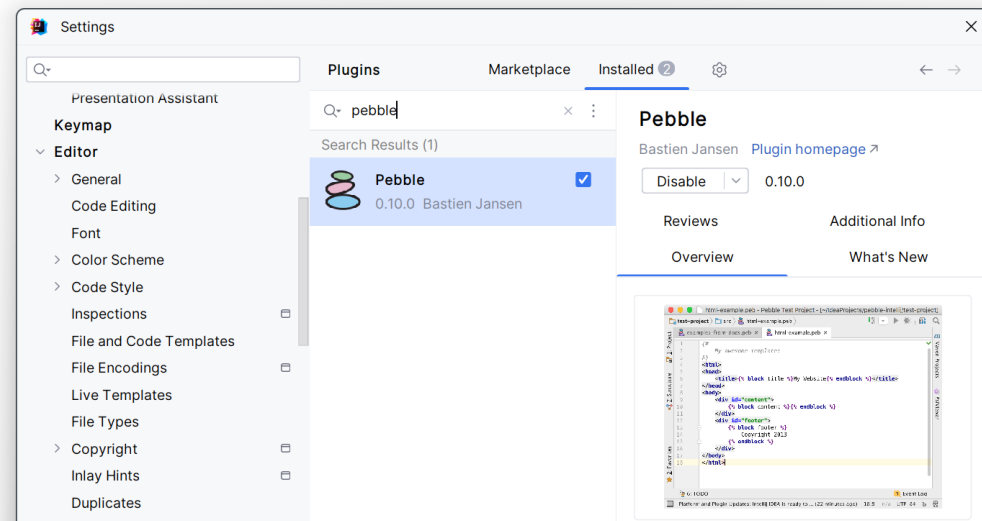
**src/main/resources/...**

- **static:** Statische Seiten, Stylesheets, Bilder, JavaScript, usw.
- **templates:** Definitionen der Views
- **application.properties:** Konfiguration

**src/test/java:** Unit Tests, Integration Tests, E2E-Tests

# Übung 2: Entwicklungsumgebung

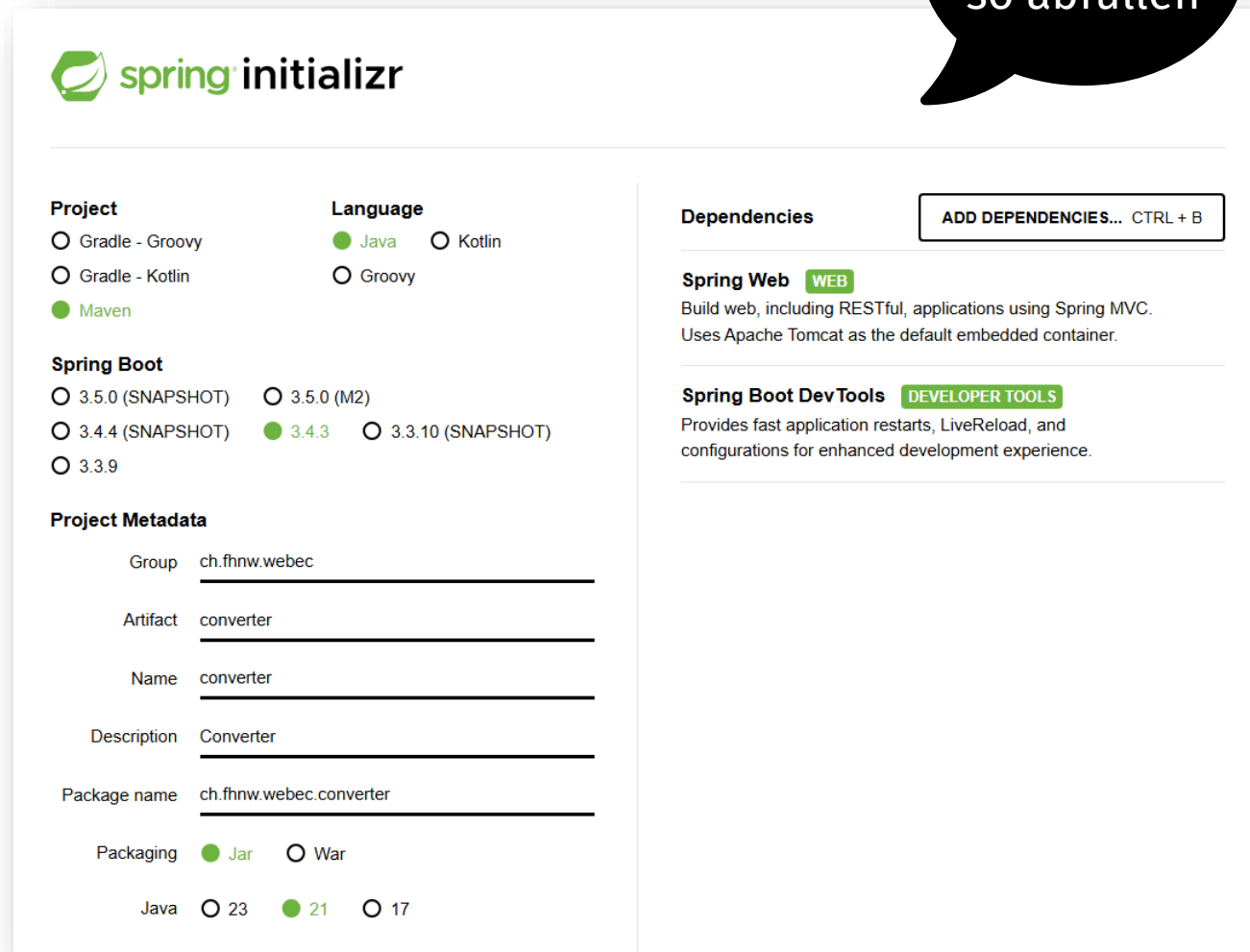
1. Installiere die **IntelliJ**-Entwicklungsumgebung. Empfohlen ist die Ultimate-Edition, welche Studierende gratis beziehen können: [www.jetbrains.com/de-de/community/education/](https://www.jetbrains.com/de-de/community/education/).
2. Stelle unter *File* → *Project Structure* sicher, dass **JDK 21** installiert und ausgewählt ist.
3. Installiere zudem das **Pebble-Plugin** in IntelliJ, das es einfacher macht, Views zu erstellen.



# Übung 3: Hello, Spring!

1. Erstelle mit *IntelliJ* oder dem *Initializr* ([start.spring.io](https://start.spring.io)) eine frische Spring Boot Applikation.

Alles genau so abfüllen

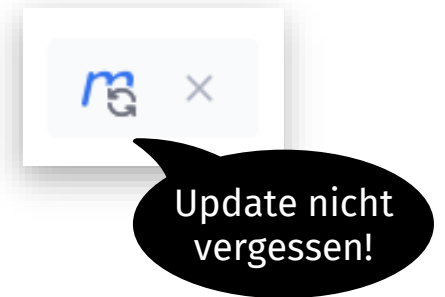


The screenshot shows the Spring Initializr web form with the following configuration:

- Project:** ☒ Maven
- Language:** ☒ Java
- Spring Boot:** ☒ 3.4.3
- Project Metadata:**
  - Group: ch.fhnw.webec
  - Artifact: converter
  - Name: converter
  - Description: Converter
  - Package name: ch.fhnw.webec.converter
  - Packaging: ☒ Jar
  - Java: ☒ 21
- Dependencies:** ☒ Spring Web, ☒ Spring Boot DevTools

## 2. Füge folgende Dependency zur POM-Datei hinzu:

```
<dependency>
  <groupId>io.pebbletemplates</groupId>
  <artifactId>pebble-spring-boot-starter</artifactId>
  <version>3.2.2</version>
</dependency>
```



## 3. Füge eine statische `index.html` Seite hinzu, starte die *ConverterApplication* und kontrolliere, dass die Seite unter <http://localhost:8080> erreichbar ist.

# Übung 4: Der erste Controller

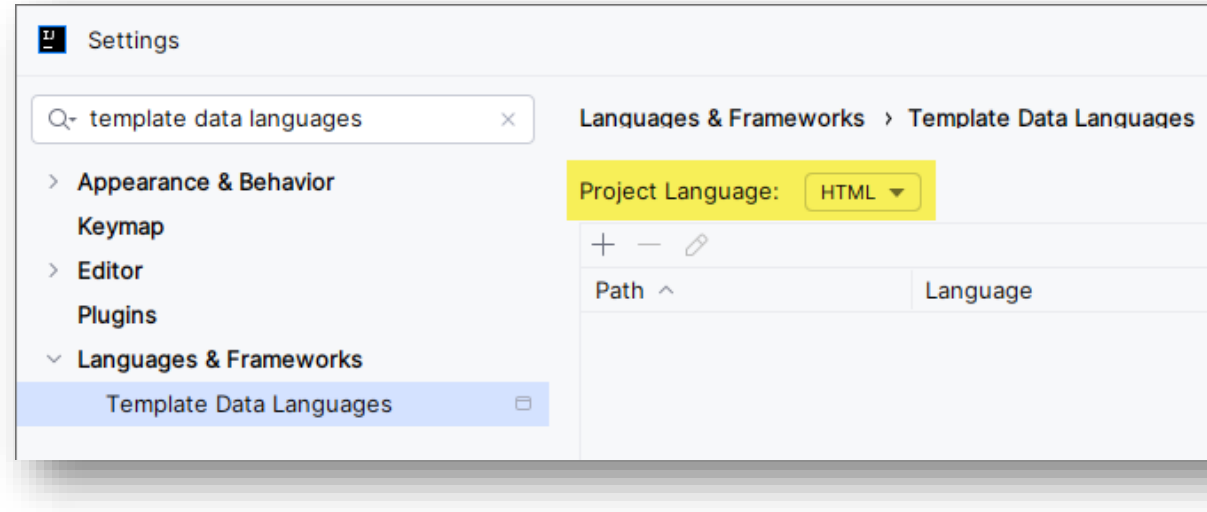
1. Füge folgende Klasse zum Projekt hinzu und ergänze die nötigen Imports:
2. Rufe [localhost:8080/time](http://localhost:8080/time) auf und lade ein paar Mal neu, um zu überprüfen, dass der dynamische Inhalt funktioniert.

```
@Controller
public class TimeController {

    @GetMapping("time")
    public ResponseEntity<String> time() {
        var time = LocalDateTime.now();
        return ResponseEntity.ok()
            .contentType(MediaType.TEXT_HTML)
            .body(
                <!DOCTYPE html>
                <title>Time</title>
                <h1>Current Time</h1>
                <p>
                    The current time is: %s
                </p>
                """).formatted(time));
    }
}
```

# Übung 5: MVC

1. Aktiviere HTML Syntax Highlighting in Pebble Templates:



2. Erstelle die Datei `time-view.peb` im Ordner `templates`:

```
<!DOCTYPE html>
<title>Time</title>
<h1>Current Time with Pebble</h1>
<p>
    {{ time }}
</p>
```

Platzhalter



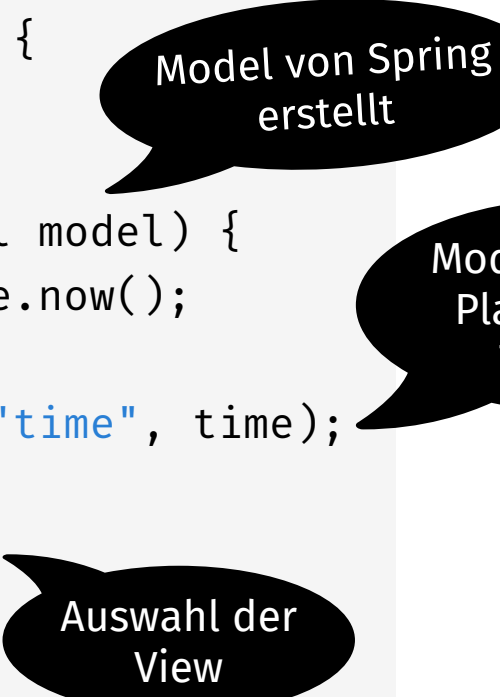
### 3. Ändere den Controller so ab:

```
@Controller
public class TimeController {

    @GetMapping("time")
    public String time(Model model) {
        var time = LocalDateTime.now();

        model.addAttribute("time", time);

        return "time-view";
    }
}
```

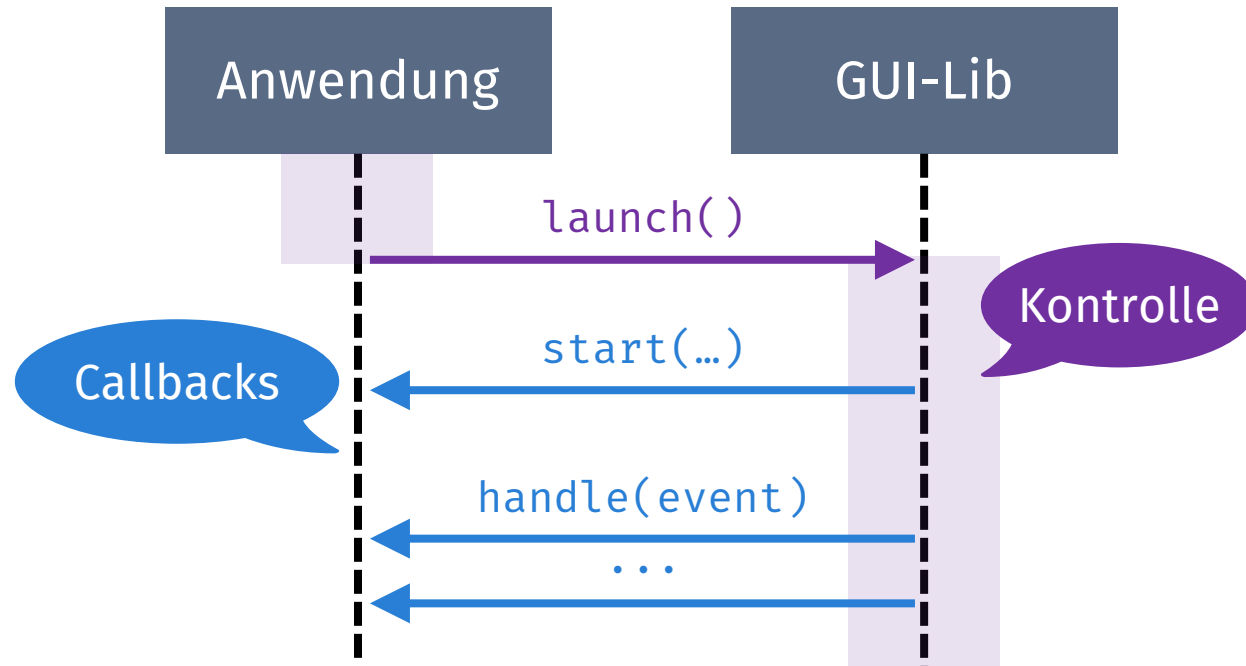


Beim Aufruf von [localhost:8080/time](http://localhost:8080/time) sollte wieder der gleiche dynamische Inhalt wie zuvor erscheinen, jetzt aber durch eine komplette MVC-Struktur erzeugt.

# Dependency Injection

# Inversion of Control

**Inversion of Control (IoC):** «Umkehren der Kontrolle»



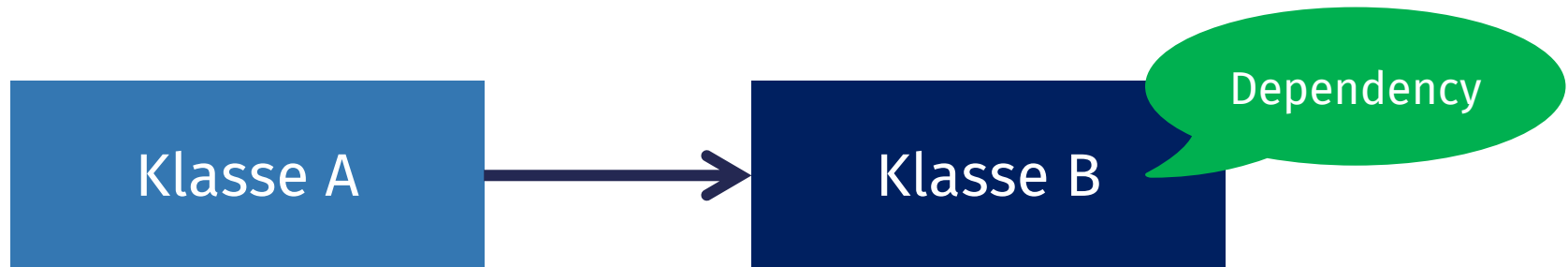
**In OOP2:** Durch Implementieren von Interfaces

**In Spring:** Durch Annotationen. Spring analysiert Code!

# Dependency Injection

- *Was ist eine Dependency?*

Typischerweise eine Klasse B, welche von Klasse A benötigt wird.



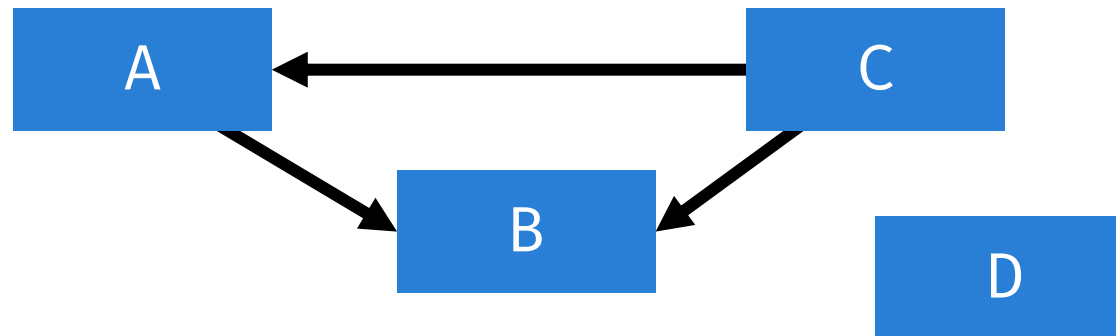
- Wir haben zwei Möglichkeiten:
  - **Ohne DI:** Klasse A instanziert Klasse B selber
  - **Mit DI:** Klasse A erhält eine Instanz von Klasse B injected (also von aussen zur Verfügung gestellt → eine Form von **IoC**)

*DI ist auch ohne DI Framework (a.k.a. IoC Container) möglich!*

# Dependency Injection Framework

Spring DI Framework macht zwei Schritte:

1. Analysiert verwaltete Klassen/Objekte (Beans) und erstellt daraus einen Abhängigkeitsgraph.



2. Instanziert Objekte (Beans) in richtiger Reihenfolge und fügt Abhängigkeiten ein.

```
var b = new B();  
var a = new A(b);  
var c = new C(a, b);  
var d = new D();
```

automatisch  
«generiert»

# @Component

Klasse wird durch Spring als Singleton instanziiert und an den passenden Stellen injected.

```
@Component
public class TimeProvider {

    public LocalDateTime getTime() {
        return LocalDateTime.now();
    }
}
```

```
@Controller
public class TimeController {

    private TimeProvider timeProvider;

    public TimeController(TimeProvider timeProvider) {
        this.timeProvider = timeProvider;
    }
}
```



Instanz wird als  
Konstruktor-Argument  
injected

# @Configuration & @Bean

Flexiblere Variante, um Beans mit Factory-Methode zu konfigurieren.

Vorteile sind unter anderem:

- **Scope** kann angegeben werden: singleton, prototype, request, session, etc.
- **Name** für Beans: ermöglicht z.B. Unterscheidung mehrere Beans vom gleichen Typ
- Objekte von **Klassen, die nicht unter eigener Kontrolle stehen** (z.B. von anderen Libraries), können als Beans zur Verfügung gestellt werden:

```
@Configuration
public class TimeFormatter {

    @Bean
    public DateTimeFormatter timeFormatter() {
        return DateTimeFormatter.ofPattern("HH:mm:ss");
    }
}
```

# @Configuration & @Bean: Beispiel

```
@Bean // using method name ("idProvider") as name
// using default scope "singleton"
public IdProvider idProvider() { // no dependencies to other beans
    return new IdProvider();
}

@Bean(name = "priorityTaskProcessor") // configuring a custom name
@Scope("prototype") // defining a custom scope
public TaskProcessor processor(IdProvider idProvider) { // injecting dependency
    String taskId = idProvider.getId();
    return new TaskProcessor(taskId); // no singleton because class has state
}

public class TaskProcessor {
    private String taskId;

    public TaskProcessor(String taskId) {
        this.taskId = taskId;
    }

    public void processTask() { ... }
}
```



Alles innerhalb  
einer Klasse mit  
@Configuration



# Komponenten & Controller

Wo verwendet man diese Beans und Komponenten überhaupt?

Zum Beispiel in  
einem Controller:

```
@Controller
public class TimeController {

    private final TimeProvider timeProvider;

    public TimeController(TimeProvider timeProvider) {
        this.timeProvider = timeProvider;
    }

    @GetMapping("time")
    public String time(Model model) {
        model.addAttribute("time", timeProvider.getTime());
        return "time-view";
    }
}
```

Komponente wird  
eingefügt!

Controller sind ebenfalls Komponenten!

# DI und Unit Tests

Dependency Injection macht es möglich, Abhängigkeiten bei Bedarf in einem Test zu ersetzen.

```
public class TimeControllerTest {  
  
    @Test  
    public void testTimeController() {  
  
        TimeProvider timeProvider = new TestTimeProvider();  
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm:ss");  
  
        TimeController tc = new TimeController(timeProvider, formatter);  
  
        String view = tc.time4(new ConcurrentModel());  
  
        assertEquals("time-view", view);  
        assertEquals("10:30:00", m.getAttribute("time"));  
    }  
}
```



TestTimeProvider  
gibt immer die gleiche  
Zeit zurück

# Übung 6: Length Converter

Implementiere einen neuen Controller, der eine Längenangabe in Inches in Zentimeter und Millimeter umwandelt.

1. Um Parameter an einen Controller zu übergeben, braucht es ein HTML-Formular:

```
<form method="get" action="convert">  
  <p>  
    <input type="text" name="inches">  
  </p>  
  <p>  
    <input type="submit" value="Convert">  
  </p>  
</form>
```


Ziel-URL

Parametername

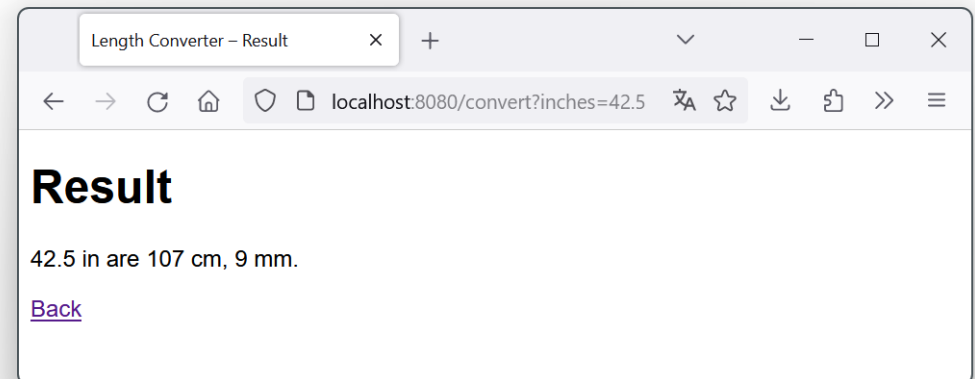
2. Nehme im Controller den Parameter entgegen, indem du einen normalen Methoden-Parameter mit dem gleichen Namen hinzufügst:

```
@Controller
public class ConvertController {

    @GetMapping("convert")
    public String convert(double inches, Model model) {
        ...
    }
}
```



3. Führe die Umwandlung im Controller durch und erweitere Model & View, um das Resultat anzuzeigen:



# Fragen?

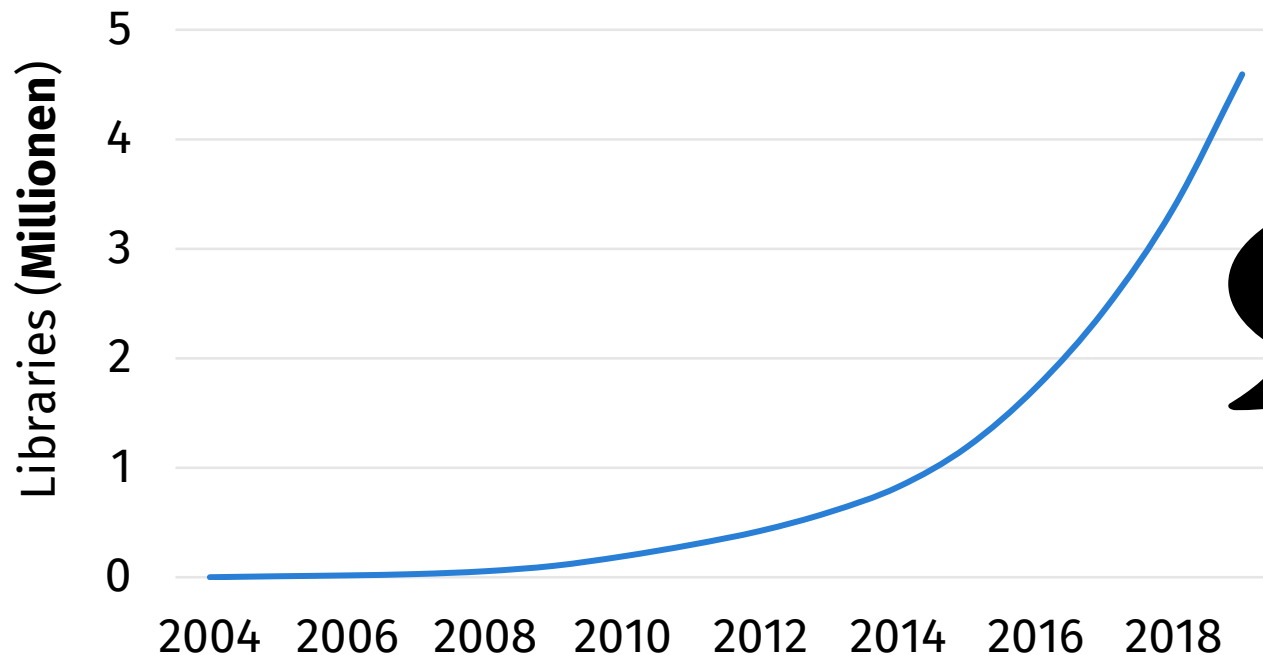


Anhang: *Maven*<sup>TM</sup>



Build-Tool (vor allem) für Java-Projekte

- Plattform-unabhängig: auf Windows genau wie auf macOS, Linux, ...
- Verwaltet Abhängigkeiten automatisch dank «Maven Central»: riesige Online-Datenbank mit vorkompilierten Java-Libraries!

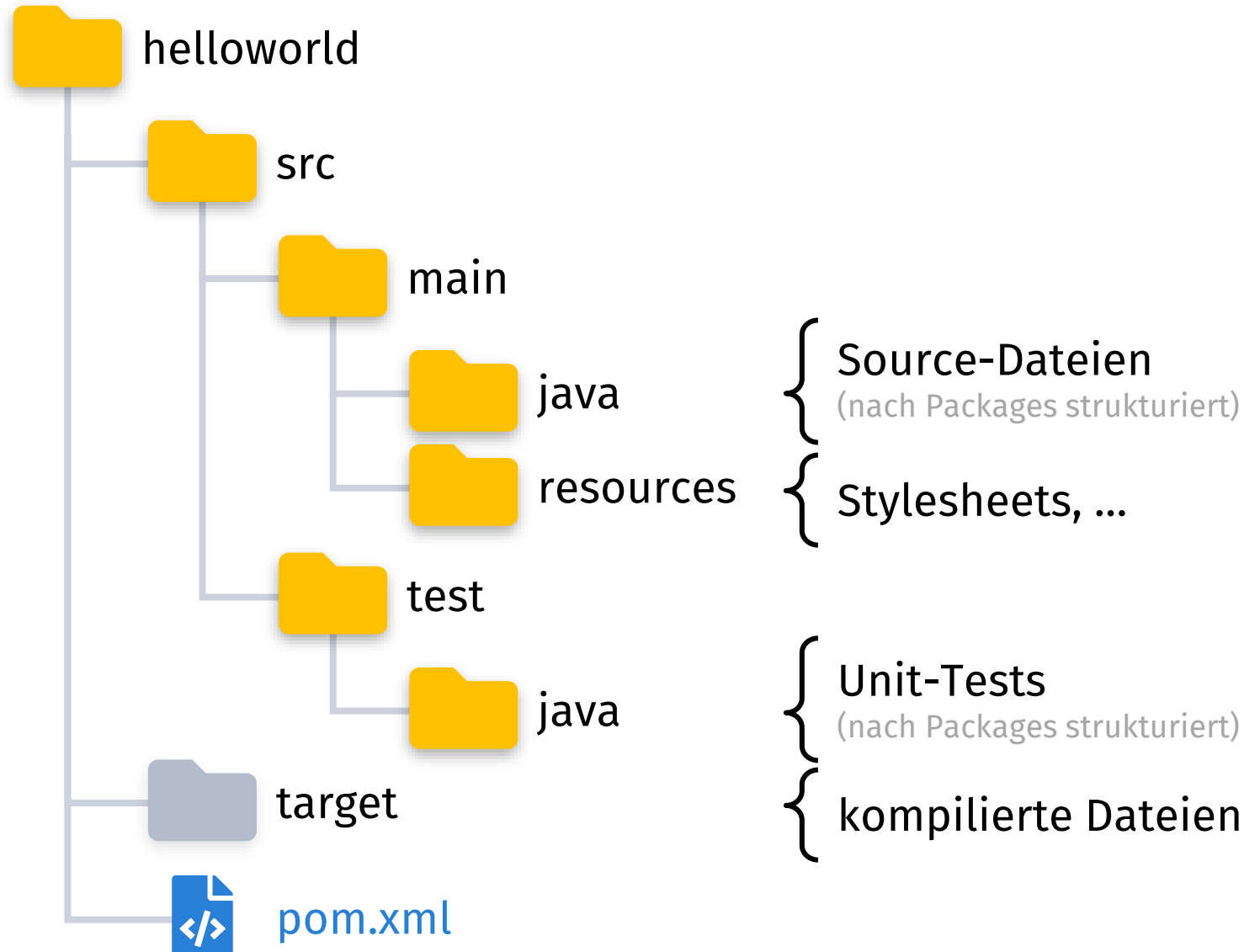


mehrere Versionen  
der selben Library...

## Maven-Philosophie: «Konvention vor Konfiguration»

- Ordner-Struktur, Build-Ablauf, Einstellungen sind standardisiert.

### Ordnerstruktur:





# Die «pom.xml»-Datei

Alle Informationen zum Projekt sind in einer Datei namens «pom.xml»

Immer gleich	{	<pre>&lt;project xmlns="http://maven.apache.org/POM/4.0.0" ... &gt;   &lt;modelVersion&gt;4.0.0&lt;/modelVersion&gt;   &lt;groupId&gt;ch.fhnw.webec&lt;/groupId&gt;   &lt;artifactId&gt;helloworld&lt;/artifactId&gt;   &lt;version&gt;1.0-SNAPSHOT&lt;/version&gt;    &lt;properties&gt;     &lt;maven.compiler.source&gt;21&lt;/maven.compiler.source&gt;     &lt;maven.compiler.target&gt;21&lt;/maven.compiler.target&gt;   &lt;/properties&gt; &lt;/project&gt;</pre>
«Koordinaten» (obligatorisch)	{	
Eigenschaften (optional, z. B. Java-Vers.)	{	
Immer gleich	{	

Obligatorisch sind nur **Projekt-«Koordinaten»**: Gruppe, Name, Version.

- Andere Projekte können dieses Projekt mit diesen Koordinaten als Abhängigkeit hinzufügen

# Einige Maven-Befehle

Standard-Befehle (bauen aufeinander auf):

Befehl	Beschreibung
<code>mvn compile</code>	Kompiliert die «Haupt»-Dateien
<code>mvn test</code>	Kompiliert die Test-Dateien und führt die Tests aus
<code>mvn package</code>	Verpackt das Projekt, z.B. als jar-Datei
<code>mvn install</code>	«Installiert» das Projekt im lokalen Repository
<code>mvn deploy</code>	Veröffentlicht das Projekt in einem Remote-Repository

Zusatzbefehle:

Befehl	Beschreibung
<code>mvn clean</code>	Entfernt alle erstellten Dateien
<code>spring-boot:run</code>	Führt eine Spring-Boot-Applikation aus