

Webbasierte Anwendungen

Wolfgang Weitz

HSRM / Medieninformatik

Sommersemester 2022



Stand: 20. April 2022



Organisatorisches

Formalitäten

- Modul besteht aus Vorlesung und Praktikum, insgesamt 6 CP
- Prüfungsmodus:
 - Modul-Prüfung zum Semesterende (Prüfungsleistung)
 - Hinweis: Modul-Prüfung umfasst Inhalte aus *gesamtem* Modul (das ist immer so)
 - Praktikum (separate Studienleistung):
 - verpflichtende, aufeinander aufbauende Übungsaufgaben,
 - Lösungen sind jeweils innerhalb von 12 Tagen ab Erscheinen des Übungsblatts selbständig zu bearbeiten und per Git gemäß Vorgabe bis Fristende abzugeben
 - und auf Nachfrage zu erläutern
 - keine Gruppenarbeit
 - Anwesenheitspflicht gem. Prüfungsordnung ($\geq 75\%$ der Praktikumstermine)
- Unterlagen: <https://read.mi.hs-rm.de>

Voraussetzungen

Viele benötigte Inhalte können Sie schon . . .

- **HTML, CSS** aus “Auszeichnungssprachen”
- **Webdesign** aus “Gestaltung elektronische Medien”
- **OO-Programmierung mit Java** aus “Programmieren 1, 2”
- Skriptsprache **Python** aus “Programmieren 3”
- Relationale **Datenbanken, SQL** aus “Datenbanksysteme”
- **Interaktionsschlaueheit** aus “EIBO” und den Gestaltungsfächern



Themen dieses Semesters

- **Grundprobleme** von Webanwendungen
- Architekturen und **softwaretechnische Konzepte**
(Buildautomation, Schichtung, ORM, Reactive Programming, ...)
- **CGI** als einfache, programmiersprach-unabhängige serverseitige API
- Web-relevante Teile des **Spring Frameworks** als Basis für große (kommerzielle) Anwendungssysteme
- Formularbasierte Webanwendungen am Beispiel von **Spring MVC**
(Eingabevalidierung, Internationalisierung, Konfigurierbarkeit)
- Datenbankintegration mit dem **Java Persistence API (JPA)**
- **REST-Services** und asynchrones **Messaging** als Grundlage für moderne Web-Frontends
- **Single-Page-Applications** am Beispiel **Vue.js** und *TypeScript*
- **Sicherheit** bei Webanwendungen



Literatur

Ergänzende Bücher und Klickbarkeiten

... es gibt (?) leider nicht "das" Buch für Personen mit Ihrem Vorkenntnis-Profil

Michael Simons

Spring Boot 2: Moderne Softwareentwicklung mit Spring 5
dpunkt, 2018



Eberhard Wolff

Microservices – Grundlagen flexibler Softwarearchitekturen
dpunkt, 2018



Elektrische Quellen u.a.:

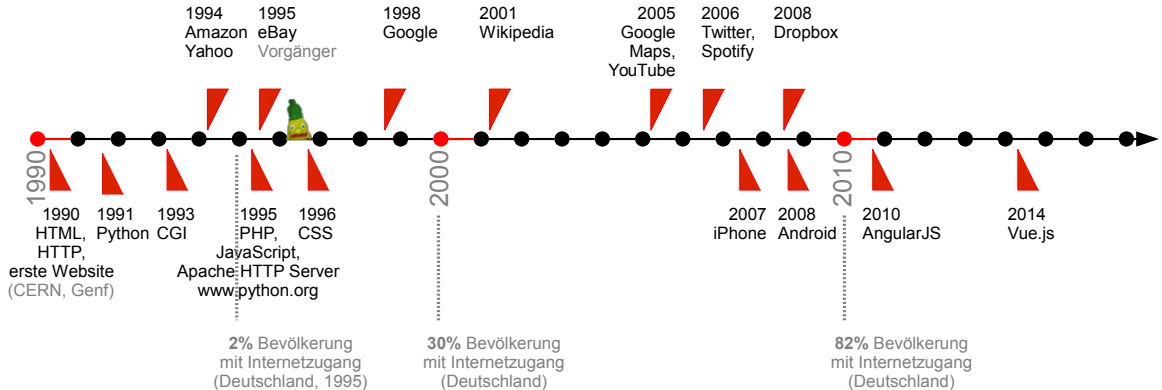
- CGI-Scripting mit Python – <https://docs.python.org/3.9/library/cgi.html>
- Serverseitiges **Spring Framework** – <https://spring.io>
- Frontend- (Browser-)seitiges Framework **Vue.js** – <https://vuejs.org/>
- Programmiersprache **TypeScript** – <https://www.typescriptlang.org/>



Einführung

Web-Historie

Technologien und bekannte Dienste

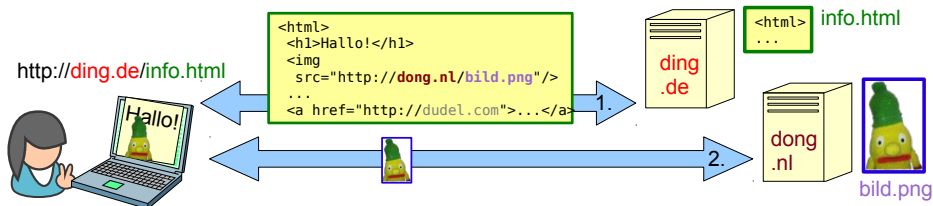


- Internet \neq World-Wide-Web (WWW), vgl. Vorlesung “Rechnernetze”
- Wie sah das WWW früher aus? Waybackmachine <http://web.archive.org/>



Web-Grundlagen

Grundbausteine: HTML über HTTP



- **URLs** zur eindeutigen, einheitlichen **Bezeichnung** von Ressourcen
- **HTTP** als Netzwerkprotokoll zur Client/Server **Kommunikation**
- **HTML** als **Datenformat** für strukturierte Dokumente
- Einfaches **Zusammenziehen von Informationen** aus und **Querverweise** (Links) zwischen Ressourcen von verschiedenen Servern, für Endanwender “unsichtbar”.
- Offen für andere Datenformate und Protokollvarianten
- Client sieht **nicht**, ob abgefragte Inhalte als **statische** Dateien auf Server hinterlegt sind oder **dynamisch** “auf Zuruf” für ihn erzeugt werden (das ist gut!)

URL-Anatomie und “Prozent-Codierung”

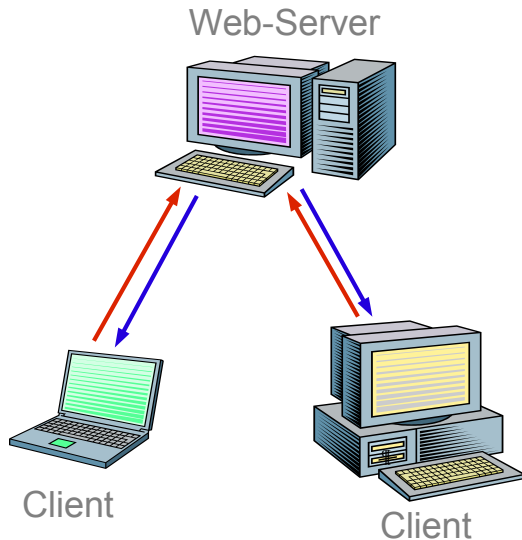
Beispiele

```
http://www.mi.hs-rm.de/
http://localhost:8080/beispiel.html
ftp://www.mi.hs-rm.de/pub/download.zip

http://www.mi.hs-rm.de/docs/sachen/wie+das+geht.html#abschnitt-1
http://www.mi.hs-rm.de/cgi-bin/programm.cgi?name=J%F6hard%20Biffel&alter=17
http://jbiff001:geheim@meine.website.de/progs/abruf.cgi?dok=buch1#toc
```

- **Schema:** Art der URL (z.B. http, ftp, mail)
- Schema-spezifischer Teil (beginnt oft mit “//”), z.B. für http
 - optional: Benutzername[:Passwort]@
 - **Zielrechner**, optional mit :portnummer
- **Pfad** – wird protokollabhängig vom Zielrechner **interpretiert**, ggf gefolgt von
 - **Query-String:** nach “?” Folge von “&”-getrennten “Schlüssel=Wert”-Paaren
 - **Fragment-Identifizier** nach “#”, bezeichnet Stelle innerhalb Ressource
- Buchstaben a-z, Ziffern und Zeichen “. ~ - _” ok, Leerzeichen entweder als “+” oder “%20”, weitere Zeichen als hexadezimaler Zeichencode der Form %xx
- Details siehe <https://tools.ietf.org/html/rfc3986>

HTTP



- **HyperText Transfer Protocol**

Spezifiziert in RFC 1945, RFC 2616

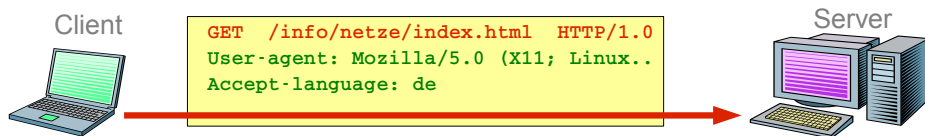
- **Ablauf (HTTP 1.0)**

- Client baut TCP-Verbindung zu Server auf (default: Port 80)
- Client schickt Anfrage
- Server schickt Antwort
- Verbindung wird geschlossen

- **HTTP 1.1:** Gleiche Verbindung kann mehrfach wiederverwendet werden (z.B. für HTML-Seite mit 17 Bildern zwar 18 Anfragen, aber nicht 18x Verbindungsauf-/abbau)

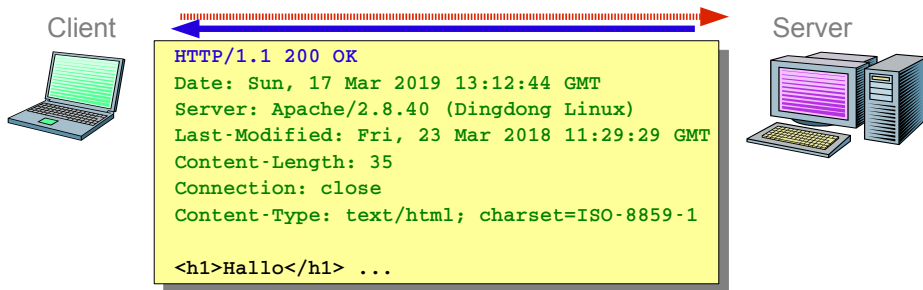
- **Zustandsloses** Protokoll, aufeinanderfolgenden Anfragen sind unabhängig voneinander.

HTTP Anfrage (request)



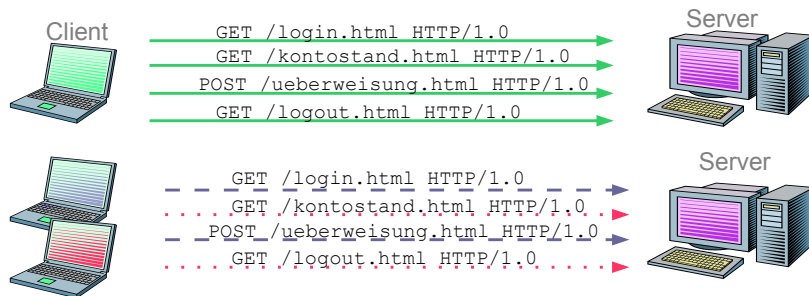
- Erste Zeile **request line** mit drei Feldern
 - HTTP Methode: **was** soll getan werden? (GET, POST, PUT, HEAD, DELETE, ...)
 - “Einfacher” Abruf einer URL z.B. im Browser verwendet üblicherweise GET
 - **worauf** soll die Operation angewandt werden? (**Pfad**, ggf. mit **Query-Parametern**)
 - Protokoll/Version (z.B. HTTP/1.0, HTTP/1.1)
- Beliebige viele **Header Zeilen** mit ergänzenden Schlüssel-/Wert-Paaren
 - z.B. Accept-language: Wunschsprache(n) für angefordertes Objekt
(sonst: Default-Sprache - Server bestimmt selbst, was er liefern kann bzw. will)
- **Leerzeile** (!) als Abschluss der Anfrage

HTTP Antwort (response)



- **Status-Zeile** (Protokoll/Version, **Statuscode** und -Meldung)
- Statuscode: 2xx Erfolg, 3xx Umleitung, 4xx Fehler des Clients, 5xx Serverfehler
- **Header-Felder** mit Zusatzinformationen (Datum/Zeit der Antwort, Servertyp, Änderungsdatum des abgerufenen Dokuments, Länge / Typ des Inhalts, ...)
- **Leerzeile** (!) markiert Ende des Blocks, darf nicht fehlen
- Abgerufener **Inhalt** folgt ggf. nach der Leerzeile (HTML, Grafik,...; binäre Daten ok)

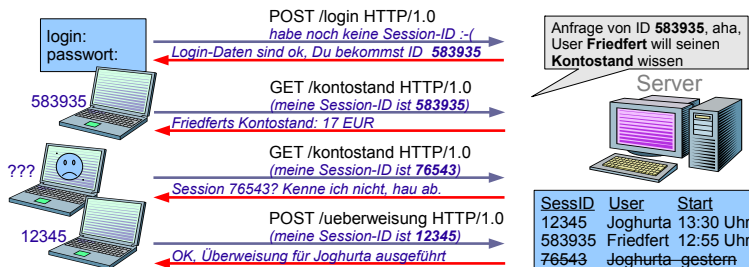
Problem: Zustandslosigkeit von HTTP



- HTTP ist **zustandslos**, Anfragen sind unabhängig voneinander, können ohne Zusatzmaßnahmen nicht einem Anwender (anonym oder nicht) zugeordnet werden.
- “Sitzungs-Zustand” relevant für serverseitige Anwendung? Beispiele
 - Benutzer muss **eingeloggt** sein, bevor er **seinen** Kontostand sieht.
 - Warenkorbinhalt sollte je Nutzer über mehrere Klicks hinweg erhalten bleiben.
- Tausende von Benutzern können “gleichzeitig” auf Webserver zugreifen.
- Wie kann der Web-Server erkennen, welche Zugriffe zusammen gehören?

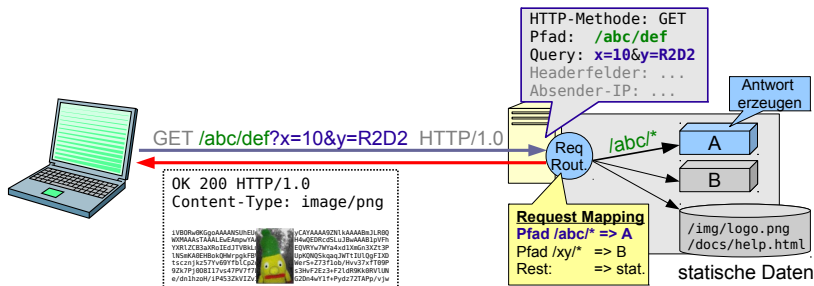
Hinweis: Nein, “gleiche Absender-Rechneradresse (IP) = gleicher User” reicht bei weitem nicht

Ansatz: Sessions – serverseitig Sitzungsdaten auseinanderhalten



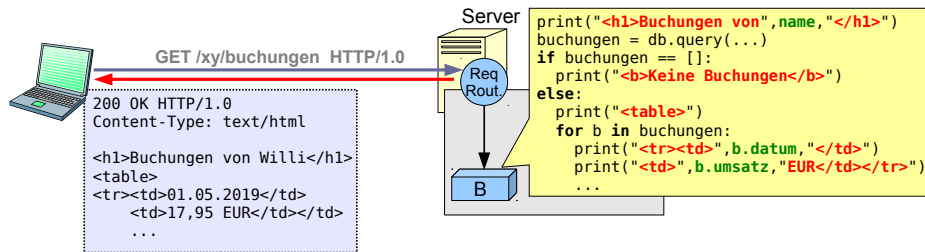
- **Server** vergibt (aktuell) eindeutige, **zufällige SessionID** in Antwort an Client **nach erstem Zugriff** und **speichert Zuordnung** "SessionID → Benutzersitzung" (neue Sitzung – neue SessionID)
- **Server** teilt Client in (von ihm generierter) HTTP-Antwort seine **SessionID** mit: **Cookie** setzen, als **Query-Parameter in URLs** flächendeckend in Antwort-HTML integriert ("URL-Rewriting") oder **verstecktes Formularfeld** einbauen - Hauptsache, Folgeanfrage vom Client liefert ID mit.
- **Client** schickt erhaltene SessionID **mit jeder Folgeanfrage** mit, **Server** kann damit die Anfrage einer konkreten Sitzung zuordnen.
- **Server erlaubt Zugriff** auf Sitzungs-relevante Daten (Warenkorb, benutzerbezogene Daten) nur dann, wenn Client aktuell **gültige SessionID** mitschickt.
- "Ausloggen" erfordert nur serverseitiges Löschen der SessionID (explizit oder nach Timeout)

Request Routing – serverseitige Anfrageverarbeitung



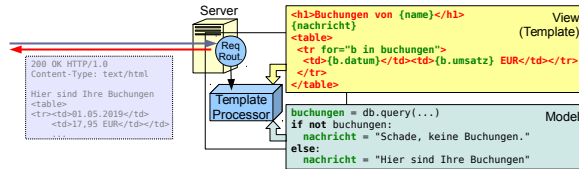
- HTTP-Server **router** HTTP-Request gemäß Konfiguration an zuständige Handler-Komponente (in Webserver eingebaut oder ladbare Erweiterung oder externes Programm oder ...)
- **Kriterien** für (statisches oder dynamisches) **Request-Mapping** z.B. HTTP-Methode (GET, POST usw.), angefragter Pfad und ggf. Query-Parameter, Formulardaten, HTTP-Header-Einträge, Anfrager-IP, ...
- **Handler** hat Zugriff auf Daten aus Anfrage, **erzeugt Antwort** "irgendwie" (oder leitet Request an andere Handler weiter oder... wie er was erzeugt, ist seine Sache)
- HTTP-Response geht **zurück an Client**, Response-Headerfeld **Content-Type** gibt Hinweis auf **Art des Inhalts** (MIME-Type), damit Client korrekt damit umgehen kann.
- (Wichtig, z.B. Anfragepfad /xy/abc.html kann *irgendein* Antwortformat ergeben, *nicht unbedingt* HTML)

Problem: HTML im Code verstreut



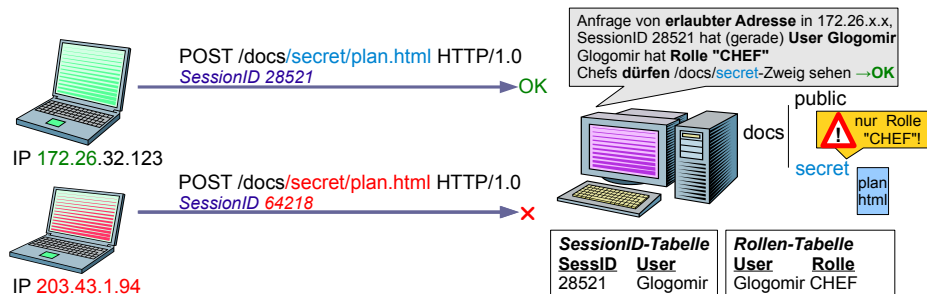
- Erstellung und Pflege mühsam
- Vermischung von Darstellung und Anwendungslogik
- Übliche HTML-Editoren kaum einsetzbar
- Arbeitsteilung Entwickler/Web-Designer:
 - die einen machen das Design kaputt,
 - die anderen den Code :-)

Ansatz: Templating – Vorlagen dynamisch mit Inhalt füllen



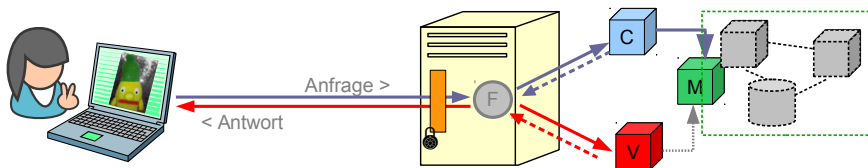
- **HTML Vorlagen** (Templates) mit Platzhaltern werden von einem **Template Processor** mit dynamischen Daten gefüllt, um Antwort zu erzeugen.
- Der Template Processor kann **eigenständiges Programm** sein (ggf. mit eigener Programmiersprache, z.B. PHP) oder **eingebettete Software-Komponente** (z.B. Jinja für Python, Thymeleaf für Java).
- Je nach verwendetem System mehr oder weniger Logik in HTML sichtbar.
In manchen (frühen) Systemen kann Anwendung komplett in den Templates gecodet werden (z.B. PHP), incl. DB-Abfragen und Verarbeitung – na ja...
- Templates sind idealerweise gültiges HTML, HTML-Editoren damit normal einsetzbar.
- Konfliktärmere Arbeitsteilung: Entwickler erstellen Code (Model), Designer HTMLen und CSSen (View)
- Trennung Darstellung ("View") von darzustellenden Anwendungslogik/-daten ("Model") ist guter Stil (vgl. "Model-View-Controller" Konzept aus EIBO)

Problem: Zugriffsschutz – Beispiele für Kriterien



- Schutz für alle oder nur bestimmte Zweige? ⇒ angefragten **Zugriffspfad** in HTTP-Anfrage checken
- Benutzer eingeloggt / SessionID gültig? ⇒ gültiger Eintrag in **Session-Tabelle**?
- Mit Session assoziierter User (bzw. dessen Rolle) für gewünschte Ressource berechtigt?
⇒ **Zuordnung User** → Rolle(n) → **Rechte** (für Ressource)
- Client-IP bzw. dessen Teilnetz zugelassen? ⇒ Client **IP-Adresse** für eingehende Anfragen **filtern**
- Kriterien kombinierbar ("nur *Chefs* aus *Subnetz 172.26.32.x* dürfen auf ... zugreifen")

Zusammenfassende Skizze zur Requestverarbeitung

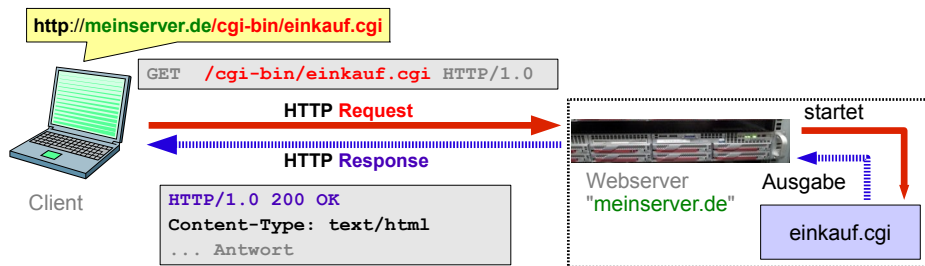


- Entgegennehmen der “rohen” Anfrage, z.B. durch *allgemeinen* **Front-Controller** (F)
 - **Zerlegen** der Anfrage (Pfad, Parameter, ...)
 - **Zugriffsschutz**: Filtern / Blockieren unberechtigter Anfragen
 - **Request-Routing**: Weiterleiten der Anfrage an den “richtigen”, spezialisierten Controller
 - kann sich (je nach Implementierung) um **Session-Management**-Aspekte kümmern
 - oft konfigurierbarer, anwendungsunabhängiger Bestandteil eines Web-Frameworks
- *Spezialisierte* **Controller** (C) bereiten Web-Anfragen für Operationen auf (Web-neutralem) **Model** (M) auf und wählen (ggf. eingabeabhängig) eine passende View aus
- **View** greift auf (geändertes) Model zu und erstellt die Antwort (z.B. per Templating)
- Antwort fließt zurück an Client.



Common Gateway Interface (CGI)

Einfache serverseitige Schnittstelle: CGI



- Sehr frühe Methode zur **dynamischen** Verarbeitung von Anfragen.
- Idee: **Webserver startet** zur Verarbeitung einer HTTP-Anfrage ein vorkonfiguriertes, ggf. durch die Anfrage identifiziertes **CGI-Programm** (immer als **neuen Prozess**),
- dieses erhält vom Server die **Anfragedaten** durch vorbelegte **Environment-Variablen** und/oder über die **Standardeingabe** hereingereicht,
- die **Standardausgabe** des CGI-Programms wird als **Antwort an Client** zurückgeleitet, Programm **endet** nach Abarbeitung seines (einen!) Requests.
- **Programmiersprachunabhängig** (nur stdin/-ausgabe und Environmentvariablen)

Wichtige Environment-Variablen

Beispiel: HTTP GET-Anfrage für folgende URL:

<http://www.mi.hs-rm.de/~jbiff017/cgi-bin/show.cgi/docs/webbuch.html?version=17&farbe=rot>

Das CGI-Programm bekommt hier u.a. folgenden Environment-Variablen gesetzt:

- HTTP_HOST ist **Server**-Hostname (www.mi.hs-rm.de)
- REMOTE_ADDR ist IP-Adresse des **Clients** (also z.B. die IP Ihres Rechners)
- REQUEST_METHOD ist verwendete **HTTP-Methode** (GET)
- REQUEST_URI ist gesamte **Pfad+Query-String**-Komponente der URL
(/~jbiff017/cgi-bin/show.cgi/docs/webbuch.html?version=17&farbe=rot)
- SCRIPT_NAME ist **Pfadanteil zu CGI-Skript** (/~jbiff017/cgi-bin/show.cgi)
- PATH_INFO ist **verbleibender Pfad-Anteil** der URL (/docs/webbuch.html)
- QUERY_STRING ist **Query**-Teil der Anfrage (version=17&farbe=rot)

Hinweise:

- “Pfad-Teil” der URL *muss nicht* mit dem CGI-Programm show.cgi enden – CGI-Programm kann folgende Pfadkomponenten (PATH_INFO) frei interpretieren.
- Es muss also im URL-Beispiel oben weder einen Ordner docs noch eine Datei webbuch.html geben, noch muss die Anfrage HTML zurückliefern

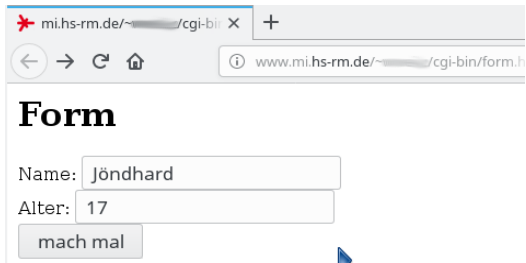
Beispiel: Formulardaten-Codierung "urlencoded" (default)

1. Client erhält HTML-Formular

```
<h1>Form</h1>

<form action="/~jbiff017/cgi-bin/dump.cgi" method="post">
  Name: <input type="text" name="name"/><br/>
  Alter: <input type="text" name="alter"/><br/>
  <input type="submit" name="okbutton" value="mach mal"/>
</form>
```

2. Client schickt Formular ab



The screenshot shows a web browser window with the address bar displaying 'www.mi.hs-rm.de/~jbiff017/cgi-bin/form.h'. The page content includes a heading 'Form' and two text input fields: 'Name: Jöndhard' and 'Alter: 17'. Below these fields is a submit button labeled 'mach mal'.

3a Server: CGI-Skript erhält über Environmentvariablen (Auswahl):

```
CONTENT_LENGTH=42
CONTENT_TYPE=application/x-www-form-urlencoded
GATEWAY_INTERFACE=CGI/1.1
HTTP_ACCEPT=text/html,application/xhtml+xml,...
HTTP_ACCEPT_LANGUAGE=de-DE,en-US;q=0.8,en;q=0.5,nl;q=0.3
HTTP_HOST=www.mi.hs-rm.de
PWD=/home/mi/jbiff017/public_html/cgi-bin
QUERY_STRING=
REMOTE_ADDR=217.224.160.39
REMOTE_PORT=48496
REQUEST_METHOD=POST
REQUEST_SCHEME=http
REQUEST_URI=/~jbiff017/cgi-bin/dump.cgi
SERVER_ADDR=195.72.105.32
SERVER_NAME=www.mi.hs-rm.de
SERVER_PORT=80
SERVER_PROTOCOL=HTTP/1.1
SERVER_SIGNATURE=<address>Apache/2.4.25 (Debian) ...
SERVER_SOFTWARE=Apache/2.4.25 (Debian)
```

3b Server: CGI-Skript erhält als Standardeingabe übergeben:

```
name=J%F6ndhard&alter=17&okbutton=mach+mal
```

Beispiel: Formulardaten-Codierung "multipart/form-data"

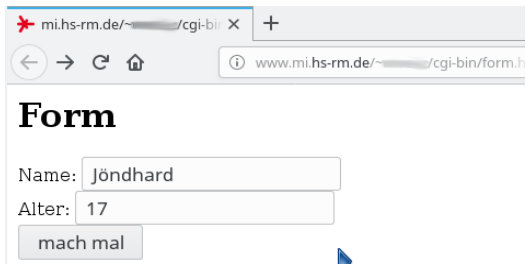
1. Client erhält HTML-Formular

```
<h1>Form</h1>

<form action="/~jbiff017/cgi-bin/dump.cgi"
  method="post" enctype="multipart/form-data">

  Name: <input type="text" name="name"/><br/>
  Alter: <input type="text" name="alter"/><br/>
  <input type="submit" name="okbutton" value="mach mal"/>
</form>
```

2. Client schickt Formular ab



The screenshot shows a web browser window with the address bar displaying 'mi.hs-rm.de/~jbiff017/cgi-bin/form.h'. The page content includes a heading 'Form' and two text input fields. The first field is labeled 'Name:' and contains the text 'Jöndhard'. The second field is labeled 'Alter:' and contains the number '17'. Below these fields is a submit button with the text 'mach mal'.

3a Server: CGI Environmentvariablen

```
CONTENT_LENGTH=418
CONTENT_TYPE=multipart/form-data; boundary=-----1096268442..
GATEWAY_INTERFACE=CGI/1.1
HTTP_ACCEPT=text/html,application/xhtml+xml,...
HTTP_ACCEPT_LANGUAGE=de-DE,en-US;q=0.8,en;q=0.5,nl;q=0.3
HTTP_HOST=www.mi.hs-rm.de
HTTP_USER_AGENT=Mozilla/5.0 (X11; Linux x86_64; rv:66.0) ...
PWD=/home/mi/jbiff017/public_html/cgi-bin
QUERY_STRING=
REMOTE_ADDR=217.224.160.39
REMOTE_PORT=48106
REQUEST_METHOD=POST
REQUEST_SCHEME=http
REQUEST_URI=/~jbiff017/cgi-bin/dump.cgi
SERVER_ADDR=195.72.105.32
SERVER_NAME=www.mi.hs-rm.de
SERVER_PORT=80
SERVER_PROTOCOL=HTTP/1.1
SERVER_SOFTWARE=Apache/2.4.25 (Debian)
```

3b Server: CGI Standardeingabe

```
-----9807423231438503121657809642
Content-Disposition: form-data; name="name"

Jöndhard
-----9807423231438503121657809642
Content-Disposition: form-data; name="alter"

17
-----9807423231438503121657809642
Content-Disposition: form-data; name="okbutton"

mach mal
-----9807423231438503121657809642--
```

Python CGI-Modul

- Erzeugung der **Rückantwort** per einfachen Ausgaben auf **Standardausgabe** des Skripts (z.B. mit `print()`)
- Ergebnis-Aufbau beachten – **mindestens Content-Type** Header, **Leerzeile** (!) und ggf. Inhalt der Antwort selbst auszugeben.
- cgi-Modul stellt Klasse `FieldStorage` zur Verfügung, die einfachen Zugriff auf **Parameter** (aus URL oder Formular) ermöglicht
- Dictionary-artige Schnittstelle `params[name]` (bei Fehler `KeyError-Exception`),
oder `params.getvalue(name [, default])` (falls `name` nicht existiert: `None` bzw. optionaler Default-Wert)
- `params.getlist(name)` liefert *immer* Liste.
- `cgitb`-Modul zur Laufzeitfehlersuche
- Auslesen von **Environment-Variablen** über `os.environ`-Objekt (dict-artige Schnittstelle)

```
#!/usr/bin/env python3
import cgi, cgitb, os
cgitb.enable() # NUR während Entwicklung

### HTTP Header ausgeben, mindestens Inhaltstyp
print("Content-Type: text/html")
# ... hier ggf weitere Antwort-Header-Felder ...
# Wichtig: Leerzeile schließt Header ab
print()

# Infos aus Environmentvariable holen
anfrager = os.environ["REMOTE_ADDR"]
httpmethode = os.environ["REQUEST_METHOD"]

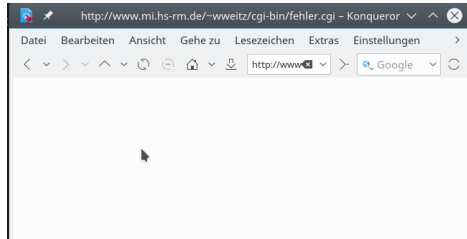
### Ab hier: Eigentlicher Antwort-Inhalt ###
print("Anfrage von IP-Adresse", anfrager)
print("mit HTTP-Methode", httpmethode)

# Empfangene Param. als Schlüssel/Wert-Paare ausg.
params = cgi.FieldStorage()
print("<ul>")
for key in sorted(params.keys()):
    print("<li>",key, params[key].value)
# print("<li>",key, params.getvalue(key, "nix"))
print("</ul>")
```

cgitb unterstützt Fehlersuche während CGI-Entwicklung

```
print("Content-Type: text/html")
print()
print(gibtsnicht)          # <-- Problembar
```

Abruf ohne cgitb.enable()

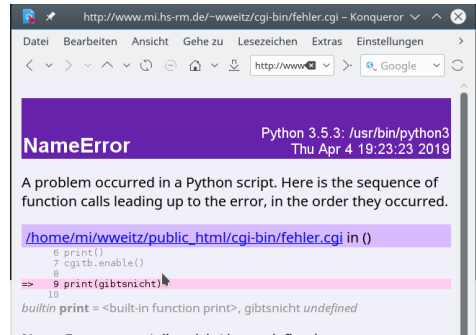


(Fehlermeldung läuft im *Webserver-Log* auf – keine Ausgabe im *Client* (Browser). cgitb hilft während Entwicklung, für Produktionsbetrieb ausschalten!)

```
import cgitb, os
cgitb.enable()
```

```
print("Content-Type: text/html")
print()
print(gibtsnicht)
```

Abruf mit cgitb.enable()



Serverseitig Session-Daten halten mit versteckten Formularfeldern

- Bei erstem Abruf mit HTTP GET gibt es keine **Session-ID** \Rightarrow neue ID erzeugen und in **verstecktes** (hidden) **Formularfeld** einbetten.
- Formular-Abschicken erfolgt per **HTTP POST** auf selbes Skript, **Session-ID** wird bei nachfolgenden POST des Formulars **durchgereicht**.
- Aber: Bei einem GET-Abruf würde wieder eine **neue** Session angelegt

```
import os, uuid, cgi, cgitb
import shelve # einfaches "persistentes Dictionary"
               # unsicher - nur zu Demo-Zwecken

cgitb.enable(logdir=os.path.expanduser("~/logs"))

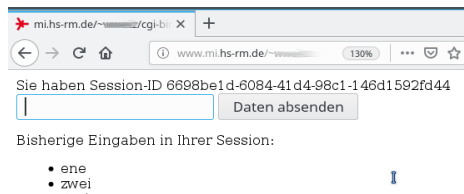
MYSESSIONKEY = "mysessionID"
DBPFAD = os.path.expanduser("~/sessiondata")

forminput = cgi.FieldStorage()
print("Content-Type: text/html")
print()

# wenn SessionID empfangen - verwenden; sonst neue erzeugen
sessionid = forminput.getvalue(MYSESSIONKEY, str(uuid.uuid4()))

with shelve.open(DBPFAD, writeback=True) as db:
    if sessionid not in db:
        db[sessionid] = []
    if "eingabe" in forminput:
        db[sessionid].append(forminput["eingabe"].value)

# SessionID ausgeben u. durchreichen in verstecktem Eingabefeld
print('Sie haben Session-ID {sval}'
      '<form action="{contextroot}" method=POST>'
      '  <input type="hidden" name="{skey}" value="{sval}" />'
      '  <input type="text" name="eingabe" />'
      '  <input type="submit" />'
      '</form>'
      'Bisherige Eingaben in Ihrer Session:'
      '<ul>'.format(skey=MYSESSIONKEY, sval=sessionid,
                    contextroot=os.environ["SCRIPT_NAME"]))
for eintrag in db[sessionid]:
    print("<li>", eintrag)
print("</ul>")
```



Serverseitig Session-Daten halten mit HTTP Cookies

- HTTP-Cookies spezifiziert in RFC6265
- **Server** schickt in HTTP-Antwort Set-Cookie:-Header mit Schlüssel/Wert-Paaren an Client
- Ggf. Beschränkungen wie Gültigkeitsdauer (Expires / Max-Age), zugehörige Domain des Servers, Path usw. möglich
- **Client merkt sich** Cookies und fügt (unter Berücksichtigung der Beschränkungen) **automatisch** Cookie:-Header mit den Werten **bei Folgeanfragen** hinzu.
- Python: http.cookies-Modul als Unterstützung
- `cookie.output()` erzeugt HTTP Set-Cookie Header, enthält hier nur Session-ID

```
from http.cookies import SimpleCookie
import os, uuid, time, shelve

MYSESSIONID = "mysessionID"
DBPFAD = os.path.expanduser("~/sessiondata")

httpcookie = os.environ.get("HTTP_COOKIE", "")
cookie = SimpleCookie(httpcookie)
if MYSESSIONID in cookie:
    # Mitgeschickte Session-ID aus Cookie holen
    sessionid = cookie[MYSESSIONID].value
else:
    # Keine Session-ID geliefert - neue erzeugen
    sessionid = str(uuid.uuid4())
    cookie[MYSESSIONID] = sessionid

with shelve.open(DBPFAD, writeback=True) as db:
    now = time.ctime(time.time())
    if sessionid not in db:
        # neue Session - DB-Eintrag initialisieren
        db[sessionid] = []

    # Abrufzeitpunkt an Liste der Session hängen
    db[sessionid].append(now)

    # Antwort zurueckschicken
    print("Content-Type: text/plain")
    print(cookie.output())
    print()
    print("ID Ihrer Sitzung aus Cookie:", sessionid)
    print("Serverseitige Liste Ihrer Abrufe:")
    for eintrag in db[sessionid]:
        print("-", eintrag)
```

CGI: Vor- und Nachteile

Das CGI ist eine einfache Schnittstelle, um einem Webserver **eigene** (externen) Code zur **dynamischen Requestverarbeitung** zugänglich zu machen.

• Vorteile

- einfach (im Sinne von “schlicht”),
- von vielen Webservern unterstützt,
- fast **jede Programmiersprache** verwendbar (geringe Anforderungen; nur Zugriff auf Environmentvariablen und Standardein-/ausgabe nötig)
- pragmatische Lösung z.B. zur Anbindung von Systemen an das Web, bei denen z.B. lediglich eine einfache Daten-Konvertierung (ohne viel Ablauflogik) nötig ist
- Kann als Grundlage für komfortablere Frameworks dienen
(z.B. für einfache “Front-Controller”-artige Komponente)

• Nachteile

- “Ein Prozess je HTTP-Request” teuer, ggf. serverseitige **Lastprobleme**
- **Vermischung** von Ablauflogik und Benutzungsoberfläche (HTML-Erzeugung im Programmcode, Skriptnamen als URLs in HTML-Seiten) führen zu **Unübersichtlichkeit**.
- Fehlersuche (ohne Unterstützung) mühsam
- Verschiedene Varianten der **Parameterübergabe** bei HTTP-Methoden GET bzw. POST sind zu berücksichtigen (z.B. “von Hand” oder durch Nutzung des Python cgi-Moduls)

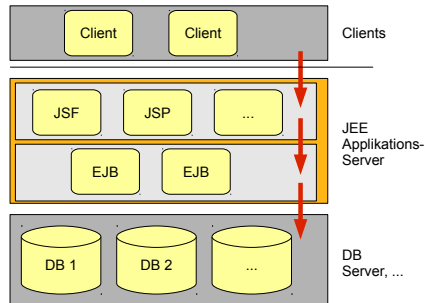


Java und das Web

Java / Jakarta Enterprise Edition (JEE) – Schichten (*tiers*)

Die JEE ist eine Menge von Teil-Spezifikationen verschiedener Bausteine

- **Client** Schicht
 - Browser, andere Anwendungen, ...
- **Web** Schicht
 - JSF (Java Server Faces) – komplexere Template-Engine, UI-Komponenten
 - JSP (Java Server Page) – “Java in HTML”
 - **Servlets** – “low level” API zur Requestbearbeitung
- **Business** Schicht (Anwendungslogik)
 - EJB Komponenten (“Enterprise Java Beans”)
 - JPA (“Java Persistence API”) *Entities*
- **EIS** (“Enterprise Information Systems”) Schicht
 - **JPA** Objekt-relationales Mapping, DB-Zugang
 - **JDBC** (kennen Sie aus der DB-Vorlesung)
 - Transaktionsmanagement, Batch-Verarbeitung, Mail, Messaging...



[Java EE Tutorial]

Servlet – Grundbaustein für Java-Webanwendungen

Ein **Servlet** ist eine Java-Klasse, die serverseitig Methoden zur Verarbeitung (insb. von HTTP-)Anfragen bereitstellt.

Vorteile gegenüber CGI:

- Das Servlet wird vom Servlet-Container **einmal geladen** und bleibt im Speicher,
- für jede Anfrage wird ein **Java-Thread** gestartet
(leichtgewichtiger Thread, nicht “teurer” Prozess wie bei CGI)
- Einfacher Zugang zur großen Auswahl an Java-APIs
- **Effizienz:** Servlets werden compiliert (Geschwindigkeit; Compilierung nur einmal)
- Plattformunabhängigkeit (Java)
- Komplexere Web-Technologien (wie JSF) basieren auf Servlets

Nachteile:

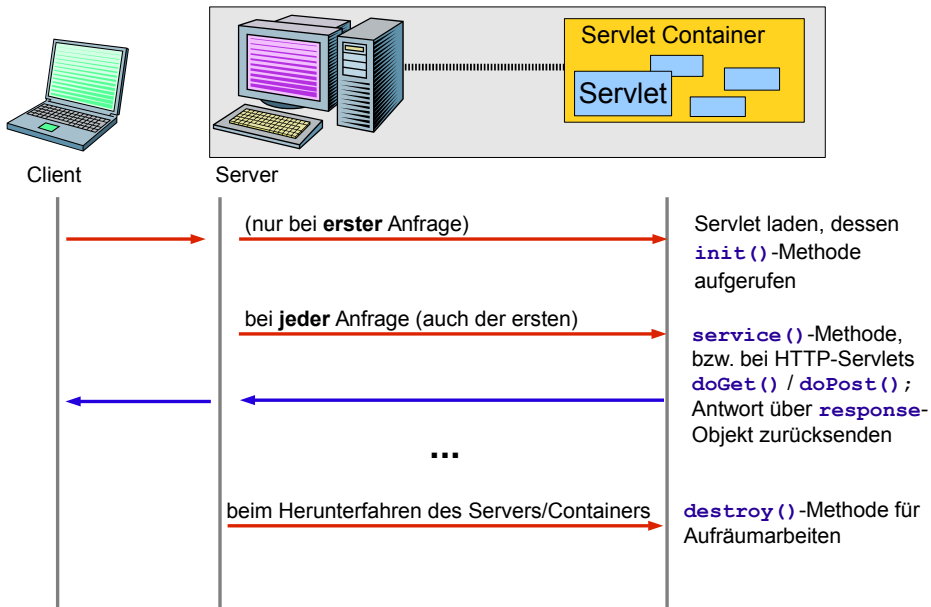
- Festlegung auf Java bzw. die JVM
- Abstraktionshöhe noch relativ “low level” (aber etwas höher als CGI)

Servlet-Container

- **Servlet Container:** Ablaufumgebung für Servlets, Lifecycle-Management
- Populäre Implementierungen: Apache Tomcat, Jetty (enthalten HTTP-Server)
- Wesentliche Dienste:
 - **(Re-)Compilieren** von JSPs/Servlets (bei Änderungen)
 - **Weiterleiten einer Anfrage** mit einer bestimmten URL (URI) an das “richtige” Servlet
 - Bereitstellung und **Übergabe** zweier Objekte an das Servlet bei jedem Aufruf:
request für Zugriff auf ein- und **response** für ausgehende Daten
 - **Session-Management** (HttpSession-Objekt, hält Schlüssel/Wert-Paare *je Session*)
- **Paketierung:** Servlet-Klasse(n) werden mit Hilfsklassen und Konfigurationsdaten in Zip-Datei gepackt (“Web ARchiv” war-Datei – vgl. jar),
- war-Datei wird in Applikationsserver geladen (**Deployment** der Anwendung)



Servlet-Lebenszyklus



Einfaches HTTP-Servlet – Beispiel

- HttpServlet als Basisklasse
- Überschreibbare doGet(), doPost(), doDelete() usw. je HTTP-Methode
- Vorimplementierte service()-Methode leitet eingehende Anfrage an "richtige" do-Methode weiter (gemäß Konfiguration)
- HttpServletRequest ermöglicht Zugriff auf Parameter, Cookies, Session etc.
- HttpServletResponse für Rückantwort (PrintWriter erlaubt print() wie bei System.out)
- Ähnlichkeiten/Unterschiede zu CGI?

```
package de.hsrmi.web;

import ...

public class SimpleServlet extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
        throws ServletException, IOException {

        PrintWriter ausgabe = response.getWriter();

        HttpSession session = request.getSession();

        String name = request.getParameter("name");
        if (name == null) {
            // kein Name übergeben - in Session nachsehen
            Object n = session.getAttribute("name");
            name = n==null? "nobody" : (String)n;
        } else {
            // Name übergeben - in Session speichern
            session.setAttribute("name", name);
        }

        ausgabe.print("<h1>Beispiel</h1>");
        ausgabe.print("Hallo " + name + ", wie geht's?");

    }
}
```

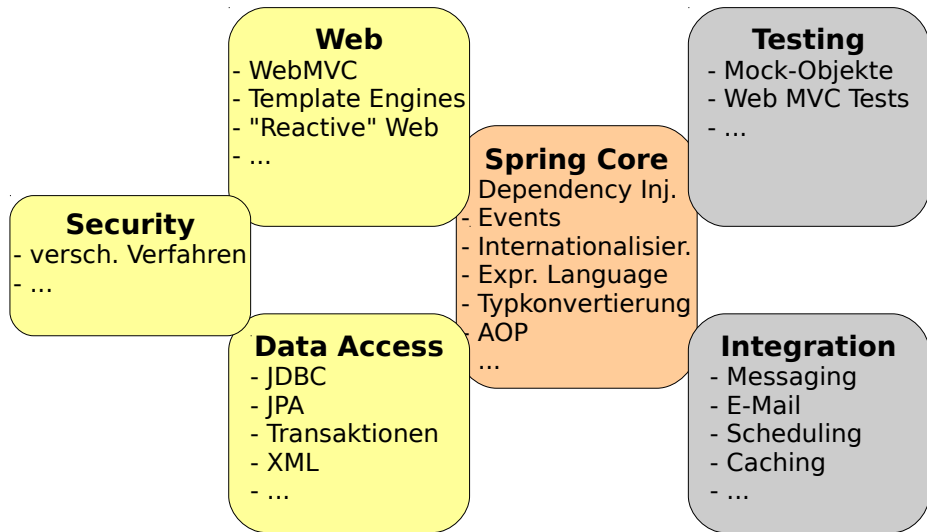


Das Spring Framework

Spring Framework

- Ausgangspunkt: Standard J2EE (Java2 Enterprise Edition) kompliziert, unhandlich
- Entwicklungsbeginn 2002 (Rod Johnson, bei Arbeit an J2EE-Buch)
- Framework insb. für Unternehmensanwendungen
- **Auswahlmöglichkeiten** auf allen Ebenen: **Integration** verschiedener bestehender Technologien auf Java-Basis (*einschließlich* JEE-Standards)
- besteht aus mehreren Teilprojekten (**Baukasten**-Prinzip)
- Homepage <http://spring.io>,
Sourcen auf <https://github.com/spring-projects/spring-framework>

Spring Framework Überblick (Auswahl)



Umfang: >640.000 Lines of Code, >10.000 Klassen (Spring Framework 5.2)

Spring Core

- Container für “Spring-Bean” Java-Klassen (Lifecycle-Management)
- POJOs (“plain old java objects”) mit sparsamen Framework-Abhängigkeiten
- Durchgängige Unterstützung für
 - **Dependency Injection** (autom. “Verdrahtung” von Softwarekomponenten)
 - Aspekt-orientierte Programmierung (AOP)
 - Kommunikation zwischen Komponenten über **Events**
 - hohe **Konfigurierbarkeit**
 - “Spring Expression Language” (SpEL)
- Konfiguration der Beans wahlweise über...
 - **Konfigurationsdateien** (XML / YAML)
(dadurch Umkonfiguration der Anwendung ohne Codeänderung möglich)
 - **Annotationen** im Code oder
 - **als Code** über Konfigurations-Klassen
- ApplicationContext (und andere Contexte)

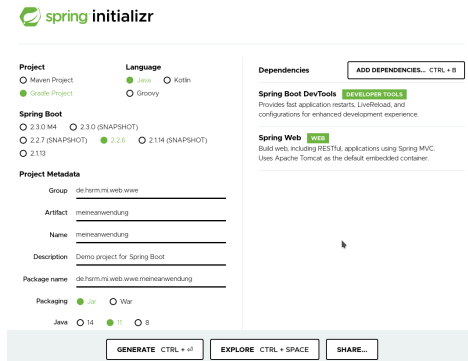
Wozu “Spring Boot”?

Spring ist ein **komplexes**, mächtiges Framework mit **vielen Komponenten** und **Variationsmöglichkeiten** – Aufsetzen eines neuen Projekts ist nicht so trivial.

Spring Boot...

- Vereinfacht **Einrichten** und Entwicklung eines Spring-Projekts incl. Verzeichnisstruktur, Build-Konfiguration, Anfangsdateien etc
- Aktivierung benötigter Teile über **Abhängigkeiten** in Build-Konfiguration (für Tools *Maven* oder *Gradle*), Anwendung kann jederzeit (ohne IDE) gebaut / gestartet werden
- Vorgefertigte “**starter**”-**Abhängigkeiten** mit “vernünftigen”, aufeinander abgestimmten Basiskonfigurationen (“opinionated”)
- Umfassende **Autokonfiguration**
- Erzeugt lauffähige **standalone-Anwendungen**, integriert ggf. Servlet-Container
- kein Deployment der Anwendung auf *separatem* Applikationsserver nötig (aber auf Wunsch möglich)

Spring Initializr



The screenshot shows the Spring Initializr web form. It is divided into several sections: Project, Language, Spring Boot, Project Metadata, Dependencies, and a bottom bar with action buttons. The 'Project' section has radio buttons for 'Maven Project' and 'Gradle Project', with 'Gradle Project' selected. The 'Language' section has radio buttons for 'Java', 'Kotlin', and 'Groovy', with 'Java' selected. The 'Spring Boot' section has radio buttons for versions 2.3.0.M4, 2.3.0 (SNAPSHOT), 2.2.7 (SNAPSHOT), 2.2.6 (selected), and 2.1.14 (SNAPSHOT). The 'Project Metadata' section includes fields for Group (de.harm.mi.web.www), Artifact (meineanwendung), Name (meineanwendung), Description (Demo project for Spring Boot), and Package name (de.harm.mi.web.www.meineanwendung). The 'Packaging' section has radio buttons for 'jar' (selected) and 'War'. The 'Dependencies' section has a button 'ADD DEPENDENCIES... CTRL + B'. The bottom bar has buttons 'GENERATE CTRL + G', 'EXPLORE CTRL + SPACE', and 'SHARE...'. A URL 'https://start.spring.io' is visible on the right side.

- Web-basierter Generator für Spring Boot Projekte
- <https://start.spring.io/>
- Build-Tool, Java-Version, **Spring Boot Starter** (“Dependencies”) auswählbar, “Generate Project” erzeugt Download eines zip-Archivs.
- Während Entwicklung stets **Developer Tools** (DevTools) einbinden
(für automatischen Restart der Anwendung bei Codeänderung, Exception-Ausgabe im Browser usw)

Gradle Build-Tool

- **Konventions-basierter** Ansatz

(wie bei älterem Alternativtool Maven, teilweise von Maven übernommen)

- **Dependency-Management** (kann u.a. Maven-Infrastruktur nutzen)

(automatisches Herunterladen und Einbinden benötigter Bibliotheken/Jars)

- In der JVM-basierten Skriptsprache *Groovy* geschrieben.
- Build-Skripte sind tatsächlich Groovy-Programme, daher
- freie **Skripting**-Möglichkeiten im Build-Skript.
- Effizientere Build-Strategie für große Builds.
- IDE-Unterstützung, aber vollständig **ohne IDE nutzbar**.
- Unterstützt auch nicht-JVM-Sprachen wie C/C++.
- *Das* von Google ausgewählte Build-System für Android und *das* Buildtool, das für das *spring framework* selbst verwendet wird.

Mehr unter <http://gradle.org/>

Von Initializr generierte build.gradle Konfiguration

```
plugins {  
    id 'org.springframework.boot' version '2.6.6'  
    id 'io.spring.dependency-management' version '1.0.11.RELEASE'  
    id 'java'  
}  
  
group = 'de.hsrmi.mi.web'  
version = '0.0.1-SNAPSHOT'  
sourceCompatibility = '17'  
  
repositories {  
    mavenCentral()  
}  
  
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    developmentOnly 'org.springframework.boot:spring-boot-devtools'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
}  
  
tasks.named('test') {  
    useJUnitPlatform()  
}
```

Einfach Kommandos:

- ./gradlew clean
- ./gradlew build
- ./gradlew bootRun
- ./gradlew test
- ./gradlew javadoc

... komplette Liste mit ./gradlew tasks

Verzeichnisstruktur von Maven übernommen:

src/main/java – Anwendungscode

src/test/java – (Unit-)Tests

Ergebnisdateien in build-Verzeichnis

Auswirkung der spring-boot-starter-web Dependency

- ./gradlew dependencies zeigt Build-Abhängigkeiten:
- Die (wenigen) **Starter**-Dependencies ziehen für diesen Projekttyp **automatisch alle benötigte Abhängigkeiten** (jar-Bibliotheken) nach,
- Automatischer **Download** von jar-Files aus Internet-Repository (z.B. “Maven Central”), falls noch nicht lokal vorhanden.
- Versionen **aufeinander abgestimmt**
- “opinionated” – von den vielen möglichen Zusammenstellungs-Varianten wird eine **sinnvolle** ausgewählt (z.B. eingebetteten tomcat als Servlet Container für Web-Projekt)
- Zusammenstellung kann beliebig umkonfiguriert werden.

```

...
compileClasspath - Compile classpath for source set 'main'.
\--- org.springframework.boot:spring-boot-starter-web:2.1.4.RELEASE
|    +- org.springframework.boot:spring-boot-starter:2.1.4.RELEASE
|    |    +- org.springframework.boot:spring-boot:2.1.4.RELEASE
|    |    |    +- org.springframework:spring-core:5.1.6.RELEASE
|    |    |    |    \--- org.springframework:spring-context:5.1.6.RELEASE
|    |    |    +- org.springframework:spring-aop:5.1.6.RELEASE
|    |    |    |    +- org.springframework:spring-beans:5.1.6.RELEASE
|    |    |    |    |    \--- org.springframework:spring-core:5.1.6.RELEASE (*)
|    |    |    |    \--- org.springframework:spring-core:5.1.6.RELEASE (*)
|    |    |    +- org.springframework:spring-core:5.1.6.RELEASE (*)
|    |    |    \--- org.springframework:spring-expression:5.1.6.RELEASE (*)
|    |    \--- org.springframework:spring-core:5.1.6.RELEASE (*)
|    +- org.springframework.boot:spring-boot-autoconfigure:2.1.4.RELEASE
|    |    \--- org.springframework.boot:spring-boot:2.1.4.RELEASE (*)
|    +- org.springframework.boot:spring-boot-starter-logging:2.1.4.RELEASE
|    |    +- ch.qos.logback:logback-classic:1.2.3
|    |    |    +- ch.qos.logback:logback-core:1.2.3
|    |    |    |    \--- org.slf4j:slf4j-api:1.7.25 -> 1.7.26
|    |    |    +- org.apache.logging.log4j:log4j-to-slf4j:2.11.2
|    |    |    |    +- org.slf4j:slf4j-api:1.7.25 -> 1.7.26
|    |    |    |    \--- org.apache.logging.log4j:log4j-api:2.11.2
|    |    |    \--- org.slf4j:jul-to-slf4j:1.7.26
|    |    |        \--- org.slf4j:slf4j-api:1.7.26
|    |    \--- javax.annotation:javax.annotation-api:1.3.2
|    \--- org.springframework:spring-core:5.1.6.RELEASE (*)
+- org.springframework.boot:spring-boot-starter-jackson:2.1.4.RELEASE
|    +- org.springframework.boot:spring-boot-starter:2.1.4.RELEASE (*)
|    +- org.springframework:spring-web:5.1.6.RELEASE
|    |    +- org.springframework:spring-beans:5.1.6.RELEASE (*)
|    |    \--- org.springframework:spring-core:5.1.6.RELEASE (*)
|    +- com.fasterxml.jackson.core:jackson-databind:2.9.8
|    |    +- com.fasterxml.jackson.core:jackson-annotations:2.9.0
|    |    |    \--- com.fasterxml.jackson.core:jackson-core:2.9.8
|    |    +- com.fasterxml.jackson.datatype:jackson-datatype-jdk8:2.9.8
|    |    |    \--- com.fasterxml.jackson.core:jackson-core:2.9.8
|    |    \--- com.fasterxml.jackson.core:jackson-databind:2.9.8 (*)
|    +- com.fasterxml.jackson.datatype:jackson-datatype-jdk8:2.9.8
|    |    \--- com.fasterxml.jackson.core:jackson-annotations:2.9.0
|    +- com.fasterxml.jackson.core:jackson-annotations:2.9.0
|    |    \--- com.fasterxml.jackson.core:jackson-core:2.9.8
|    \--- com.fasterxml.jackson.module:jackson-module-parameter-names:2.9.8
|        +- com.fasterxml.jackson.core:jackson-core:2.9.8
|        \--- com.fasterxml.jackson.core:jackson-databind:2.9.8 (*)
+- org.springframework.boot:spring-boot-starter-tomcat:2.1.4.RELEASE
|    +- javax.annotation:javax.annotation-api:1.3.2
|    +- org.apache.tomcat.embed:tomcat-embed-core:9.0.17
|    \--- org.apache.tomcat.embed:tomcat-embed-websocket:9.0.17

```

Einstiegsklasse – @SpringBootApplication

```
package de.hsrmi.mi.web;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MeineWebanwendung {

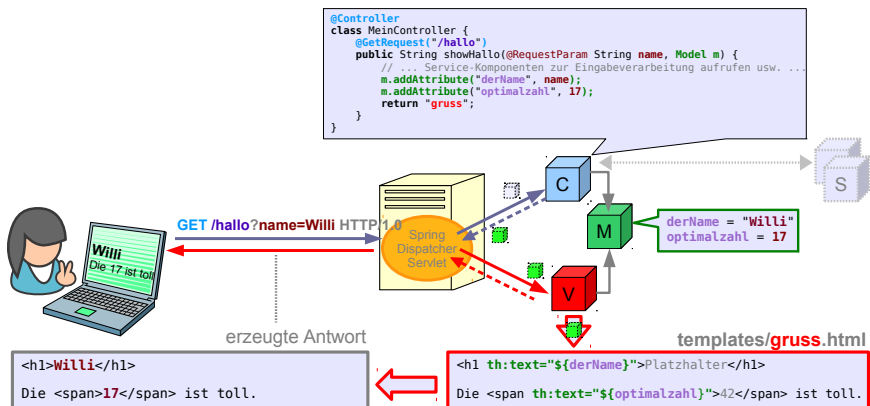
    public static void main(String[] args) {
        SpringApplication.run(MeinServiceApplication.class, args);
    }
}
```

- Einstiegs-Klasse mit `main()`-Methode ist mit `@SpringBootApplication` annotiert.
- `SpringApplication.run()` startet Anwendung,
- Spring Boot sucht Classpath nach (z.B. passend annotierten) Komponenten ab (standardmäßig im Package der Hauptklasse und dessen Unterpackages)
- initialisiert und verbindet sie (*dependency injection*) und fährt Anwendung hoch,
- in diesem Fall einschließlich tomcat Servletcontainer (mit HTTP-Server)



Spring Web MVC

Spring Web MVC Requestverarbeitung (Skizze)



- Zentrales **Spring Dispatcher Servlet** empfängt und zerlegt HTTP-Anfrage, sucht im zugehörigen **WebApplicationContext**-Objekt konfigurierten passenden Controller
- **Controller** verarbeitet Eingabe, füllt **Model**-Attribute mit Ergebnisdaten und gibt z.B. **View-Namen** zurück
- Spring Servlet **sucht** dazu mit **WebApplicationContext** konfigurierte **View**-Komponente
- **View** rendert übergebenes Model, Servlet liefert Ergebnis als **HTTP-Antwort** zurück.

Ein einfacher @Controller zur Verarbeitung von Web-Requests

- @Controller-Annotation markiert Klasse als (Web-)Controller-Komponente
- Wird bei *component scan* bei Start der Anwendung gefunden und eingebunden.
- @RequestMapping auf **Klassen-Ebene** legt URL-Basispfad fest, für den der Controller zuständig sein soll.
- Auf **Methoden-Ebene** lieber spezialisierte Sub-Annotationen verwenden: @GetMapping bildet HTTP GET-Anfragen ab, analog @PostMapping, @DeleteMapping usw.
- Handlermethode kann optional Model-Parameter deklarieren und mit Schlüssel/Wert-Paaren füllen,
- liefert View-**Namen** als String-Rückgabe.
- Gewählte View hat Zugriff auf Model-Objekt.

```
package de.hsrm.mi.web;
...
@Controller
@RequestMapping("/leute")
public class GrussKlasse {

    // http://localhost:8080/leute/verabschieden
    @GetMapping("/verabschieden")
    public String tschuess(Model m) {
        ...
        m.addAttribute("ausruf", "Oh Jammer!");
        m.addAttribute("abstand", 2);
        ...
        return "verabschiedung"; // View-Name
    }
}
```

Request-Parameter in Handler-Methoden auswerten

- `@RequestParam("name")` für benannte Parameter (Query-Parameter oder Formulardaten)
- Falls Variablenname von Requestparameternamen abweicht, muss er explizit angegeben werden.
- Eingehende Daten sind technisch Strings – Spring konvertiert automatisch auf deklarierten Parametertyp (z.B. `int`)
- “Beliebig viele” Request-Parameter möglich.

```
@Controller
@RequestMapping("/leute")
public class GrussKlasse {
    ...
    // GET auf http://localhost:8080/leute/gruss
    @GetMapping("/gruss")
    public String gruss_get() {
        ...
        return "grussview";
    }
    ...
    // POST auf http://localhost:8080/leute/gruss
    @PostMapping("/gruss")
    public String gruss_post(Model m,
                             @RequestParam String name,
                             @RequestParam("age") int alter) {

        String a = alter > 17? "alter" : "junger";
        String g = "Hallo, " + a + " " + name + "!";

        m.addAttribute("ausgabe", g);
        return "grussview"; // View-Name
    }
    ...
}
```

View: z.B. HTML-Templates mit Thymeleaf

- Ziel: “natural templates” – als “normales HTML” im Browser guckbar
- Unterstützt **Ausdrücke**, z.B. `${...}` mit Spring Expression Language
- In HTML-Templates deklarieren:
`xmlns:th="http://www.thymeleaf.org"`
- Thymeleaf-Jars einbinden, dazu in `build.gradle` Dependency:
`org.springframework.boot:spring-boot-starter-thymeleaf`
- Templates liegen im Projekt unter `/resources/templates/`
- Templatename aus View-**Namen** (=Controller-Rückgabe) abgeleitet.

HalloController.java

```
@GetMapping("/hallo")
public String showHallo(Model m) {
    m.addAttribute("ueberschrift", "Hallo Leute");
    m.addAttribute("name", "Joghurta");
    return "hallo"; } ...
```

/resources/templates/hallo.html

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
    ...
    <body>
        <h1 th:text="${ueberschrift}">Ein Titel</h1>
        Endlich, <span th:text="${name}">Dummy</span> ist da.
    </body>
</html>
```

Ausgabe

Hallo Leute

Endlich, Joghurta ist da.