Hochschule **Rosenheim**
University of Applied Sciences

Verteilte Verarbeitung

Kapitel 11

HTTP-Protokoll

HTTP Protokoll

- ***HTTP = HyperText Transfer Protocol***
- = Protokoll zum Zugriff auf Dokumente auf anderen Rechnern
  - Auf der Grundlage von TCP, Port 80 oder 8080
  - Request --> Response (Server ist passiv)
  - ***Zustandslos***
- Nachrichten (Request und Reply)
  - **`Message = Header <cr> <lf> <cr> <lf> Body`**
  - **`Header`** = Request / Statuszeile + HTTP headers
  - **`Body`** = Irgendwelche Informationen (Bytes)
- Aktuell: Praktisch nur noch HTTPS
  - = HTTP über TLS 1.2 oder TLS 1.3
  - SSL 3.0 lange schon nicht mehr aktuell
  - Port: 443

HTTP – Protokoll Versionen

- 1989 begann Tim Berners Lee mit der Entwicklung
  - am CERN in der Schweiz
  - Zusammen mit HTML und URL
- HTTP 0.9 (aus 1991)
- HTTP 1.0: RFC 1945 (aus 1996)
  - http://www.ietf.org/rfc/rfc1945.txt
- HTTP 1.1: RFC 2616 (aus 1999)
  - http://www.w3.org/Protocols/rfc2616/rfc2616.html)
- HTTP 2.0: RFC 7540 (aus 2015)
  - aktuell: Optimiertes HTTP 1.1
  - https://tools.ietf.org/html/rfc7540
- HTTP 3.0 (In Arbeit)

Beispiel Hochschule Rosenheim

(über Developer Tools von Chrome)

Request → Response

Request Headers view parsed

```
GET / HTTP/1.1
Host: www.th-rosenheim.de
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit
i/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,ima
nge;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
```

Response Headers view parsed

```
HTTP/1.1 200 OK
Date: Mon, 01 Jun 2020 13:13:04 GMT
Server: Apache
Last-Modified: Mon, 01 Jun 2020 12:40:01 GMT
ETag: "8f25-5a70517e74b71-gzip"
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Encoding: gzip
Cache-Control: max-age=1616
Expires: Mon, 01 Jun 2020 13:40:01 GMT
Content-Length: 7504
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
```

HTTP Request-Typen (Verben)

- HTTP bietet Request-Typen zum Anlegen, Ändern und Löschen von Daten …
  - **`GET`** Laden von Daten, Query
  - **`PUT`** Anlegen und Ändern von Daten
  - **`POST`** Anlegen von Unterknoten
  - **`DELETE`** Löschen von Daten
- Weitere Verben sind: **`OPTIONS, HEAD, TRACE, CONNECT`**
- Daten werden jeweils durch URI identifiziert

HTTP 1.1 Status Codes

- HTTP hat Statuscodes, in jedem Response:

  1xx: Informell

  2xx: Erfolgreicher Aufruf

  3xx: Redirection / Umleitung

  4xx: Fehler auf Client-Seite

  5xx: Interne Server Fehler

  …

- Siehe dazu:
  http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html#sec6

HTTP: Status Codes

- ***2xx: Erfolgreicher Aufruf***
  - 200: OK alles prima, häufigste Antwort
  - 201: CREATED neue Ressource angelegt
  - 202: ACCEPTED Verarbeitung läuft noch, z.B. Videoupload
- ***4xx: Problem am Client***
  - 400: BAD REQUEST, Syntaxfehler im Request
  - 401: UNAUTHORIZED, Authentifikation notwendig, z.B. Login
  - 403: FORBIDDEN, Server hat Anfrage verstanden, Login notw.
  - 404: NOT FOUND, Ressource nicht gefunden, ggf. später
  - 406: INCOMPATIBLE, inkompatibler http-Header
  - 409: CONFLICT

Parameter in HTTP

- URL Parameter: `http://example.com/such/heititei/1`
- Query Parameter in URL (Form-Data)
  - = Einfache Name / Wert – Paare

  `http://example.com/such?vorname=gerd&nachname=beneken`
- Header Parameter
  - Darüber Autorisierung (Tokens, Basic Auth, OAuth)
    - **`Authorization`**
      - Basic (Username und Passwort Base64 codiert)
      - Bearer (z.B. über Oauth bereitgestelltes Token)
  - Darüber Content-Negotiation
    - **`Accept`**
    - **`Content-Type`**
- Parameter im Body z.B. anzulegende Daten

Ressourcen

Ressourcen

- Haben eine eindeutige Identifikation (URI)
  - = ***Fachliche Identität***
  - Primärressource = Zentraler, Autonomer Bestandteil der Anwendung
- können verknüpft werden (Hypermedia-Links)
- Können mehrere Darstellungen haben (XML, Text, JSON,…), Sie kennen JSON
- Beispiele:
  - HTML-Seiten
  - Download-Inhalte
  - Entitäten: Kunde, Konto, Vertrag
  - Listen: Kundenliste, Stornierungen, …
  - „Künstliche“ Ressourcen: Projektionen / Aggregationen

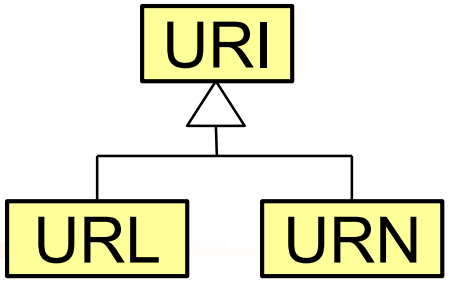
Eindeutige Identifikation über die URI

- Jede Ressource hat eine URI
- URI = Uniform Ressource Identifier
  - = Globaler Namensraum für Ressourcen
  - Schema zur Vergabe von IDs im Web
- URI enthält
  - Protokoll: **`https`**
  - Quelle: **`example.com`**
  - Was genau: Bestellung(**`orders`**) mit Id **`/2007/10/776654`**
- Beispiele

```
https://example.com/customers/1234
https://example.com/orders/2007/10/776654
https://example.com/products/4554
https://example.com/processes/salary-increase-234
```

*) Quelle Tilkov: Rest und HTTP, dpunkt, 2009

URI, URL und URN



- URI = Universal Ressource Identifier
  - ***URL = Uniform Ressource Locator***
  - URN = Uniform Ressource Name
- Schema für URLs:

```
http://example.com:8042/helloworld/there?name=frag#13
```

Schema Authority Pfad Query Fragment

- Beispiele für URI

```
ftp://ftp.is.co.za/rfc/rfc1808.txt
http://rudi:password@www.ietf.org/rfc/rfc2396.txt
ldap://[2001:db8::7]/c=GB?objectClass?one
mailto:John.Doe@example.com
news:comp.infosystems.www.servers.unix
tel:+1-816-555-1212
telnet://192.0.2.16:80/
urn:oasis:names:specification:docbook:dtd:xml:4.1.2
```

Ressourcen über Verweise verbinden

- Verweise stellen Assoziation / Komposition / Aggregation dar
- Beispiel

```
<order self='http://example.com/customers/1234' >
   <amount>23</amount>
   <product ref='http://example.com/products/4554' />
   <customer ref='http://example.com/customers/1234' />
   <link rel='cancel' ref='./cancellations' />
</order>
```

- Vorteil der Verweise:
  - anwendungsübergreifend (Integration)
  - unternehmensübergreifend
  - können auch Aktionen beinhalten (wie LINK)
- Vgl. z.B. Vorteile Namensschema bei Amazon
  - Verweise per E-Mail versendbar, auf Todo-Liste kopierbar

Ressourcen können verschiedene Darstellungen haben

- HTTP spezifiziert Datenformat über Media Types (MIME)
  - Im ***Header*** spezifiziert über ***`Accept`*** und ***`Content-Type`***
  - MIME = Multipurpose Internet Mail Extensions
- HTTP-GET-Request: applikationsspezifisches XML

```
GET /customers/1234
HOST www.example.com
Accept: application/xml
```

→

```
<customer>
...
</customer>
```

- HTTP-GET-Request: html-Format

```
GET /customers/1234
HOST www.example.com
Accept: text/html
```

→

```
<html>
   <body> Customer ...
   </body>
</html>
```

- ***Wichtig: Trennung Identität (URI) von Repräsentation (MIME)***
- Spezifizierte Formate sind unter anderem: csv, json, xml, html, .. Siehe: http://www.iana.org/assignments/media-types/
- Details unter dem Stichwort „Content Negotiation“

Werkzeuge für HTTP - Requests

cURL: HTTP auf der Kommandozeile

Quelle: cURL (http://curl.haxx.se/)

Usage: curl [options...] <url>

- **`-X <command>`**

  Command = **`GET`** | **`PUT`** | **`POST`** | **`DELETE`** | **`OPTIONS`** | **`HEAD`**

- **`-i`**

  Anzeigen des HTTP-Headers

- **`-H <header>`**

  Optionen für den Header für den Server
  z.B. **`-H "Content-Type: application/xml"`**

- **`-d <data>`**

  HTTP PUT/POST Daten mit dem Content-Type application/x-www-form-urlencoded

cURL Beispiele

```
curl -XGET -i http://localhost:9998/helloworld
curl -XHEAD -i http://www.fh-rosenheim.de

curl -XPOST -i -H "Content-Type:application/xml"
   -d "<participant
   xmlns=\"http://doodle.com/xsd1\">
   <name>Paule</name><preferences>
   <option>1</option></preferences>
   </participant>" http://doodle-test.com/
   api1WithoutAccessControl/polls/
   ne3sw3edkh8y4x2x/participants
```

Datentransfer im „Lieblingsformat“

- Kunden anlegen im XML Format

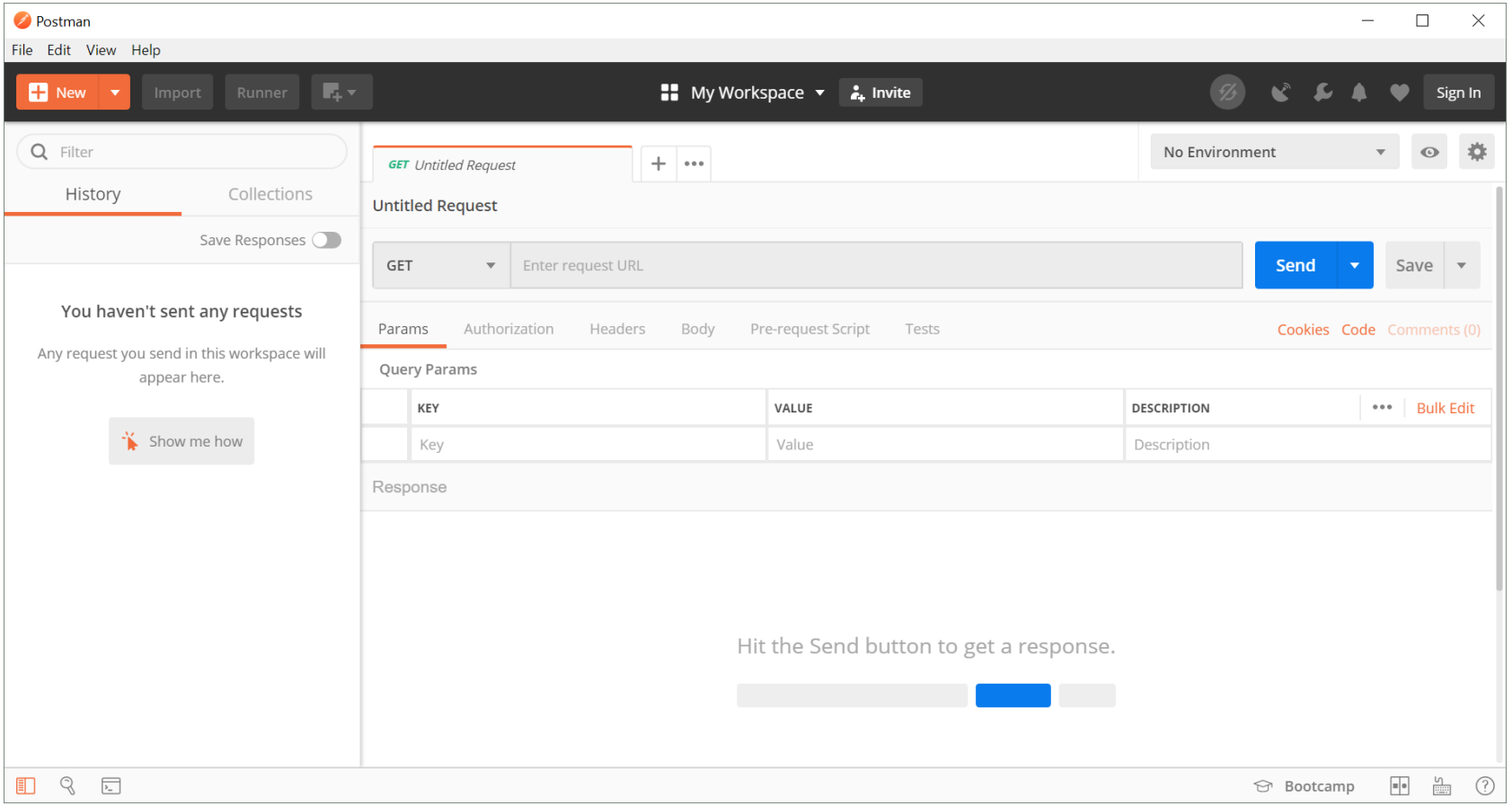
```
curl -XPOST -H "Content-Type:application/xml" -i
  -d "<kunde><nummer>4710</nummer><name>Egon
Balder</name><adresse><ort>Hamburg</ort><plz>23084</plz>
<strasse>Alsterstr. 47</strasse></adresse></kunde>„
http://localhost:9998/kunden
```

- Kunden anlegen im JSON Format

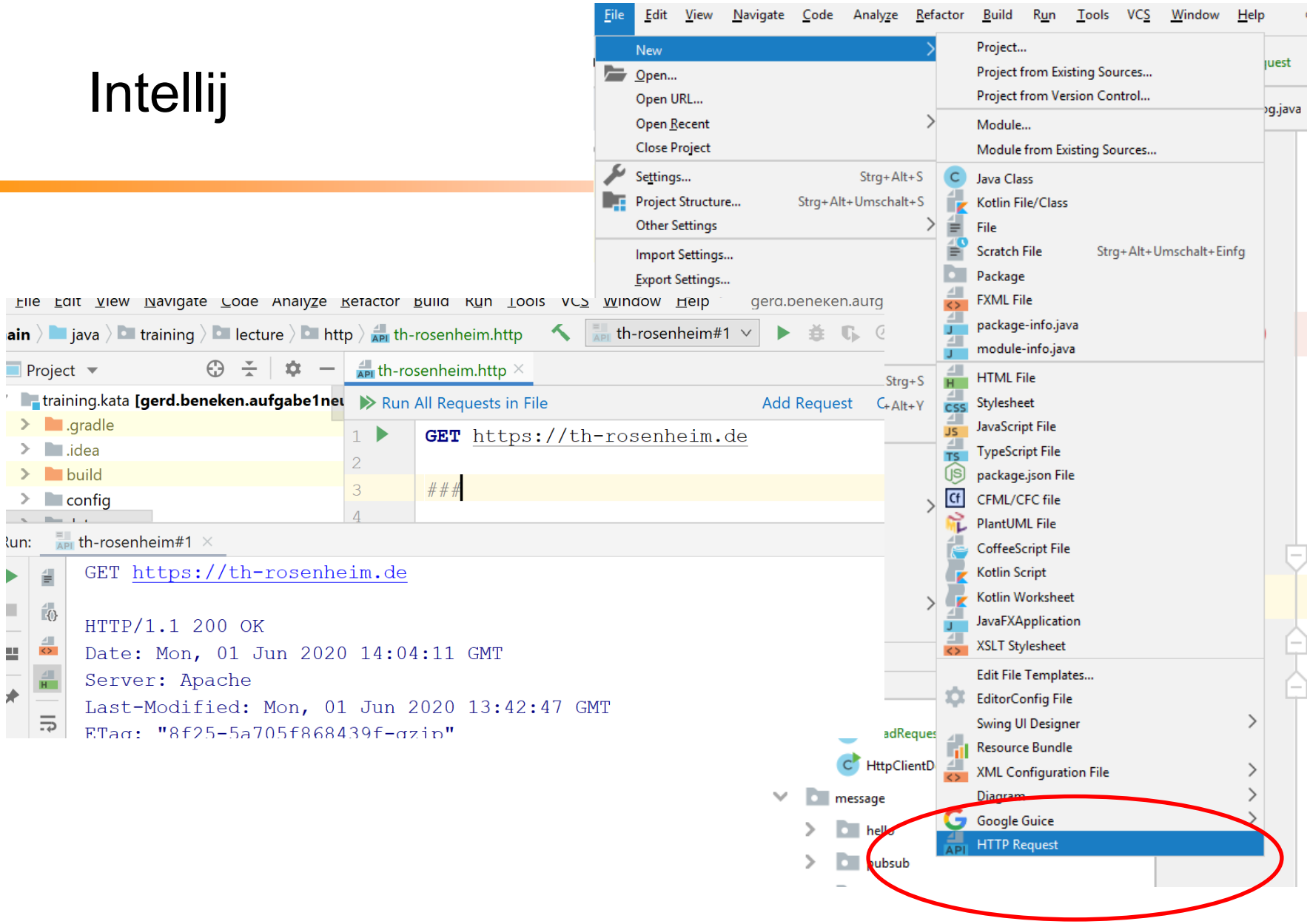
```
curl -XPOST -H "Content-Type:application/json" -i
  -d "{\"nummer\":\"4712\",\"name\":\"Harry Hirsch\",
     \"adresse\":{\"ort\":\"Guetersloh\",\"plz\":\"23456\",
     \"strasse\":\"Tiefschulstr.2\"}}"
  http://localhost:9998/kunden
```

Standardwerkzeug: Postman

https://www.postman.com/



Intellij


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
New
Open...
Open URL...
Open Recent
Close Project
Settings... Strg+Alt+S
Project Structure... Strg+Alt+Umschalt+S
Other Settings
Import Settings...
Export Settings...
Project...
Project from Existing Sources...
Project from Version Control...
Module...
Module from Existing Sources...
Java Class
Kotlin File/Class
File
Scratch File Strg+Alt+Umschalt+Einfg
Package
FXML File
package-info.java
module-info.java
HTML File
Stylesheet
JavaScript File
TypeScript File
package.json File
CFML/CFC file
PlantUML File
CoffeeScript File
Kotlin Script
Kotlin Worksheet
JavaFXApplication
XSLT Stylesheet
Edit File Templates...
EditorConfig File
Swing UI Designer
Resource Bundle
XML Configuration File
Diagram
Google Guice
HTTP Request
ain java training lecture http th-rosenheim.http
th-rosenheim#1
Project
th-rosenheim.http
training.kata [gerd.beneken.aufgabe1neu
.gradle
.idea
build
config
Run All Requests in File
Add Request
1 GET https://th-rosenheim.de
2
3 ###
4
Run: th-rosenheim#1
GET https://th-rosenheim.de
HTTP/1.1 200 OK
Date: Mon, 01 Jun 2020 14:04:11 GMT
Server: Apache
Last-Modified: Mon, 01 Jun 2020 13:42:47 GMT
ETag: "8f25-5a705f868439f-gzip"
message
hello
pubsub