



SoSe 2019

Dynamische Webanwendungen und IOT (2. Hörsaalübung)

Prof. Dr.-Ing. Uwe Rüppel
Meiling Shi, M.Sc.

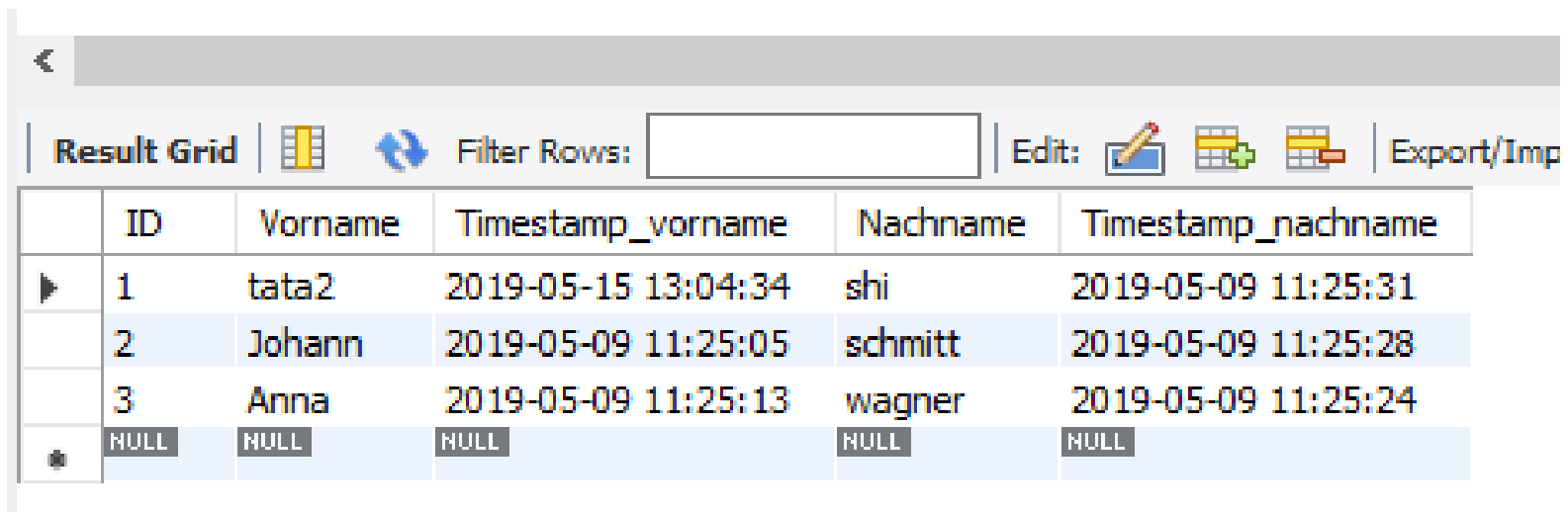
Inhalte der Hörsaalübungen zur 1. Blockübung

- 1. Hörsaalübung
 - Entwurf Relationaler Datenbanken
 - Einführung in Python
 - Datenbankschnittstelle der Python-Plattform. Erzeugen, Füllen, Manipulieren und Abfragen von Datenbanken mit MySQL-Python Connector
- 2. Hörsaalübung
 - Mehrbenutzerbetrieb
 - Grundstruktur und Aufbau einer www-Seite (HTML & CSS)
 - Erzeugen von dynamischen Inhalt für die www-Seite (Flask)
 - Arduino IDE
 - Sensoren und ESP8266
 - Sensordaten übers Netzwerk in den Server übertragen

- **Python** (ab Version 3.6.7) installieren
 - <https://www.python.org/downloads/>
 - Installationsanweisung: <https://de.wikihow.com/Python-installieren>
(Customize: default)
 - IDE
 - PyCharm (Student Version: <https://www.jetbrains.com/student/>)
 - Jupyter Notebook (<https://jupyter.org/install> Install with pip)
- **MySQL Server** (aktuelle Version 8.0.16)
 - <http://dev.mysql.com/downloads/mysql/>
- **Arduino IDE** (Version 1.8.9)
 - <https://www.arduino.cc/en/Main/Software>

MySQL Tabelle „multiusingbesp“

- Zusätzliche Spalte „Timestamp“



	ID	Vorname	Timestamp_vorname	Nachname	Timestamp_nachname
▶	1	tata2	2019-05-15 13:04:34	shi	2019-05-09 11:25:31
	2	Johann	2019-05-09 11:25:05	schmitt	2019-05-09 11:25:28
	3	Anna	2019-05-09 11:25:13	wagner	2019-05-09 11:25:24
✱	NULL	NULL	NULL	NULL	NULL

Mehrbenutzerbetrieb (2)



```
import mysql.connector
from mysql.connector import Error
import datetime

def multiusingSelect(database = "multiusingbesp", mySQLconnection = None):
    try:
        timestamp1 = datetime.datetime.now()
        cursor = mySQLconnection.cursor()
        sql_select_Query = "select * from " + database
        cursor.execute(sql_select_Query)
        records = cursor.fetchall()
        print ("Printing each row's column values i.e. developer record")
        for row in records:
            print("Id = ", row[0] )
            print("Vorname = ", row[1])
            print("Nachname = ", row[3], "\n")
        cursor.close()
    except Error as e :
        print ("Error while connecting to MySQL", e)
    finally:
        #closing database connection.
        if(mySQLconnection .is_connected()):
            connection.close()
            print("MySQL connection is closed")
        return timestamp1
```

Mehrbenutzerbetrieb (3)



```
def multiusingUpdate(ID, connection, timestampSel, value_new = "Meliing"):
    try:

        cursor = connection.cursor(prepared=True)
        sql_insert_query = """UPDATE multiusingbesp SET
                               Vorname = %s WHERE ID = %s""" # mehrzeilige
        insert_tuple = ( value_new, ID)
        result = cursor.execute(sql_insert_query, insert_tuple)
        cursor.execute("select Timestamp_vorname from multiusingbesp where ID = %s", (ID,))
        row_time_old = cursor.fetchone()
        print ("last edited at: ", row_time_old[0])
        if (timestampSel > row_time_old[0]):
            timestamp2 = datetime.datetime.now()
            updatetimestamp = """UPDATE multiusingbesp SET
                               Timestamp_vorname = %s WHERE ID = %s""" # mehrzeilige
            cursor.execute(updatetimestamp, (timestamp2, ID))
            connection.commit()
            print ("Record inserted successfully into python users table at time " , timestamp2)
        else:
            print ("Database is being edited by others, please try again")
    except mysql.connector.Error as error :
        connection.rollback()
        print("Failed to insert into MySQL table {}".format(error))
    finally:
        #closing database connection.
        if(connection.is_connected()):
            cursor.close()
            connection.close()
            print("MySQL connection is closed")
```

Wenn der Zeitstempel des Sichten
später als die letzte Änderung ist

Beispiel



```
timestampSelect2 = multiusingSelect("multiusingbesp",connecting())
```

Printing each row's column values i.e. developer record

Id = 1

Vorname = tata1

Nachname = shi

Id = 2

Vorname = Johann

Nachname = schmitt

Id = 3

Vorname = Anna

Nachname = wagner

MySQL connection is closed

Nutzer 2 sieht DB und bekommt Zeitstempel
„timestampSelect2“

```
timestampSelect1 = multiusingSelect("multiusingbesp",connecting())
```

Printing each row's column values i.e. developer record

Id = 1

Vorname = tata1

Nachname = shi

Id = 2

Vorname = Johann

Nachname = schmitt

Id = 3

Vorname = Anna

Nachname = wagner

MySQL connection is closed

Nutzer 1 sieht DB und
bekommt Zeitstempel
„timestampSelect1“

Nutzer 1 ändert DB

```
multiusingUpdate (1,connecting(), timestampSelect1, value_new = "tata2")
```

```
last edited at: 2019-05-09 11:39:33
```

```
Record inserted successfully into python_users table at time 2019-05-09 19:24:11.4!
```

```
MySQL connection is closed
```

Nutzer 2 ändert DB, geht nicht, da DB nach dem Sichten von Nutzer 1 geändert wurde

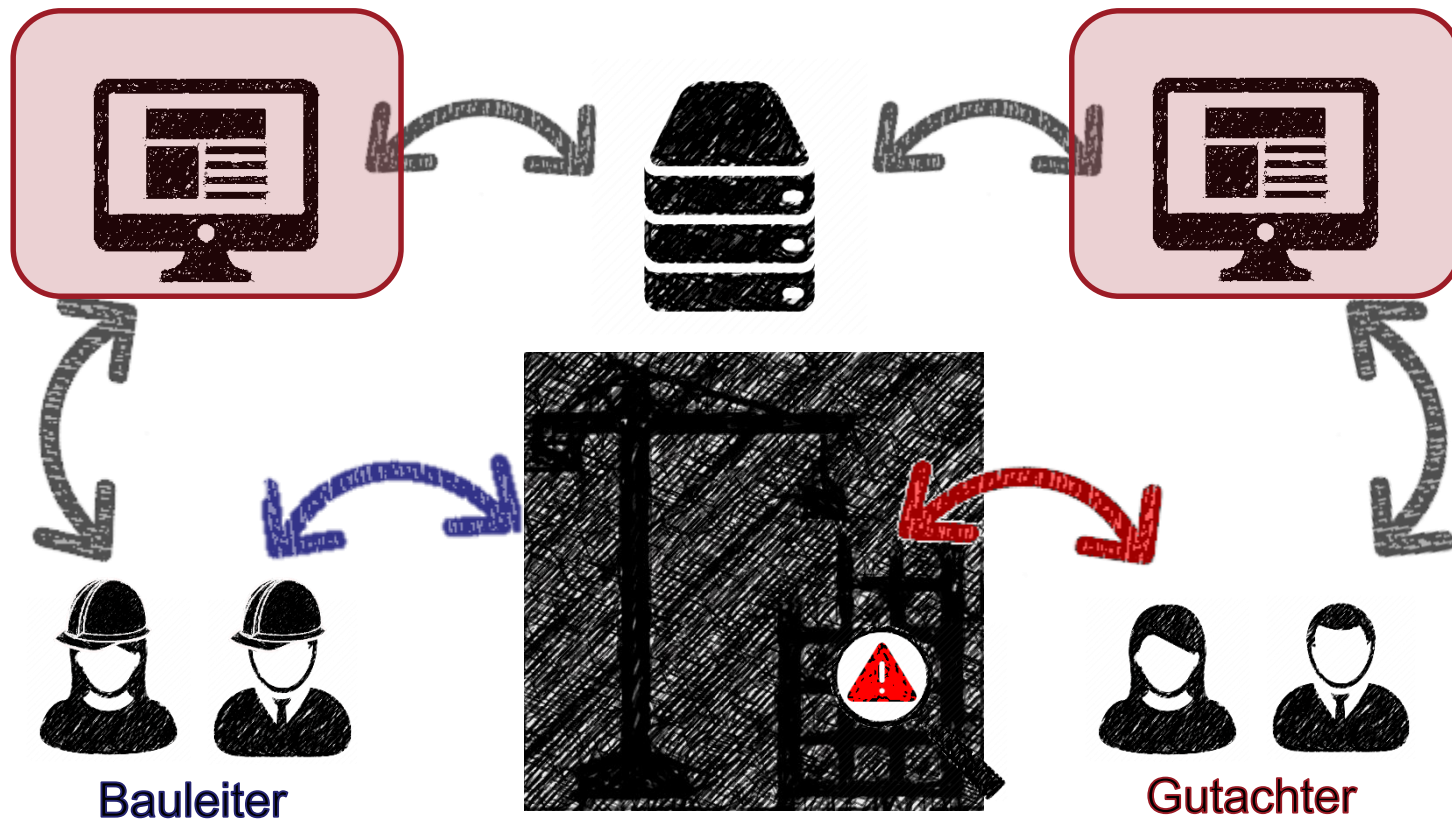
```
multiusingUpdate (1,connecting(), timestampSelect2, value_new = "tata1")
```

```
last edited at: 2019-05-09 19:24:11
```

```
Database is being edited by others, please try again
```

```
MySQL connection is closed
```


Was wollen wir machen?



Dynamische Webanwendungen



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- HTTP
- HTML
- Flask

Statisch vs. dynamisch

Statische Webseiten

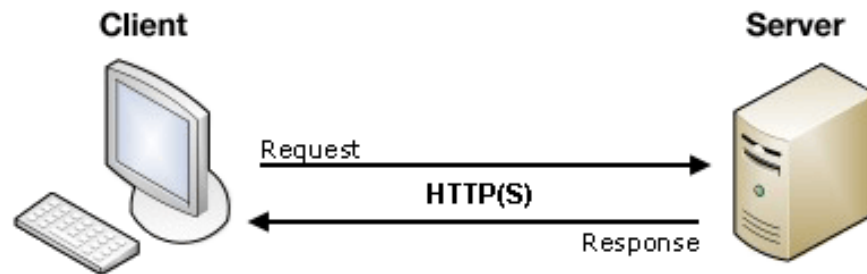
- Relativ einfach
- Bestehen aus HTML Code
- Aufbau im Browser durch Kopieren
- Keine Datenbank im Hintergrund
- Modifikationen relativ umständlich
- Erhaltung des Designs problematisch
- Geringe Anfangsinvestitionen, unter Umständen hohe Folgekosten

Dynamische Webseiten

- Baukastenprinzip
- Basiert auf einer Datenbank
- Seite wird erst während des Aufrufs erzeugt
- Änderungen/ Erweiterungen vergleichsweise einfach
- Hohe Anfangsinvestitionen, geringe Folgekosten

Client-Server-Prinzip

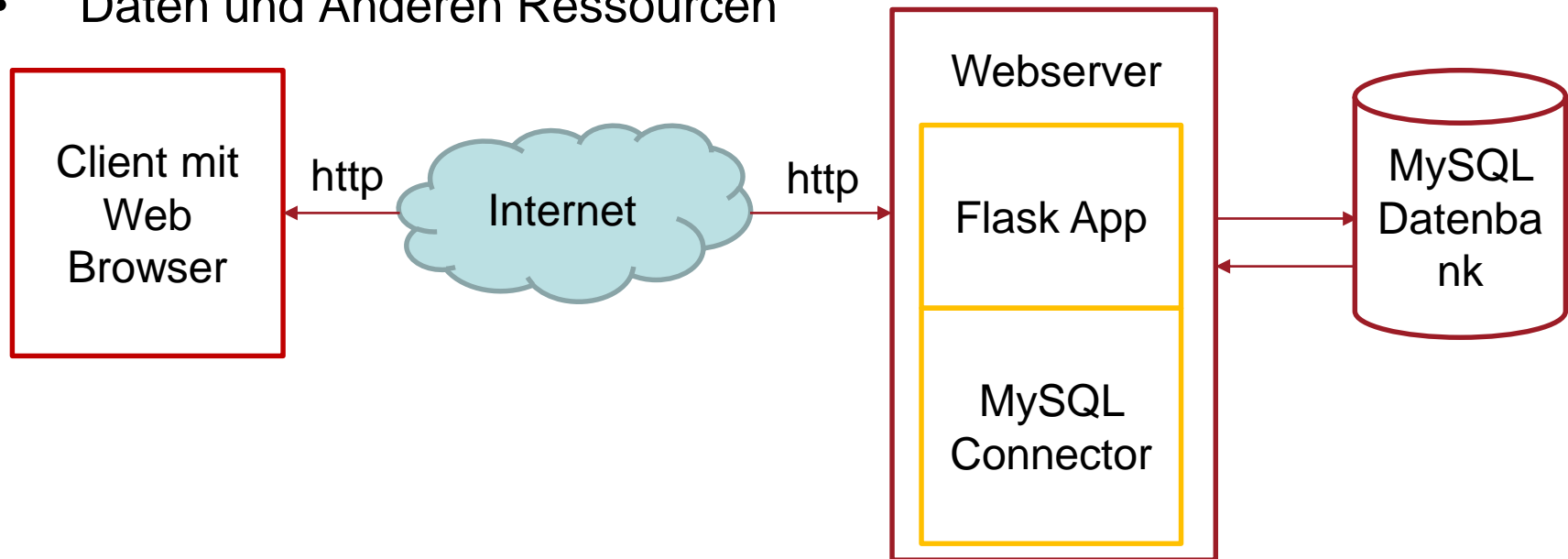
- Client stellt Anfrage an Server (Request)
- Server beantwortet Anfrage (Response)
- Client (zu nächst) schickt Anfragen an Server
- Server kann Code ausführen und generiert Webseite
- Server schickt Client die Webseite
- Client stellt Webseite im Browser dar



Quelle: <http://www.dev-grades.com/Development/ClientServer.html>

3-Schichte Architektur:

- Darstellung und Eingabeschicht
- Geschäftsprozess
- Daten und Anderen Ressourcen



Hypertext Transfer Protocol



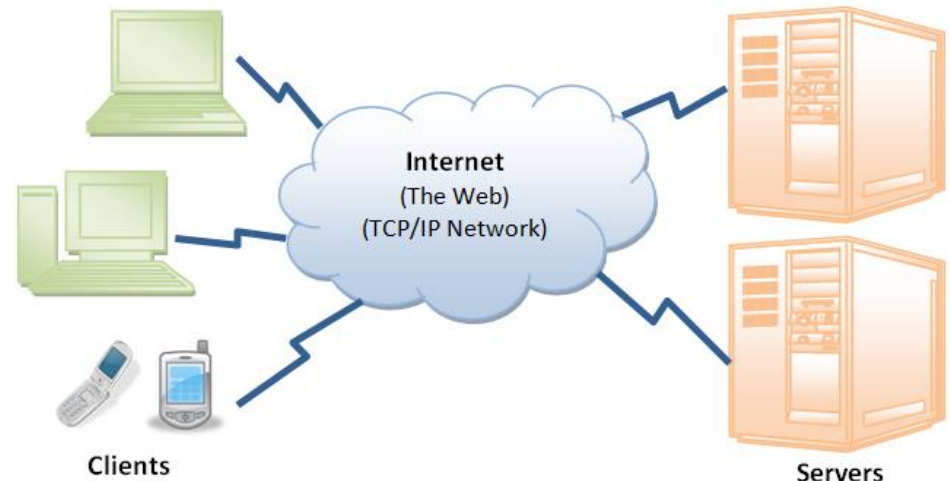
TECHNISCHE
UNIVERSITÄT
DARMSTADT

http://

HTTP – Grundlagen (1)

- Internet = riesiges verteiltes Client/Server Informationssystem
- Verschiedenste Anwendungen: browsing, E-Mail, File Transfer, Audio/Video Streaming, etc.
- Für eine funktionierende Kommunikation zwischen Client und Server, müssen sich die Applikationen auf ein Protokoll verständigen, z.B. HTTP, FTP, SMTP, POP, etc.

OSI-Schicht	TCP/IP-Schicht	Beispiel
Anwendungen (7)	Anwendungen	HTTP, UDS, FTP, SMTP, POP, Telnet, OPC UA
Darstellung (6)		
Sitzung (5)		
		SOCKS
Transport (4)	Transport	TCP, UDP, SCTP
Vermittlung (3)	Internet	IP (IPv4, IPv6), ICMP (über IP)
Sicherung (2)	Netzzugang	Ethernet, Token Bus, Token Ring, FDDI, IPoAC
Bitübertragung (1)		



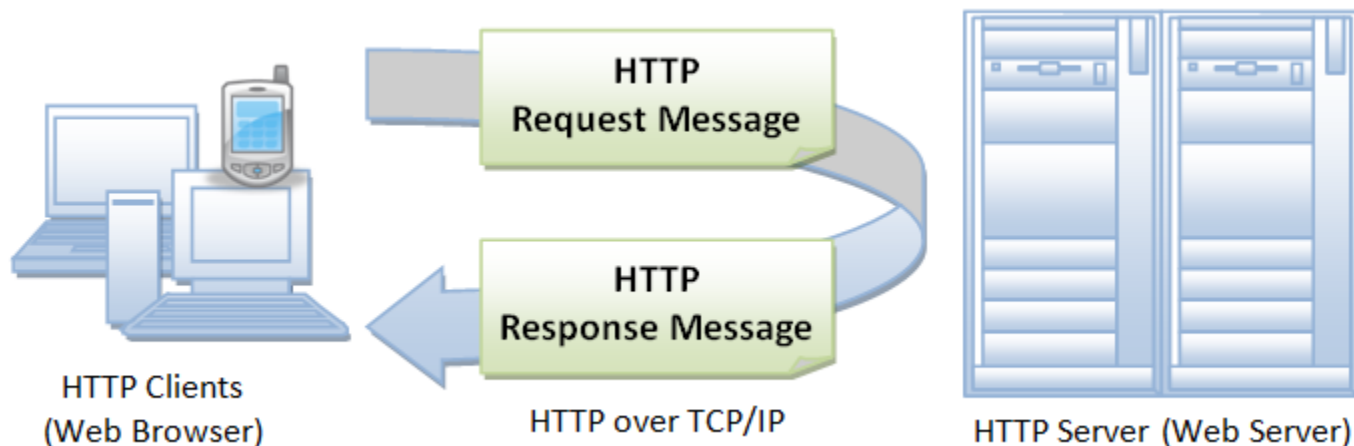
Quelle: https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

HTTP – Grundlagen (2)

- HTTP → Hypertext Transfer Protocol
- Zustandsloses Protokoll zum Übertragen von Daten über ein Netzwerk
- Zustandslos: es werden keine Sitzungsinformationen verwaltet, alle Transaktionen werden unabhängig voneinander behandelt
- Häufigster Anwendungsfall ist das Laden von Webseiten in einen Webbrowser

HTTP – Grundlagen (3)

- Nachrichtenaustausch erfolgt nach dem REQUEST / RESPONSE Prinzip
- Jede Nachricht besteht aus 2 Teilen: Header & Body
- HTTP-Anfragemethoden: GET, POST, HEAD, PUT, DELETE, TRACE, OPTIONS, CONNECT




Uniform Resource Locator (URL)

- Dient zur eindeutigen Lokalisierung einer Ressource im Web

Syntax

`protocol://hostname:port/path-and-file-name`

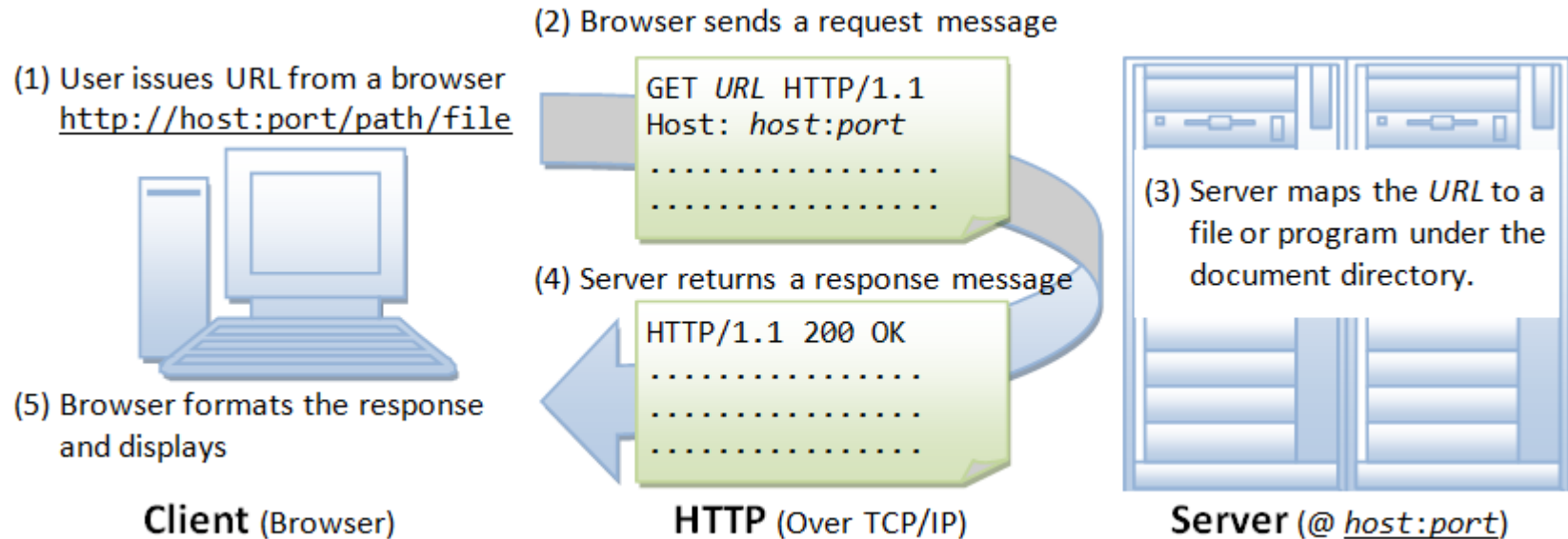
- URL besteht aus 4 Teilen:
 1. Protocol: genutztes Applikations-Level Protokoll (z.B. HTTP, FTP, etc.)
 2. Hostname: DNS Domain-Name (z.B. www.iib.tu-darmstadt.de)
 3. Port:  Nummer des TCP-Port auf den der Server „horcht“
 4. Path-and-file-name: Name und Ort der Ressource auf dem Server

Beispiel

`http://www.iib.tu-darmstadt.de/willkommen/index.de.jsp`

- TCP/IP = Transmission Control Protocol / Internet Protocol
- Ein Set von Netzwerk-Layer Protokollen für die Kommunikation in einem Netzwerk
- IP: Behandelt Netzwerkadressen und Routing
 - Jede Maschine im Netzwerk hat eine eindeutige Adresse (IP)
 - IPv4 → 4 Bytes (je 0-255) durch Punkte getrennt → z.B. 172.20.4.10
 - DNS (Domain Name Service) → Ersetzung von IP-Adressen durch Namen → z.B. 127.0.0.1 = localhost
- TCP: ein Transport-Layer Protokoll
 - verantwortlich für die Erstellung der Connection zwischen Client und Server
 - „reliable“ → Datenpaket nicht angekommen? → erneut senden

HTTP over TCP/IP



HTTP – Request Message

Request-Line (Syntax)

request-method-name request-URI HTTP-version

- *request-method-name*: HTTP-Anfragemethode
- *request-URI*: die angefragte Ressource
- *HTTP-version*: HTTP/1.0 bzw. HTTP/1.1

Request-Headers (Syntax)

request-header-name: request-header-value1, ...



HTTP – Response Message

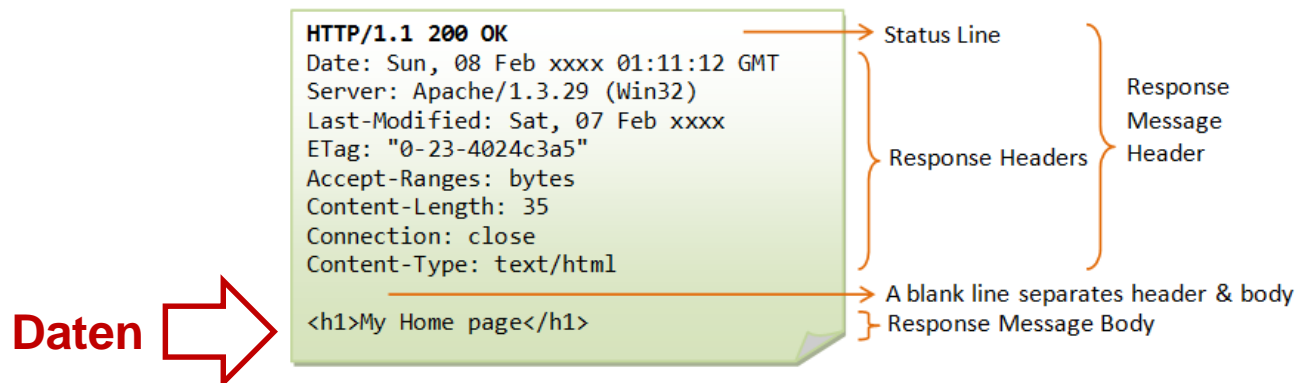
Status-Line (Syntax)

HTTP-version status-code reason-phrase

- *HTTP-version*: HTTP/1.0 bzw. HTTP/1.1
- *status-code*: 3-stellige Nummer, gibt Auskunft über Ergebnis der Anfrage
- *reason-phrase*: kurze Erklärung des Status-Code

Response-Headers (Syntax)

response-header-name: response-header-value1, ...



HTTP - Statuscodes

- Server sendet Fehlermeldung + Fehlercode, wenn etwas schief gelaufen ist. Darüber hinaus werden aber auch Statuscodes verwendet. (siehe HTTP-Spezifikation)



100
Continue



301
Moved Permanently



200
OK



502
Bad Gateway



404
Not Found

Quelle: <https://www.flickr.com/photos/girliemac/sets/72157628409467125/with/6512768893/>

GET vs POST (1)

- GET
 - Sendet Daten/Parameter in der URL
 - Üblicherweise verwendet um Daten zu „holen“
 - Alle Informationen zum auffinden der Daten sind in der URL
 - Länge der URL beschränkt (2 Kilobyte = 2048 Zeichen)
- POST
 - Daten/Parameter im Request-Body
 - Datenmenge nicht begrenzt
 - Üblicherweise eingesetzt um Formulardaten an den Server zu übertragen, bzw. für Operationen, die Daten verändern können
 - Enthält das Formular Werte, die nicht in der URL angezeigt werden sollen (z.B. ‚Password‘)

HTTP – GET

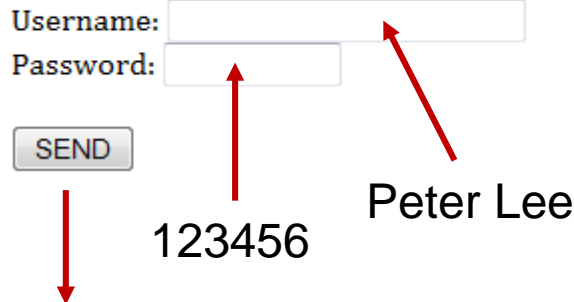
LOGIN

Username:

Password:

123456

Peter Lee



```
<html>
<head><title>Login</title></head>
<body>
  <h2>LOGTN</h2>
  <form method="get" action="/bin/login">
    Username: <input type="text" name="user" size="25" /><br />
    Password: <input type="password" name="pw" size="10" /><br /><br />
    <input type="hidden" name="action" value="login" />
    <input type="submit" value="SEND" />
  </form>
</body>
</html>
```

<http://127.0.0.1:8000/bin/login?user=Peter+Lee&pw=123456&action=login>

**Achtung: Hier werden
Passwort-Informationen
direkt in der URL
übertragen!
Sensible Informationen
niemals unverschlüsselt
übertragen!**

```
GET /bin/login?user=Peter+Lee&pw=123456&action=login HTTP/1.1
Accept: image/gif, image/jpeg, */*
Referer: http://127.0.0.1:8000/login.html
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 127.0.0.1:8000
Connection: Keep-Alive
```

HTTP – POST



TECHNISCHE
UNIVERSITÄT
DARMSTADT

LOGIN

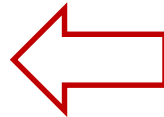
Username:

Password:

SEND

123456

Peter Lee



```
<html>
<head><title>Login</title></head>
<body>
  <h2>LOGIN</h2>
  <form method="post" action="/bin/login">
    Username: <input type="text" name="user" size="25" /><br />
    Password: <input type="password" name="pw" size="10" /><br /><br />
    <input type="hidden" name="action" value="login" />
    <input type="submit" value="SEND" />
  </form>
</body>
</html>
```

POST /bin/login HTTP/1.1

Host: 127.0.0.1:8000

Accept: image/gif, image/jpeg, */*

Referer: http://127.0.0.1:8000/login.html

Accept-Language: en-us

Content-Type: application/x-www-form-urlencoded

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)

Content-Length: 37

Connection: Keep-Alive

Cache-Control: no-cache

User=Peter+Lee&pw=123456&action=login

Hypertext Markup Language



TECHNISCHE
UNIVERSITÄT
DARMSTADT

< HTML />

- **HyperText Markup Language**
- Standardisiert durch das W3C (<http://www.w3c.org>)
- Auszeichnungssprache (markup language)
- HTML-Datei ist eine einfache Text-Datei
- Beschreibt den logischen Inhalt einer WWW-Seite
- HTML-Befehle haben einen festgelegten Erstreckungsraum, d.h. einen definierten Beginn und ein definiertes Ende

Tags:

- Alle HTML-Befehle stehen in sogenannten „Tags“
- Tags werden durch „< ... >“ definiert
- Einleitender Tag:
 - `<HTMLBefehl>`
- Abschließender Tag:
 - `</HTMLBefehl>`
- Beispiel: Überschrift 2. Ordnung
 - `<h2>` Dies ist eine Überschrift `</h2>`
- Tags können verschachtelt werden
 - `<h2>` Dies ist eine `` Überschrift `` `</h2>`

Aufbau einer HTML-Seite

- Kopf (Head):
 - Definiert allgemeine Angaben für die Seite
 - Definiert Informationen über den Ersteller usw.
 - Definiert Informationen für Web-Suchmaschinen
 - Legt Text der Titelzeile fest
- Körper (Body):
 - Definiert den eigentlichen Inhalt der Webseite
 - Enthält alle darzustellenden Elemente, wie Überschriften, Tabellen, etc.



HTML – Grundlagen (4)

`<html>`

`<head>`

`<title>`Text des Titels`</title>`

`</head>`

} Kopf

`<body>`

Text, Verweise, Grafikreferenzen usw.

`</body>`

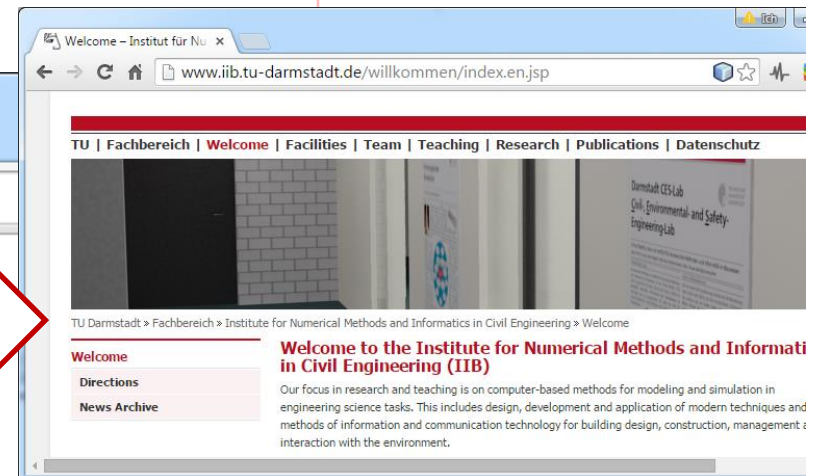
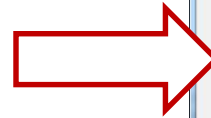
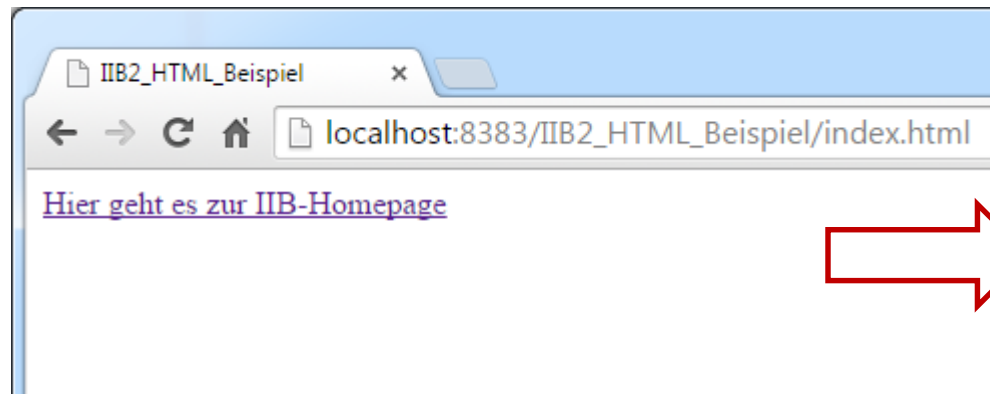
} Körper

`</html>`

Verweise <a>

- Durch Verweise (Links) können HTML-Seiten miteinander verknüpft werden

```
<html>
  <head>
    <title>IIB2_HTML_Beispiel</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <a href="http://www.iib.tu-darmstadt.de">Hier geht es zur IIB-Homepage</a>
  </body>
</html>
```

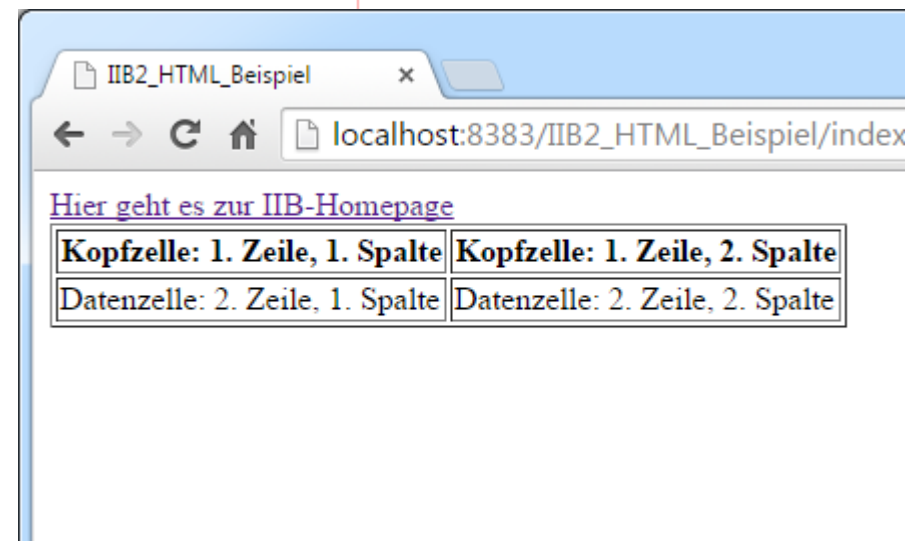


Tabellen <table>

- Tabellen sind in HTML ein wichtiges Element um den Inhalt einer Webseite zu strukturieren

```
<body>
  <a href="http://www.iib.tu-darmstadt.de">Hier geht es zur IIB-Homepage</a>

  <table border="1">
    <tr>
      <th>Kopfzelle: 1. Zeile, 1. Spalte</th>
      <th>Kopfzelle: 1. Zeile, 2. Spalte</th>
    </tr>
    <tr>
      <td>Datenzelle: 2. Zeile, 1. Spalte</td>
      <td>Datenzelle: 2. Zeile, 2. Spalte</td>
    </tr>
  </table>
</body>
```



Formulare <form> (1)

- Mit Hilfe von Formularen ist es möglich Daten zu erfassen

`<form> ... </form>`

- Formular-Elemente:

- Einzeilige Eingabefenster

`<input type=text ...>`

- Mehrzeilige Eingabefenster

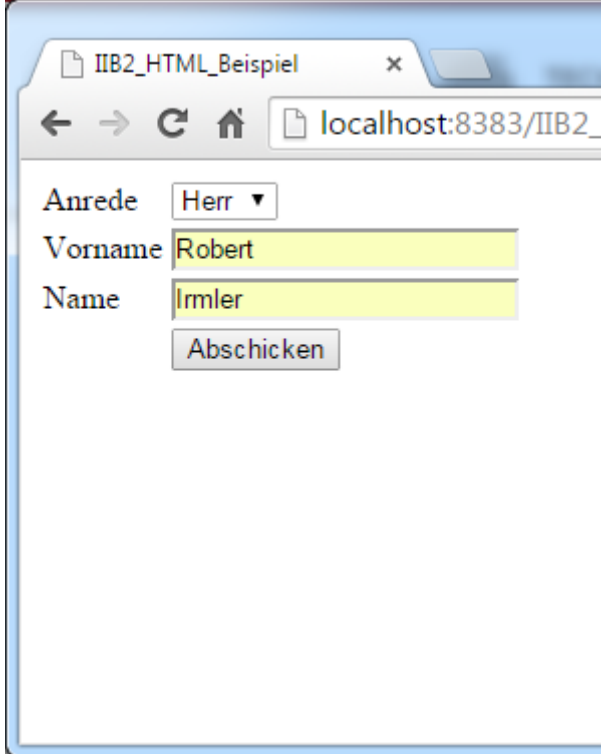
`<textarea> ... </textarea>`

- Listenfenster

`<select> ... </select>`

- Buttons

`<input type=button ...>`



The screenshot shows a web browser window with the title 'IIB2_HTML_Beispiel'. The address bar shows 'localhost:8383/IIB2_'. The form contains the following elements:

- Anrede:** A dropdown menu with the value 'Herr'.
- Vorname:** A text input field containing the text 'Robert'.
- Name:** A text input field containing the text 'Irmeler'.
- Abschicken:** A button to submit the form.

Formulare <form> (2)

```
<form name="form1" method="post" action="next.html">
  <table>
    <tr>
      <td>Anrede</td>
      <td>
        <select name="anrede">
          <option>Herr</option>
          <option>Frau</option>
        </select>
      </td>
    </tr>
    <tr>
      <td>Vorname</td>
      <td><input type="text" name="vorname"></td>
    </tr>
    <tr>
      <td>Name</td>
      <td><input type="text" name="name"></td>
    </tr>
    <tr>
      <td></td>
      <td><input type="submit" name="senden" value="Abschicken"></td>
    </tr>
  </table>
</form>
```

form

- name: Name des Formulars
- method: get/post/etc.
- action: Zielseite

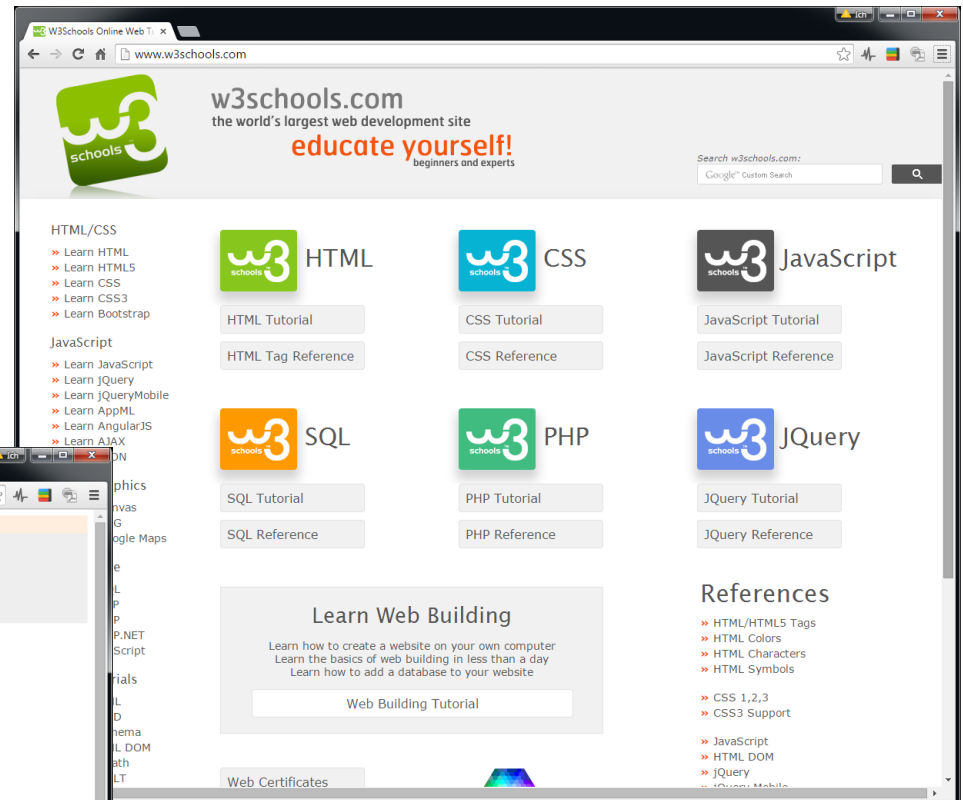
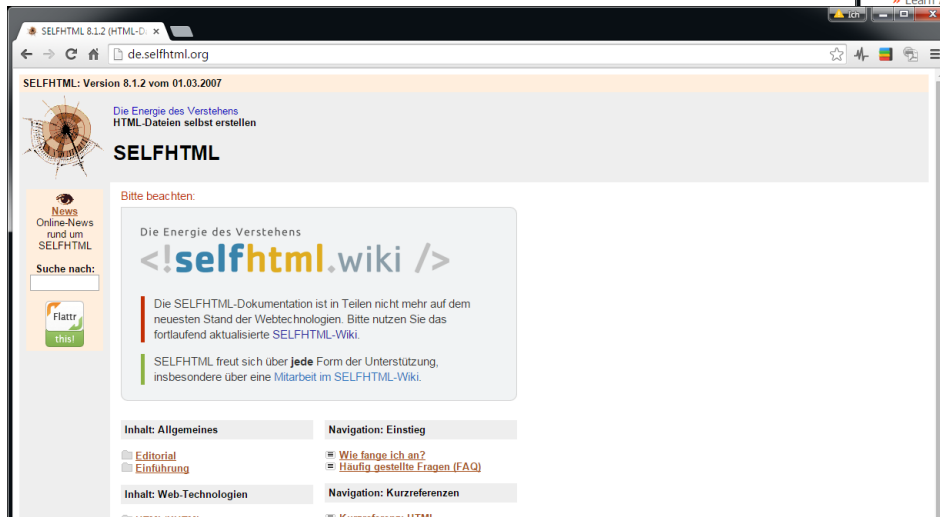
input

- name: Name des Elements
- type: Typ des Elements
- value: Anzeige-Text

Klicken des submit-Buttons versendet das ausgefüllte Formular...

HTML Tutorials & Hilfen

- SelfHTML von Stefan Münz
<http://de.selfhtml.org>
- W3Schools
<http://www.w3schools.com>



Flask



TECHNISCHE
UNIVERSITÄT
DARMSTADT

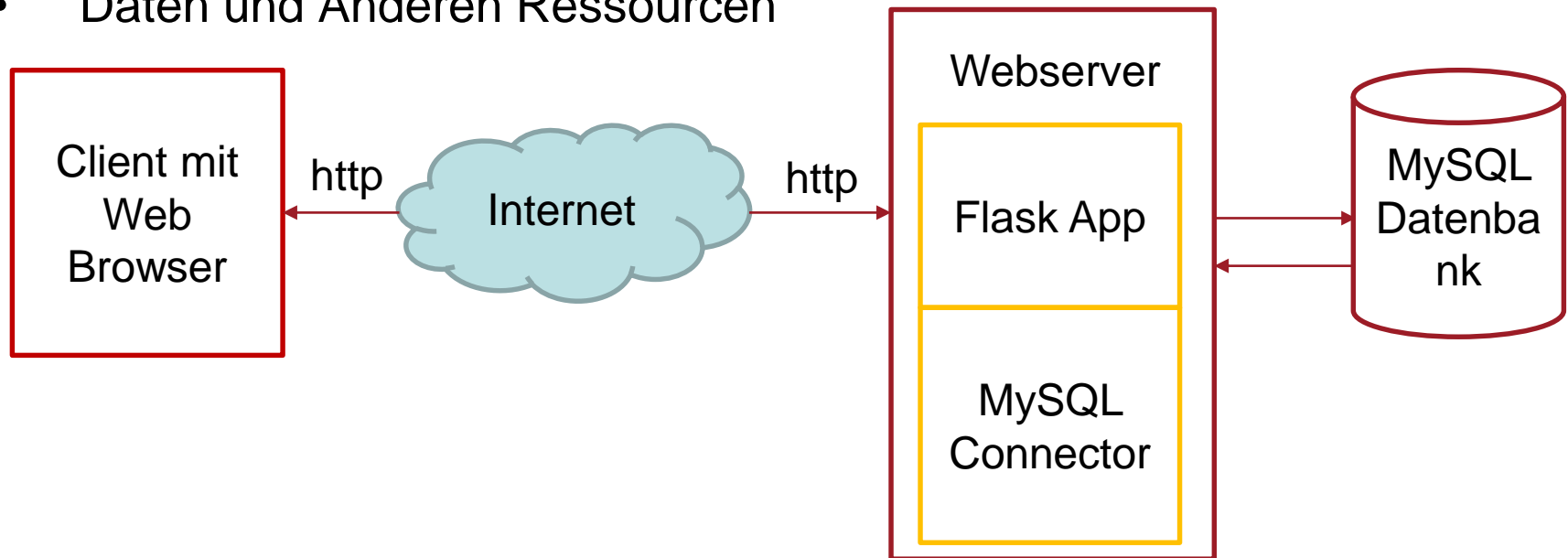


Flask

web development,
one drop at a time

3-Schichte Architektur:

- Darstellung und Eingabeschicht
- Geschäftsprozess
- Daten und Anderen Ressourcen



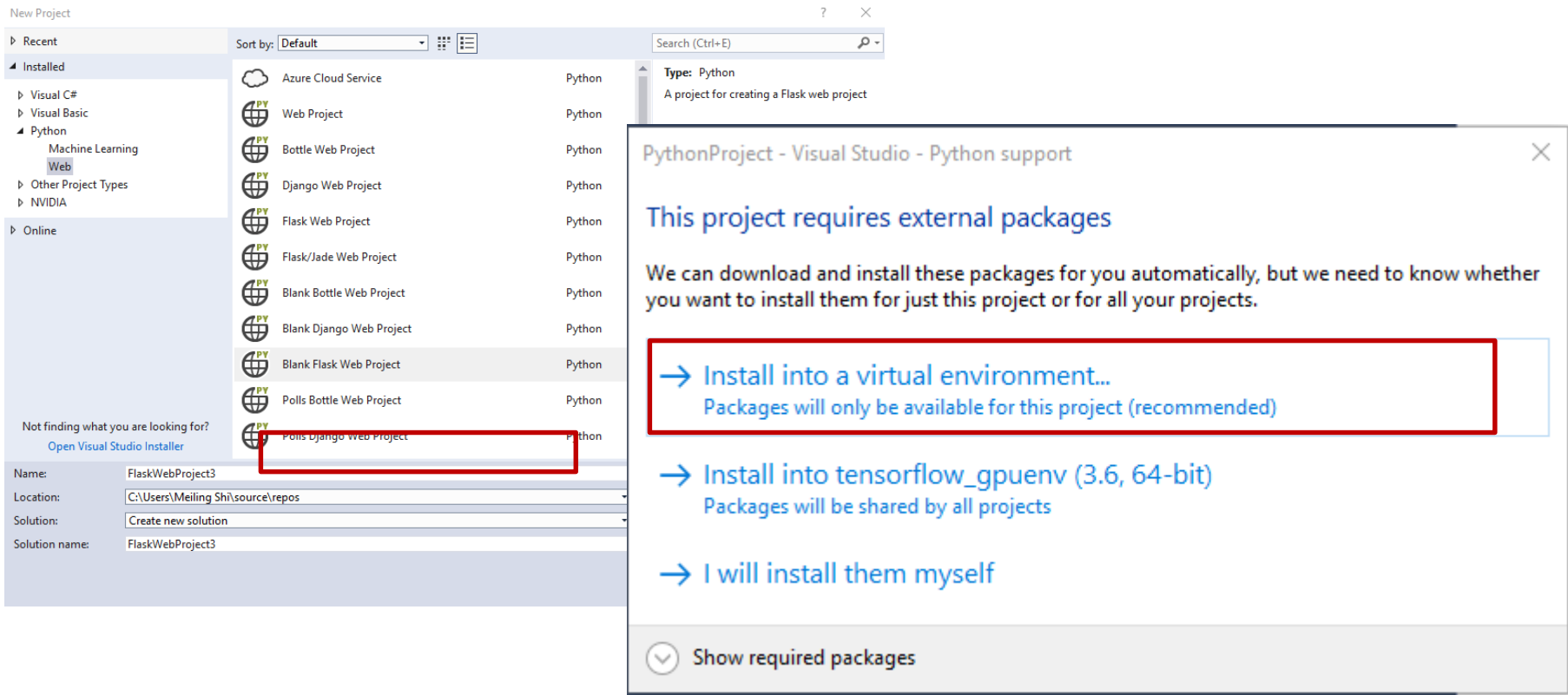
Option 1:

- Install pip falls nicht vorhanden
<https://pip.pypa.io/en/stable/installing/>
- Install flask in Kommandozeile mit pip (click on the Windows icon and type **cmd**, then click on the command prompt icon)):

```
pip install Flask
```

Installation Option 2

In Visual Studio: erstellt unter python->Web-> Create „Blank Flask Web Project“



The screenshot shows the Visual Studio 'New Project' dialog and a subsequent 'PythonProject - Visual Studio - Python support' dialog.

New Project Dialog:

- Left pane:** 'Installed' > 'Python' > 'Web'.
- Right pane:** List of project types. 'Blank Django Web Project' is highlighted with a red box.
- Bottom:** Project details for 'FlaskWebProject3'.

PythonProject Dialog:

- Title:** PythonProject - Visual Studio - Python support
- Text:** 'This project requires external packages. We can download and install these packages for you automatically, but we need to know whether you want to install them for just this project or for all your projects.'
- Options:**
 - Install into a virtual environment...** (highlighted with a red box): Packages will only be available for this project (recommended).
 - Install into tensorflow_gpuenv (3.6, 64-bit): Packages will be shared by all projects.
 - I will install them myself.
- Bottom:** 'Show required packages' button.

Flask- Micro Webframework für Python

- Kein Webserver, sondern ein Entwicklungsframework, besteht aus Tools und Bibliotheken
- Öffnet einen Socket, wartet auf TCP-Verbindungen, nimmt HTTP-Requests an
- Analysiert die Anfrage und ruft definierte Funktionen auf, die Antwort-HTML generieren
- Schickt eine entsprechende HTTP-Antwort Aufgabe: Funktionen, die abhängig vom jeweiligen URL sind
- Auf dem Server Code ausführen und HTML zurückgeben

Basiert auf...

- Werkzeug: Python Web Server Gateway Interface
 - Schnittstelle, die beschreibt wie ein Webserver mit Webanwendungen kommuniziert und wie Webanwendungen miteinander verknüpft werden können, um eine Anforderung zu verarbeiten.
- Jinja2: Template Engine: Unterstützt HTML-Seite bei Darstellung
 - Gerüst bauen, in das bestimmte Werte eingesetzt werden können



Syntax:

- {%.....%}: Anweisungen
- {{.....}}: Ausdrücke, die zum Drucken auf die Vorlagenausgabe verwendet werden.
- {#.....#}: Kommentare, die nicht in der Vorlagenausgabe enthalten sind.
- #.....## : werden als Zeilenanweisungen verwendet.
- Mehr Befehle siehe: <https://overiq.com/flask-101/basics-of-jinja-template-language/>

Template-Vererbung verwendet `{% Block %}` Tag, um der Template-Engine mitzuteilen, dass sie die gemeinsamen Elemente Ihrer Website über untergeordnete Templates überschreiben soll.

Beispiel (s.f.)

- **base.html** ist die übergeordnete Template, die das Basis-Layout ist und auf der Sie mit Hilfe der untergeordneten Templates ändern können, die verwendet werden, um leere Blöcke mit Inhalt zu füllen. base.html ist ein allgemeines Layout der Webseite.

Beispiel: base.html



```
<!DOCTYPE html>
<html>
<nav>
  <h1><a href="{{ url_for('application.index') }}">Homepage</a></h1>
  <ul>
    {% if g.user %}
      <li><span>{{ g.user[1] }}</span>
      <li><a href="{{ url_for('auth.logout') }}">Log Out</a>
    {% else %}
      <li><a href="{{ url_for('auth.register') }}">Register</a>
      <li><a href="{{ url_for('auth.login') }}">Log In</a>
    {% endif %}
  </ul>
</nav>
<head>
  {% block head %}
  <title>{% block title %}{% endblock %}</title>
  {% endblock %}
</head>
<body>
  {% block body %}{% endblock %}
</body>
</html>
```

Beispiel: index.html (vererbt von base.html)



```
{% extends 'base.html' %}
```

```
{% block head %}Homepage{% endblock %}
```

```
{% block body %}
```

```
{% if records %}
```

```
  <table>
```

```
    {% for rec in records %}
```

```
    <tr>
```

```
      <th>{{rec[2]}}</th>
```

```
      <th>{{rec[3]}}</th>
```

```
      <th>{{rec[1]}}</th>
```

```
      <th>{{rec[4]}}</th>
```

```
.....
```

```
    </tr>
```

```
    {% endfor %}
```

```
  </table>
```

```
{%else%}
```

```
<p>Please Log in</p>
```

```
{% endif %}
```

```
{% endblock %}
```

Flask - Hallo World!



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
from flask import Flask
app = Flask(__name__)

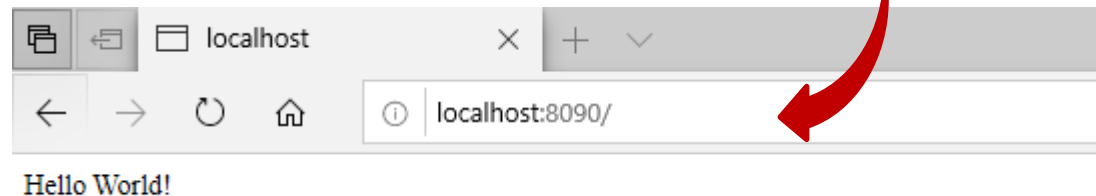
@app.route('/')
def hello():
    return "Hello World!"

if __name__ == '__main__':
    app.run(host = '0.0.0.0', port=8090)
```



Import Flask Klasse
Flask-klasse „app“ instanzieren
@app.route('/'): Decorator, markiert die Funktion als „Route“

```
* Serving Flask app "__main__" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: off

* Running on http://0.0.0.0:8090/ (Press CTRL+C to quit)
```



- Ein Dekorateur ist eine Funktion, die eine andere Funktion übernimmt und das Verhalten dieser Funktion erweitert, ohne sie explizit zu modifizieren.
- `@**.route(„/“)` -- Flask & Blueprint Instanz
 - Mit `app.route` als Dekorateur wird die Funktion `index` für die Route registriert / so dass bei Anforderung dieser Route der Index aufgerufen und das Ergebnis `"Hello world"` an den Client zurückgegeben wird (Webbrowser, Curl, etc.).
- `@login_required`
- `@**.before_app_request` – Blueprint Instanz

- Um Instanz (Application) der Flask-Klasse zu erweitern
- Blueprint ist ideal für größere Anwendungen
- Ein Projekt könnte ein Anwendungsobjekt instanziiieren, mehrere Erweiterungen initialisieren und eine Sammlung von Blueprints registrieren. 
- Eine Blueprint wird für eine Anwendung unter einem URL-Präfix und/oder einer Subdomain registriert. 
- Parameter im URL-Präfix/Subdomain werden zu gemeinsamen View-Argumenten (mit Standardeinstellungen) über alle View-Funktionen im Blueprint.

Beispiel



application.py

```
from flask import Blueprint
bp = Blueprint("application", "application")

@bp.route("/index")
def index():
    """Show all the buildings"""

    if (session.get("user_id") != None):
        id = session.get("user_id")
        sql_query = "select geb_id, geb_plz, geb_strasse,
geb_hausnummer, geb_ort from gebaeude where geb_blt_id = " +
str(id)
        records = databasemanager.getAll(sql_query)
        return render_template("/index.html", records =
records)
    else:
        return render_template("/index.html")
```

app.py (Start-Datei)

```
import auth, ... ..
def create_app(test_config=None):
    """Create and configure an instance of the Flask
    application."""
    app = Flask(__name__, instance_relative_config=True)
    ... ..
    # apply the blueprints to the app
    app.register_blueprint(auth.bp)
    app.register_blueprint(application.bp)
    return app

if __name__ == '__main__':
    app = create_app()
    app.run(host = '0.0.0.0', port=8090)
```

auth.py

```
Import ...  
from flask import Blueprint  
  
bp = Blueprint("auth", "auth", url_prefix="/auth")  
@bp.route("/logout")  
def logout():  
    """Clear the current session, including the stored user  
    id."""  
    session.clear()  
    return redirect(url_for("application.index"))  
...
```

- HTTP ist ein zustandsloses Protokoll
- Sessions/Cookies: Um Nutzer spezifische Daten zwischen Request zu speichern
- Das Session-Objekt des Flask-Pakets wird verwendet, um Session-Daten zu setzen und abzurufen. Es funktioniert wie ein Dictionary, kann aber auch Änderungen verfolgen.
- **Secret Key** wird gebraucht

Beispiel



app.py

```
def create_app(test_config=None):  
    """Create and configure an instance of the Flask  
    application."""  
    app = Flask(__name__, instance_relative_config=True)  
    app.config.from_mapping(  
        # a default secret that should be overridden by  
        instance config  
        SECRET_KEY="dev")  
    ...  
  
    # apply the blueprints to the app  
    app.register_blueprint(auth.bp)  
    app.register_blueprint(application.bp)  
    return app
```

Beispiel



auth.py

```
@bp.route("/login", methods=("GET", "POST"))
def login():
    """Log in a registered user by adding the user id to the session."""
    if request.method == "POST":
        username = request.form["username"]
        password = request.form["password"]
        error = None
        user = databasemanager.getBauleiterLogin(username)
        password_db = user[4]
        if password_db is None:
            error = "Incorrect username."
        # elif not check_password_hash(password_db, password): must register first
        elif not password_db == password:
            error = "Incorrect password."
            print(error = "Incorrect password.")
        if error is None:
            # store the user id in a new session and return to the index
            session.clear()
            session["user id"] = user[0]
            return redirect(url_for("application.index"))
        flash(error)
    return render_template("/login.html")
```

Attribute „g“

- „g“ ist ein globaler Namensraum für die Speicherung beliebiger Daten in einem einzigen Applikationskontext
- Ein App-Kontext dauert für einen Anfrage-/Antwortzyklus („Request-Response“)
- g ist nicht geeignet, um Daten über Anfragen hinweg zu speichern.
- Für persistente Daten eine Datenbank, Redis, die Session oder eine andere externe Datenquelle verwenden

auth.py

```
@bp.before_app_request
def load_logged_in_user():
    """If a user id is stored in the session, load the user
    object from the database into ``g.user``."""
    user_id = session.get("user_id")
    print(type(user_id))
    if user_id is None:
        g.user = None
    else:
        sql = "select * from bauleiter where blt_id = " +
str(user_id)
        g.user = databasemanager.getOne(sql)
        print(g.user)
```

Redirect



```
render_template("/index.html")  
render_template("/index.html", records = records)
```

Oder

```
redirect(url_for('application.index'))
```

Blueprint name, „.“ wenn
die blueprint local ist

Funktionsname

```
@bp.route("/index")  
@login_required  
def index():  
    """Show all the buildings"""  
  
    if (session.get("user_id") != None):  
        id = session.get("user_id")  
        sql_query = "select geb_id, geb_plz, geb_strasse,  
geb_hausnummer, geb_ort from gebaeude where geb_blt_id = " + str(id)  
        records = databasemanager.getAll(sql_query)  
        return render_template("/index.html", records = records)  
    else:  
        return render_template("/index.html")
```

Redirect Beispiel



```
@bp.route("/login", methods=("GET", "POST"))
def login():
    """Log in a registered user by adding the user id to the session."""
    if request.method == "POST":
        username = request.form["username"]
        password = request.form["password"]
        error = None
        user = databasemanager.getBauleiterLogin(username)
        password_db = user[4]
        if password_db is None:
            error = "Incorrect username."
        elif not password_db == password:
            error = "Incorrect password."
            print(error = "Incorrect password.")

        if error is None:
            # store the user id in a new session and return to the index
            session.clear()
            session["user_id"] = user[0]
            return redirect(url_for("application.index"))
        flash(error)
    return render_template("/login.html")
```

Blueprint name, „.“
wenn die blueprint
local ist

Funktionsname

Projekt Struktur



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Projektordner

--static

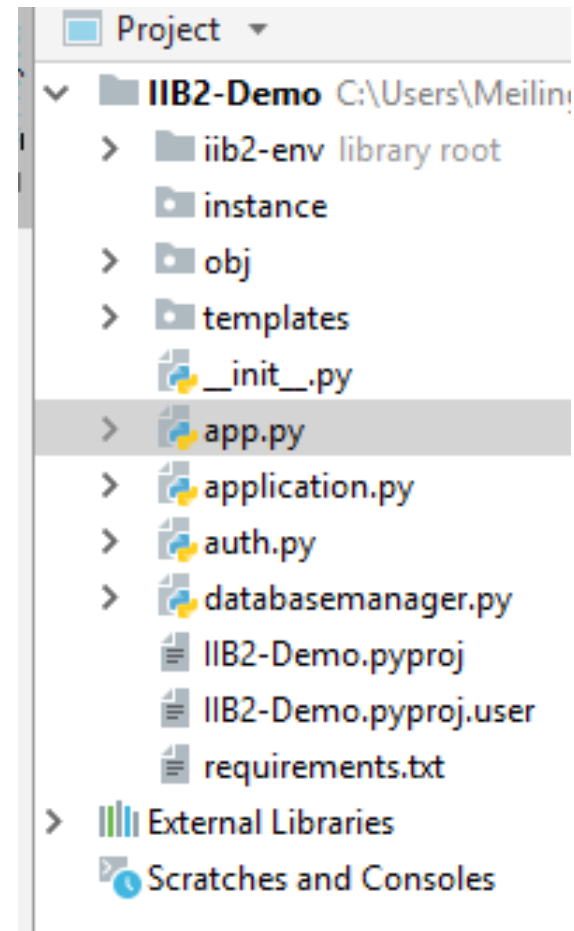
--.CSS...

--templates

--.html

--logikmodule

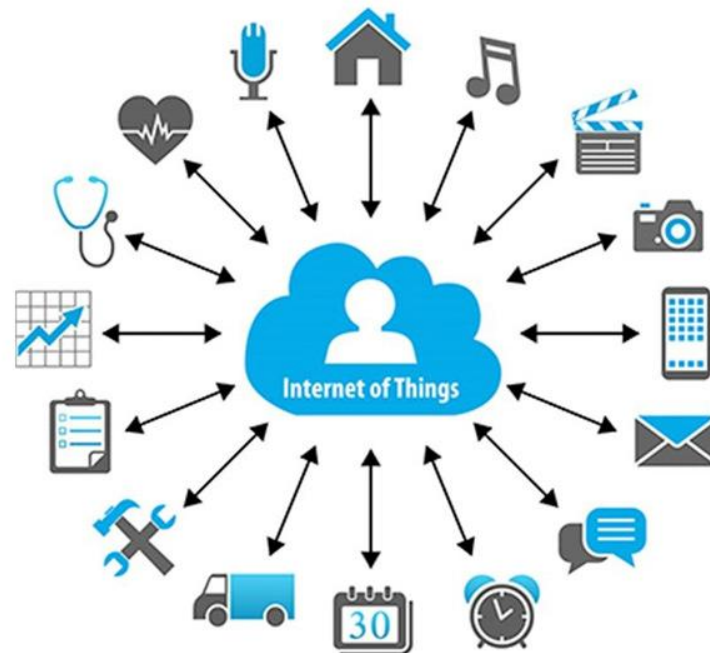
--.py



Internet der Dinge



TECHNISCHE
UNIVERSITÄT
DARMSTADT

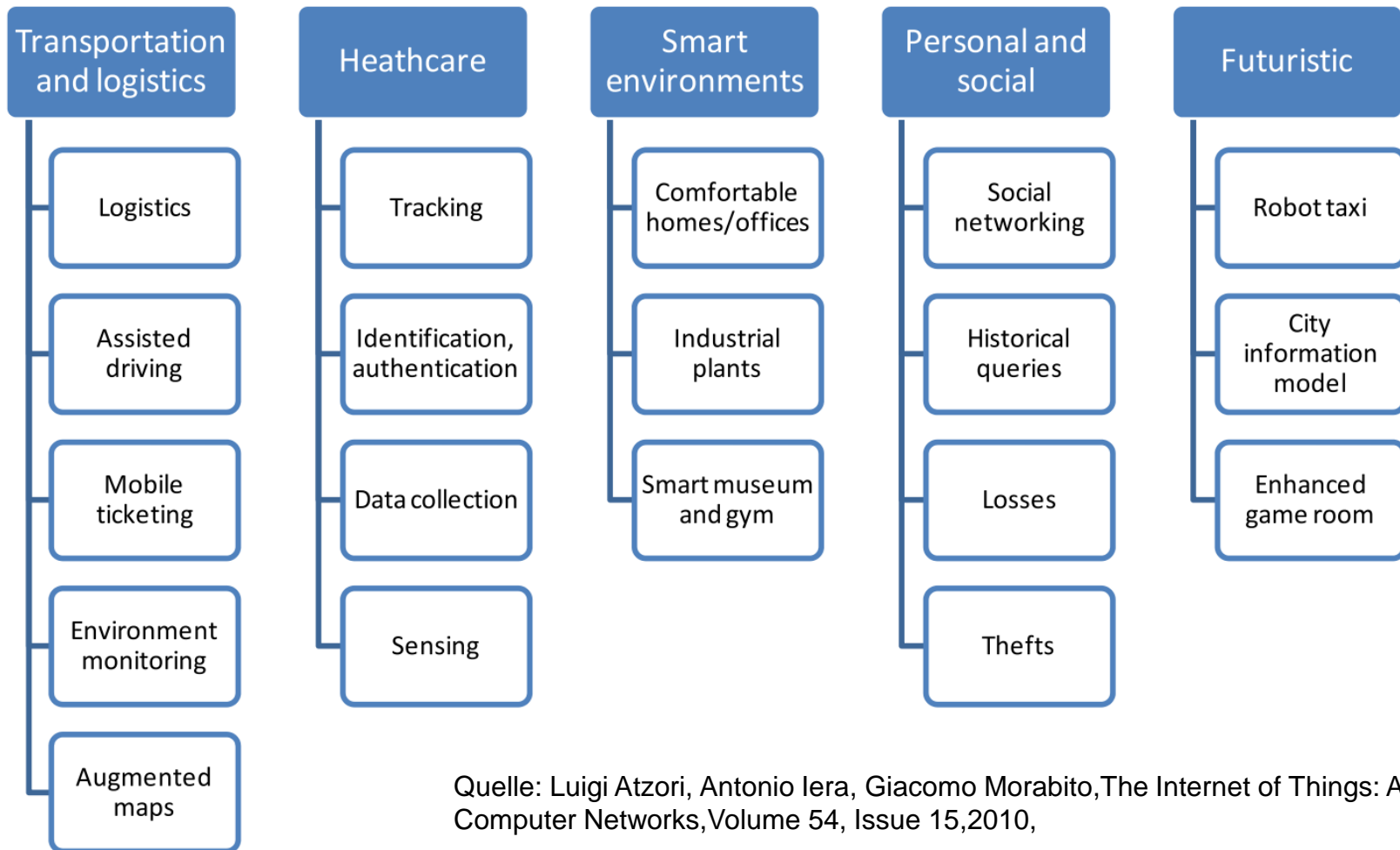


- Vernetzung vieler Geräte
- Dinge werden „intelligenter“ und können mit einander kommunizieren
- Menschen können dadurch entlastet
- **Zustände** der realen Welt digitalisieren und über Internet bereitgestellt nutzen
- Technologien: RFID, QR, Barcode, **Sensoren**, Aktoren,...
- Webtechnologien: TCP, UDP, IPv4, IPv6, **HTTP**, **Web Services**, IOT Protokolle...

Anwendungen

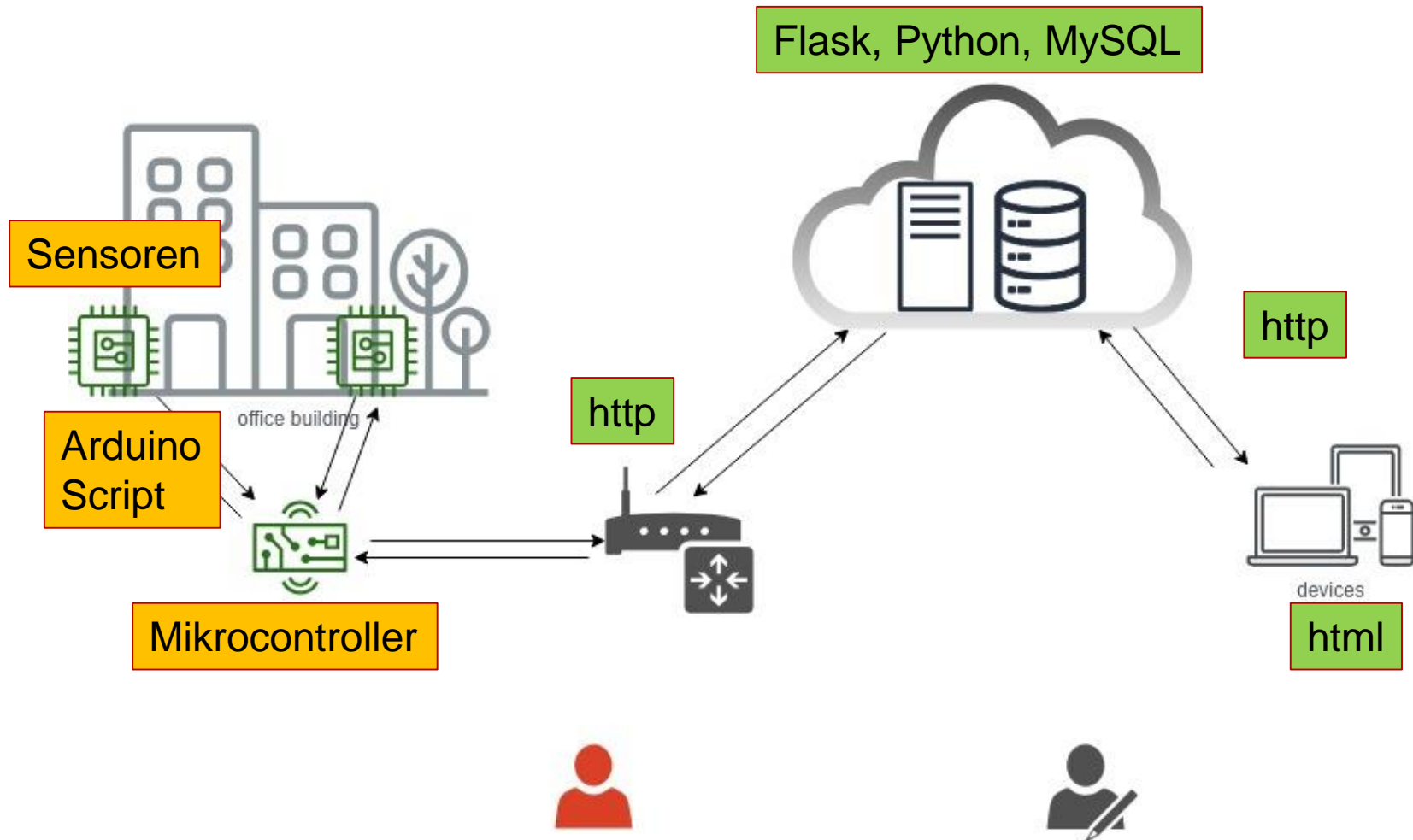


TECHNISCHE
UNIVERSITÄT
DARMSTADT



Quelle: Luigi Atzori, Antonio Iera, Giacomo Morabito, The Internet of Things: A survey, Computer Networks, Volume 54, Issue 15, 2010,

Bestandteile am Beispiel der HÜ

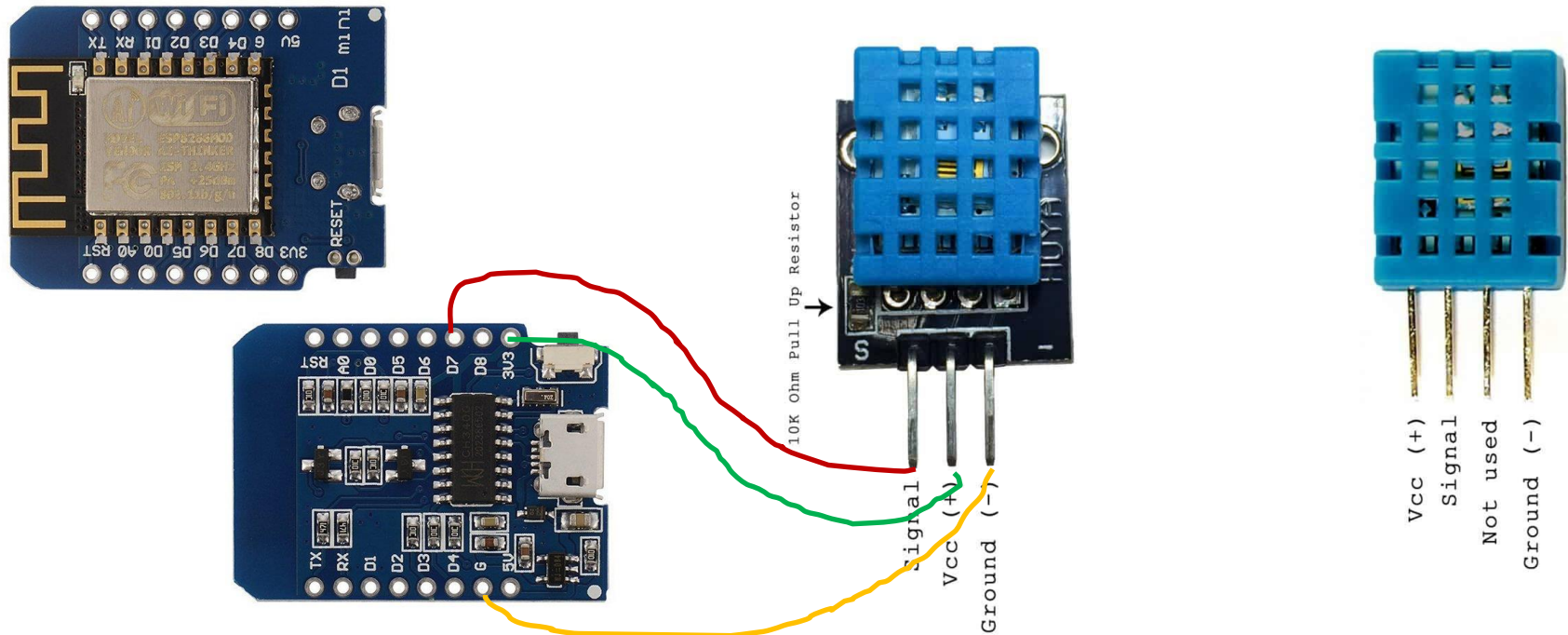


Mikrocontroller: Ein Mikrocontroller ist ein Ein-Chip-Computersystem, der einen Prozessor und zugleich auch Peripheriefunktionen enthalten wie Arbeits- und Programmspeicher, WLAN (Waschmaschinen, Fernbedienungen..)

Sensoren: Technisches Bauteil, das bestimmte physikalische oder chemische Eigenschaften und/oder die stoffliche Beschaffenheit seiner Umgebung qualitativ oder als Messgröße quantitativ erfassen kann. Diese Größen werden mittels physikalischer oder chemischer Effekte erfasst und in ein weiterverarbeitbares elektrisches Signal umgeformt.

-- Wiki

Mikrocontroller und Sensor verbinden



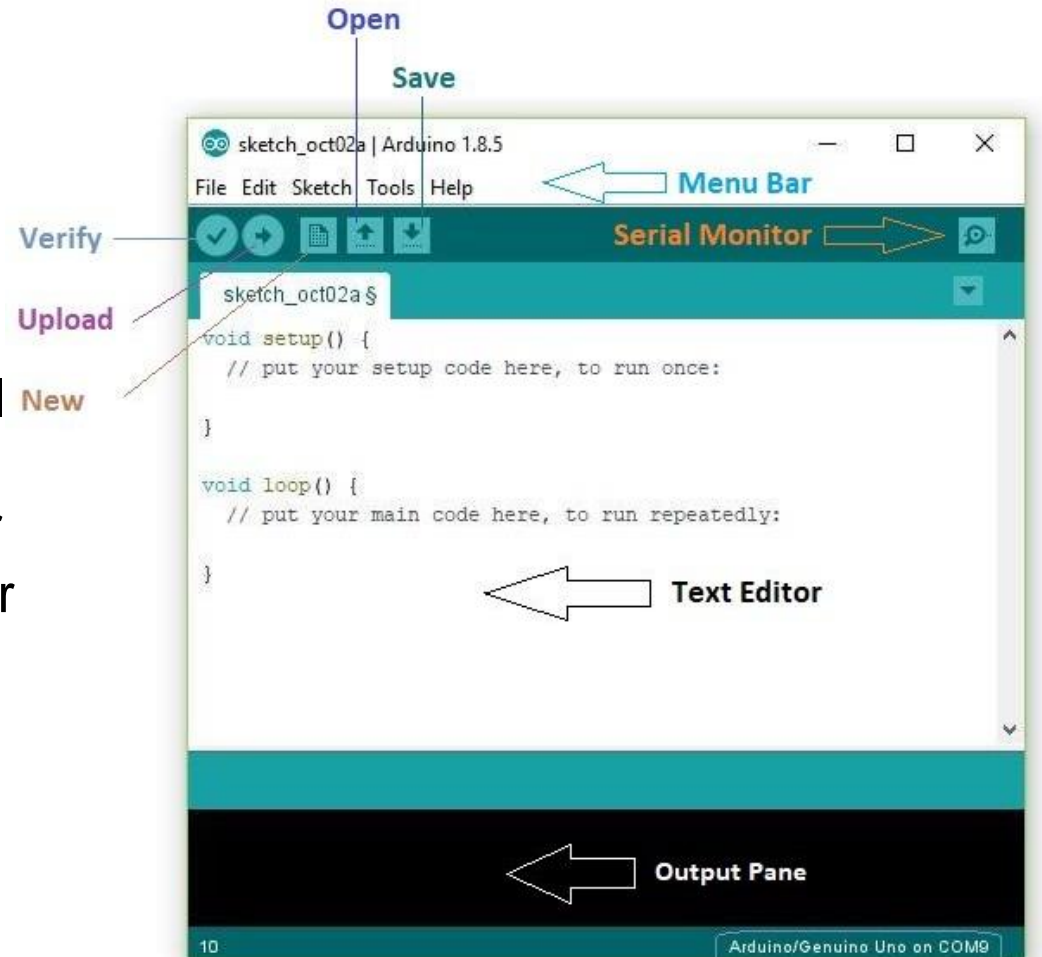
WeMos D1 Mini

Temperature Humidity Sensor Module DHT11

Arduino Plattform

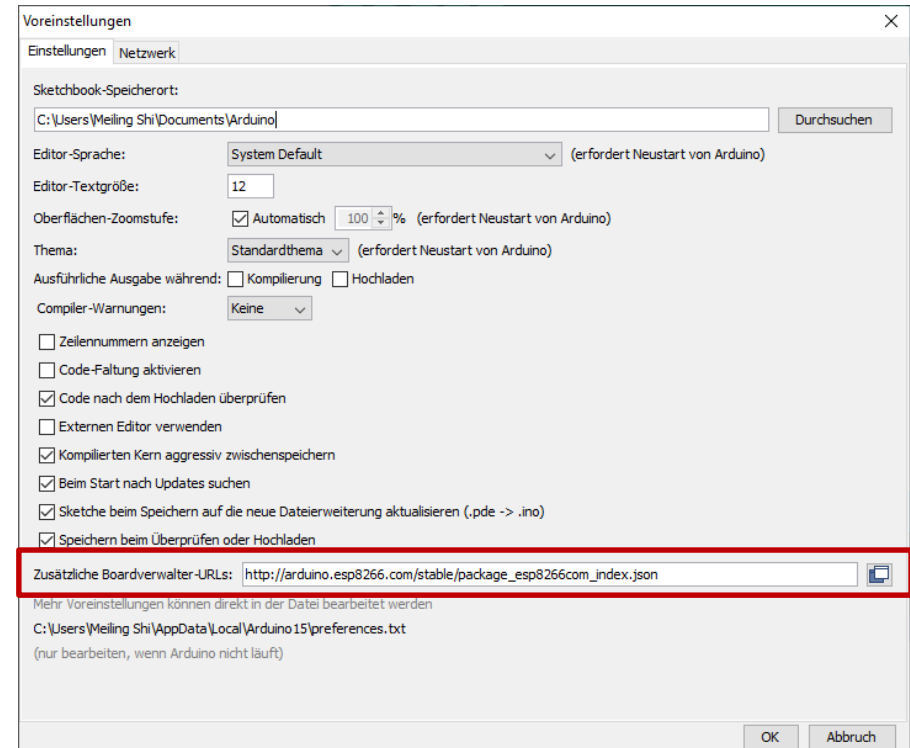
Basiert auf Processing und
erleichtert den Zugang zur
Programmierung mit
Mikrocontrollern

- `setup()` : beim Start einmal aufgerufen
- `loop()` : durchgehend immer wieder durchlaufen solange der Board an ist



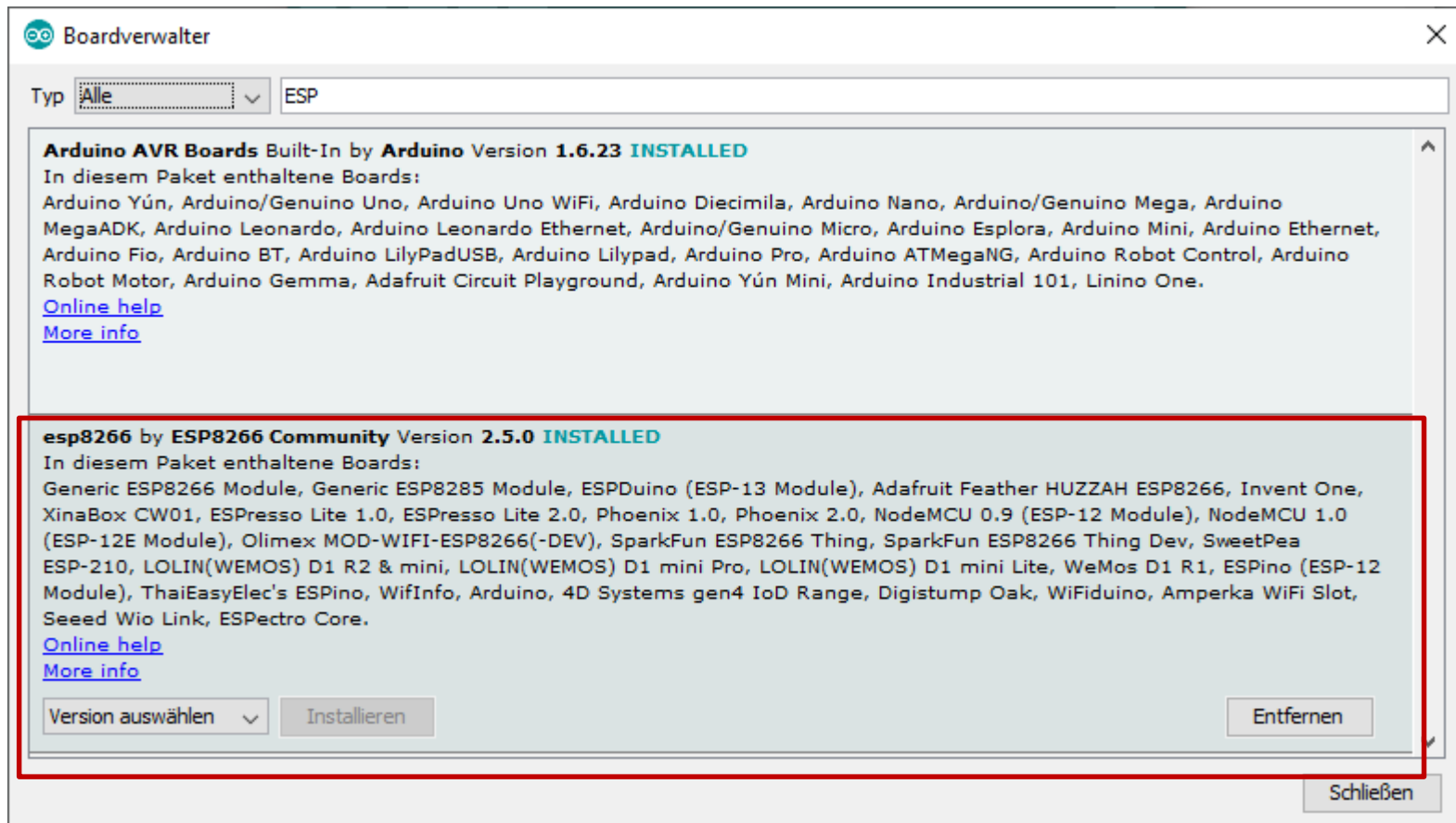
Installation - Arduino

- Download:
<https://www.arduino.cc/en/Main/Software>
- Datei-> Voreinstellung->Zusätzliche Boardverwalter-URLs:
http://arduino.esp8266.com/stable/package_esp8266com_index.json eingeben



Installation - Arduino

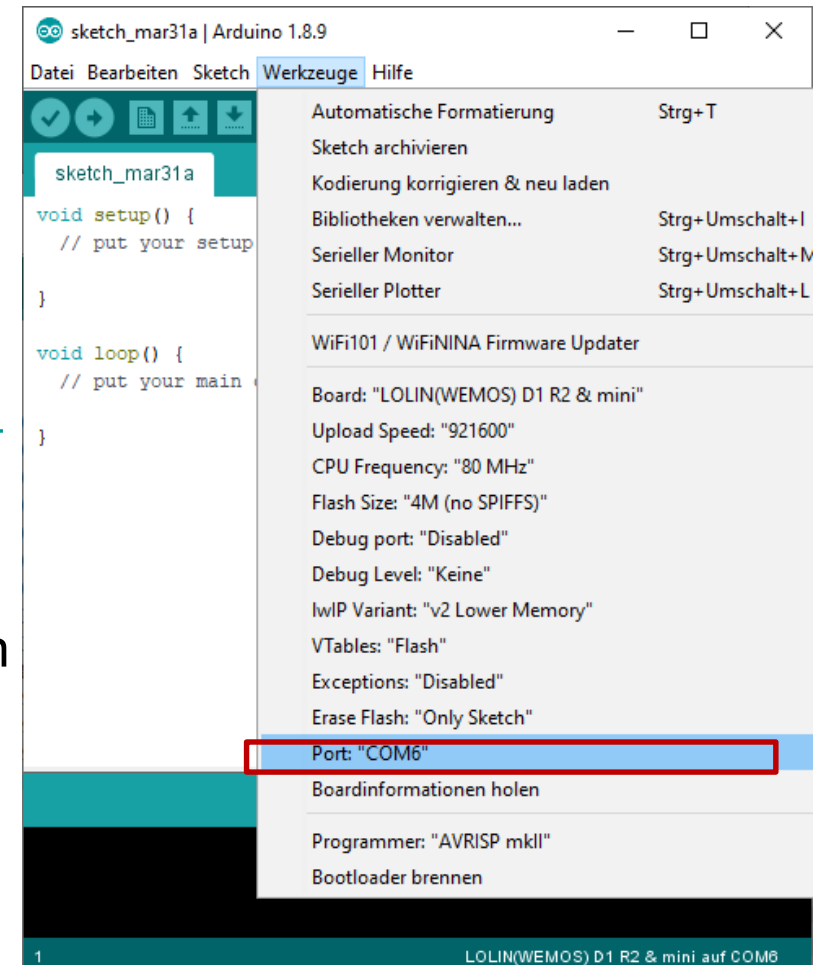
Werkzeuge->Boardverwalter->esp8266 installieren



Installation- Wemos Treiber

Treiber für Microcontroller Wemos herunterladen:

- <https://wiki.wemos.cc/downloads> (Windows)
- http://www.wch.cn/download/CH341SER_ZIP.html (wenn die erste nicht funktionieren sollte)
- Dann sollte eine „Port“ erkannt werden



Installation – Bib in Arduino

Tools -> Manage Libraries ->

- Mysql Connector Arduino
- WifiManager
- NTP Client
- ArduinoJson
- DHT sensor library
- ESP8266 Microgear

- Lokale Netzwerk erstellen durch Hotspot (mit Laptop oder Handy)
- Verbinden Mikrokontroller und dein Laptop mit gleichen Netzwerk (Hotspot)
- Finden die **IP Adresse** deiner Laptop im lokalen Netzwerk: <https://kb.netgear.com/20878/Finding-your-IP-address-without-using-the-command-prompt>
 - Meistens : http://192.168.137.1
- Mikrocontroller „brennen“
- Flask Applikation starten

Skript: Mikrocontroller mit WLAN verbinden



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
const long utcOffsetInSeconds = 3600;
// A UDP instance to let us send and receive packets over UDP
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org", utcOffsetInSeconds); //wifi nötig

void setup()
{
  // put your setup code here, to run once:
  Serial.begin(115200);
  WiFi.begin("testiibesp", "12345678");
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(1000);
    Serial.println("Connecting..");
  }
  // timeClient.begin();
  Serial.println("Connected to WiFi Network");
}
```

Information der Hotspot anpassen

Skript: Daten über lokale Netzwerk übertragen

Laptop IP-Adresse im lokalen Netzwerk, Ports 8090 muss eventuell in Firewall freigeschaltet werden

.....

```
HTTPClient http: //Declare object of class HTTPClient
http.begin("http://192.168.137.1:8090/postjson"); //Specify request destination
http.addHeader("Content-Type", "application/json"); //Specify content-type header
int httpCode = http.POST(JSONmessageBuffer); //Send the request
String payload = http.getString(); //Get the
Serial.println(httpCode); //Print HTTP return code
Serial.println(payload); //Print request response payload
http.end(); //Close connection
} else {
    Serial.println("Error in WiFi connection");
}
delay(10000); //Send a request every 10 seconds
}
```

Ports freischalten in Win10:

<https://www.tomshardware.com/news/how-to-open-firewall-ports-in-windows-10,36451.html>

Anzeige in Serial Monitor



TECHNISCHE
UNIVERSITÄT
DARMSTADT

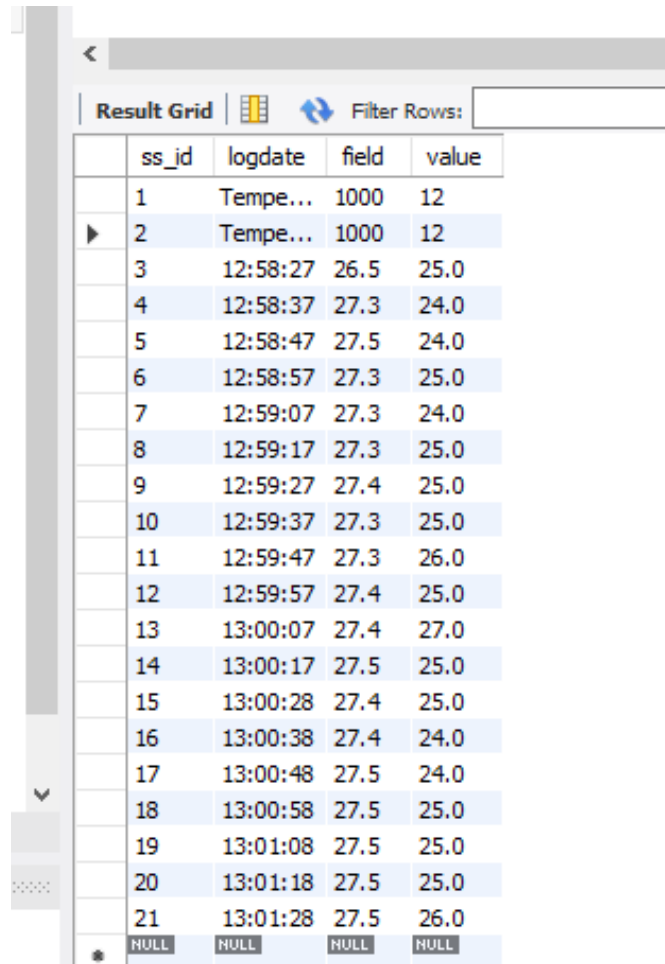
COM6

JSON posted
Temperature = 27.50
Humidity = 24.00
{
 "time": "12:58:47",
 "field": 27.50,
 "value": 24.00
}
200
JSON posted
Temperature = 27.30
Humidity = 25.00
{
 "time": "12:58:57",
 "field": 27.30,
 "value": 25.00
}

```
const long utcOffsetInSeconds = 3600;  
// A UDP instance to let us send and receive packets over UDP  
WiFiUDP ntpUDP;  
NTPClient timeClient(ntpUDP, "pool.ntp.org", utcOffsetInSeconds); //wifi nötig  
  
void setup()  
{  
  // put your setup code here, to run once:  
  Serial.begin(115200);  
  WiFi.begin("testiibes", "12345678");  
  while (WiFi.status() != WL_CONNECTED)  
  {  
    delay(1000);  
    Serial.println("Connecting..");  
  }  
  // timeClient.begin();  
  Serial.println("Connected to WiFi Network");  
}
```

☒ Autoscroll ☐ Show timestamp Both NL & CR 115200 baud Clear output

Resultat in MySQL



	ss_id	logdate	field	value
1		Tempe...	1000	12
2		Tempe...	1000	12
3		12:58:27	26.5	25.0
4		12:58:37	27.3	24.0
5		12:58:47	27.5	24.0
6		12:58:57	27.3	25.0
7		12:59:07	27.3	24.0
8		12:59:17	27.3	25.0
9		12:59:27	27.4	25.0
10		12:59:37	27.3	25.0
11		12:59:47	27.3	26.0
12		12:59:57	27.4	25.0
13		13:00:07	27.4	27.0
14		13:00:17	27.5	25.0
15		13:00:28	27.4	25.0
16		13:00:38	27.4	24.0
17		13:00:48	27.5	24.0
18		13:00:58	27.5	25.0
19		13:01:08	27.5	25.0
20		13:01:18	27.5	25.0
21		13:01:28	27.5	26.0
	NULL	NULL	NULL	NULL

Kochrezept für eine Webanwendung

- ERM
- Datenbank erstellen.
- Verbinden Datenbank mit Python Anwendung durch mysql-connector:
- Hilfsklassen z.B. für immer wieder auftretende Datenbankfunktionen
- Datenbankspezifischen Pythoncode auslagern!
- Keine manuelle Eingabe von IDs!
- Dynamische Webseiten mit Flask erstellen, Interaktion mit DB
- Sensordaten über Internet übertragen und in DB speichern
- (die Demo-Code darf 1-1 verwendet werden!!!!)

- Ende der Vorlesungszeit Mitte July
- Abends gegen 18:00
- Snacks und Getränke

