WBE: JAVASCRIPT

WEBSERVER

ÜBERSICHT

- Internet-Protokolle
- Das HTTP-Protokoll
- Node.js Webserver
- REST APIs
- Express.js

ÜBERSICHT

- Internet-Protokolle
- Das HTTP-Protokoll
- Node.js Webserver
- REST APIs
- Express.js

INTERNET

- Netzwerk von Internet-Geräten
- Internet-Protokoll-Stack (TCP/IP, ...)

ISO/OSI	Internet	Protokolle	typische Angaben
Application (Anwendung)	Application	HTTP FTP SMTP Telnet	URL: http://www.zhwin.ch Mailadresse: mustepet@zhwin.ch
Presentation (Darstellung)			
Session (Sitzung)			
Transport	Transport	TCP UDP	Portnummer 80 = HTTP 25 = SMTP
Network (Netzwerk)	Internet	IP	IP-Adresse 192.168.0.1
Data Link (Sicherung)	Physical / Access	Ethernet Wireless LAN Token Ring PPP/(Modem, ISDN, xDSL)	MAC-Adresse 00:0F:7F:23:45:67 Telefonnummer: 0878/123456
Physical (Bitübertragung)			

SERVER IM INTERNET

- Wartet auf Anfragen auf bestimmtem Port
- Client stellt Verbindung her und sendet Anfrage
- Server beantwortet Anfrage

Port	Service
20	FTP -- Data
21	FTP -- Control
22	SSH Remote Login Protocol
23	Telnet
25	Simple Mail Transfer Protocol (SMTP)
53	Domain Name System (DNS)
80	HTTP
443	HTTPS

WEBSERVER

- Wartet auf HTTP/S-Anfragen
- Standard-Ports: 80, 443
- Beispiele: Nginx, Apache Webserver, Apache Tomcat

```
GET /index.html HTTP/1.1
```

DAS WEB (WH)

- Client: Browser (oder allgemein: User Agent)
- Server: Web Server

- Protokoll: HTTP/S
- Sprachen: HTML, CSS, ...
- Adressierung: URL/URI

```
 http://eloquentjavascript.net/13_browser.html
|      |                        |              |
protocol       server               path
```

SERVER AN DER ZHAW

https://dublin.zhaw.ch/~<kurzzeichen>

- Laborserver: CGI, PHP, MySQL, Postgres
- Zugang nur noch innerhalb des ZHAW-Netzes (oder VPN)

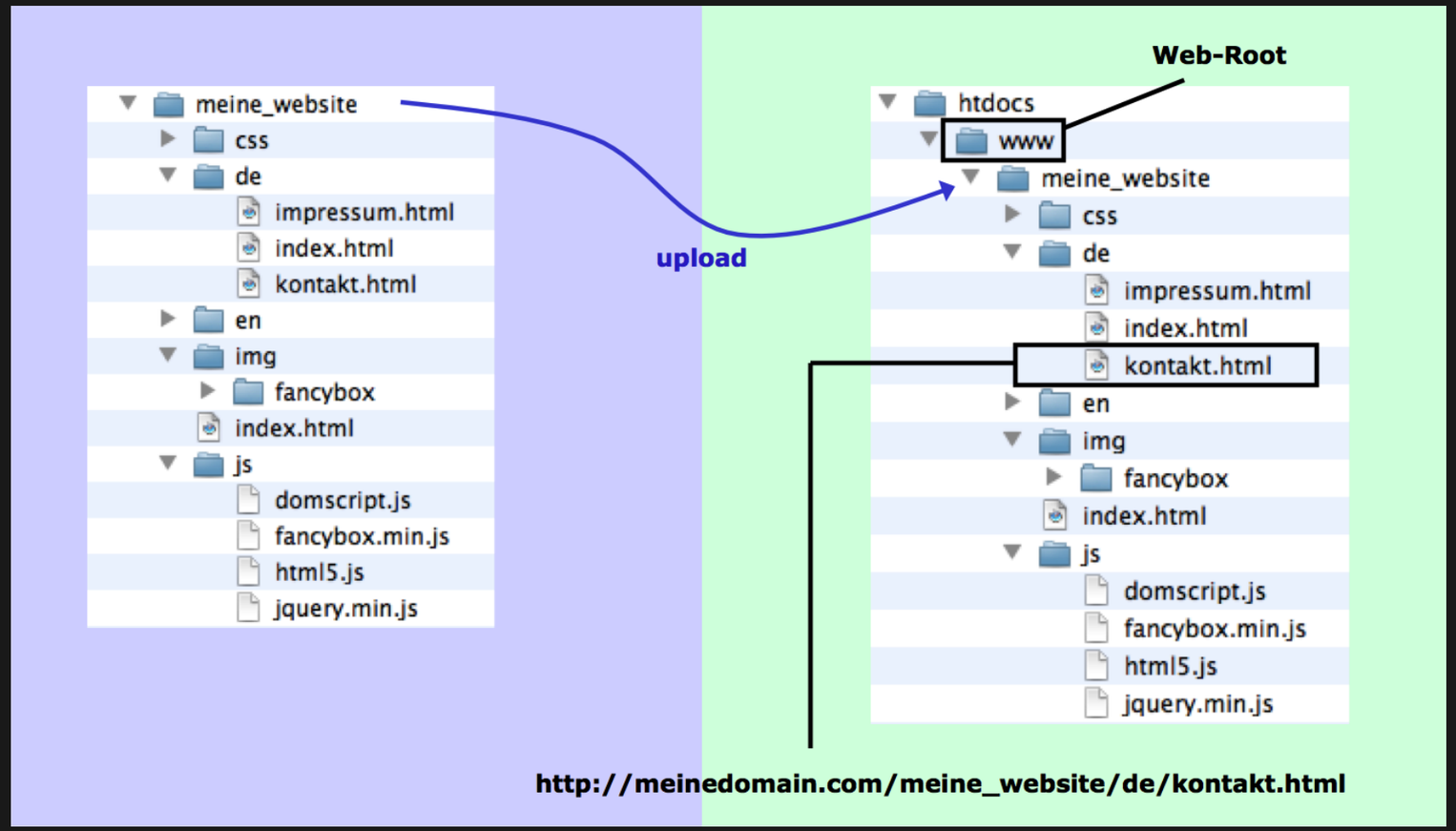
https://github.zhaw.ch/

- Github Pages

Zahlreiche weitere Labor- und Test-Server für bestimmte Aufgaben, teilweise von ausserhalb des ZHAW-Netzes erreichbar, teilweise nur über VPN

WEB-ROOT

- Einstellung des Web-Servers
- Stelle im Server-Verzeichnis, welche Wurzel des Web-Verzeichnisses ist



FILE-TRANSFER

- FTP (File Transfer Protocol)
- SFTP (SSH File Transfer Protocol)
- Anwendungen mit GUI und auf der Kommandozeile (ftp, sftp)

```
$ sftp bkrt@dublin.zhaw.ch
password:
Connected to dublin.zhaw.ch.

sftp> dir
ggt.py      ggt.pyc     index.html  private     public      www

sftp> get ggt.py
Fetching /home/staff/bkrt/ggt.py to ggt.py
/home/staff/bkrt/ggt.py
100%   58     0.1KB/s   00:00

sftp> quit
$
```

SECURE SHELL: SSH

- Sichere Verbindung zum Server herstellen
- Dort auf der Kommandozeile arbeiten

```
$ ssh dublin.zhaw.ch -l bkrt
bkrt@dublin.zhaw.ch's password:
Last login: Tue Jul 16 13:47:05 2013 from ...

$ ls
ggt.py  ggt.pyc  index.html  ine1  private  public  www

$ cd www
$ mv index.html old.html
$ exit
```

ÜBERSICHT

- Internet-Protokolle
- Das HTTP-Protokoll
- Node.js Webserver
- REST APIs
- Express.js

HTTP

Aufruf http://dublin.zhaw.ch/~bkrt/hallo.html im Browser

- DNS-Abfrage `dublin.zhaw.ch`
- Liefert IP-Adresse, z.B.: `160.85.67.138`
- Verbindung zu Host auf Port 80 herstellen
- HTTP-Anfrage senden: `GET /~bkrt/hallo.html HTTP/1.1`
- Server sendet Antwort und beendet Verbindung

HTTP REQUEST

```
GET /~bkrt/hallo.html HTTP/1.1
Host: dublin.zhaw.ch
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X...) Gecko/20100101 Firefox
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

HTTP REQUEST: METHODEN

- `GET`: Ressource laden
- `POST`: Informationen senden
- `PUT`: Ressource anlegen, überschreiben
- `PATCH`: Ressource anpassen
- `DELETE`: Ressource löschen ...

https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods

HTTP RESPONSE

```
HTTP/1.1 200 OK
Date: Mon, 15 Jul 2013 17:10:56 GMT
Server: Apache/2.2.15 (CentOS)
Last-Modified: Wed, 17 Oct 2012 08:10:22 GMT
ETag: "5b018a-af-4cc3ccd575780"
Accept-Ranges: bytes
Content-Length: 175
Connection: close
Content-Type: text/html; charset=UTF-8

<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Hallo</title>
  </head>
  <body>
    <h1>Hallo</h1>
    <p>Ich bin eine Webseite</p>
  </body>
</html>
```

HTTP RESPONSE: STATUS CODES

- `1XX`: Information (z.B. 101 Switching Protocols)
- `2XX`: Erfolg (z.B. 200 Ok, 204 No Content)
- `3XX`: Weiterleitung (z.B. 301 Moved Permanently)
- `4XX`: Fehler in Anfrage (z.B. 403 Forbidden, 404 Not Found)
- `5XX`: Server-Fehler (z.B. 501 Not Implemented)

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

ÜBERSICHT

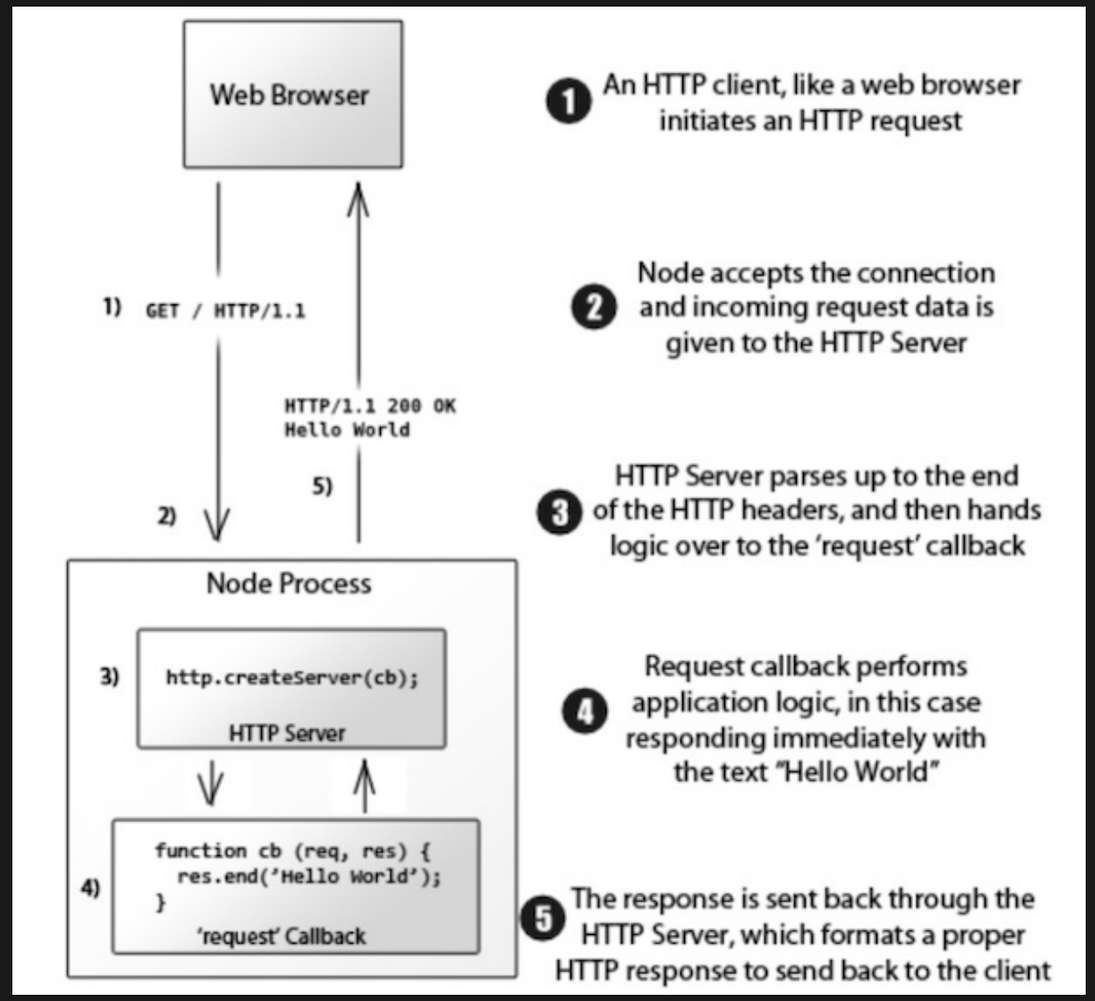
- Internet-Protokolle
- Das HTTP-Protokoll
- Node.js Webserver
- REST APIs
- Express.js

EINFACHER WEBSERVER

```
const {createServer} = require("http")

let server = createServer((request, response) => {
  response.writeHead(200, {"Content-Type": "text/html"})
  response.write(`
    <h1>Hello!</h1>
    <p>You asked for <code>${request.url}</code></p>`)
  response.end()
})
server.listen(8000)
console.log("Listening! (port 8000)")
```

EINFACHER WEBSERVER



EINFACHER WEB-CLIENT

```
const {request} = require("http")

let requestStream = request({
  hostname: "eloquentjavascript.net",
  path: "/20_node.html",
  method: "GET",
  headers: {Accept: "text/html"}
}, response => {
  console.log("Server responded with status code", response.statusCode)
})

requestStream.end()
```

NODE.JS WEB-CLIENT

- Einfache Variante mit `http`-Modul (letztes Beispiel)
- Paket `https` für HTTPS-Zugriffe
- Seit Node.js 18 wird auch die Fetch API unterstützt (mehr dazu beim Thema „Client-Server-Interaktion”)
- Alternative: Axios, HTTP-Client für Browser und Node.js

STREAMS: SERVER

```
const {createServer} = require("http")

createServer((request, response) => {
  response.writeHead(200, {"Content-Type": "text/plain"})
  request.on("data", chunk =>
    response.write(chunk.toString().toUpperCase()))
  request.on("end", () => response.end())
}).listen(8000)
```

- Eingehende Daten als Stream gelesen
- `data`-Event: nächster Teil verfügbar
- `end`-Event: alle Daten wurden übertragen

STREAMS: CLIENT

```
const {request} = require("http")

let rq = request({
  hostname: "localhost",
  port: 8000,
  method: "POST"
}, response => {
  response.on("data", chunk =>
    process.stdout.write(chunk.toString()));
})

rq.write("Hello server\n")
rq.write("And good bye\n")
rq.end()
```

BEISPIEL: FILE-SERVER (1)

(Exkurs)

- Kleiner Server zum Zugriff auf Files
- HTTP-Methoden `GET`, `DELETE` und `PUT`
- Im Unterricht nur kurzer Überblick über Funktionsweise
- Kompletter Code in Demos, Erklärungen in Lecture Notes
- Beispiel, wie bestimmte Features umgesetzt werden können
- Achtung: nicht für produktiven Einsatz im Web geeignet

BEISPIEL: FILE-SERVER (2)

```
1  const {createServer} = require("http")
2  const methods = Object.create(null)
3
4  createServer((request, response) => {
5    let handler = methods[request.method] || notAllowed;
6    handler(request)
7      .catch(error => {
8        if (error.status != null) return error
9        return { body: String(error), status: 500 }
10     })
11     .then(({body, status=200, type="text/plain"}) => {
12       response.writeHead(status, {"Content-Type": type})
13       if (body && body.pipe) body.pipe(response)
14       else response.end(body)
15     })
16 }).listen(8000)
```

BEISPIEL: FILE-SERVER (3)

```
async function notAllowed (request) {
  return {
    status: 405,
    body: `Method ${request.method} not allowed.`
  }
}
```

- Unbekannter Handler
- `notAllowed` (405) senden

BEISPIEL: FILE-SERVER (4)

```
const {parse} = require("url")
const {resolve, sep} = require("path")

const baseDirectory = process.cwd()

function urlPath (url) {
  let {pathname} = parse(url)
  let path = resolve(decodeURIComponent(pathname).slice(1))
  if (path != baseDirectory && !path.startsWith(baseDirectory + sep)) {
    throw {status: 403, body: "Forbidden"}
  }
  return path
}
```

BEISPIEL: FILE-SERVER (5)

```
const {createReadStream} = require("fs")
const {stat, readdir} = require("fs").promises
const mime = require("mime")

methods.GET = async function (request) {
  let path = urlPath(request.url)
  let stats
  try {
    stats = await stat(path)
  } catch (error) {
    if (error.code != "ENOENT") throw error
    else return {status: 404, body: "File not found"}
  }
  if (stats.isDirectory()) {
    return {body: (await readdir(path)).join("\n")}
  } else {
    return {body: createReadStream(path),
            type: mime.getType(path)}
  }
}
```

BEISPIEL: FILE-SERVER (6)

```
const {rmdir, unlink} = require("fs").promises

methods.DELETE = async function (request) {
  let path = urlPath(request.url)
  let stats
  try {
    stats = await stat(path)
  } catch (error) {
    if (error.code != "ENOENT") throw error
    else return {status: 204}
  }
  if (stats.isDirectory()) await rmdir(path)
  else await unlink(path)
  return {status: 204}
}
```

BEISPIEL: FILE-SERVER (7)

```
const {createWriteStream} = require("fs");

function pipeStream (from, to) {
  return new Promise((resolve, reject) => {
    from.on("error", reject)
    to.on("error", reject)
    to.on("finish", resolve)
    from.pipe(to)
  })
}

methods.PUT = async function (request) {
  let path = urlPath(request.url)
  await pipeStream(request, createWriteStream(path))
  return {status: 204}
}
```

BEISPIEL: FILE-SERVER (8)

Test des Servers:

```
$ curl http://localhost:8000/file.txt
File not found
$ curl -X PUT -d hello http://localhost:8000/file.txt
$ curl http://localhost:8000/file.txt
hello
$ curl -X DELETE http://localhost:8000/file.txt
$ curl http://localhost:8000/file.txt
File not found
```

https://eloquentjavascript.net/20_node.html#h_yAdw1Y7bgN

ÜBERSICHT

- Internet-Protokolle
- Das HTTP-Protokoll
- Node.js Webserver
- REST APIs
- Express.js

REST APIS

- REST: Representational State Transfer
- Programmierparadigma für verteilte Systeme
- Grundlage: Web-Architektur und HTTP
- Leichtgewichtig (im Vergleich zu RPC oder SOAP/WSDL)

REST EIGENSCHAFTEN

- Zugriff auf Ressourcen über ihre Adresse (URI)
- Kein Zustand: jede Anfrage komplett unabhängig
- Kein Bezug zu vorhergehenden Anfragen
- Alle nötigen Informationen in Anfrage enthalten
- Verwenden der HTTP-Methoden: `GET`, `PUT`, `POST`, ...

RESTFUL APIS

- Basisadresse, z.B. http://example.com/api/
- Sammlung von Ressourcen, z.B. http://example.com/api/products/
- Einzelne Ressource, z.B. http://example.com/api/products/17
- Medientyp für Ressource/n, z.B. JSON
- Zulässige Operationen, z.B. GET, PUT, POST, or DELETE

HTTP-METHODEN IN RESTFUL APIS

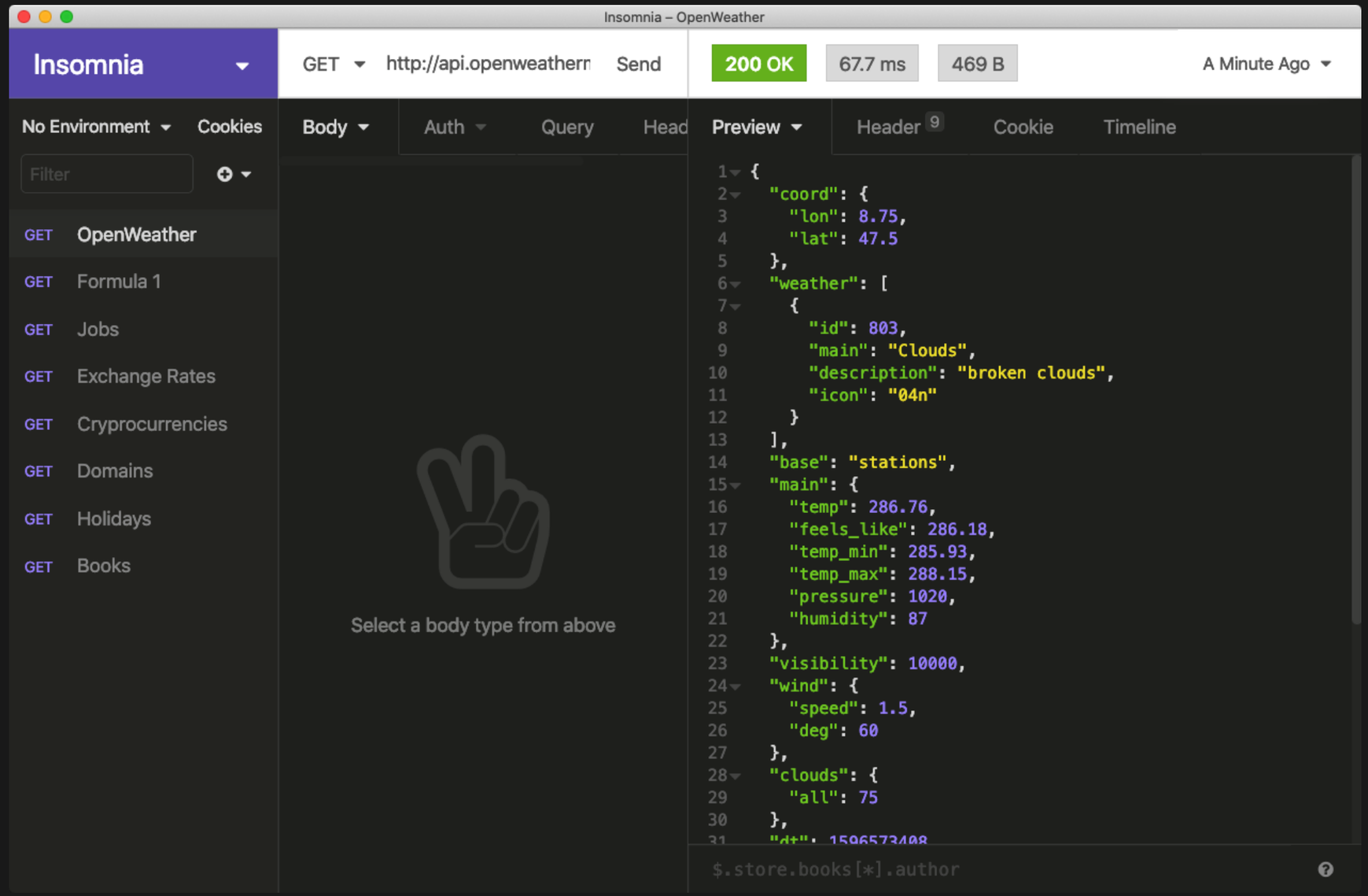
HTTP-Methode	Sammlung (Collection)	Einzel-Ressource
GET	Repräsentation für die Collection laden	Repräsentation für die Ressource laden
POST	Ressource unterhalb der angegebenen anlegen	Ressource in der angegebenen anlegen
PUT	Sammlung ersetzen oder anlegen	Ressource ersetzen oder anlegen
DELETE	Löscht die angegebene Sammlung	Löscht die angegebene Ressource
PATCH	Sammlung anpassen oder anlegen	Ressource anpassen oder anlegen

REST APIS

- Viele Services stellen REST-APIs zur Verfügung
- In der Regel natürlich nur GET-Requests
- Beispiel: OpenWeather (Registrierung erforderlich)

```
$ curl "http://api.openweathermap.org/data/2.5/weather?q=Winterthur,ch&appid=674..."
{"coord":{"lon":8.75,"lat":47.5},"weather":[{"id":803,"main":"Clouds","description":
"broken clouds","icon":"04n"}],"base":"stations","main":{"temp":286.76,
"feels_like":286.18, "temp_min":285.93,"temp_max":288.15,"pressure":1020,...}}
```

REST TOOLS



https://insomnia.rest

REST-ALTERNATIVE: GRAPHQL

```
{
  hero {
    name
    friends {
      name
    }
  }
}
```

- Neueres Konzept, Facebook 2015
- Anfragesprache mit mächtigeren Auswahlmöglichkeiten
- Reihe von Werkzeugen zu diesem Zweck
- Im Beispiel: liefere alle `hero`-Einträge mit `name` und `friends`, von diesen aber auch nur `name`

ÜBERSICHT

- Internet-Protokolle
- Das HTTP-Protokoll
- Node.js Webserver
- REST APIs
- Express.js

EXPRESS.JS

- Minimales, flexibles Framework für Web-Apps
- Zahlreiche Utilities und Erweiterungen
- Grundlage: Node.js
- Grundlage für zahlreiche weitere Frameworks

http://expressjs.com

INSTALLATION

```
$ mkdir myapp
$ cd myapp
$ npm init
$ npm install express --save
```

- Der Schritt `npm init` fragt eine Reihe von Informationen (Projektname, Version, ...) zum Projekt ab
- Als *Entry Point* ist hier `index.js` voreingestellt
- Das kann zum Beispiel in `app.js` geändert werden.

HELLO WORLD

```
1  const express = require('express')
2  const app = express()
3  const port = 3000
4
5  app.get('/', (req, res) => {
6    res.send('Hello World!')
7  })
8
9  app.listen(port, () => {
10   console.log(`Example app listening at http://localhost:${port}`)
11 })
```

EXPRESS APP GENERATOR

- App-Gerüst mit häufig benötigten Komponenten anlegen
- Schnelle Variante zum Projektstart

```
app
├── app.js
├── bin
│   └── www
├── package.json
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └── style.css
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── error.jade
    ├── index.jade
    └── layout.jade
```

```
# Hilfetext ausgeben
npx express-generator -h

# Generator starten
npx express-generator
```

http://expressjs.com/en/starter/generator.html

ROUTING

```
app.get('/', function (req, res) {
  res.send('Hello World!')
})
app.post('/', function (req, res) {
  res.send('Got a POST request')
})
app.put('/user', function (req, res) {
  res.send('Got a PUT request at /user')
})
app.delete('/user', function (req, res) {
  res.send('Got a DELETE request at /user')
})
```

http://expressjs.com/en/guide/routing.html

STATISCHE DATEIEN

- Middleware `express.static`
- Pfadangabe für Dateien als erstes Argument

```
1 app.use(express.static('public'))
2 /* http://localhost:3000/css/style.css
3 /* Pfad zur Datei: public/css/style.css
4 */
5 app.use('/static', express.static('public'))
6 /* http://localhost:3000/static/css/style.css
7 /* Pfad zur Datei: public/css/style.css
8 */
```

MIDDLEWARE

- Funktionen mit Zugriff auf `request` und `response`
- Express-App ist eigentlich eine Folge von Middleware-Aufrufen

```
app.use(function (req, res, next) {
  console.log('Time:', Date.now())
  next()
})

app.use('/user/:id', function (req, res, next) {
  console.log('Request Type:', req.method)
  next()
})
```

MIDDLEWARE

Module	Description
body-parser	Parse HTTP request body
compression	Compress HTTP responses
cookie-parser	Parse cookie header and populate req.cookies
cors	Enable cross-origin resource sharing (CORS)
passport	Authentication using “strategies” such as OAuth
…	…

http://expressjs.com/en/resources/middleware.html
http://www.passportjs.org

SERVER MIT NPM STARTEN

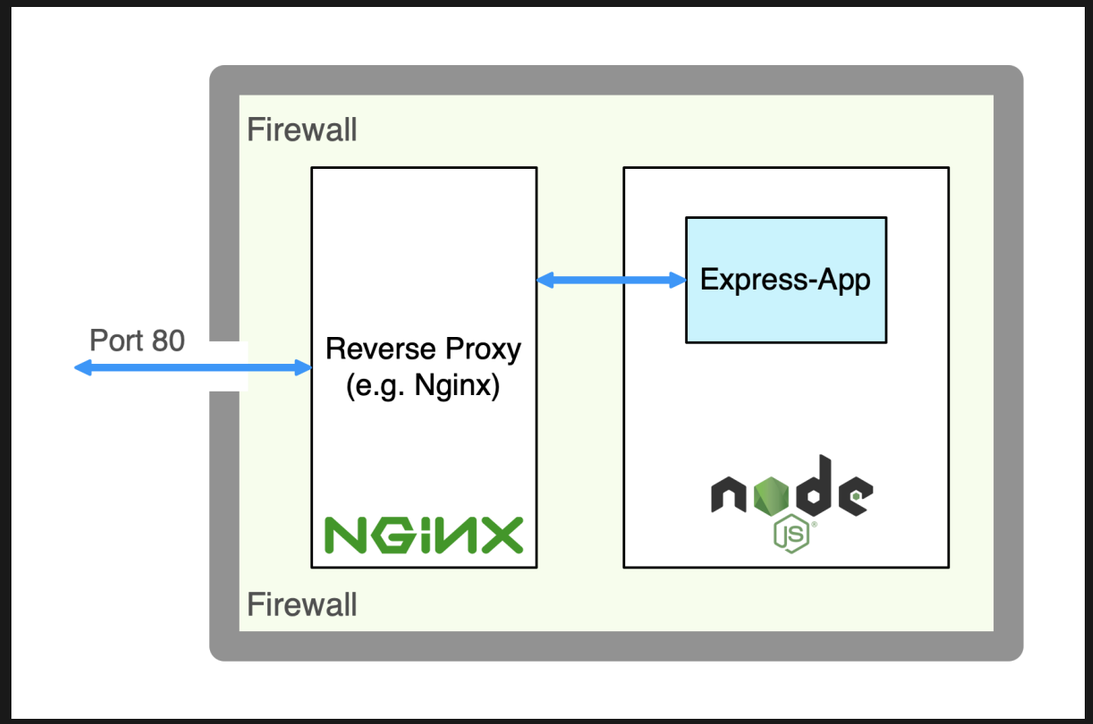
Eintrag in `package.json`:

```
"scripts": {
  "start": "node ./express_server.js"
}
```

Der Server kann dann so gestartet werden:

```
$ npm start
```

REVERSE PROXY



- Express-App wird übers Internet nicht direkt angesprochen
- Zugang erfolgt über Reverse Proxy, z.B. ein nginx-Server
- Dieser leitet Anfragen an die Express-App weiter
- Zusätzliche Services: Fehlerseiten, Komprimierung, Cache

QUELLEN

- Marijn Haverbeke: Eloquent JavaScript
  https://eloquentjavascript.net/
- Ältere Slides aus WEB2 und WEB3
- Dokumentationen, u.a. zu Node.js

LESESTOFF

Geeignet zur Ergänzung und Vertiefung

- Einzelne Abschnitte in Kapitel 20 von:
  Marijn Haverbeke: Eloquent JavaScript
  https://eloquentjavascript.net/

Stand:
19.10.2024 16:20