

A thick red vertical bar is positioned on the left side of the slide.

# Webbasierte Anwendungen

## Node.js

Prof. Dr. Ludger Martin

# Content

---

- ◆ Introduction
- ◆ Web Server
- ◆ Promise
- ◆ Database Access

# Introduction

---

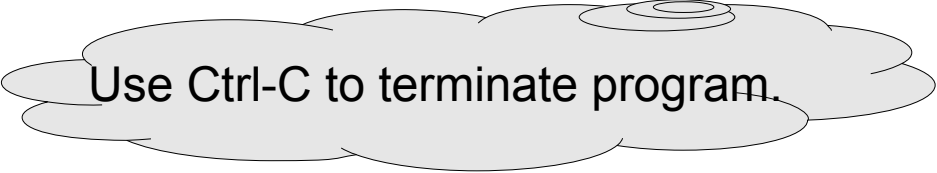
- ◆ Node.js is a JavaScript based runtime environment outside of a browser.
- ◆ Based on V8 JavaScript execution engine, initially built for Google Chrome
- ◆ Also usable as Web server. It does not need Apache.
- ◆ Allows access to databases.
- ◆ <https://nodejs.org>
- ◆ <https://expressjs.com> (Module for Web server)

# Introduction

---

- ♦ Call Node.js program

```
node helloworld.js
```

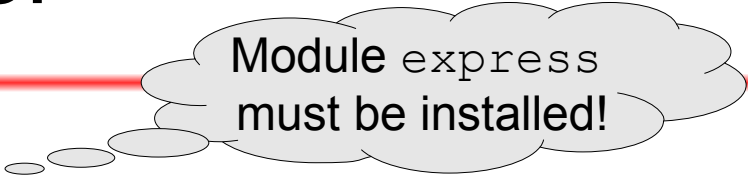


Use Ctrl-C to terminate program.

# Web Server

- ◆ Import Module express

```
const express = require('express');
```




Module express  
must be installed!

- ◆ Create express app

```
let app = express();
```

- ◆ Bind and listen for connection. Every port on a server can be used only once.

```
app.listen([port[, host[, backlog]]]  
          [, callback]);
```



Callback on success

# Web Server

---

## ♦ Example

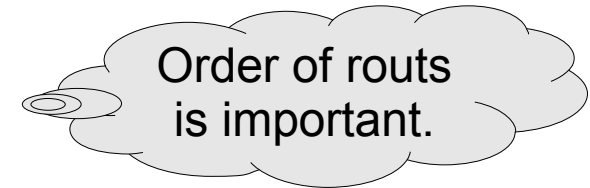
```
app.listen(3000, function () {  
    console.log(  
        'Listening on port 3000!');  
});
```

# Web Server

- ◆ **Routing** refers to how an application's endpoints (URIs) respond to client requests.

- ◆ HTTP GET Requests

```
app.get(path, callback);
```



- ◆ HTTP POST Requests

```
app.post(path, callback);
```

**optional for parsing** `application/x-www-form-urlencoded`  

```
app.use(express.urlencoded({extended: true}));
```

- ◆ HTTP PUT and DELETE similar

- ◆ `app.all(path, callback);` – matches all HTTP methods

# Web Server

## ◆ Callback Function



## ◆ Request Object

★ Use `req.body.variable` to access a post variable



# Web Server

---

## ◆ Response Object

- ★ Sets the HTTP status for the response.

`res.status(code)`

- ★ Sets the Content-Type HTTP header.

`res.type(type)`

- ★ Sends the HTTP response.

`res.send(body)`

- ★ Transfers the file at the given path. Content-Type based on filename's extension.

`res.sendFile(path)`

# Web Server

---

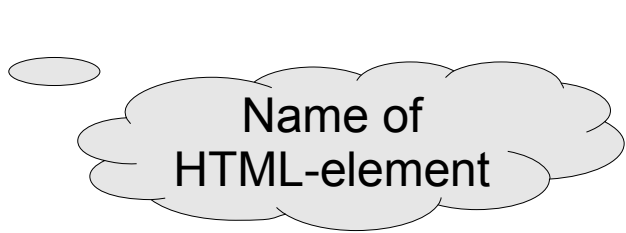
## ♦ Example

```
app.get('/', function (req, res) {  
    res.sendFile('/path/index.html');  
});
```

# Web Server

## ◆ Example

```
app.use(express.urlencoded({extended: true}));  
app.post('/', function (req, res, next) {  
  res.set('Content-Type', 'text/html');  
  res.send('<!DOCTYPE html>...' +  
    '<p>Your input was ' +  
    req.body.text +  
    '</p>...');  
});
```



Name of  
HTML-element

# Promise

---

- ◆ New in ECMAScript 2015 (ES6)
- ◆ This lets asynchronous methods return values like synchronous methods
- ◆ JavaScript Engine has one task-queue and one micro task queue.
- ◆ After every event in the task queue all entries in the micro task queue are processed.
- ◆ Promises are micro tasks.

# Promise

- ◆ A promise is in one of these states:
  - ★ **pending**: initial state
  - ★ **fulfilled**: operation was completed successfully
  - ★ **rejected**: operation failed
- ◆ Created with `Promise` constructor. A function is passed to initiate a task. Function calls callbacks `resolve` or `reject`.
  - ★ `resolve(result)` – task is fulfilled
  - ★ `reject(error)` – task is rejected

# Promise

---

- ◆ The following methods are used to associate further action with a promise.
  - ★ `then(onFulfill, onReject)`
  - ★ `catch(onReject)`
  - ★ `finally(onFinally)`
  - ★ As these methods return promises, they can be chained.

# Promise

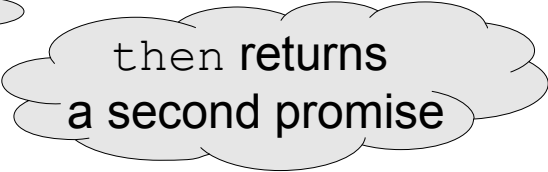
## ◆ Example

```
let promise = new Promise(  
  function (resolve, reject) {  
    // some functionality with a result  
    if (/* successful? */) {  
      resolve(result);  
    } else {  
      reject("error message");  
    }  
  }  
);  
promise.then(function (val) {  
  console.log("then val:", val);  
}, function (err) {  
  console.log("then err: ", err);  
});  
promise.catch(function (err) {  
  console.log("catch err: " + err);  
});
```

# Promise

## ◆ Example chained

```
new Promise(  
  function (resolve, reject) {  
    // some functionality with a result  
    if (/* successful? */) {  
      resolve(result);  
    } else {  
      reject("error message");  
    }  
  }  
)  
.then(function (val) {  
  console.log("then val:", val);  
}, function (err) {  
  console.log("then err: ", err);  
}).catch(function (err) {  
  console.log("catch err: " + err);  
});
```



then returns  
a second promise



# Promise

- ◆ `async` and `await` introduced by ECMAScript 2017
  - ★ `async function` creates a binding of a new `async function`
  - ★ `await` keyword is permitted within the function body. Simulates `.then()` callback.
  - ★ `try { } catch (err) { }` handles reject as exception.
  - ★ Promise-based behavior to be written in a cleaner style and avoiding the need to explicitly configure promise chains.

# Promise

◆ **Example** `async/await`

```
async function p () {
  try {
    let val = await new Promise(
      function (resolve, reject) {
        // some functionality with a result
        if (/* successful? */) {
          resolve(result);
        } else {
          reject("error message");
        }
      }
    );
    console.log("then val:", val);
  } catch (err) {
    console.log("catch err: "+ err);
  }
}
```

`p();`

# Database Access

Module mariadb  
must be installed!

- ◆ Include module for MariaDB.

```
const mariadb =  
    require('mariadb');
```

- ◆ Configure database access

```
let conn = await mariadb.createConnection({  
    host: 'localhost',  
    database: 'schema',  
    user: 'username',  
    password: 'password'  
});
```

To use socket connection, replace host by  
socketPath: '/var/run/mysqld/mysqld.sock'

# Database Access

---

## ◆ Query database

```
let title = '%Pirate%';  
let sql = 'SELECT * FROM medienartikel  
          WHERE titel LIKE ?';  
let rows = await conn.query(  
    sql, [title]  
);
```

# Documentation

---

- ◆ Only a small part of Node.js presented
- ◆ Please check the documentation for more details:  
<https://nodejs.org/en/docs/>

# References

- ◆ Ackermann, P.: JavaScript – Das umfassende Handbuch, Auflage 1, Rheinwerk, 2016
- ◆ OpenJS Foundation: Node.js, <https://nodejs.org> accessed 23.06.2020
- ◆ Express Node.js web application framework, <https://expressjs.com> accessed 23.06.2020
- ◆ MariaDB: Connector/Node.js, <https://mariadb.com/docs/skysql-previous-release/connect/programming-languages/nodejs/>, accessed 15.05.2023
- ◆ mdn: Promise, [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise), accessed 23.05.2024