

RUHR-UNIVERSITÄT BOCHUM

# WEB-ENGINEERING

Sommersemester 2023



Informatik  
im Bauwesen

# Web-Engineering

## Koa.js

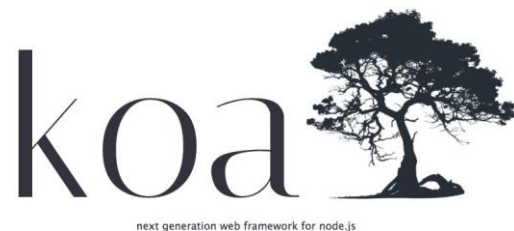
# Koa.js

Koa ist ein Web-Framework für Node.js, dass die Entwicklung moderner Webanwendungen vereinfacht

- Es ist für ES6 ausgelegt, beispielsweise mit `async & await`
- Koa zielt darauf ab kleiner, ausdrucksstärker und robuster als andere Web-Frameworks zu sein
  - Koa ist mit nur ca. 550 Zeilen Code sehr leichtgewichtig
- Eine Koa Applikation bietet einen Zusammenschluss von `Middleware Methoden`
- Ein Kerngedanke bei der Entwicklung Koa's war es high level Spracheigenschaften in einer low level middleware Umgebung zu nutzen

Kommandozeile

```
C:\Users\...> npm install koa  
C:\Users\...> npm install koa-router  
C:\Users\...> npm install koa-json
```



# Koa.js

## Welche Vorteile bringt Koa?

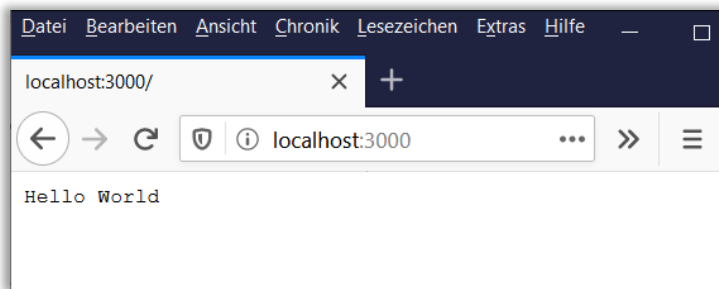
- Promise-basierter Kontrollfluss
  - Keine callback hell → `async & await`
  - Besseres error handling → `try/catch`
- Koa ist minimalistisch
  - Koa enthält nur den Middleware Kernel
  - Koa ist modular aufgebaut
  - Kontext Objekt `ctx` wird bereitgestellt  
(`ctx.request & ctx.response` ersetzen dabei NodeJS `req & res`)

# Koa.js

## Hello World in Koa

```
const Koa = require('koa');  
const app = new Koa();  
app.use(async ctx => {  
  ctx.body = 'Hello World';  
});  
app.listen('3000', () => {  
  console.log('Server lauscht auf Port: 3000');  
});
```

Schreibt „Hello World“  
über das Kontext-Objekt  
ctx in den body.



# Koa.js

## Häufig genutzte Koa Module

- `koa-router`: Ermöglicht es ein einfaches routing-Modul zu erzeugen
- `koa-bodyparser`: Ein Modul zum parsen von eintreffenden Daten → `json`, `form`, `text`
- **Templating** engines wie `koa-pug` oder `koa-views`
- Die Middleware Routinen können mit `use` zu einer selbst gestalteten `request-response pipeline` verkettet werden:

# Koa.js

```
const Koa = require('koa');  
const Router = require('koa-router');  
const bodyParser = require('koa-bodyparser');
```

```
const app = new Koa();  
const router = new Router();
```

router

```
.post('/', (ctx) => {  
  ctx.response.body = ctx.request.body;  
})  
.get('/12', (ctx) => {  
  ctx.body = 'GET request';  
});
```

Mittles des `koa-router`  
Moduls können `http-Request`  
gesendet werden.

app

```
.use(router.allowedMethods())  
.use(router.routes())
```

Einhängen der Komponenten in die `use-verkettung`.  
Ohne die `Router.allowedMethods()` Methode  
könnten `nur GET & POST` Anfragen gesendet werden.

# Koa.js

Grundsätzlich gibt es **5 verschiedene Klassen von HTTP Statuscodes**

- **1xx informational response**
  - Die Anfrage wurde **empfangen** und der Prozess fortgesetzt
- **2xx successful**
  - Die Anfrage wurde erfolgreich empfangen, verstanden und akzeptiert
- **3xx redirection**
  - Maßnahmen sind nötig um die Anforderung abzuschließen
- **4xx client error**
  - Die Anforderung enthält eine falsche Syntax oder kann nicht erfüllt werden
- **5xx server error**
  - Der Server konnte eine scheinbar gültige Anforderung nicht erfüllen

**Besonders Die Fehlermeldungen 4xx & 5xx werden häufig geworfen**



# Koa.js

## Fehler abfangen

- Innerhalb von Koa werden Fehler durch das Hinzufügen einer Middleware abgefangen
  - `try {await next()} → Fehlerbehandlung` mit Hilfe von Promises
  - Folgender Code wird als Standard Koa Fehlerbehandlung genutzt:

```
const errorMiddleware = async(ctx, next) => {
```

```
  try {
```

```
    await next();
```

Promise: **Wartet auf den reject oder resolve Status einzelner Aufrufe in der use-Verkettung.**

```
  } catch (err) {
```

```
    ctx.status = err.status || 500;
```

Der Fehlercode 500 ist der default Wert.

```
    ctx.body = err.message;
```

```
    ctx.app.emit('error', err, ctx);
```

Gibt ein Ergebnis mit einem Typ aus.

```
  }
```

```
};
```

Die Fehlerbehandlung muss als erstes Element in der **use-Verkettung** angegeben werden.

```
app.use(errorMiddleware()).use(...);
```

# Koa.js

## Fehler abfangen

- Der Standard `error handler` gibt bei einem 404 Statuscode Fehler keine Fehlermeldung aus
- Um **eigene Logik für die Ausgabe von Fehlermeldungen** zu implementieren können `event listener` verwendet werden

```
app.on('error', err => {  
  log.error('server error', err)  
});
```

- Um Fehlerbehandlung besser zu kontrollieren verfügt der Kontext in Koa über ausgeprägte `throw` und `assert` Methoden

```
app.use(async (ctx, next) => {  
  ctx.throw(  
    500, ←  
    'Error Messege'  
  );  
});
```

Notiert den Fehler und wirft einen Internal Server Error!

```
app.use(async (ctx, next) => {  
  ctx.throw(  
    400, ←  
    'Error Messege'  
  );  
});
```

Notiert den Fehler NICHT und gibt einen Antwort mit Status 400 zurück.

# Web-Engineering

## pug

# Pug

## Pug ist eine **Template Engine**

- Pug erweitert die **HTML-Funktionalitäten**
- Wird standardmäßig in **JavaScript** eingebunden
- Durch pug können Seiten je nach Datensatz dynamisch gestaltet werden
  - Ist im Gegensatz zu einfachem HTML durch Programmcode strukturierbar
  - Verfügt über Kontrollstrukturen und Schleifensyntax
  - Mixins → wiederverwendbare Blöcke von pug-Code

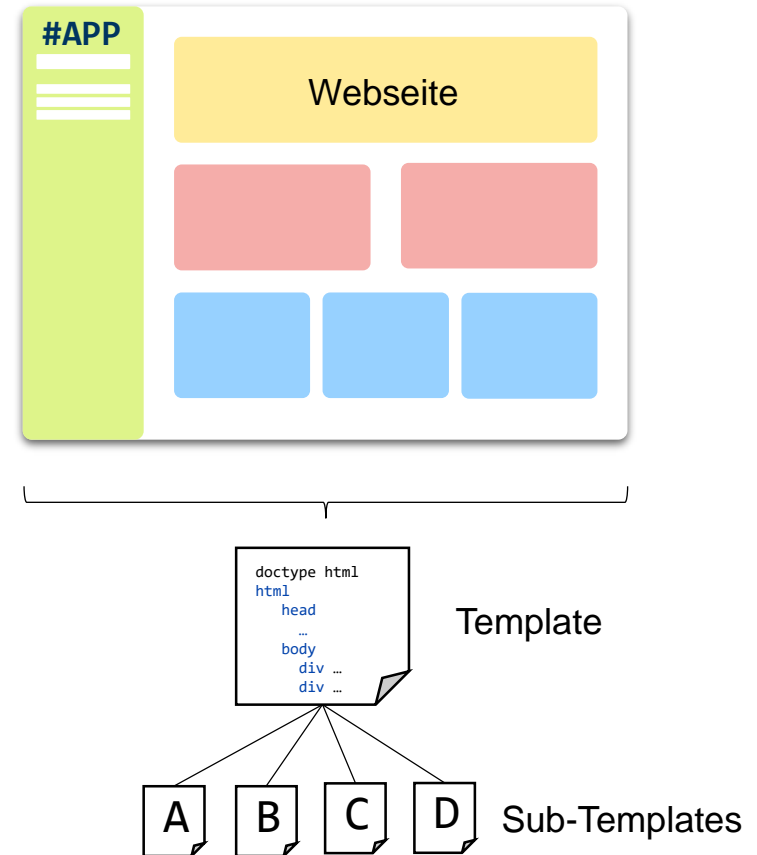


Quelle: <https://github.com/pugjs/pug>

# Pug

## Was ist eine Template?

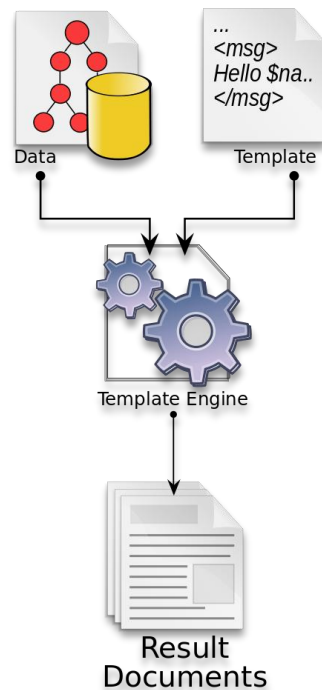
- Ein Template (de. Vorlage) ist eine Blaupause für ein spezifisches Layout, das für eine spätere Verwendung in einem Dokument gespeichert wird
- Ein Template generalisiert einen strukturierbaren Inhalt und beschreibt dessen Aufbau
- Erleichtert die Wartung von Layouts die über mehrere Seiten wiederverwendet werden
- In der Regel setzt ein Template sich aus mehreren Sub-Templates zusammen



# Pug

## Was ist eine Template Engine?

- Als Library oder Framework
  - Interpretiert Daten unter der Verwendung von Regeln
  - Rendert und erzeugt dynamisch Ansichten (`views`)
  - Ein `View` ist eine HTML Seite oder ein Teil dieser
- Pug ist eine Template Engine, welche spezialisiert ist HTML Dokumente zu generieren



Quelle:  
[https://en.wikipedia.org/wiki/Template\\_processor](https://en.wikipedia.org/wiki/Template_processor)

# Pug

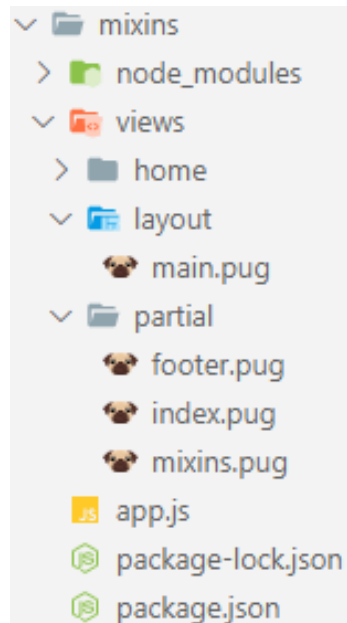
## Pug's Rolle im Node.JS Umfeld

- Pug-Dateien werden der Konvention nach in dem Unterordner `views` gespeichert
- Templates als `.pug`-Dateien angelegt
- Pug in einem Projekt einbinden:

Kommandozeile

```
C:\Users\...> npm install pug
```

Die Ordnerstruktur könnte so aussehen:



# Pug

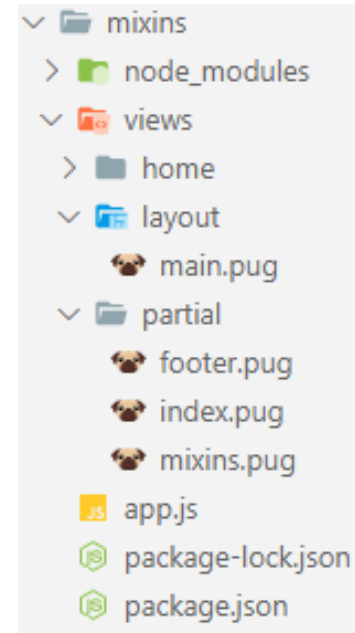
## Pug auf Seite des Servers

- Pug kann web Frameworks wie `koa` in ihrer Funktionalität erweitern
- HTML Seiten können so ohne das hinzuziehen von front-end Frameworks von dem `backend` gerendert werden

Kommandozeile

```
C:\Users\...> npm install pug
```

Die Ordnerstruktur könnte so aussehen:





# Pug

## Pug grundlegende Syntax

Innerhalb von pug müssen ...

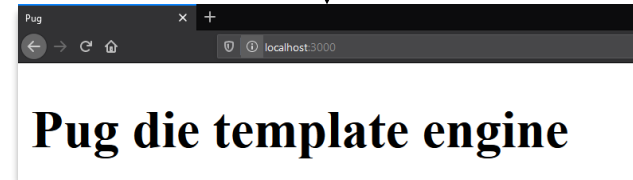
- ...keine `< >` genutzt werden
- ...Tags nicht geschlossen werden
- ...Einrückung je nach Verschachtelung gesetzt werden

Durch Einrücken wird die Verschachtelung ausgedrückt

- Durch Tab oder die Leertaste
- Die Tiefe der Einrückung darf nicht variieren

## Pug-Datei

```
doctype html
html
  → head
    → title Pug
    → body
      → h1 Pug die template Engine
```



# Pug

Es gibt mehrere Möglichkeiten JavaScript-Code in Pug zu verwenden und Variablen, Schleifen, Listen etc. zu verwenden.

- **Unbuffered**: Beginnt mit Bindestrich(-), nicht direkt Teil der Ausgabe
- **Buffered**: Beginnt mit Gleichheitszeichen(=), evaluiert den Ausdruck und wird der Ausgabe hinzugefügt
- **Unescaped Buffered**: Beginnt mit Ungleichheitszeichen (!=) und wird hier nicht weiter betrachtet, da es als unsicher für Benutzereingaben gilt

# Pug

Unbuffered:

```
- var alter = 3;  
- for (var x = 0; x < alter; x++)  
  li item
```

Output:

```
<li>item</li>  
<li>item</li>  
<li>item</li>
```

Buffered:

```
p='This code is' + ' <escaped>!'  
p(style="background: blue")= 'A message  
with a ' + 'blue' + ' background'
```

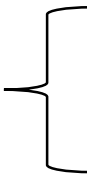
Output:

```
<p>This code is &lt;escaped>!</p>  
<p style="background: blue">A message  
with a blue background</p>
```

# Pug

## Schleifen in Pug

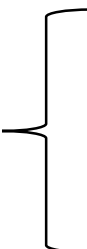
- Iteriert über eine Datenstruktur und erstellt aus jedem darin vorhanden Eintrag ein Paragraph Objekt



```
each user in users  
  p = user
```

## Kontrollstrukturen in Pug

- Wenn (`if`) eine Bedingung (`authenticated`) erfüllt wurde tue etwas, ansonsten (`else`) etwas anderes
- In pug Version 1 musste die unbuffered Schreibweise mit dem Bindestrich verwendet werden, was nun nicht mehr erforderlich ist



```
if authenticated  
  p = Hallo User  
else  
  p = Zurück zum Login
```

# Pug

## Pug CSS Styling

Wenn das Styling im Vorfeld definiert wurde, kann es pug-Elementen zugewiesen werden:

- `comments#title` → `comments` hat nun die `id title`
- `contacts.footer` → `contacts` wird die Klasse `footer` zugewiesen

|                              |   |  |
|------------------------------|---|--|
| <code>div#content</code>     | } | <code>&lt;div id="content"&gt;</code>              |
| <code>div.block</code>       |   | <code>&lt;div class="block"&gt;</code>             |
| <code>input#bar.f1.f2</code> |   | <code>&lt;input id="bar" class="f1 f2"/&gt;</code> |
|                              |   | <code>&lt;/div&gt;</code>                          |
|                              |   | <code>&lt;/div&gt;</code>                          |

# Pug

## Pug **Mixin** sind **wiederverwendbare Blöcke** von pug-Code


- Mixin's können, wie auch Funktionen, über eine Parameterliste verfügen
- Durch ein anführendes **Plus (+)** wird das Mixin aufgerufen
- Ein Mixin verfügt auch über ein implizites `attributes` Argument, welches zusätzlich genutzt wird um Inhalt zu spezifizieren

```
mixin pet(name)  
  span.pet= name
```




Definiert ein Mixin namens `pet`.

```
div  
  +pet('Alex')
```



Aufruf des `pet`-Mixin.

```
mixin link(href, name)  
  a(class!=attributes.class href=href)= name  
  +link('/foo', 'foo')(class="btn")
```



`attributes` beinhaltet die zusätzlich übergebenen Argumente.

# Pug

```

mixin vorlesung(datum, beschreibung)
  div.customStyle
    div.datumStyle= datum
    div.beschreibungStyle= beschreibung
doctype html
html
  head
    title Mixins in Pug
    style.
      .customStyle{
        padding: 10px;
        border: 1px solid #555;
        width: 300px;
      }
      .beschreibungStyle {font-size: 100%;
        font-family: helvetica;}
      .datumStyle {text-align: right;
        font-weight: bold;}

```

Erstellung des Mixin vorlesung

div-Element die Klasse customStyle hinzufügen

Styling Angaben für die class customStyle

Seperate Stylings für beschreibung & datum

# Pug

Die **später sichtbaren Elemente** werden wie bei HTML in den body geschrieben

- Die Elemente `v3` & `v6` werden jeweils **an das Mixin `vorlesung`** übergeben und anschließend gerendert
- Mixins können beliebig häufig wiederverwendet werden

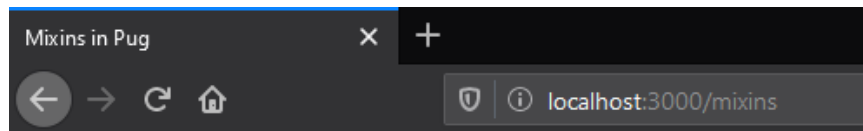
b ← Beginn des **HTML-body**

```
- const v3 = {  
  d: '04-05-2020',  
  b: 'JavaScript'}
```

```
+vorlesung(v3.d, v3.b)
```

```
- const v6 = {  
  d: '22-05-2020',  
  b: 'templating mit koa-pug'}
```

```
+vorlesung(v6.d, v6.b)
```



|                        |            |
|------------------------|------------|
| JavaScript             | 04-05-2020 |
| templating mit koa-pug | 22-05-2020 |

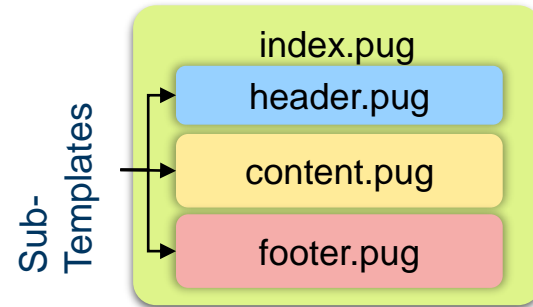
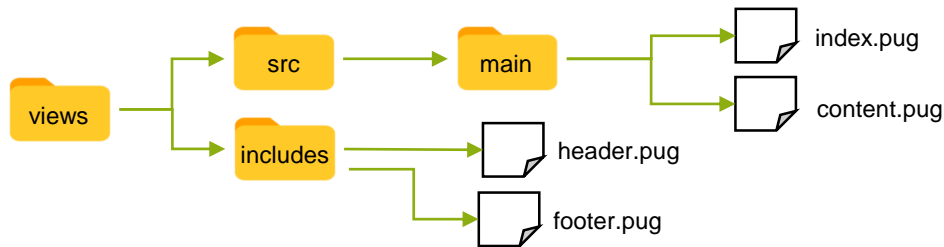


# Pug

## Einbinden von Pug Dateien

- Über das `include` Schlüsselwort können pug-Dateien durch Angabe ihres Pfades ineinander verschachtelt werden
- Dabei Kann Der Pfad relative angegeben werden (in index.pug):

```
include ../../includes/header.pug
include ../../includes/footer.pug
include content.pug
```



# Pug

## Einbinden von Pug Dateien

```
//- index.pug
```

```
doctype html
```

```
html
```

```
include includes/head.pug  
b
```

```
h1 Meine Seite
```

```
p Willkommen.
```

```
include includes/foot.pug
```

### Pug-Datei

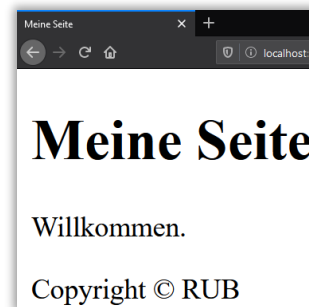
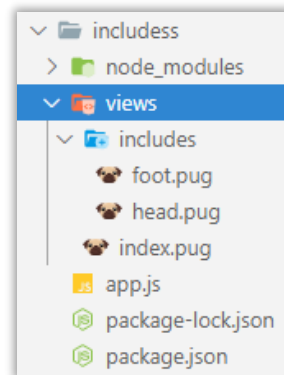
```
head
```

```
  title Meine Seite
```

```
  script(src='/...')
```

```
footer
```

```
  p Copyright (c) RUB
```



# Pug

## Eine Node.js Programm das Pug als Template Engine nutzt

`p #{name}'s Pug source code!` Pug-Datei

```
const pug = require('pug');  
const compFunc = pug.compileFile('template.pug');
```

```
console.log(compFunc({  
  name: 'Timothy'  
}));
```

```
console.log(compFunc({  
  name: 'Forbes'  
}));
```

Gibt den Inhalt der kompilierten Datei in der Konsole aus. Es wäre auch möglich den kompilierten Code in ein HTML-Datei zu schreiben.

Kommandozeile

```
C:\Users\...> node ./app.js
```

```
<p>Timothy's Pug source code!</p>  
<p>Forbes's Pug source code!</p>
```

# Pug

## Eine Pug Datei `kochbuch.pug`

```
b
  each rezept in rezepte
    ul
      li= rezept.titel
      li= rezept.vegetarisch
```

Ein pug-Template welches durch einen Datensatz names `rezepte` iteriert.

```
const chili = {
  titel: "Chili con Carne",
  vegetarisch: "Nein"};

const gemuese = {
  titel: "Gemüsepfanne",
  vegetarisch: "ja"};

var rezepte = [chili, gemuese];
```

Anlegen zweier Objekte `chili` & `gemuese`. Die Objekte werden in einem Array namens `rezepte` hinterlegt. Stehen NICHT direkt in der pug Datei, sondern sollen dem Template von außen übergeben werden.

# Pug

## Ein express-Server der die Pug Datei `kochbuch.pug` nutzt

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

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

Das `path` Modul wird benötigt um den Pfad zu den views anzugeben.

```
const app = express();
```

```
app.set('views', path.join(__dirname, 'views'));
```

Angabe des views Ordner.

```
app.set('view engine', 'pug');
```

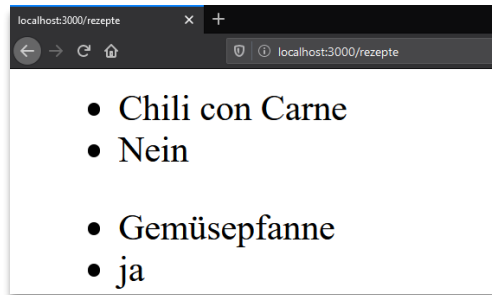
Pug als view engine angeben.

```
app.get('/rezepte', (req, res) => {  
  res.render('kochbuch', rezepte);  
});
```

```
app.listen(3000);
```

Die `kochbuch.pug` wird aus dem views Ordner ausgelesen.

`Rezepte` wird als Kontext für die pug-Datei mit übergeben.



# Pug

## koa-Server der koa-pug nutzt

```
var Koa = require('koa');  
var Router = require('koa-router');  
const path = require('path');  
var Pug = require('koa-pug');
```

Einbindung des koa-  
pug Moduls

```
const app = new Koa()  
const router = new Router()
```

Erstellung des  
Pug Elements

```
const pug = new Pug({  
  viewPath: path.resolve(__dirname, 'views'),  
  app: app  
})
```

Angabe des views  
Ordner

Der Server nutzt nun das Pug Element

# Pug

## koa-Server der koa-pug nutzt

router

```
.get('/rezepte', async ctx => {  
  await ctx.render('kochbuch', rezepte)  
});
```

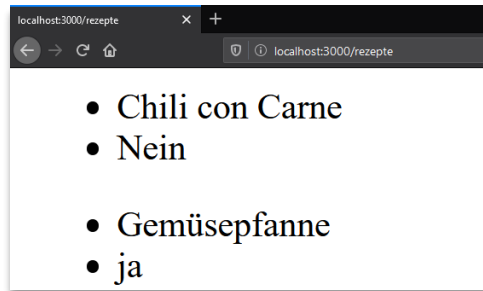
```
app.use(router.routes());
```

```
app.listen('3000');
```

render-Befehl über das Koa  
ctx Element

Rezepte wird als  
Kontext für die pug-Datei  
mit übergeben.

Die kochbuch.pug wird  
aus dem views Ordner  
ausgelesen.



# Web-Engineering

## Datenbanken



# Datenbanken

## Einleitung Datenbanken

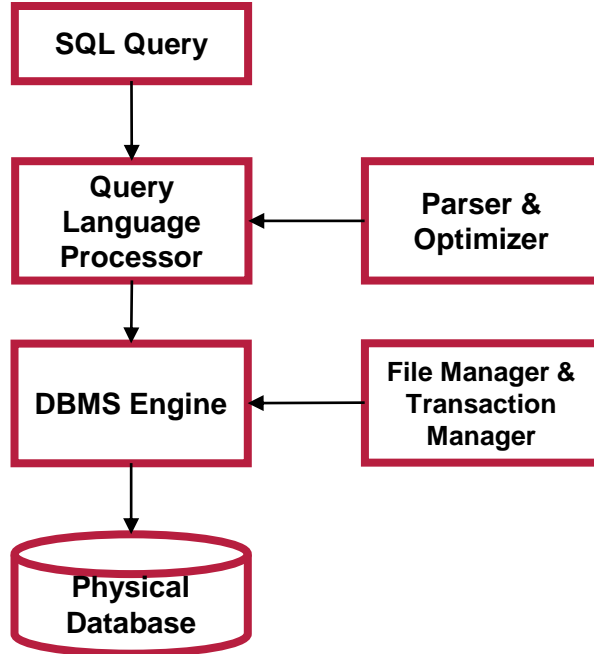
Ein Datenbanksystem **oder** DBs **gewährleistet die** Vollständigkeit, Richtigkeit **und** Verfügbarkeit **von Daten.**

Wofür DBs?

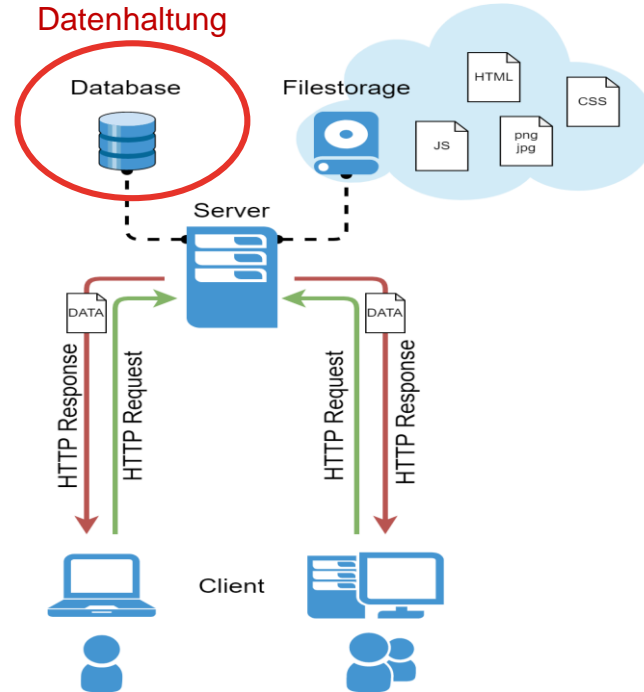
- Große Datenmengen werden nicht programmintern zwischengehalten, sondern in externen Datenbanken gelagert.
- DBs dienen als `persistenter` Datenspeicher
- Daten in DBs werden in `Relation` zueinander gesetzt (`Relationale Datenbank`)
- Mit einer `Anfragesprache (SQL)` können Teilmengen gefiltert werden (`Sicht`)

# Datenbanken

## Einleitung Datenbanken



Fokus auf serverseitige  
Datenhaltung



# Pug

## SQL steht für Structured Query Language

- SQL ist eine typisierte Abfragesprache für Datenbanken
- Mittels SQL Sprachen werden `queries` abgeschickt mit denen `Datensätze` erweitert, gelöscht, abgefragt oder manipuliert werden können
- Die Basis für SQL Sprachen ist ein `Relational Database Management System (RDBMS)`
  - Wie beispielsweise `Access`, `SQL Server` oder `MySQL`
- Mit SQL wird der Inhalt von Datenbanken gesteuert und verändert

# Datenbanken

## Die SQL Grundlagen:

- `CREATE TABLE`: Legt eine neue Tabelle in einer Datenbank an
- `PRIMARY KEY`: Legt fest welche Spalte zur Identifikation dient
- `FOREIGN KEY`: Verweist auf den Primärschlüssel einer anderen Tabelle und verknüpft diese
- `NOT NULL`: Signalisiert das bei einem Eintrag dieser Wert nicht optional ist
- `Int, varchar, float, boolean, date, usw.`: Erwartete Datentyp einer Spalte inklusive maximal erlaubter Zeichensatzlänge

# Datenbanken

- `INSERT INTO [Tabelle(Spalten)] VALUES [(Werte)]:`  
Fügt Rohe Datensätze in eine Tabelle ein.
  - Die Reihenfolge der angegebenen Daten ist relevant
  - Jede Spalte die mit `NOT NULL` ausgezeichnet wurde muss einen Wert erhalten
- `SELECT [*|Spalte] FROM [Tabelle] WHERE [Kondition]:`  
Wählt Daten einer spezifischen Tabelle als Datensatz aus und wird in der Regel mit `WHERE` zusätzlich gefiltert
- `DELETE FROM [Tabelle] WHERE [Kondition]:`  
Löscht alle Einträge aus einer Tabelle, welche die Konditionen im `WHERE`-Teil erfüllen

# Datenbanken

```
CREATE TABLE authors (  
  id int(11) NOT NULL,  
  name varchar(50),  
  city varchar(50),  
  PRIMARY KEY (id)  
);
```

```
CREATE TABLE books (  
  authorID int(11) NOT NULL,  
  titel varchar(50),  
  FOREIGN KEY (authorID) REFERENCES authors(id)  
);
```

```
INSERT INTO authors(id, name, city) VALUES  
(1, 'Michaela Lehr', 'Berlin'),  
(2, 'Michael Wanyoike', 'Nairobi'),  
(3, 'James Hibbard', 'Munich'),  
(4, 'Karolina Gawron', 'Wrocław');
```

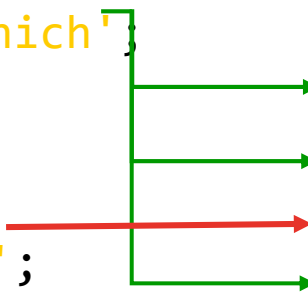
Tabelle: authors

| <u>id</u> | name             | city    |
|-----------|------------------|---------|
| 1         | Michaela Lehr    | Berlin  |
| 2         | Michael Wanyoike | Nairobi |
| 3         | James Hibbard    | Munich  |
| 4         | Karolina Gawron  | Wrocław |

# Datenbanken

```
SELECT id, name FROM authors  
WHERE NOT authors.city = 'Munich';
```

```
DELETE FROM authors  
WHERE authors.city = 'Munich';
```



| <u>id</u>    | name                     | city              |
|--------------|--------------------------|-------------------|
| 1            | Michaela Lehr            | Berlin            |
| 2            | Michael Wanyoike         | Nairobi           |
| <del>3</del> | <del>James Hibbard</del> | <del>Munich</del> |
| 4            | Karolina Gawron          | Wrocław           |

# Datenbanken

## FEHLER!

Das Buch „Kochen für Anfänger“ verweist auf den Autor mit id=3 und würde bei seinem Löschen ins Leere zeigen.

## Lösung:

Erst Buch löschen, dann Autor.

```
DELETE FROM authors  
WHERE authors.city = 'Munich';
```

Tabelle: books

| <u>autorsID</u> | titel               |
|-----------------|---------------------|
| 3               | Kochen für Anfänger |

referenziert

Tabelle: authors

| <u>id</u>    | name                     | city              |
|--------------|--------------------------|-------------------|
| 1            | Michaela Lehr            | Berlin            |
| 2            | Michael Wanyoike         | Nairobi           |
| <del>3</del> | <del>James Hibbard</del> | <del>Munich</del> |
| 4            | Karolina Gawron          | Wrocław           |



# Datenbanken

## Programminterne Nutzung von Datenbanken

- Für den Zugriff auf Datenbanken wird in der Regel eine Art von Treiber verwendet, der als Schnittstelle zur Datenbank dient
- Diese Treiber sind kein fester Bestandteil der Node.js Plattform, sondern liegen als Module vor, die je nach Bedarf installiert werden
- Es liegen Module für gängigen `relationale` Datenbanken vor
  - Beispielsweise für: `MySQL`, `MSSQL`, `SQLite`
- Ebenso liegen Module für gängigen `nicht-relationale` Datenbanken vor
  - Beispielsweise für: `CouchDB` oder `MongoDB`

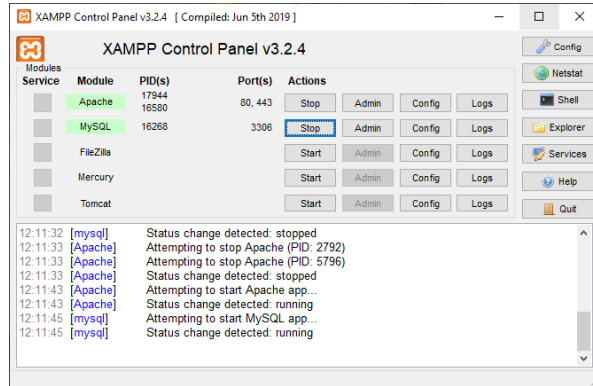
Kommandozeile

```
C:\Users\...> npm install mysql
```

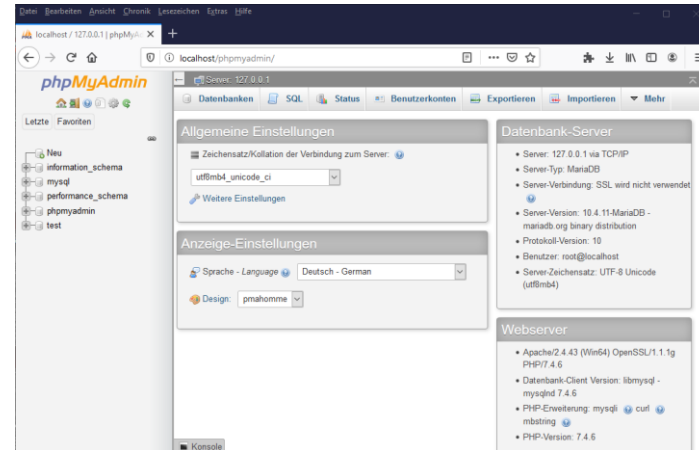
# Datenbanken

**Der MySQL Treiber erfordert die Verwendung einer gehosteten Datenbank.**

- Durch XAMPP kann eine Datenbanken über die integrierten phpMyAdmin Anwendung lokal angelegt werden. Sobald gestartet im Browser erreichbar über: `http://localhost/phpmyadmin/`



Quelle: <https://www.apachefriends.org/de/index.html>



Quelle: <https://www.phpmyadmin.net/>

# Datenbanken

## Datenbank Einbindung und verwenden über Node.js

- Zunächst wird eine Verbindung zu einer laufenden Datenbank hergestellt
- Manche Treiber erlauben es eine lokale Datei als Datenbank einzubinden
  - In der Regel werden DBs `online` hinterlegt und verbunden



# Datenbanken

## Datenbank Einbindung und verwenden über Node.js

```
const mysql = require('mysql');
const con = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'root',
  database: 'mydb'
});
con.connect((err) => {
  if (err) throw err;
  console.log('Verbindung hergestellt!');
});
//DO SOMETHING HERE
con.end();
```

Erstellung des Verbindungsobjekts zur Datenbank namens `mydb`.

Aufbau der Verbindung über das Verbindungsobjekt.

Verbindung wieder freigeben, bei fertiger Anfrage.

# Datenbanken

## SQL Query absenden

```
con.query('SELECT * FROM mydb.authors', (err, rows) => {  
  if (err) throw err;  
  console.log('erhaltene Daten:');  
  rows.forEach((row) => {  
    console.log(`${row.name} lebt in ${row.city}`);  
  });  
});
```

Kommandozeile

```
C:\Users\...> node ./app.js
```

```
Verbindung hergestellt  
erhaltene Daten:  
Michaela Lehr lebt in Berlin  
Michael Wanyoike lebt in Nairobi  
James Hibbard lebt in Munich  
Karolina Gawron lebt in Wrocław
```

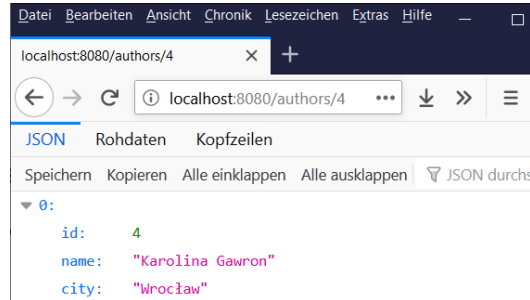
name und city werden iterative  
in der Konsole ausgegeben.

Abgefragt werden die  
Daten innerhalb von  
authors der mydb  
Datenbank.

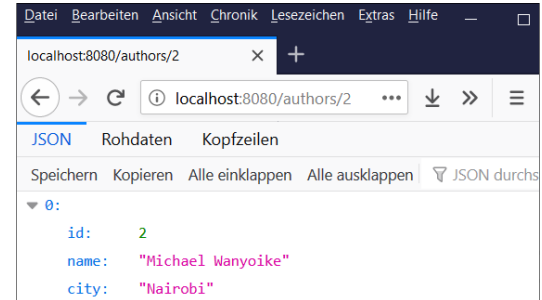
# Datenbanken

In Kombination mit Rest-Development können so Endpunkte zur Datenbankverwaltung erstellt werden. Beispiel in Express:

```
app.get("/authors/:id", (req, res) => {  
  const userId = req.params.id;  
  const querystring = 'SELECT * FROM authors WHERE id = ?';  
  connection.query(querystring, [userId], (err, rows, fields)=>  
  {  
    res.json(rows);  
    res.end();  
  })  
});
```



GET: http://localhost:8080/authors/4



GET: http://localhost:8080/authors/2