

Verteilte Systeme 1

Technologien des World Wide Web

christian.zirpins@hs-karlsruhe.de

Web Entwicklung mit Ajax & Co



Hochschule Karlsruhe
Technik und Wirtschaft

UNIVERSITY OF APPLIED SCIENCES



VS1 Termine im Sommer 2017

Termin (ca.)	Thema	Vorbereitung (Begleitbuch)	Raum
KW11: 13.03, 15.03	HTTP , die Sprache des Web		E 301/304
KW12: 20.03, 22.03	Web Apps mit HTML5	Web App Development Kapitel 2	E 301/304
KW13: 27.03, 29.03	Gestaltung von Web Apps mit CSS3	Web App Development Kapitel 3	E 301/304
KW14: 03.04, 05.04	<i>Übung: Webseite mit HTML5/CSS3 erstellen</i>		LI 137
KW15: 10.04, 12.04	Browser Interaktion mit JavaScript	Web App Development Kapitel 4	E 301/304
KW16	Ostern		
KW17: 24.04, 26.04	<i>Übung: Formulare mit JavaScript / HTML5 APIs</i>		LI 137
KW18	Maifeiertag		
KW19: 08.05, 10.05	JavaScript auf dem Server mit Node.js	Web App Development Kapitel 6	E 301/304
KW20: 15.05, 17.05	<i>Übung: Node.js / Express Web App erstellen</i>		LI 137
KW21: 22.05, 24.05	Web Entwicklung mit Ajax & Co	Web App Development Kapitel 5	E 301/304
KW22: 29.05, 31.05	<i>Übung: Web App mit REST und AJAX erweitern</i>		LI 137
KW23	Pfingsten		
KW24: 12.06, 14.06	Web Apps Personalisieren	Web App Development Kapitel 9	E 301/304
KW25: 19.06, 21.06	Web App Sicherheit		E 301/304
KW26: 26.06, 28.06	Klausurvorbereitung, Q&A		E 301/304

Heutige Lernziele

Nach dieser Vorlesung können Sie...

- ...“**reines**” **Ajax** (ohne Hilfe von jQuery) **implementieren**
- ...den Zusammenhang von **Express und Connect** **erklären**
- ...die **Template Sprache ejs** **verwenden**

Node + Connect + Express:

Web Ressourcen
bereitstellen

EJS:
HTML Seiten
dynamisch
generieren



html,
css, js

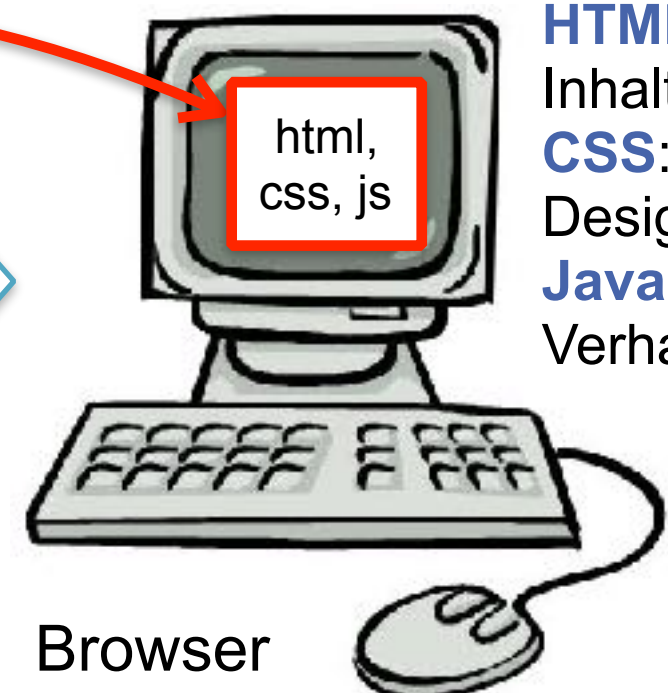


Server

AJAX:

Interaktion mit dem
Web Server *aus
einer Seite heraus*

HTTP: Kommunikation



Browser

HTML:
Inhalte
CSS:
Design
JavaScript:
Verhalten

html,
css, js

Ajax

Dynamische Updates auf dem Client

Ajax

Nur im Namen

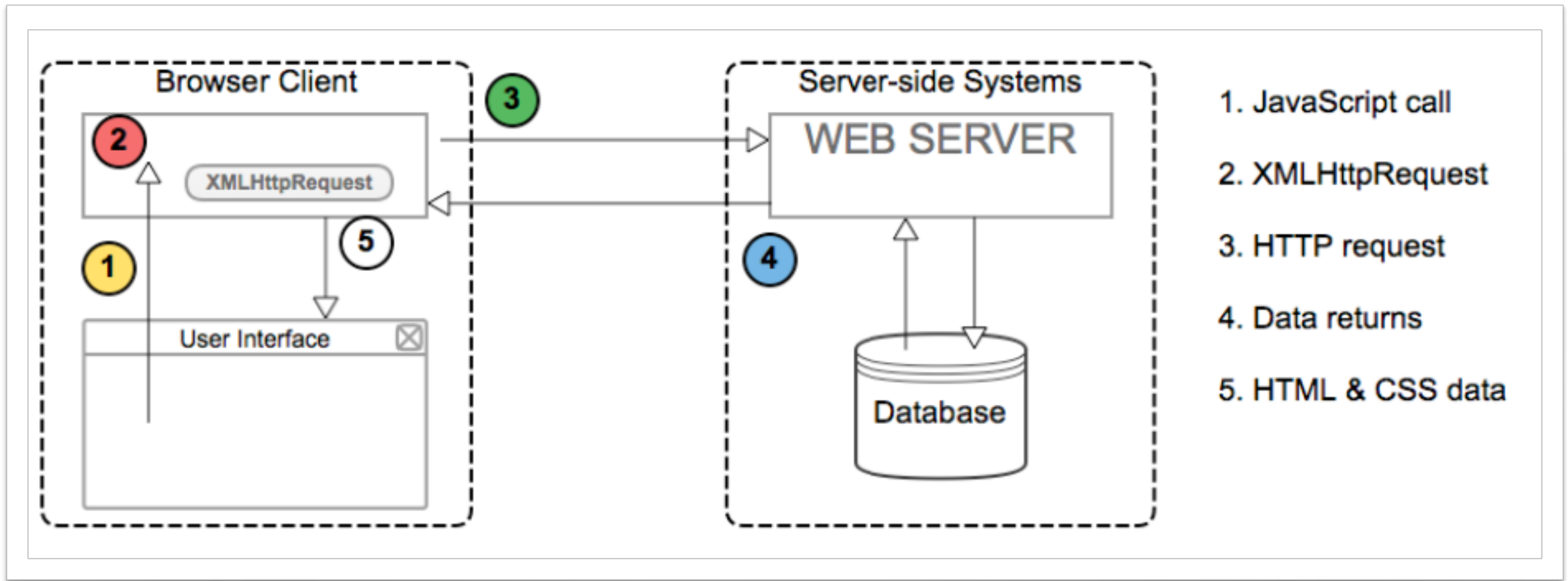
Asynchronous JavaScript and XML

- Ajax ist ein **JavaScript-Mechanismus**, der das dynamische Laden von Inhalten ermöglicht, ohne die Seite manuell neu zu laden oder abrufen zu müssen.
- Ajax: Technologie, um neue Daten in eine bestehende Web-Seite zu laden (keine Sprache oder Produkt)
- Sie sehen diese Technologie jeden Tag: *Chats, endloses Scrollen*
- Ajax dreht sich um `XMLHttpRequest`, ein JavaScript Objekt
- **JQuery verbirgt alle Komplexität, macht Ajax-Anrufe einfach**

Ajax - wie funktioniert das?

1. Webbrowser erstellt ein `XMLHttpRequest`-Objekt
2. `XMLHttpRequest` fordert Daten von einem Webserver an
3. Daten werden vom Server zurückgesendet
4. Auf dem Client fügt JavaScript-Code die Daten in die Seite ein

Ajax - wie funktioniert das?



Ohne Ajax...

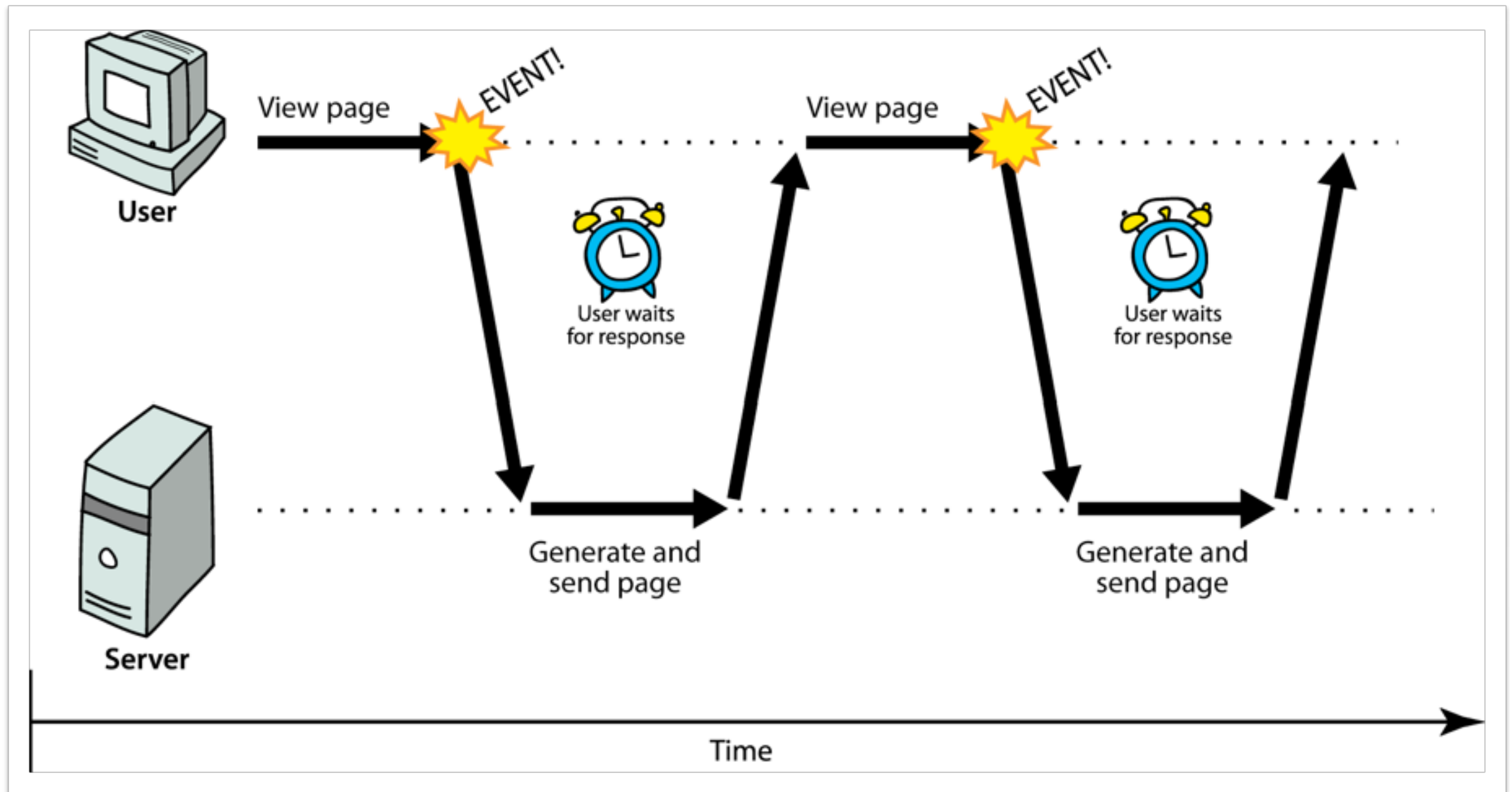


Abb. aus <http://www.webstepbook.com/supplements-2ed/slides/chapter12-ajax-xml-json.shtml#slide2>

Ajax funktioniert anders

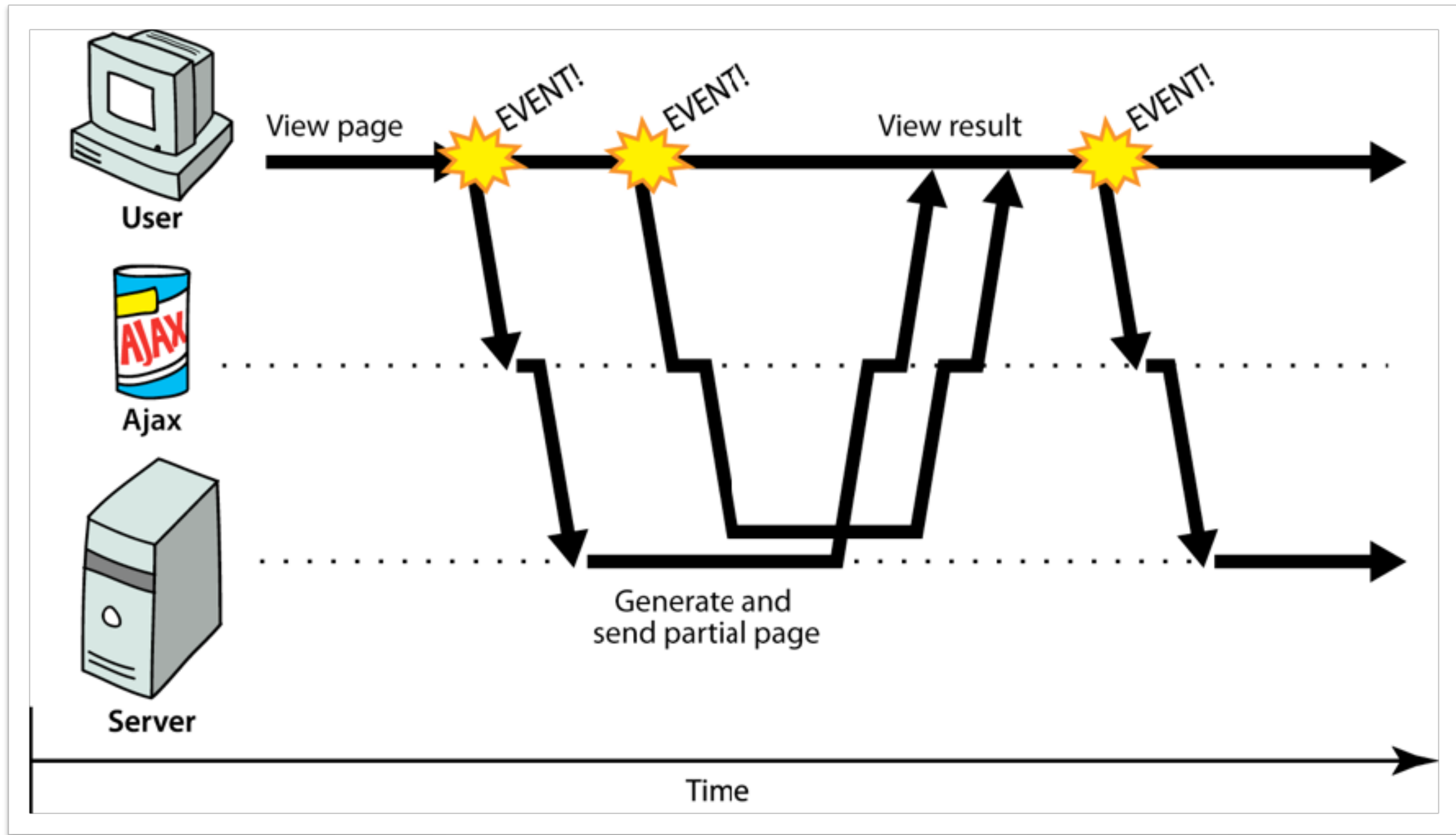


Abb. aus <http://www.webstepbook.com/supplements-2ed/slides/chapter12-ajax-xml-json.shtml#slide2>

Ajax: synchroner Request

```
// IE6 und frühere IE Versionen nutzen  
// stattdessen Microsoft XMLHttpRequest  
var ajax = new XMLHttpRequest();  
  
// Lade Daten von URL (Datei)  
// "false" Parameter: synchroner Request  
ajax.open('GET', 'example.txt', false);  
ajax.send(null);  
  
// Antwortdaten in ajax.responseText  
document.getElementById('ttExampleText').value =  
    ajax.responseText;
```

Ausführung erst nach
Ende von `ajax.send`

reines JS

Ajax: XMLHttpRequest

<i>Eigenschaft</i>	<i>Beschreibung</i>
onreadystatechange	Funktion, die aufgerufen wird, wenn sich die Eigenschaft readyState ändert
readyState	Hält den Status des XMLHttpRequest : 0: Request nicht initialisiert 1: Serververbindung hergestellt 2: Request eingegangen 3: Request wird verarbeitet 4: Request beendet, Antwort liegt vor
responseText	Antwortdaten als String
responseXML	Antwortdaten als XML
status	HTTP Status Code (z.B. "404" oder "200")
statusText	HTTP Status Text (z.B. "Not Found" oder "OK")

reines JS

Ajax: XMLHttpRequest

<i>Eigenschaft</i>		<i>Beschreibung</i>
onreadystatechange		Funktion, die aufgerufen wird, wenn sich die Eigenschaft readyState ändert
readyState		Hält den Status des XMLHttpRequest: 0: Request nicht initialisiert
	<i>Methode</i>	<i>Beschreibung</i>
	abort ()	Abbruch des aktuellen Requests
	getAllResponseHeaders ()	Gibt Header Informationen zurück
response	getResponseHeader ()	Gibt einzelne Header Informationen zurück
response	open (method, url, async, username, pswd)	Bestimmt Request Typ, URL, ob der Request asynchron ausgeführt werden soll u.a. - <i>method</i> : GET, POST etc. - <i>url</i> : Ort der Datei auf dem Server - <i>async</i> : true (asynchron), false (synchron)
status		
status		
status		
	send (string)	Sendet den Request an den Server - <i>string</i> : Nur für POST Requests
	setRequestHeader ()	Fügt Name/Wert-Paar zum Header hinzu

reines JS

Ajax: asynchroner Request

```
var ajax = new XMLHttpRequest();  
  
// Funktion, die bei Statusänderung  
// aufgerufen wird  
ajax.onreadystatechange = function() {  
    // Zustand von Interesse  
    if (ajax.readyState == 4) {  
        // Verarbeite eingehende Daten  
    }  
}; //Ende der Funktion  
  
ajax.open("GET", "url", true);  
//true bedeutet asynchroner Request  
  
ajax.send(null);
```

Ajax: asynchroner Request

```
var ajax = new XMLHttpRequest();
```

```
// Funktion, die bei Statusänderung  
// aufgerufen wird
```

onreadystatechange
feuert, bei jeder (!)
Änderung des Status

```
ajax.onreadystatechange = function() {  
    // Zustand von Interesse  
    if (ajax.readyState == 4) {  
        // Verarbeite eingehende Daten  
    }  
}; //Ende der Funktion
```

```
ajax.open("GET", "url", true);  
//true bedeutet asynchroner Request
```

```
ajax.send(null);
```

reines JS

Ajax Sicherheit

- Praktischerweise kamen alle Daten von "*unserem* Webserver"
- **Sicherheitsbeschränkung von Ajax:** kann nur Dateien vom selben Webserver abrufen wie die aufrufende Seite (**Same-Origin Policy**)
 - Gleicher Ursprung, wenn *Protokoll*, *Port* und *Host* für zwei Seiten gleich sind
- Ajax **kann nicht** von einer lokal auf der Festplatte gespeicherten Webseite ausgeführt werden

LWAD Buch erklärt, wie man dies umgeht (sollte man sein lassen)

Representational State Transfer“ (REST)

Was ist REST?

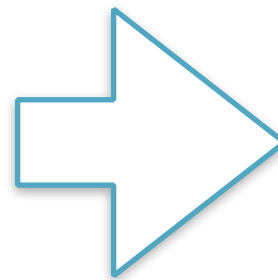
- **REST** ist ein **Architektur-Stil** für lose gekoppelte (Software) Systeme meist umgesetzt mit Web Techniken wie HTTP, URLs, XML/JSON
- Alle **Ressourcen** (Informationen) haben eine **URL** und können per **HTTP** erzeugt, gelesen, geändert oder gelöscht werden
- Darstellung wahlweise per XML, **JSON**, HTML und ggf. mit **Links**

REST und AJAX

- Mit Ajax können alle REST-Ressourcen direkt aus dem (Web) Client heraus genutzt werden
- Die Todo-Liste im vorherigen Beispiel ist eine REST-Resource
 - Neben Lesen (GET) und Speichern (POST) von Todos könnten leicht weitere Routen für Änderung und Löschen ergänzt werden

Node.js

Organisation und Wiederverwendung von Code



Bislang

Serverseitiger Code in einer einzigen Datei

- Für kleine Projekte
- Größere Projekte leiden aber darunter, denn
 - das **Debugging** ist schwerfällig,
 - **Teamarbeit** ist umständlich,
 - die **Programmierung** ist umständlich.

node.js Module

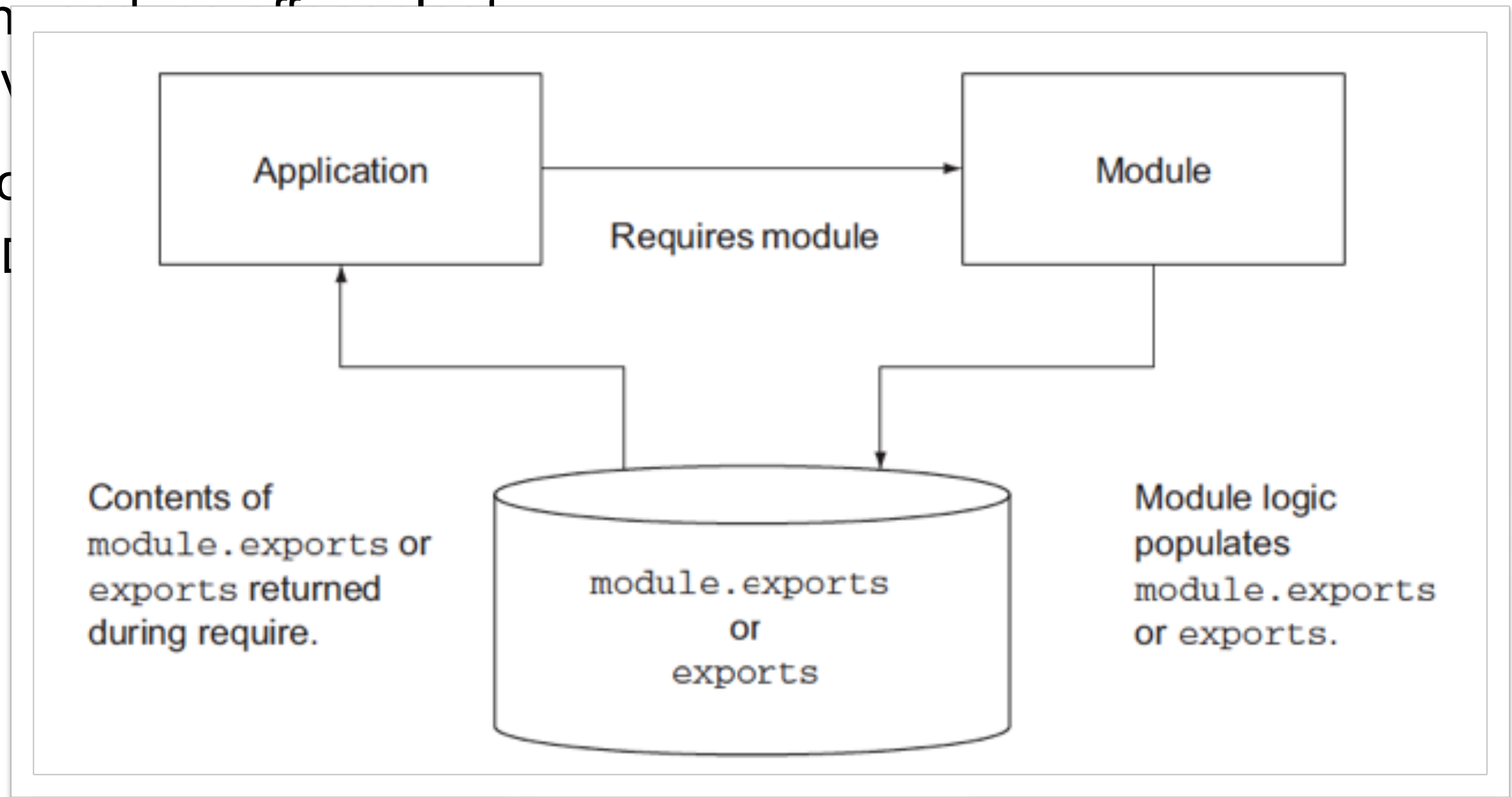
- Code kann in Modulen organisiert werden
- Nicht alle Funktionen und Variablen in einem Modul werden der Anwendung offengelegt
 - Veröffentlichte Elemente werden über **exports** deklariert
- Module können via **npm veröffentlicht** werden
 - Die Verbreitung von Modulen für andere Entwickler ist einfach

node.js Module

- Code kann in Modulen organisiert werden
- Nicht alle Funktionen und Variablen in einem Modul werden der

An

- V
- Mo
- I



Quelle: "Node.js in Action"

Ein Modul erstellen

Ein Modul besteht aus ...

- einer einzelnen **Datei** oder
- einem **Verzeichnis** von Dateien (das eine Datei `index.js` enthält)

```
function roundGrade(grade) {  
    return Math.round(grade);  
}
```

Nicht exportiert! Eine Anwendung kann die Funktion nicht nutzen

```
function roundGradeUp(grade) {  
    return Math.round(0.5 + parseFloat(grade));  
}
```

```
exports.maxGrade = 10;  
exports.roundGradeUp = roundGradeUp;  
exports.roundGradeDown = function(grade) {  
    return Math.round(grade - 0.5);  
}
```

Ein Modul verwenden

```
var express = require("express");  
var url = require("url");  
var http = require("http");  
var grading = require("./grades");  
var app;
```

`require` liefert den Inhalt
des `exports` Objektes

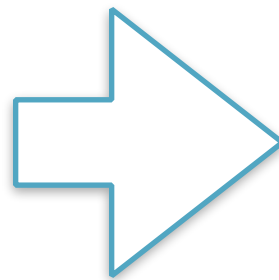
```
var port = process.argv[2];  
app = express();  
http.createServer(app).listen(port);
```

Das Modul hinzufügen
(gleiches Verzeichnis)

```
app.get("/round", function(req, res) {  
  var query = url.parse(req.url, true).query;  
  var grade = (query["grade"] != undefined) ?  
    query["grade"] : "0";  
  res.send("Rounding up: " +  
    grading.roundGradeUp(grade) + ", and down: " +  
    grading.roundGradeDown(grade));  
});
```

Modulfunktionen
verwenden

Express und Connect

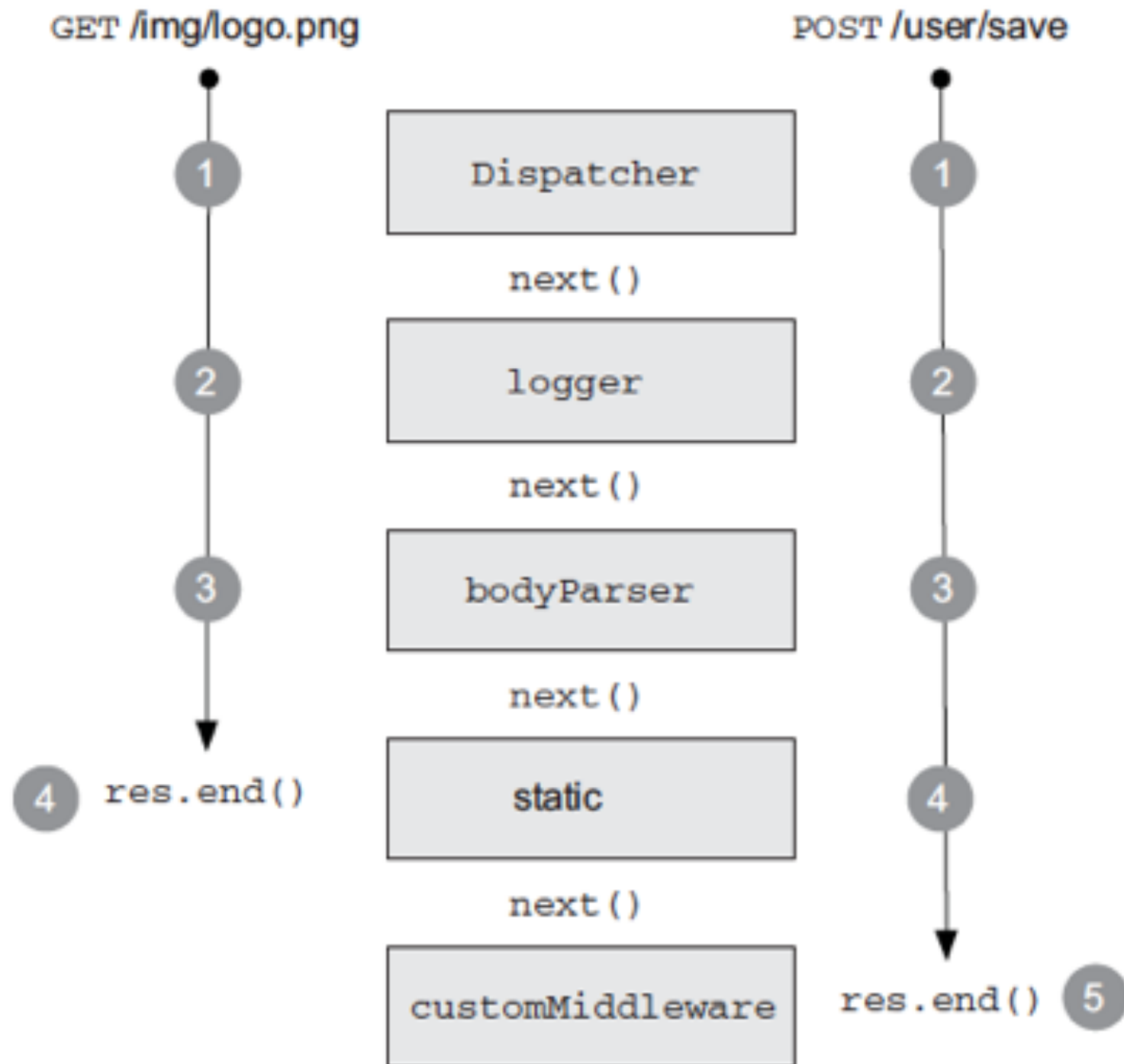


Was ist **Connect**?

Express basiert auf **Connect**

- **Connect: Framework**, dessen Komponenten ("**Middleware**") verwendet werden, um die Logik einer Web-Anwendung zu erstellen (inspiriert von Ruby's Rack-Framework)
- **Middleware: Funktion**, die die Request- und Reply-Objekte des HTTP-Servers abfängt, die Logik ausführt und die Response beendet oder an die nächste Komponente übergibt
- **Dispatcher** verbindet die Komponenten (daher "Connect")
- Typische Komponenten: Request Logging (Protokollierung), Bereitstellung statischer Dateien, Request-Body-Parsing, Session-Verwaltung usw.

Connect Beispiel



- 1 Dispatcher receives request and passes it to the first middleware
- 2 Request is logged and passed to the next middleware using `next()`
- 3 Request body is parsed if any exists and then passed to the next middleware using `next()`
- 4 If request is for a static file, response is sent with that file and `next()` is not called; otherwise, the request moves to the next middleware
- 5 Request is handled with a custom middleware and response is ended

Ein **minimales** Connect Beispiel

```
var connect = require('connect');  
var app = connect();  
app.listen(3000)
```

Erinnert an
Express Setup...

- Im Beispiel wird keine Middleware deklariert
- Dispatcher ruft jede Middleware-Komponente auf, bis eine antwortet
 - Wenn keine antwortet (oder keine existiert), sendet der Dispatcher schließlich eine 404-Antwort an den Client zurück

Middleware Komponenten

- Middleware-Komponenten haben drei Argumente:
 - HTTP-Request Objekt
 - HTTP-Response Objekt
 - Callback-Funktion (`next()`), die anzeigt, dass die Komponente fertig ist und der Dispatcher zur nächsten Komponente wechseln kann
- Middleware-Komponenten sind **klein**, in sich **geschlossen** und **wiederverwendbar** zwischen Anwendungen

Einfache **Logger** Komponente

- **Ziel:** Erstellen einer Log Datei, die die Request-Methode und die URL von Requests aufzeichnet, die an den Server gehen
- **Erforderlich:** JavaScript-Funktion, die die Request- und Response-Objekte und die `next()` Callback Funktion entgegennimmt

```
var connect = require('connect');  
  
// Middleware Logger Komponente  
function logger(request, response, next) {  
    console.log('%s\t%s\t%s', new Date(),  
        request.method, request.url);  
    next();  
}  
var app = connect();  
app.use(logger);  
app.listen(3001);
```

Einfache **Logger** Komponente

- **Ziel:** Erstellen einer Log Datei, die die Request-Methode und die URL von Requests aufzeichnet die an den Server gehen
- **Erforderlich:** JavaScript-Funktion, die die Request- und Response-Objekte und die `next()` Callback Funktion entgegennimmt

```
var connect = require('connect');  
  
// Middleware Logger Komponente  
function logger(request, response, next) {  
    console.log('%s\t%s\t%s', new Date(),  
        request.method, request.url);  
    next();  
}  
var app = connect();  
app.use(logger);  
app.listen(3001);
```

Kontrolle geht an den
Dispatcher zurück

Middleware registrieren

Nun noch eine **Antwort** an den Client

- **Ziel:** antworte "Hello World" bei jedem Request
- **Erforderlich:** JavaScript-Funktion, die die Request- und Response-Objekte und die `next()` Callback Funktion entgegennimmt

```
var connect = require('connect');  
  
function logger(request, response, next) { ... }  
  
function helloWorld(request, response, next) {  
  response.setHeader('Content-Type',  
    'text/plain');  
  response.end('Hello World!');  
}  
  
var app = connect();  
app.use(logger);  
app.use(helloWorld);  
app.listen(3001);
```

Kein Aufruf von `next()`; Kontrolle geht nicht an Dispatcher zurück!

Es können beliebig viele Komponenten registriert werden

Frage: Was tut dieser Code?

```
var connect = require('connect');
connect()
  .use(logger)
  .use('/admin', restrict)
  .use(serveStaticFiles)
  .use(hello);
```

```
function restrict(req, res, next) {
  var auth = req.headers.authorization;
  if (!auth)
    return next(new Error('Unauthorized'));

  var parts = auth.split(' ');
  var scheme = parts[0];
  var auth = new Buffer(parts[1], 'base64')
    .toString().split(':');

  var user = auth[0]
  var pass = auth[1];
  if (user == "user" && pass == "password") {
    next();
  }
}
```


Frage: Was tut dieser Code?

```
var connect = require('connect');
connect()
  .use(logger)
  .use('/admin', restrict)
  .use(serveStaticFiles)
  .use(helper);
```

Jeder Aufruf von `use()` gibt das Connect Objekt selbst zurück - ermöglicht Methodenverkettung.

```
function restrict(req, res, next) {
  var url = req.url;
  if (url.startsWith('/admin')) {
    next();
  } else {
    res.status(403).send('Access Denied');
  }
}
```

Middleware Komponente wird nur dann aufgerufen, wenn der URL Präfix zutrifft

```
curl --user user:password http://localhost:3001/admin
```

```
var auth = new Buffer(parts[1], 'base64')
  .toString().split(':');

var user = auth[0]
var pass = auth[1];
if (user == "user" && pass == "password") {
  next();
}
}
```

Komponenten **konfigurierbar** machen

- **Ziel:** Middleware-Komponenten sollten in mehreren Anwendungen ohne zusätzlichen Aufwand wiederverwendet werden
- **Ansatz:** ursprüngliche Middleware-Funktion in einer Setup-Funktion verpacken, die mit Parametern konfigurierbar ist

```
function setup(options) {  
  // Setup Logik  
  return function(req, res, next) {  
    // Middleware Logik!  
  }  
}  
  
app.use(setup({param1: 'value1'}));
```

Wichtig: dies ist ein
Funktionsaufruf!

Express Routen als Middleware

```
var express = require("express");
var url = require("url");
var http = require("http");

var port = 3000;
var app = express();
app.use(express.static(__dirname + "/client"));
http.createServer(app).listen(port);

var todos = [];
var t1 = {message: "Mow the cat",type: 1,deadline: "12/12/2016"};
var t2 = {message: "Feed the lawn",type: 3,deadline: "20/12/2016"};
todos.push(t1);
todos.push(t2);

function logger(request, response, next) {
  console.log('%s\t%s\t%s', new Date(), request.method, request.url);
  next();
}

function getTodos(request, response, next) {
  console.log("todos requested!");
  response.json(todos);
}

app.get("/todos", logger, getTodos);
```

Für eine Route können beliebig viele Komponenten registriert werden

Templates mit ejs

Express und HTML...

```
var express = require("express");
var url = require("url");
var http = require("http");
var app;

var port = process.argv[2];
app = express();
http.createServer(app).listen(port);

app.get("/greetme", function(req, res) {
  var query = url.parse(req.url, true).query;
  var name = (query["name"] !== undefined) ?
    query["name"] : "Anonymous";

  res.send("<html><head></head><body><h1> Greetings "
    + name + "</h1></body></html>");
});

app.get("/goodbye", function(req, res) {
  res.send("Goodbye you!");
});
```

Express und HTML...

```
var express = require("express");
var url = require("url");
var http = require("http");
var app;

var port = process.argv[2];
app = express();
http.createServer(app).listen(port);

app.get("/greetme", function(req, res) {
  var query = url.parse(req.url, true).query;
  var name = (query["name"] !== undefined) ?
    query["name"] : "Anonymous";

  res.send("<html><head></head><body><h1> Greetings "
    + name + "</h1></body></html>");
});

app.get("/goodbye", function(req, res) {
  res.send("Goodbye you!");
});
```

Fehleranfällig, nicht
wartbar, scheitert bei
jedem ernsthaften
Projekt

Stattdessen: **Templates**

- **Ziel:** schreibe so wenig HTML "von Hand" wie möglich



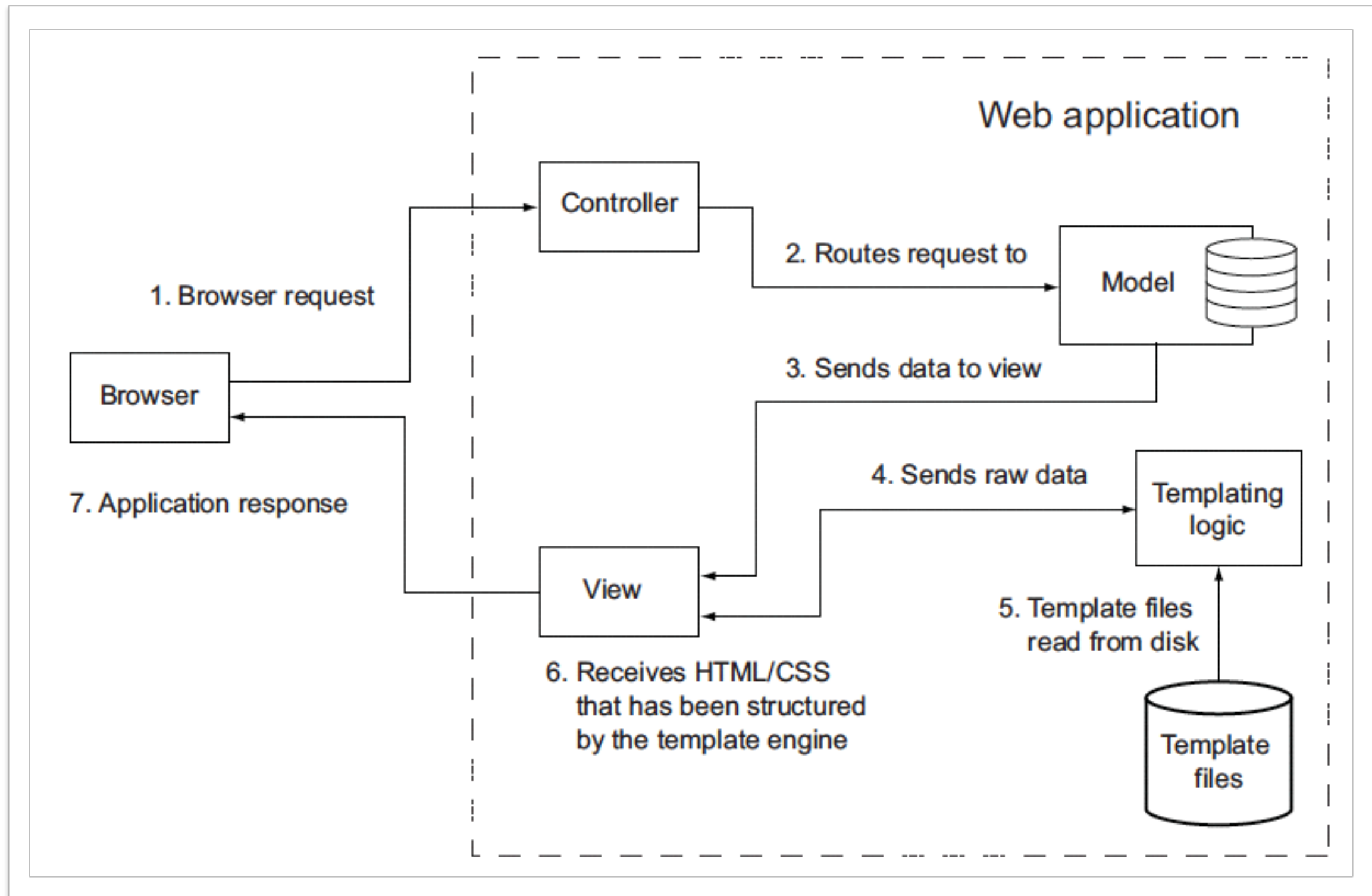
- Hält den Code sauber - trennt Logik von Präsentation Markup
- Für node.js gibt es viele verschiedene Template-Engines
- Wir konzentrieren uns zunächst auf **EJS** (Embedded JavaScript), eine **Template Engine und Sprache**

“EJS cleans the HTML out of your JavaScript with client side templates. After EJS gets its rubber gloves on dirty code, you'll feel organized and uncluttered.” — <http://www.embeddedjs.com/>

Model-View-Controller (MVC)

- Standard Entwurfsmuster um **Logik**, **Daten** und **Präsentation** getrennt zu halten!
- Benutzer Request für eine Ressource auf dem Server bewirkt ...
 1. **Controller** fordert Anwendungsdaten aus dem **Modell** an
 2. **Controller** übergibt die Daten an die **View**
 3. **View** formatiert die Daten für den Benutzer (hierzu werden Template-Engines verwendet)

Model-View-Controller (MVC)



Quelle: "Node.js in Action"

Ein erster Blick auf EJS

Start im Terminal: "npm install ejs" dann "node test-ejs.js"

```
var ejs = require('ejs');
var template = '<%= message %>';
var context = {
  message: 'Hello template!'
};
console.log(ejs.render(template, context));
```

test-ejs.js

<%- wenn Escaping nicht gewünscht wird
(Achtung, ermöglicht Cross-Site Scripting Angriffe!)

```
var ejs = require('ejs');
var template = '<%= message %>';
var context = {
  message: "<script>alert('hi!');</script>"
};
console.log(ejs.render(template, context));
```

test-ejs2.js

Standardmäßig "escaped" ejs spezielle Werte im Kontext:

`<script>alert('hi!');</script>`

EJS Filter

- Im Template kann eine leichtgewichtige Filterung definiert werden

```
var template = '<%=: movies | last %>';  
var context = {  
  'movies': [  
    'The Hobbit',  
    'X-Men',  
    'Superman V'  
  ]  
};  
console.log(ejs.render(template, context));
```

EJS Filter

- Im Template kann eine leichtgewichtige Filterung definiert werden

```
var template = '<%=: movies | last %>';  
var context = {  
  'movies': [  
    'The Hobbit',  
    'X-Men',  
    'Superman V'  
  ]  
};  
console.log(ejs.render(template, context));
```

':' zeigt
Filterung an

Filter im Template definiert:
wähle letztes Array Element

- Filtern basierend auf Array-Position:
 - **last** - letztes Element auswählen
 - **first** - erstes Element auswählen
 - **get:n** - wähle n-tes Element (Index beginnt bei 0)

EJS Filter

- Es gibt zahlreiche **Textmanipulationsfilter**
 - `upcase` um ein Wort groß zu schreiben
 - `downcase` um ein Wort klein zu schreiben
 - `truncate:n` beschränkt die Eingabe auf die ersten n Zeichen
 - `truncate_words:n` beschränkt die Eingabe auf die ersten n Worte

```
var ejs = require('ejs');  
var template = '<%=: title | truncate:16 %>';  
var context = {  
  title: 'I should keep it short but I cannot'  
};  
console.log(ejs.render(template, context));
```

- `replace` (entweder einen String oder einen regulären Ausdruck)

EJS Filter

- Filter können im Template **verkettet** werden

```
var template = '<%=: movies | sort | get:1 %>';  
var context = {  
  'movies': [  
    'The Hobbit',  
    'X-Men',  
    'Superman V'  
  ]  
};  
console.log(ejs.render(template, context));
```

Verkettung mit 'Pipe' Symbol

EJS Filter

■ Eigene Filter definieren

```
var ejs = require('ejs');
var template = '<%=: price * 1.145 | round:2 %>';
var context = {
  price: 21
};

ejs.filters.round = function(number, decimalPlaces) {
  number = isNaN(number) ? 0 : number;
  decimalPlaces = !decimalPlaces ? 0 :
    decimalPlaces;
  var multiple = Math.pow(10, decimalPlaces);
  return Math.round(number * multiple) / multiple;
};

console.log(ejs.render(template, context));
```

Filter definieren

round ist nicht vordefiniert

Konfigurieren von Views mit Express

- Einstellen des "Views" Verzeichnisses (enthält die Templates)

```
app.set('views', __dirname + '/views');
```

- Einstellen der Template-Engine

Globale Variable in Node

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

- Hinweis: eine Anwendung kann mehrere Template-Engines gleichzeitig nutzen
- Für node.js gibt es zahlreiche Template-Engines

Daten an einen View übertragen

```
var express = require("express");
var url = require("url");
var http = require("http");
var app;

var port = process.argv[2];
app = express();
http.createServer(app).listen(port);

var todos = [];
todos.push({message: 'Midterm exam tomorrow',dueDate: '12/11/2014'});
todos.push({message: 'Prepare for assignment 5',dueDate: '05/01/2015'});
todos.push({message: 'Sign up for final exam',dueDate: '06/01/2015'});

app.set('views', __dirname + '/views');
app.set('view engine', 'ejs');

app.get("/todos", function(req, res) {
  res.render('todos', {
    title: 'My list of TODOs',
    todo_array: todos
  });
});
```

Daten an einen View übertragen

```
var express = require("express");
var url = require("url");
var http = require("http");
var app;

var port = process.argv[2];
app = express();
http.createServer(app).listen(port);
```

ToDo Liste die dem Client
bereitgestellt werden soll

```
var todos = [];
todos.push({message: 'Midterm exam tomorrow', dueDate: '12/11/2014'});
todos.push({message: 'Prepare for assignment 5', dueDate: '05/01/2015'});
todos.push({message: 'Sign up for final exam', dueDate: '06/01/2015'});
```

```
app.render('views');
```

render zeigt die Nutzung
eines Templates an

```
app.get("/todos", function(req, res) {
  res.render('todos', {
    title: 'My list of TODOs',
    todo_array: todos
  });
});
```

Name des Templates (Datei ohne Endung)

Variablen des Templates

EJS Template Datei

```
<!DOCTYPE html>
<html>

<head>
  <title>
    <%= title %>
  </title>
</head>

<body>
  <h1>TODOs</h1>
  <div>
    <% todo_array.forEach(function(todo) { %>
      <div>
        <h3><%=todo.dueDate%></h3>
        <p>
          <%=todo.message%>
        </p>
      </div>
    <% }) %>
  </div>
</body>

</html>
```

JavaScript zwischen `<% %>` wird ausgeführt.

JavaScript zwischen `<%= %>` fügt Inhalte zur Ergebnisdatei hinzu.

Ein letztes Wort zu Templates

- Ejs behält noch die ursprünglichen HTML-Tags bei
- Andere Template-Sprachen nicht, **Pug** (ehem. **Jade**) ist ein beliebtes Beispiel - welche Sie wählen, hängt von Ihren persönlichen Vorlieben ab

```
doctype html
html(lang="en")
  head
    title= pageTitle
    script(type='text/javascript').
      if (foo) {
        bar(1 + 5)
      }
  body
    h1 Jade - node template engine
    #container.col
      if youAreUsingJade
        p You are amazing
      else
        p Get on it!
      p.
        Jade is a terse and simple
        templating language with a
        strong focus on performance
        and powerful features.
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Jade</title>
    <script type="text/javascript">
      if (foo) {
        bar(1 + 5)
      }
    </script>
  </head>
  <body>
    <h1>Jade - node template engine</h1>
    <div id="container" class="col">
      <p>You are amazing</p>
      <p>
        Jade is a terse and simple
        templating language with a
        strong focus on performance
        and powerful features.
      </p>
    </div>
  </body>
</html>
```

Literatur

- Learning Web App Development, Kapitel 4
- Ethan Brown, *"Web development with Node and Express"*, O'Reilly, 2014
- Robert Prediger ; Ralph Winzinger, *"Node.js"*, Hanser, 2015 (Online verfügbar im Hochschulnetz)
- Mike Cantelon, *"Node.js in action"*, Manning, 2014
- Connect, <https://github.com/senchalabs/connect>

