

# Multimedia Engineering II

## 09 Backbone.js Clientseitig REST APIs nutzen

Johannes Konert




BEUTH HOCHSCHULE  
FÜR TECHNIK  
BERLIN


University of Applied Sciences



## Agenda

- **Wiederholung NoSQL und mongoDB**
- **(Wdh!) Statische HTML-Seite ausliefern mit require.js darin**
-  **BACKBONE.JS**
  - Router
  - Models
  - Models + REST } Übungsaufgabe
  - Collections
  - Collections + REST
  - Backbone Events mit .on(...)
  - Views
  - DOM-Events } Übungsaufgabe
- **Template Engine UNDERSCORE.JS**
- **Zusammenfassende Fragen**
- **Ausblick auf nächsten Unterricht**

# Agenda

- **Wiederholung NoSQL und mongoDB**
- **(Wdh!) Statische HTML-Seite ausliefern mit require.js darin**
-  **BACKBONE.JS**
  - Router
  - Models
  - Models + REST } Übungsaufgabe
  - Collections
  - Collections + REST
  - Backbone Events mit .on(...)
  - Views
  - DOM-Events } Übungsaufgabe
- **Template Engine** UNDERSCORE.JS
- **Zusammenfassende Fragen**
- **Ausblick auf nächsten Unterricht**

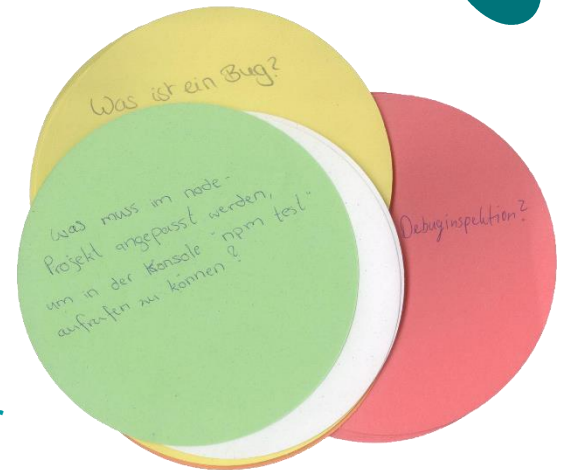
## Zusammenfassende Fragen und Wiederholung

### Wiederholung als Teamquiz

Der Raum wird in zwei Teams eingeteilt



1. Ein Teammitglied (A) zieht 2 Karten
  - Wählt eine von beiden aus
2. Stellt die Frage dem anderen Team (B)
  - Das Team darf diskutieren über die Lösung
3. Das Team (B) antwortet
  - Team A entscheidet, ob die Antwort korrekt war
4. Dozent als Schiedsrichter vergibt endgültigen Punkt an Team A oder B



**Anschließend ist das andere Team dran (1.-3.)**

Insgesamt werden **vier Fragen** besprochen.

## Agenda

- 
- 
- **Einschub: Zeitplan-Aktualisierung!**
-  **BACKBONE.JS**
  - Router
  - Models
  - Models + REST } Übungsaufgabe
  - Collections
  - Collections + REST
  - Backbone Events mit .on(...)
  - Views
  - DOM-Events } Übungsaufgabe
- **Template Engine** UNDERSCORE.JS
- **Zusammenfassende Fragen**
- **Ausblick auf nächsten Unterricht**

## ■ Zeitplan Zug 1 (vorläufig)

	Datum	Thema	Übung
1	05.04.2016		Ü1: Client-Website
2	12.04.2016	Einführung, Ziele, Ablauf, Benotung	Ü1
3	19.04.2016	Client-Server Architektur	Ü2: Server mit node.js
4	26.04.2016	REST-APIs	Ü2
5	03.05.2016	REST in node.js	- Feiertag - (5.5.)
6	10.05.2016	Debugging und Testen	Ü3: API mit node.js
7	17.05.2016	Strukturierung, Modularisierung	Ü3
8	24.05.2016	Vertiefung einzelner Themen	Ü4: Umfangreiche REST API
9	31.05.2016	Datenhaltung, SQL, NoSql, primär mit MongoDB	Ü4
10	07.06.2016	backbone.js als Gegenpart zu REST/node	Ü4
11	14.06.2016	Authentifizierung und Patterns	Ü5: mongoDB-Anbindung
12	21.06.2016	Gastdozent: mycs GmbH, CTO Claudio Bredtfeld	Ü5
13	28.06.2016	Mobile Development/Cross-Plattform-Development	(Ü5?) Ü6: Backbone.js
14	05.07.2016	Klausurvorbereitung	(Ü5?) Ü6
15	12.07.2016	Klausur PZR1 (Di, 12.07. 14:00 Uhr, Ingeborg-Meising-S.)	(Ü6?)
16	19.07.2016	Klausureinsicht	(Ü6?)
	21.09.2016	Klausur PZR2 (Mi, 21.09. 12:00 Uhr, Raum B101, H Gauß)	-

## ■ Zeitplan Zug 2 (vorläufig)

	Datum	Thema	Übung
1	06.04.2016		Ü1: Client-Website
2	13.04.2016	Einführung, Ziele, Ablauf, Benotung	Ü1
3	20.04.2016	Client-Server Architektur	Ü2: Server mit node.js
4	27.04.2016	REST-APIs	Ü2
5	04.05.2016	REST in node.js	- Feiertag - (5.5.)
6	11.05.2016	Debugging und Testen	Ü3: API mit node.js
7	18.05.2016	Strukturierung, Modularisierung	Ü3
8	25.05.2016	Vertiefung einzelner Themen	Ü4: Umfangreiche REST API
9	01.06.2016	Datenhaltung, SQL, NoSql, primär mit MongoDB	Ü4
10	08.06.2016	backbone.js als Gegenpart zu REST/node	Ü4
11	15.06.2016	Authentifizierung und Patterns	Ü5: mongoDB-Anbindung
12	22.06.2016	Gastdozent: mycs GmbH, CTO Claudio Bredtfeld	Ü5
13	29.06.2016	Mobile Development/Cross-Plattform-Development	(Ü5?) Ü6: Backbone.js
14	06.07.2016	Klausurvorbereitung	(Ü5?) Ü6
15	12.07.2016	Klausur PZR1 (Di, 12.07. 14:00 Uhr, Ingeborg-Meising-S.)	(Ü6?)
16	20.07.2016	Klausureinsicht	(Ü6?)
	21.09.2016	Klausur PZR2 (Mi, 21.09. 12:00 Uhr, Raum B101, H Gauß)	-

## Terminverschiebungen/Konflikte wegen Bewerbervorträgen

### Professur „Verteilte Systeme“ soll besetzt werden

- Einladung an Sie zur Teilnahme an den Lehrvorträgen der Bewerber/innen
- Die Stimme der Studierenden/Evaluation hat sehr hohes Gewicht bei der Auswahl

### Termine/Orte:

- |   |             |       |                            |        |
|---|-------------|-------|----------------------------|--------|
| ■ | 23.6. 14:15 | H5    | Betriebssysteme            | B-MI 2 |
| ■ | 27.6. 14:15 | H5    | Verteilte Systeme          | B-MI 4 |
| ■ | 28.6. 10:00 | B401  | Human Computer Interaction | B-MI 4 |
| ■ | 28.6. 16:00 | H3    | Betriebssysteme            | B-MI 2 |
| ■ | 29.6. 12:15 | H5    | Verteilte Systeme          | B-TI 4 |
| ■ | 29.6. 16:00 | D 209 | Betriebssysteme            | B-MI 2 |
| ■ | 30.6. 14:15 | H5    | Betriebssysteme            | B-MI 2 |
| ■ | 06.7. 12:15 | H5    | Verteilte Systeme          | B-TI 4 |



## Terminverschiebungen/Konflikte wegen Bewerbervorträgen

**Professur „Verteilte Systeme“ soll besetzt werden**

### Zug 2:

- Ausweichtermin für Mi, 29.6. statt 12:15 Uhr **bereits 10:00 Uhr in B023**
- Ausweichtermin für Mi, 06.7. statt 12:15 Uhr **bereits 10:00 Uhr in B444**

### Zug1:

- Übung 1b (Do, 14:15 Uhr) gibt es 2x Bewerbertermine stattdessen
  - **23.06.**
  - **30.06.**
- Daher: Kommen Sie zur Übung 1a (Do, 12:15 Uhr)

### Für beide Züge:

- Abgabe für Übungsblatt 5 und 6 auch bis zu 2 Wochen später möglich.

## Information zu Gastdozent:

am Di, 21.06. 12:15 Uhr in B301 und Mi, 22.06. in B042

■ Claudio Bredfeldt

■ CTO mycs GmbH

Thema:

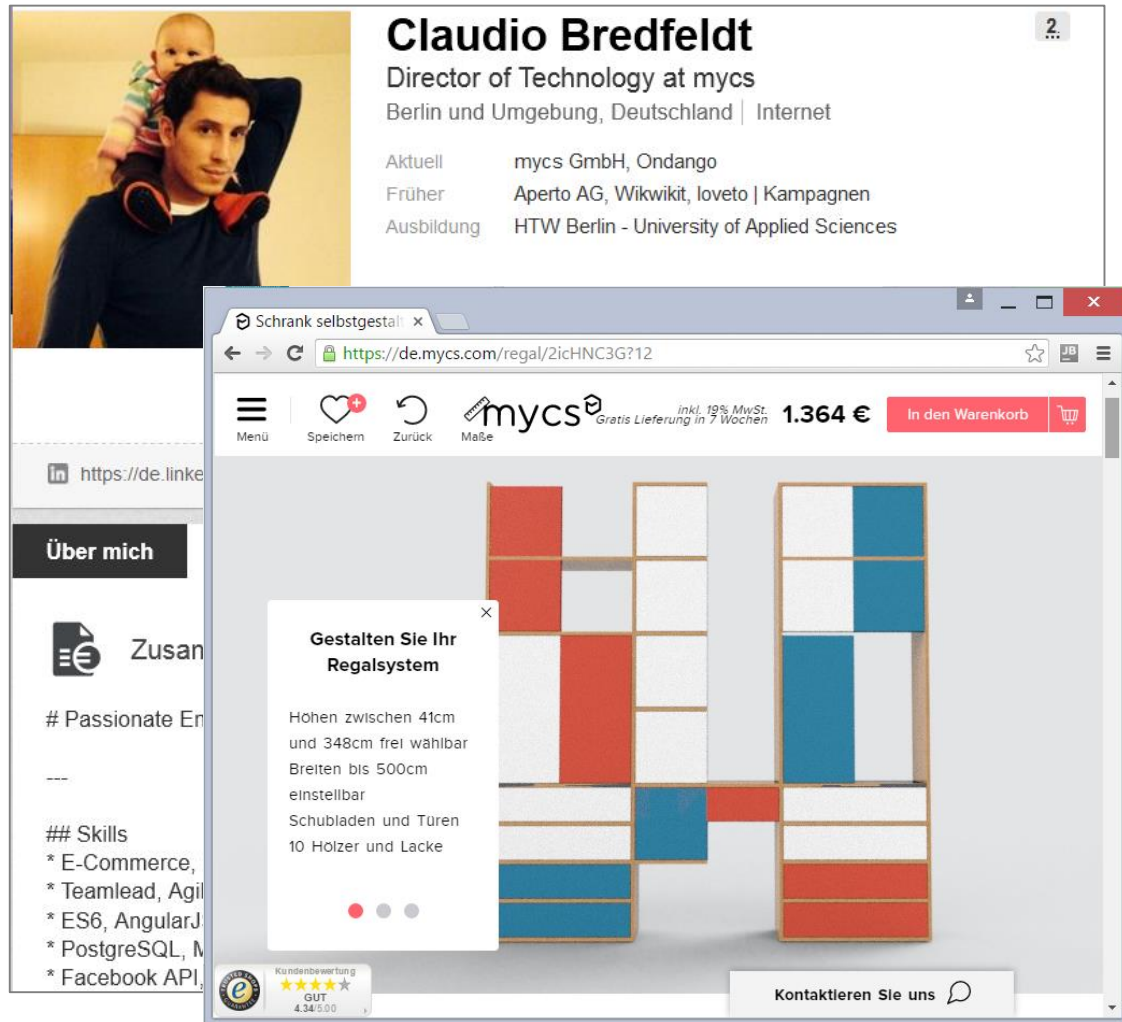
ToolChain und Einsatz von  
node.js / hapi.js bei mycs

mycs<sup>®</sup>

ondango



EUROWEB  
GROUP



The screenshot displays two overlapping windows. The background window is a LinkedIn profile for Claudio Bredfeldt, Director of Technology at mycs. His profile includes a photo of him holding a child, his current role at mycs in Berlin, and his previous experience at Aperto AG and Wikikit. The foreground window is a browser showing the mycs website, which features a 3D rendering of a modular shelving system. A pop-up window on the website provides details about the shelving system, including its dimensions and materials. The website also shows a price of 1.364 € and a 'In den Warenkorb' button.

**Claudio Bredfeldt**  
Director of Technology at mycs  
Berlin und Umgebung, Deutschland | Internet

Aktuell mycs GmbH, Ondango  
Früher Aperto AG, Wikikit, loveto | Kampagnen  
Ausbildung HTW Berlin - University of Applied Sciences

https://de.myics.com/regal/2icHNC3G?12

Menü Speichern Zurück Maße mycs<sup>®</sup> inkl. 19% MwSt. 1.364 € In den Warenkorb

**Gestalten Sie Ihr Regalsystem**

Hohen zwischen 41cm und 348cm frei wählbar  
Breiten bis 500cm einstellbar  
Schubladen und Türen  
10 Hölzer und Lacke

Kundenbewertung  
★★★★★  
GUT  
4.34/5.00

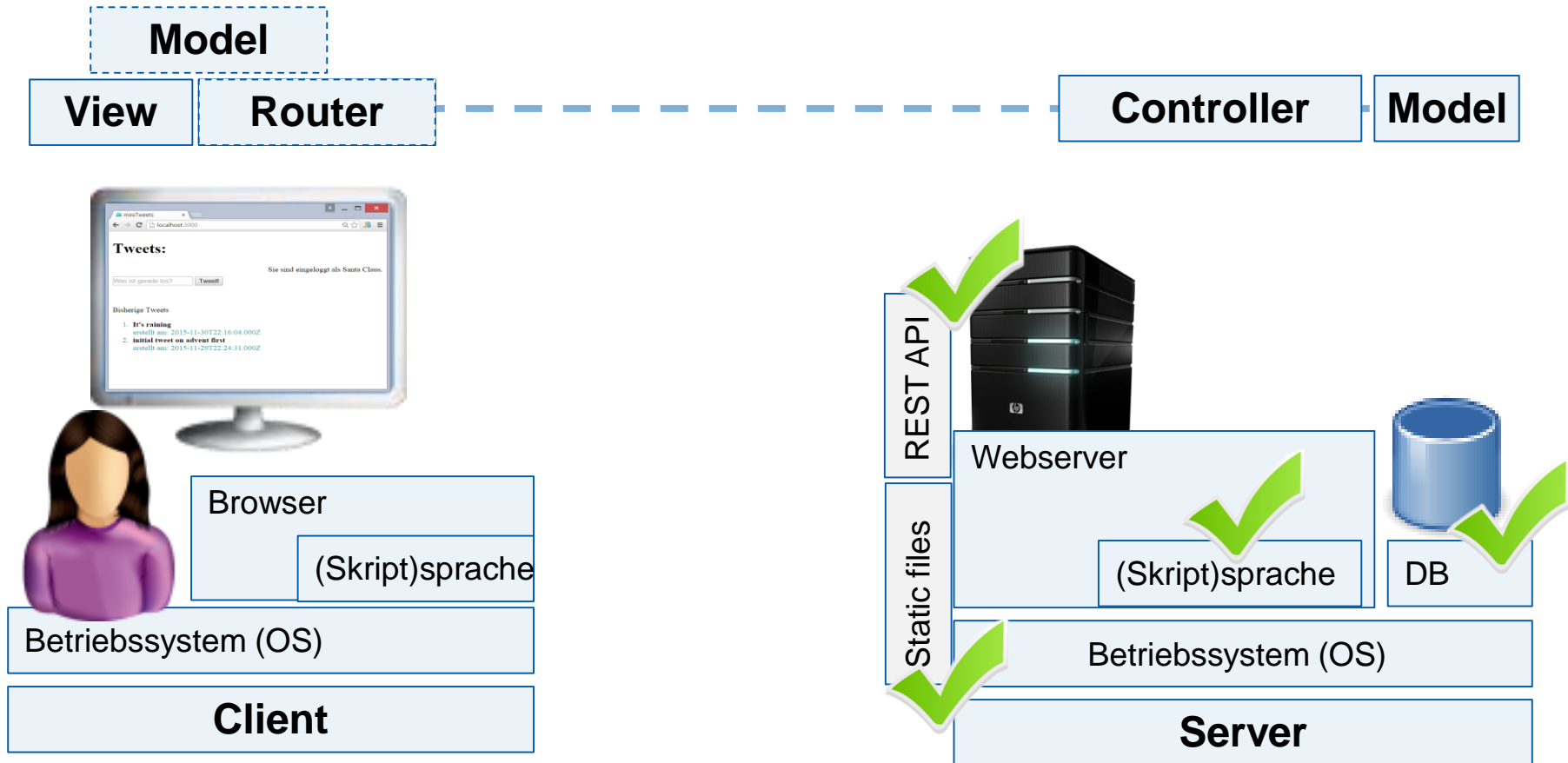
Kontaktieren Sie uns

## Agenda

- **Wiederholung NoSQL und mongoDB**
- **(Wdhl) Statische HTML-Seite ausliefern mit require.js darin**
-  **BACKBONE.JS**
  - Router
  - Models
  - Models + REST } Übungsaufgabe
  - Collections
  - Collections + REST
  - Backbone Events mit .on(...)
  - Views
  - DOM-Events } Übungsaufgabe
- **Template Engine** UNDERSCORE.JS
- **Zusammenfassende Fragen**
- **Ausblick auf nächsten Unterricht**

# Client-Server Architekturen

## ■ Rich-Client-Prinzip



## Ziel heute: Single Page App „miniTweets“

- **Laden** von REST-API /users und /tweets
- **Speichern**
- **Views** aktualisieren



## Eine erste statische Webseite ausliefern

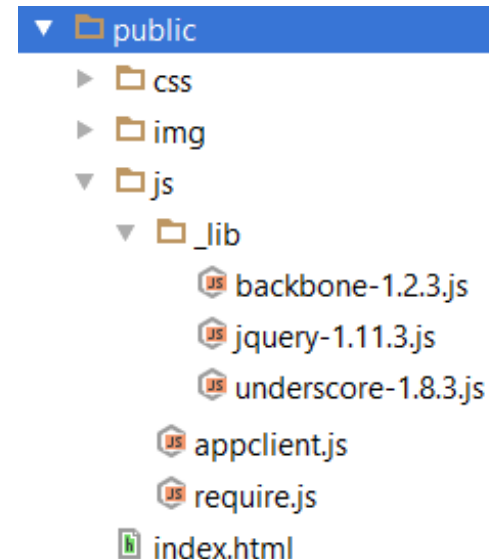


### ■ Anforderungen

- **HTML-Skelett**
- **Backbone.js** mit einbinden
  - Abhängigkeiten: **jQuery.js** für REST-API
  - Abhängigkeiten: **underscore.js** (auch als Template-Engine)
- Daher: **Require.js** für Abhängigkeiten und Kapselung nutzen

### ■ Umsetzung in node.js/express:

- **express.static** Middleware
- **index.html**
  - Bindet **require.js** ein
  - **require.js** lädt dann die **appclient.js** (und alles weitere)



## Eine erste statische Webseite ausliefern

### ■ index.html

```
<script data-main="js/appclient" src="js/require.js"></script>
```

### ■ appclient.js

```
require([ 'backbone' ], function(Backbone) {  
    console.log(Backbone); // [Object ...]  
    ...  
});
```

- leider ist Backbone nicht AMD kompatibel
- doch require.js liefert die sog. shim-Unterstützung mit

## Eine erste statische Webseite ausliefern

### ■ appclient.js enthält vorher:

```
requirejs.config({  
  baseUrl: '/js',  
  paths: {  
    jquery: './_lib/jquery-1.11.3',  
    underscore: './_lib/underscore-1.8.3',  
    backbone: './_lib/backbone-1.2.3'  
  },  
  shim: {  
    underscore: {  
      exports: '_'  
    },  
    backbone: {  
      deps: ['underscore', 'jquery'],  
      exports: 'Backbone'  
    }  
  }  
});
```


requirejs braucht ~immer~ eine Pfadangabe für Module, die nur per Namen geladen werden (um die zu finden)

underscore und backbone sind nicht AMD kompatibel. Daher wird angegeben welches Objekt aus den .js Dateien „exportiert“ wird

```
require(['backbone'], function(Backbone) { ... });
```



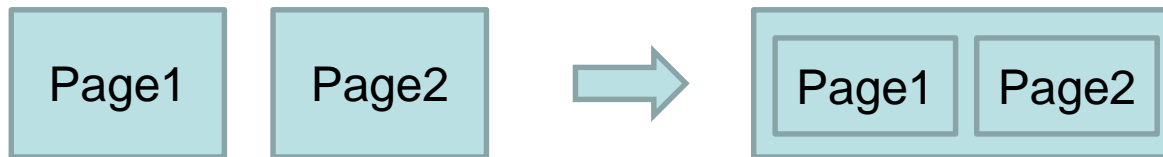
## Agenda

- Wiederholung NoSQL und mongoDB
- (Wdhl) Statische HTML-Seite ausliefern mit require.js darin
-  BACKBONE.JS
  - Router
  - Models
  - Models + REST } Übungsaufgabe
  - Collections
  - Collections + REST
  - Backbone Events mit .on(...)
  - Views
  - DOM-Events } Übungsaufgabe
- Template Engine UNDERSCORE.JS
- Zusammenfassende Fragen
- Ausblick auf nächsten Unterricht

## Backbone: Client-seitig modular entwickeln

### ■ Backbone

- Framework für Single Page Applications
- komplette Applikation in einer HTML-Seite
- asynchrones Austauschen der Views je nach Aktion
- spart Ladezeiten
- ermöglicht zentralen Anwendungszustand (Kontext)



## Backbone: Client-seitig modular entwickeln

- Die Navigation findet über einen sog. Router (ähnlich REST) statt und wird über Anchors realisiert

```
http://domain/page1.htm
```

```
http://domain/page2.htm
```



```
http://domain/index.htm#page1
```

```
http://domain/index.htm#page2
```

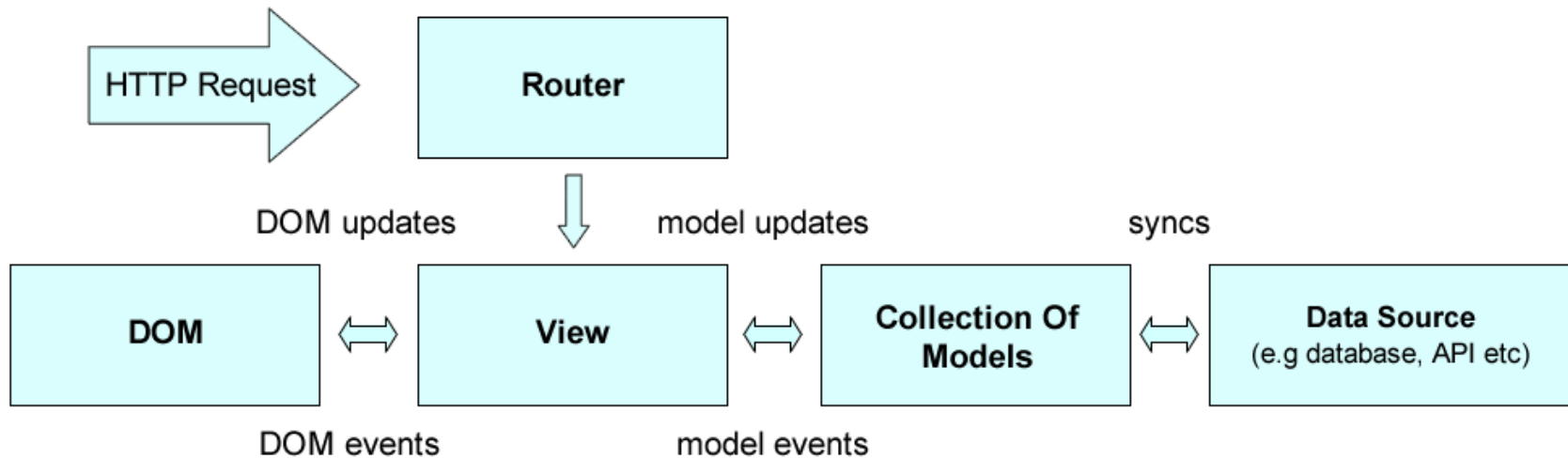
- **Fazit:**  
**Die Applikation befindet sich immer auf einer einzigen Seite!**

## Backbone: Client-seitig modular entwickeln

### ■ Backbone Komponenten

- Router
- View (DOM + View)
- Model
- Collection

### ■ Zusammenspiel (Workflow)



## Agenda

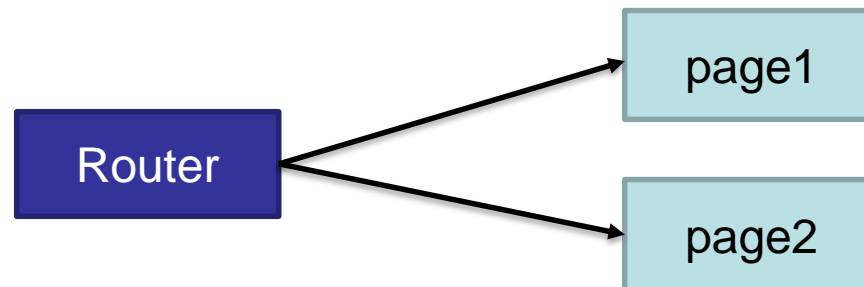
- **Wiederholung NoSQL und mongoDB**
- **(Wdh!) Statische HTML-Seite ausliefern mit require.js darin**
-  **BACKBONE.JS**
  - **Router**
  - **Models**
  - **Models + REST** } **Übungsaufgabe**
  - **Collections**
  - **Collections + REST**
  - **Backbone Events mit .on(...)**
  - **Views**
  - **DOM-Events** } **Übungsaufgabe**
- **Template Engine** UNDERSCORE.JS
- **Zusammenfassende Fragen**
- **Ausblick auf nächsten Unterricht**

## Backbone Router

- Überwachung der Navigation
- Navigation über Anchors
- Initialisiert die Views
- Führt pro Route erforderliche Schritte aus

```
http://domain/#page1
```

```
http://domain/#page2
```



## Backbone Router

- Backbone bietet eigenes Vererbungs-Management an
  - `extend( {...})`

```
var AppRouter = Backbone.Router.extend( {  
    ...  
} );
```

- Anschließend: Instanzieren des Routers

```
var myAppRouter = new AppRouter();
```

## Backbone Router

- Mapping von Routen zu Handler-Funktionen
  - im Attribut `routes`


```
var AppRouter = Backbone.Router.extend({  
  routes: {  
    'page1' : 'showPage1',  
    'page2' : 'showPage2'  
  }  
});  
  
var myAppRouter = new AppRouter();
```



## Backbone Router

- Mapping von Routen zu Handler-Funktionen
  - im Attribut `routes`
  - Ziel-Funktionen dann auf dem Konstruktor `AppRouter` ebenfalls

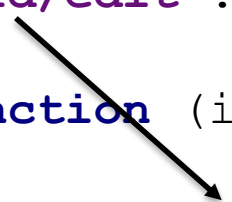
```
var AppRouter = Backbone.Router.extend({  
  routes: {  
    'page1' : 'showPage1',  
    'page2' : 'showPage2'  
  },  
  showPage1: function() {  
  },  
  showPage2: function() {  
  }  
});  
  
var myAppRouter = new AppRouter();
```



## Backbone Router

- Mapping von Routen zu Handler-Funktionen
  - erlaubt auch `:parameter`

```
var AppRouter = Backbone.Router.extend({  
  routes: {  
    'pages/:id': 'showPage',  
    'pages/:id/edit': 'editPage'  
  },  
  showPage: function (id) {  
  },  
  editPage: function (id) {  
  }  
});  
var myAppRouter = new AppRouter();
```




## Backbone Router

### ■ Mapping von Routen zu Handler-Funktionen

- erlaubt auch `:parameter`
- und am Ende auch `*variable`

```
var AppRouter = Backbone.Router.extend({  
  routes: {  
    'this/is/a/*path' : 'showPath'  
  },  
  showPath: function(path) {  
  }  
});  
var myAppRouter = new AppRouter();
```



- Aufruf: <http://myapp.com/#this/is/a/path/i/want/to/show>
- wird in Funktion showPath zu: `path` → `path/i/want/to/show`

## Backbone History

### ■ Navigation von Route zu Route über

```
myAppRouter.navigate('pages/'+page.id+'/edit', {trigger: true});
```

- **trigger** löst auch die entsprechende Router-Funktion aus
- **ohne trigger** wird URL nur im Browser (und in History) neu gesetzt

### Damit das funktioniert muss 1x :


```
Backbone.history.start({pushState: true});
```

- **pushState** versucht die Navigation über **#** durch reine URLs zu ersetzen (wenn Browser das unterstützt)
  - <http://localhost:3000/#pages/1212/edit>
  - wird zu <http://localhost:3000/pages/1212/edit>

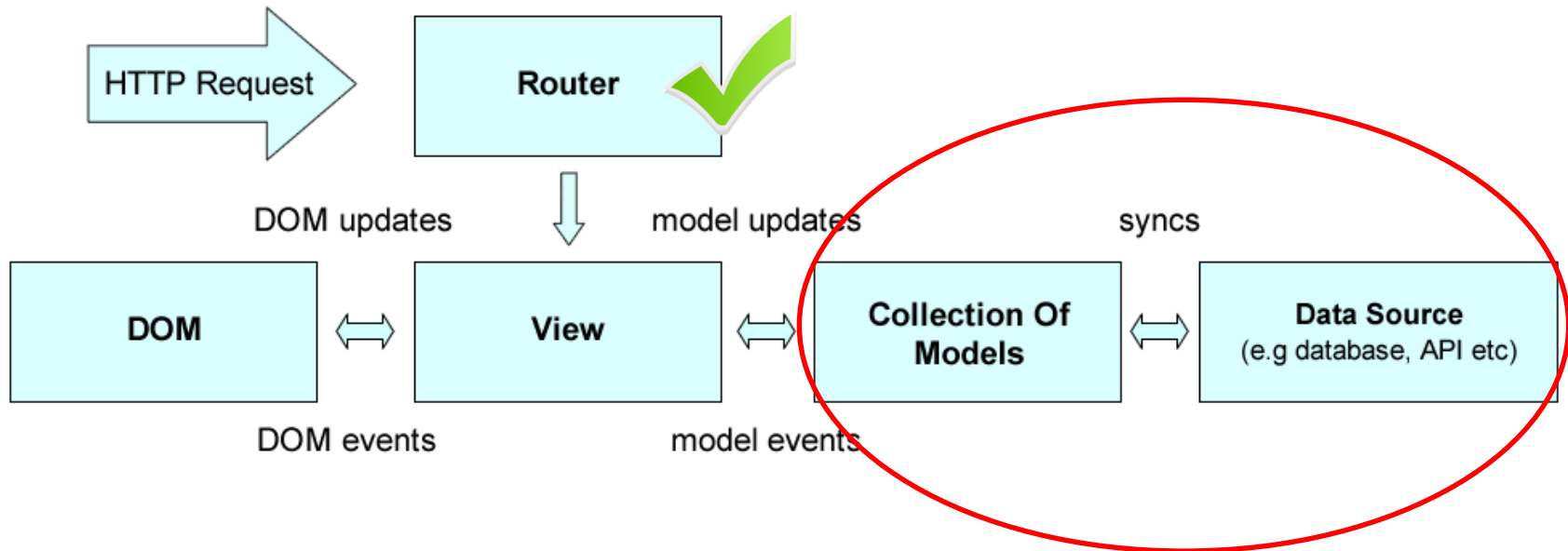
# Backbone Router

```
var AppRouter = Backbone.Router.extend({  
  routes: {  
    'this/is/a/*path' : 'showPath'  
  },  
  showPath: function(path) {  
    ??? (bisher passiert „nichts“)  
  }  
});  
var myAppRouter = new AppRouter();  
  
Backbone.history.start({pushState: true});
```

## Agenda

- **Wiederholung NoSQL und mongoDB**
- **(Wdh!) Statische HTML-Seite ausliefern mit require.js darin**
-  **BACKBONE.JS**
  - **Router**
  - **Models**
  - **Models + REST** } **Übungsaufgabe**
  - **Collections**
  - **Collections + REST**
  - **Backbone Events mit .on(...)**
  - **Views**
  - **DOM-Events** } **Übungsaufgabe**
- **Template Engine** UNDERSCORE.JS
- **Zusammenfassende Fragen**
- **Ausblick auf nächsten Unterricht**

# Backbone: Models



## Backbone: Models

- **alle erben von Backbone.Model**

```
var TweetModel = Backbone.Model.extend({  
  ...  
});
```

- **Abbildung eines Schemas und Modells**  
in einer JavaScript-Konstruktorfunktion

- **Viele Methoden** vorab dabei

- Getter/Setter
- Unset
- Default Values
- toJSON
- Event Triggers (z.B. change)
- Validation
- ...

- **Unterstützung für REST-Anbindung**



## Backbone: Models

- Bei der Definition lassen sich wichtige Attribute setzen

```
var TweetModel = Backbone.Model.extend({  
  idAttribute: "_id",  
  defaults: {  
    message: '',  
    creator: null,  
    timestamp: ''  
  },  
  initialize: function() {  
    // after constructor code  
  },  
  validate: function(attr) {  
    if (!attr.creator) {  
      return "Missing Creator ID";  
    }  
  }  
});
```

Standard-Attribute, die  
gesetzt werden sollen

erlaubt Code zum  
bspw. Initialisieren  
eines Templates etc.

wird autom. aufgerufen  
zum Prüfen gesetzter  
Model-Werte

## Backbone: Models

- Bei der Definition lassen sich wichtige Attribute setzen

```
var TweetModel = Backbone.Model.extend({  
  idAttribute: "_id",  
  defaults: {  
    message: '',  
    creator: null,  
    timestamp: ''  
  },  
  initialize: function() {  
    // after constructor code  
  },  
  validate: function(attr) {  
    if (!attr.creator) {  
      return "Missing Creator ID",  
    }  
  }  
});
```

**.id** ist  
Datenbankseitige ID.  
Wenn Name anders  
(MongoDB) hier  
angeben. Dann wird  
\_id zu .id und  
umgekehrt

**.cid** gibt's auch, ist  
Clientseitige ID  
(sinnvoll, wenn bspw.  
noch nicht in DB  
gespeichert)

## Backbone: Models

### ■ Neue Model-Instanzen anlegen und nutzen

```
var myTweet = new TweetModel();
```

```
var myTweet = new TweetModel({  
  message: 'First tweet in me2',  
  creator: '565b550960b2429c028fbbdb'  
});
```

### ■ Attribute lesen

```
var msg = myTweet.get('message');  
var msg = myTweet.id; // nur für id und cid
```

### ■ .toJSON() oder .attributes liefert für REST API fertige Daten (hier auch defaults mit drin)

```
myTweet.toJSON(); // {'message': 'First tweet in me2',  
                     'creator': '565b550960b2429c028fbbdb',  
                     'timestamp': ''}
```

## Backbone: Models

### ■ Attribute setzen

```
myTweet.set('message', 'A second tweet');  
myTweet.set({  
    message: 'a second tweet',  
    creator: '565b550960b2429c028fbbdb'  
});
```

- löst Events `change`, `change:message` und `change:creator` aus.
  - Überall in Backbone beobachtbar (Observer) via `.on(..)`

```
myTweet.on('change:message', function(tweet, message) {  
    // ...  
});
```

## Agenda

- **Wiederholung NoSQL und mongoDB**
- **(Wdh!) Statische HTML-Seite ausliefern mit require.js darin**
-  **BACKBONE.JS**
  - Router
  - Models
  - Models + REST } Übungsaufgabe
  - Collections
  - Collections + REST
  - Backbone Events mit .on(...)
  - Views
  - DOM-Events } Übungsaufgabe
- **Template Engine** UNDERSCORE.JS
- **Zusammenfassende Fragen**
- **Ausblick auf nächsten Unterricht**

## Backbone: Models mit REST-APIs verbinden

- Im Model wird für REST-Anbindung das Attribut **urlRoot** gesetzt.  
Fertig.

```
var TweetModel = Backbone.Model.extend({  
  urlRoot: '/tweets',  
  idAttribute: "_id",  
  defaults: {  
    ...  
  },  
  ...  
});
```

Funktion	HTTP	URL
create	POST	/tweets
read	GET	/tweets/id
update	PUT	/tweets/id
delete	DELETE	/tweets/id

## Backbone: Models mit REST-APIs verbinden

- **Vorteile der Model-Anbindung an REST**
  - übernimmt das HTTP-Methoden-Mapping
  - Automatische Übertragung der Daten als JSON
  - Automatische Übernahme der Antwortdaten in das Model inkl. entsprechender Events (bspw. für den View wichtig)

- **Beispiel POST (neuer Eintrag)**

```
var myTweet = new TweetModel({  
  message: 'First tweet in me2',  
  creator: '565b550960b2429c028fbbdb'  
});  
myTweet.save();
```

- Backbone stellt fest, dass dies eine neue Instanz ist (.cid existiert, keine .id) und löst ein CREATE über HTTP POST aus

```
POST /tweets HTTP/1.1  
{  
  "message": "First tweet in me2",  
  "creator": "565b550960b2429c028fbbdb",  
  "timestamp": ""  
}
```

## Backbone: Models mit REST-APIs verbinden

### ■ Beispiel GET (lesen eines Eintrags)

```
var myTweet = new TweetModel({  
  _id : '565b7b1f7675a40822c61628'  
});  
myTweet.fetch();
```

- Backbone stellt fest, dass dies eine existierende Instanz ist (.cid existiert, .id existiert) und löst ein READ über HTTP GET aus

```
GET /tweets/565b7b1f7675a40822c61628 HTTP/1.1
```

### ■ Antwort

```
{ "_id": "565b7b1f7675a40822c61628",  
  "updatedAt": "2016-06-06T22:24:31.000Z",  
  "timestamp": "2016-06-06T22:24:31.000Z",  
  "message": "first tweet in advent",  
  "creator": "565b550960b2429c028fbbdb",  
  "__v": 0 }
```



## Backbone: Models mit REST-APIs verbinden

### ■ Beispiel PUT (Überschreiben eines Eintrags)

```
...  
myTweet.set('message', 'First tweet by me');  
myTweet.save();
```

- Backbone stellt fest, dass dies eine existierende Instanz ist (.cid existiert, .id existiert), dass es geänderte Attribute gibt (.set() wurde aufgerufen) und löst ein UPDATE über HTTP PUT aus

```
PUT /tweets/565b7b1f7675a40822c61628 HTTP/1.1  
{ "_id": "565b7b1f7675a40822c61628",  
  "updatedAt": " 2016-06-06T22:24:31.000Z ",  
  "timestamp": "2016-16-06T22:24:31.000Z",  
  "message": "first tweet by me",  
  "creator": "565b550960b2429c028fbbdb",  
  "__v": 0 }
```

## Backbone: Models mit REST-APIs verbinden

### ■ Beispiel PATCH (Nur geänderte Daten senden)

```
myTweet.save(myTweet.changed, {patch: true});
```

- Backbone sammelt alle key:value Änderungen in **.changed**

```
PATCH /tweets/565b7b1f7675a40822c61628 HTTP/1.1  
{ "message": "first tweet by me" }
```

## Backbone: Models mit REST-APIs verbinden

### ■ Beispiel DELETE (Löschen eines Eintrags)


```
myTweet.destroy();
```

- **Backbone stellt fest, dass diese Instanz auch auf dem Server existiert (.id existiert) und löst ein DELETE über HTTP DELETE aus**

```
DELETE /tweets/565b7b1f7675a40822c61628 HTTP/1.1
```

- **Vorausgriff:** Alle Backbone Collections, in denen der Tweet enthalten war, werden aktualisiert.

## Agenda

- **Wiederholung NoSQL und mongoDB**
- **(Wdh!) Statische HTML-Seite ausliefern mit require.js darin**
-  **BACKBONE.JS**
  - Router
  - Models
  - Models + REST } Übungsaufgabe
  - Collections
  - Collections + REST
  - Backbone Events mit .on(...)
  - Views
  - DOM-Events } Übungsaufgabe
- **Template Engine** UNDERSCORE.JS
- **Zusammenfassende Fragen**
- **Ausblick auf nächsten Unterricht**

## Kurze Übung zu Models

### Was kommt wo rein?

1. Ordnen Sie gemeinsam mit Ihren Sitznachbarn/innen die Fragmente den Buchstaben zu (2min)
2. Anschließend Befragung zu A-F  
(je Team eine Antwort)



```
var TweetModel = ?A? ({  
  urlRoot: '/tweets',  
  idAttribute: ?B?,  
  defaults: ?C?,  
  initialize: ?D?,  
  validate: ?E?  
});
```

```
var myTweet = ?F? ();
```

- (1) `'_id'`
- (2) `new TweetModel`
- (3) `function(options) { ... }`
- (4) `Backbone.Model.extend`
- (5) `function(attr) { ... }`
- (6) `{ message: '',  
 creator: null,  
 timestamp: '' }`

## Kurze Übung zu Models: Lösung

```
var TweetModel = Backbone.Model.extend({  
  urlRoot: '/tweets',  
  idAttribute: '_id',  
  defaults: {  
    message: '',  
    creator: null,  
    timestamp: ''  
  },  
  initialize: function() {  
    // after constructor code  
  },  
  validate: function(attr) {  
    if (!attr.creator) {  
      return "Missing Creator ID";  
    }  
  }  
});
```

(4)

(1)

(6)


(3)

(5)

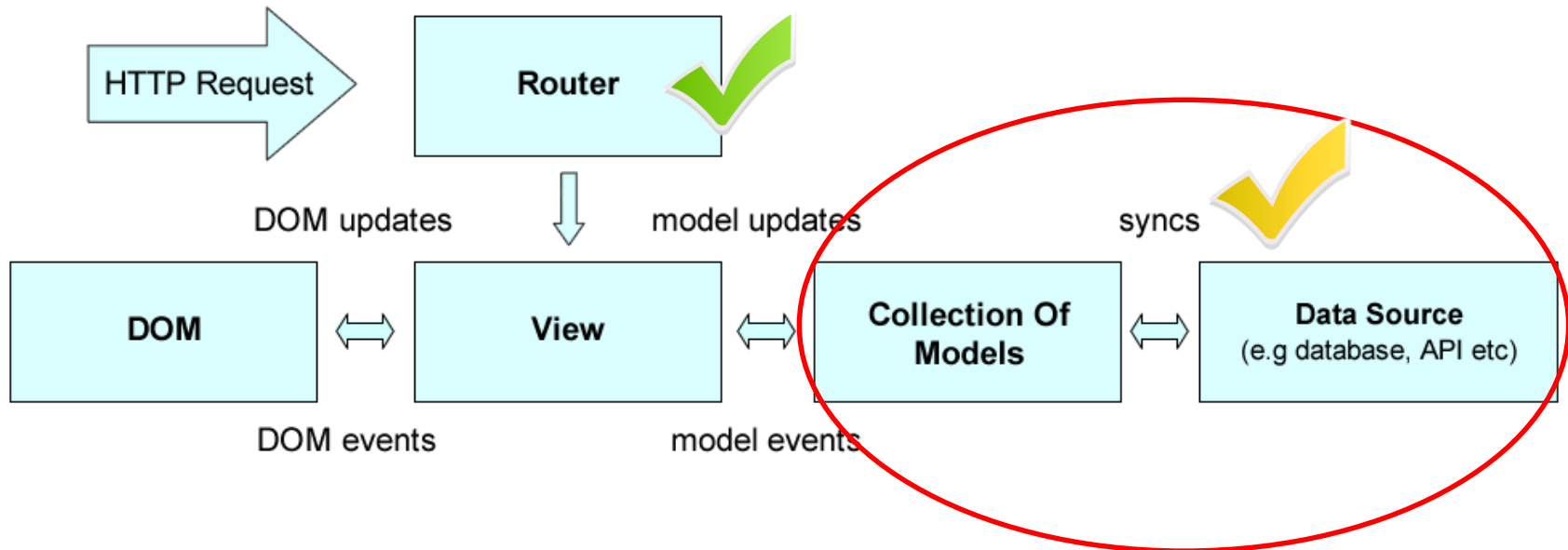
(2)

```
var myTweet = new TweetModel();
```

## Agenda

- **Wiederholung NoSQL und mongoDB**
- **(Wdh!) Statische HTML-Seite ausliefern mit require.js darin**
-  **BACKBONE.JS**
  - **Router**
  - **Models**
  - **Models + REST** } **Übungsaufgabe**
- **Collections**
- **Collections + REST**
- **Backbone Events mit .on(...)**
- **Views**
- **DOM-Events** } **Übungsaufgabe**
- **Template Engine** UNDERSCORE.JS
- **Zusammenfassende Fragen**
- **Ausblick auf nächsten Unterricht**

## Backbone: Models mit REST-APIs verbinden





## Backbone: Collections

- **Backbone Collections sind Sammlungen von Model-Instanzen**
  - **Praktisch für Listenverwaltung (und Kopplung mit Views für Listen)**
- **Collection-Definition:**

```
...  
var TweetCollection = Backbone.Collection.extend({  
  model: TweetModel,  
  url: '/tweets',  
  initialize: function() {  
    ...  
  }  
});
```

## Backbone: Collections

- Collections können
  - leer
  - oder mit Array an Modelsinitialisiert werden

```
var tweet1 = new TweetModel({message: 'Read more books',
                             creator: user.id});
var tweet2 = new TweetModel({message: 'Wake up!',
                             creator: user.id});

//no array
var tweets = new TweetCollection();
console.log("Collection size: " + tweets.length); // 0

// array
var tweets = new TweetCollection([tweet1, tweet2]);
console.log("Collection size: " + tweets.length); // 2
```

## Backbone: Collections

- Collections bieten ähnlich einem Array übliche Methoden

```
var a = new TweetModel ({message: 'Read more books', creator: user.id});  
var b = new TweetModel ({message: 'Wake up!', creator: user.id});  
var c = new TweetModel ({message: 'Tweet for me', creator: user.id});  
  
tweets.add(a);  
tweets.add([b, c]);  
  
tweets.remove(a);  
tweets.remove([b, c]);
```

- Weitere Methoden: `.push()`, `.pop()`, `.slice()`, `at()`, ...
- **Wichtig:** Ein Model, welches über `.destroy()` entfernt wurde (auch auf dem Server damit) wird automatisch aus Collections entfernt

## Backbone: Collections

- Collections fügen nicht ein, was schon drin ist (über `.id` oder `.cid` geprüft)
- Praktisch: `merge` gibt's als Option

```
var items = new Backbone.Collection();

items.add([ { id : 1, name: "Dog" , age: 3},
            { id : 2, name: "cat" , age: 2}]);

items.add([ { id : 1, name: "Bear" }], {merge: true });
items.add([ { id : 2, name: "lion" }]); // merge: false

console.log(items.toJSON());
//[{"id":1,"name":"Bear","age":3},{ "id":2,"name":"cat","age":2}]
```

## Backbone: Models aus Collections holen

- `.get()` kann mit `id` oder `cid` benutzt werden  
(Es gibt in Backbone zwei verschiedene ids für Models)
  - `id` – Persistierte id aus der DB
  - `cid` – Client-basierte id;  
beim Erzeugen der Model-Instanz jeweils sofort angelegt


## Backbone: Models aus Collections holen

### ■ .get(id)

```
var todos = new TodoCollection([a,b]);  
  
var myTodo = new TodoModel({title:'Read the book', id: 11});  
todos.add(myTodo);  
  
var todo2 = todos.get(11);  
// Models, as objects, are passed by reference  
console.log(todo2 === myTodo); // true
```

Id: 1 cid: 1234 Title: Go to bed	Id: 7 cid: 1235 Title: Go Home	Id: 11 cid: 1236 Title: Read the ...
--	--------------------------------------	--

## Agenda

- **Wiederholung NoSQL und mongoDB**
- **(Wdhl) Statische HTML-Seite ausliefern mit require.js darin**
-  **BACKBONE.JS**
  - Router
  - Models
  - Models + REST } Übungsaufgabe
  - Collections
- Collections + REST
- Backbone Events mit .on(...)
- Views
- DOM-Events } Übungsaufgabe
- **Template Engine** UNDERSCORE.JS
- **Zusammenfassende Fragen**
- **Ausblick auf nächsten Unterricht**

## Backbone Collection: auch die können REST

- In Collection wird für REST-Anbindung das Attribut **url** gesetzt. Fertig.

```
var TweetCollection = Backbone.Collection.extend({  
  model: TweetModel,  
  url: '/tweets',  
  initialize: function() {  
    ...  
  }  
});
```

Funktion	HTTP	URL
read	GET	/tweets



## Backbone Collection: auch die können REST

### ■ Lesen der Collection: GET

```
var tweets = new TweetCollection();  
tweets.fetch();
```


- Backbone wird ein Lesen über HTTP GET ausführen und die Collection mit dem Ergebnis abgleichen.

```
GET /tweets HTTP/1.1
```

- `.fetch()` erlaubt auch das Setzen bestimmter GET-Parameter, wie `?filter=` oder `?offset=` usw. (bspw. zum Blättern)

```
tweets.fetch({data:{limit: 20, offset: 10}});
```

## Agenda

- **Wiederholung NoSQL und mongoDB**
- **(Wdh!) Statische HTML-Seite ausliefern mit require.js darin**
-  **BACKBONE.JS**
  - Router
  - Models
  - Models + REST } Übungsaufgabe
  - Collections
  - Collections + REST
- Backbone Events mit .on(...)
- Views
- DOM-Events } Übungsaufgabe
- **Template Engine** UNDERSCORE.JS
- **Zusammenfassende Fragen**
- **Ausblick auf nächsten Unterricht**

## Backbone: Events zu Models und Collections

- **.on()** erlaubt es Listener (Observer) zu registrieren
  - **Geht für Collections und Models**

```
tweets.on('event', function(tweet) {  
    //do something  
});
```


- **Wesentliche Events sind**
  - **add**
  - **remove**
  - **change**
  - **change:attribute**
  - **reset**

## Backbone: Events zu Models und Collections

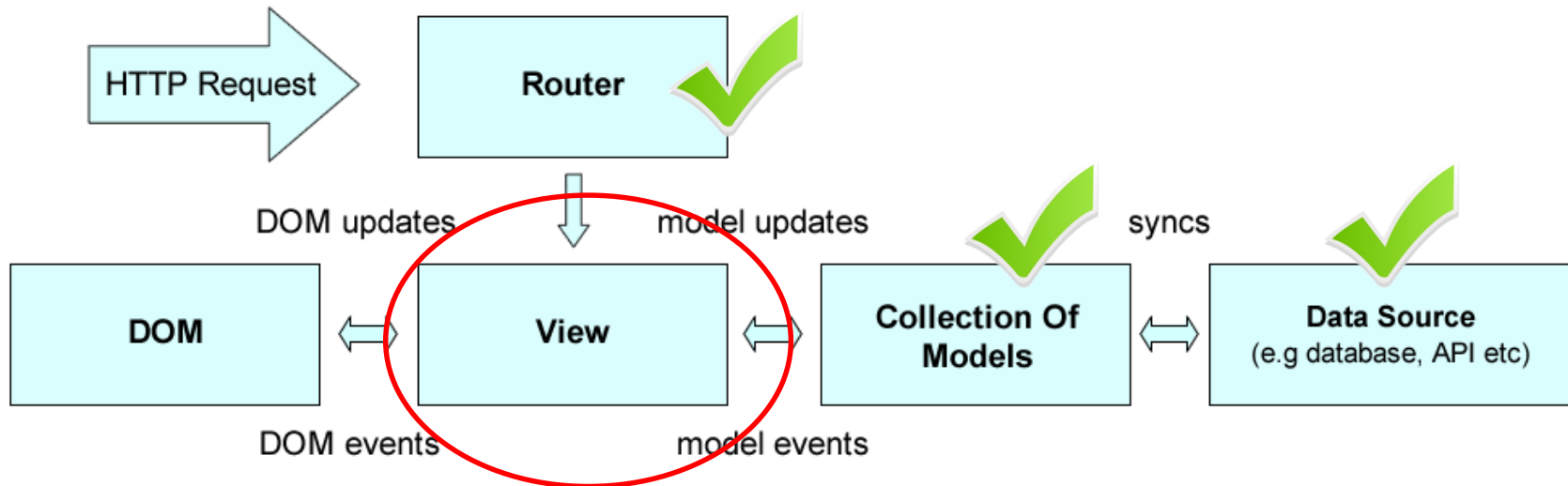
### ■ `.reset()` setzt ganze Collection zurück

```
var todos = new TodoCollection([a,b]);  
todos.on('reset', function(todos, options) {  
    console.log(options.previousModels);  
    console.log(options.previousModels[0] === a); // true  
});  
todos.reset();
```

## Agenda

- **Wiederholung NoSQL und mongoDB**
- **(Wdh!) Statische HTML-Seite ausliefern mit require.js darin**
-  **BACKBONE.JS**
  - Router
  - Models
  - Models + REST } Übungsaufgabe
  - Collections
  - Collections + REST
  - Backbone Events mit .on(...)
- Views } Übungsaufgabe
- DOM-Events }
- **Template Engine** UNDERSCORE.JS
- **Zusammenfassende Fragen**
- **Ausblick auf nächsten Unterricht**

## Backbone: View



# Backbone View

## Aufgaben des Views

- **Vermittlung zwischen DOM-Elementen und Model**
  - DOM-Events -> Model
  - Model-Events -> DOM
- Entsprechende **Erzeugung von DOM-Elementen**
- Kapselung und Verwaltung des/der **Templates**
- **Es gibt verschiedene Verwendungszwecke für Views**
  - **Enhancer**: View, der direkt an existierende DOM-Elemente anknüpft (und diese erweitert)
  - **Creator**: View, der eigene DOM-Elemente erstellt und irgendwo „eingehängt“ wird
  - **Delegator**: View, der selbst praktisch keine Elemente enthält (nur einen Referenzpunkt im DOM), jedoch weitere (Unter)Views unter sich verwaltet (Hierarchie)

## Backbone View

### ■ Definition eines **Enhancer** Views (Konstruktorfunktion)

```
var UserView = Backbone.View.extend({  
  el: '#user-login',  
  template: _.template($('#user-login-template').text()),  
  render: function() {  
    this.$el.html(this.template(this.model.attributes));  
    return this;  
  }  
});
```

```
var userView = new UserView({model: user});  
userView.render(); // result in DOM element #user-login
```

### ■ Generell wichtige View-Attribute

- **el**: Das DOM-Element, welches den View repräsentiert; quasi Root-Element des Views (ggf. leer und ggf. nicht im DOM eingehängt).
- **render**: Die Funktion, welche das el befüllt, so dass der View aktuell ist. Liefert immer this zurück, damit auf el zugegriffen werden kann.



## Backbone View

### ■ Definition eines **Enhancer** Views (Konstruktorfunktion)

```
var UserView = Backbone.View.extend({  
  el: '#user-login',  
  template: _.template($('#user-login-template').text()),  
  render: function() {  
    this.$el.html(this.template(this.model.attributes));  
    return this;  
  }  
});
```

```
var userView = new UserView({model: user});  
userView.render(); // result in DOM element #user-login
```

### ■ Warum gibt es **.el** und **.\$el** ?

- **el**: Das DOM-Element
- **.\$el**: Das DOM-Element als jQuery Objekt

## Backbone View

### ■ Definition eines **Creator Views** (Konstruktorfunktion)

```
var TweetView = Backbone.View.extend({  
  tagName: 'li',  
  className: 'tweet',  
  template: _.template($('#tweet-template').text()),  
  render: function() {  
    this.$el.html(this.template(this.model.attributes));  
    return this;  
  },  
  initialize: function() {  
    this.listenTo(this.model, 'change', this.render);  
  }  
});
```

- **tagName**: bestimmt welcher Tag als Parent/Root-Node dieses Views erstellt wird (default: div)
- **className**: und **id**: können (optional) ebenfalls gesetzt werden für den Tag
- **Dieser View ist zunächst nicht im DOM eingehängt!**

## Backbone View

### ■ Definition eines **Creator Views** (Konstruktorfunktion)

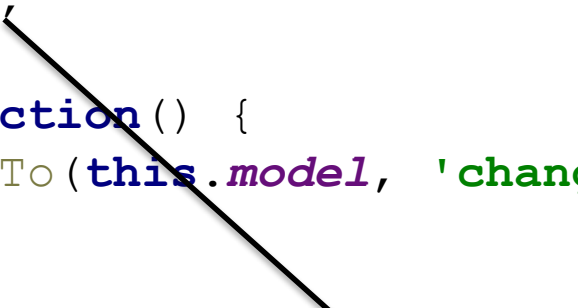
```
var TweetView = Backbone.View.extend({  
  tagName: 'li',  
  className: 'tweet',  
  template: _.template($('#tweet-template').text()),  
  render: function() {  
    this.$el.html(this.template(this.model.attributes));  
    return this;  
  },  
  initialize: function() {  
    this.listenTo(this.model, 'change', this.render);  
    this.model.on('change', function(tweet) {  
      this.render();  
    }, this);  
  }  
});
```

- **.listenTo(..)** ist eine kürzere Alternative zu **.on(..)**

## Backbone View

### ■ Definition eines **Creator Views** (Konstruktorfunktion)

```
var TweetView = Backbone.View.extend({  
  tagName: 'li',  
  className: 'tweet',  
  template: _.template($('#tweet-template').text()),  
  render: function() {  
    this.$el.html(this.template(this.model.attributes));  
    return this;  
  },  
  initialize: function() {  
    this.listenTo(this.model, 'change', this.render);  
  }  
});
```



### ■ Nutzung erfordert ein explizites Einhängen des gerenderten Elements

```
var tweetView = new TweetView({model: tweet});  
targetElement.append(tweetView.render().el);
```

## Backbone View

### ■ Definition eines **Delegator** Views (Konstruktorfunktion)

```
var TweetListView = Backbone.View.extend({  
  el: '#tweet-list',  
  template: undefined,  
  render: function() {  
    this.$el.empty();  
    this.collection.each(function(tweet) {  
      var tweetView = new TweetView({model: tweet});  
      this.$el.prepend(tweetView.render().el);  
    }, this);  
    return this;  
  },  
  initialize: function() {  
    this.listenTo(this.collection, 'add', this.render);  
  }  
});
```

- **Merkmal:** kein eigenes Template, dafür Nutzung anderer Views als Childs


## Backbone View

### ■ Definition und Nutzung v. **Delegator** Views (Konstruktorfunktion)

```
var TweetListView = Backbone.View.extend({  
  el: '#tweet-list',  
  template: undefined,  
  render: function() {  
    this.$el.empty();  
    this.collection.each(function(tweet) {  
      var tweetView = new TweetView({model: tweet});  
      this.$el.prepend(tweetView.render().el);  
    }, this);  
    return this;  
  },  
  initialize: function() {  
    this.listenTo(this.collection, 'add', this.render);  
  }  
});
```

```
var tweetListView = new TweetListView({collection: tweets});  
tweets.fetch({ ... success: tweetListView.render });
```

## Agenda

- **Wiederholung NoSQL und mongoDB**
- **(Wdh!) Statische HTML-Seite ausliefern mit require.js darin**
-  **BACKBONE.JS**
  - Router
  - Models
  - Models + REST } Übungsaufgabe
  - Collections
  - Collections + REST
  - Backbone Events mit .on(...)
  - Views
  - DOM-Events } Übungsaufgabe
- **Template Engine** UNDERSCORE.JS
- **Zusammenfassende Fragen**
- **Ausblick auf nächsten Unterricht**

## Backbone View: Reagieren auf DOM-Events

- Definition eines weiteren **Enhancer** Views (Konstruktorfunktion)
- Ziel: Eingabefeld und Button für Tweets „zum Leben erwecken“.

- HTML-Ausschnitt aus index.html

```
<section id="new-tweet">  
  <input type="text" placeholder="Was ist gerade los?">  
  <button type="button" id="#sendbutton">Tweet!</button>  
</section>
```



## Backbone View: Reagieren auf DOM-Events

### ■ Definition eines weiteren **Enhancer** Views (Konstruktorfunktion)

```
var TweetCreateView = Backbone.View.extend({  
  el: '#new-tweet',  
  events: {  
    'click #sendbutton': 'createTweet'  
  },  
  initialize: function(options) {  
    this.app = options.app; // expects a Backbone Router  
  },  
  createTweet: function() {  
    var input = this.$el.find('input');  
    if (input.val().trim()) {  
      this.collection.add({ message: input.val().trim(),  
                           creator: this.app.user.id});  
      input.val('');  
    }  
  },  
  ...  
});
```

**Dieser View hier braucht  
keine .render() Funktion!**

## Backbone View: Reagieren auf DOM-Events

- Definition eines weiteren **Enhancer Views** (Konstruktorfunktion)

```
var TweetCreateView = Backbone.View.extend({  
  el: '#new-tweet',  
  events: {  
    'click #sendbutton': 'createTweet',  
    'keypress input': 'createOnEnter'  
  }, ...  
});
```

- **events**: ermöglicht es Listener auf HTML-Kindelemente des Views im DOM zu registrieren.

- Syntax ist nach jQuery

- 'event selector': 'functionname,

- Weitere typische Events

- click, dblclick, mouseover, contextmenu, ...

## Agenda

- **Wiederholung NoSQL und mongoDB**
- **(Wdh!) Statische HTML-Seite ausliefern mit require.js darin**
-  **BACKBONE.JS**
  - Router
  - Models
  - Models + REST } Übungsaufgabe
  - Collections
  - Collections + REST
  - Backbone Events mit .on(...)
  - Views
  - DOM-Events } Übungsaufgabe
- **Template Engine** UNDERSCORE.JS
- **Zusammenfassende Fragen**
- **Ausblick auf nächsten Unterricht**

## Kurze Übung zu Views

### Was ist wofür?

1. Erörtern Sie in 2-3er Gruppen, wofür jedes der **neun** View-Attribute dient (2-3min).
2. Anschließend kurze Klärung noch offener Fragen




```
var LoginView = Backbone.View.extend({  
  (1) el: '#login',  
  (2) tagName: 'section',  
  (3) className: 'loginbox',  
  (4) id: 'render-login',  
  (5) template: _.template($('#1-templ').text()),  
  (6) model: currentUser,  
  (7) collection: userCollection,  
  (8) events: {'click #button': 'loginUser'},  
  (9) initialize: function(options) { ... },  
    loginUser: function() { ... },  
});
```

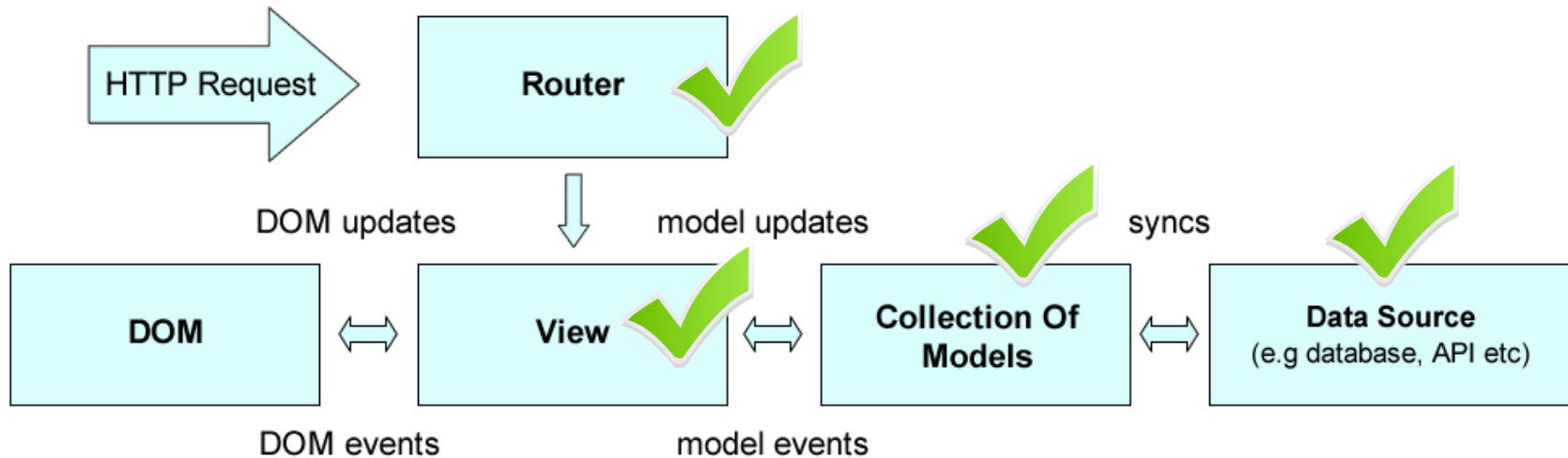
**Achtung:** Dies ist ein fiktiver (wenig) sinnvoller View, da viel zu viele Attribute gleichzeitig benutzt werden!

```
var myLoginView = new LoginView({ el: $('#login') });
```

## Agenda

- **Wiederholung NoSQL und mongoDB**
- **(Wdh!) Statische HTML-Seite ausliefern mit require.js darin**
-  **BACKBONE.JS**
  - Router
  - Models
  - Models + REST } Übungsaufgabe
  - Collections
  - Collections + REST
  - Backbone Events mit .on(...)
  - Views
  - DOM-Events } Übungsaufgabe
- **Template Engine** UNDERSCORE.JS
- **Zusammenfassende Fragen**
- **Ausblick auf nächsten Unterricht**

## Backbone: Models mit REST-APIs verbinden



# Template Engines

<b>dom.js</b> <a href="#">github</a>	<b>doT.js</b> <a href="#">project</a> (2.742k)	<b>dust.js (LinkedIn)</b> <a href="#">github</a> (9.3k)	<b>EJS</b> <a href="#">project</a> (9.8k)	<b>Handlebars.js</b> <a href="#">project</a>
<b>Hogan.js</b> <a href="#">project</a> (2.5k)	<b>ICanHaz.js</b> <a href="#">project</a> (5.445k)	<b>Jade templates</b> <a href="#">github</a> (39.687k)	<b>JsRender</b> <a href="#">project</a> (30.709k)	<b>Markup.js</b> <a href="#">github</a> (5.1k)
<b>Microtemplating</b> <a href="#">blog post</a> (1k)	<b>Mustache.js</b> <a href="#">github</a> (14.513k)	<b>Nunjucks</b> <a href="#">github</a> (20k)	<b>Plates.js</b> <a href="#">github</a> (10.811k)	<b>pure.js</b> <a href="#">project</a> (11.7k)
<b>Transparency</b> <a href="#">project</a> (5.491k)	<b>Underscore templates</b> <a href="#">project</a> (4k)			

# Template Engine Underscore

# UNDERSCORE.JS

- Sehr kleine Engine

  - 6 kB

- Einfach

- Template Beispiele

  - Ausgabe der Variable color (durch Zuweisung = )

```
<%= color %>
```

  - JavaScript im Template (durch Tags ohne Zuweisung)

```
<% _.each (people, function (name)  
{ %>  
    <li><%= name %></li>  
<% } ) ; %>
```



# Template Engine Underscore

## ■ Nutzung des Templates

1. **Kompilieren**
2. **Aufrufen**
3. **Ergebnis-String (HTML) irgendwo nutzen**

# Template Engine Underscore

## ■ Nutzung des Templates

1. Kompilieren
2. Aufrufen
3. Ergebnis-String (HTML) irgendwo nutzen

### 1. Kompilieren

```
var compiled = _.template("<div>hello: <%= name %></div>");
```

- Das Template wird durch die Template Engine vorkompiliert
- **Rückgabewert ist eine Funktion**, welche anhand eines übergebenen Parameterobjekts den entsprechenden String zurück gibt.

# Template Engine Underscore

## ■ Nutzung des Templates

1. Kompilieren
2. Aufrufen
3. Ergebnis-String (HTML) irgendwo nutzen

### 1. Kompilieren

```
var compiled = _.template("<div>hello: <%= name %></div>");
```

oder auch

```
var tweetcompiled = _.template($('#tweet-template').text())
```

(dann im HTML-Dokument dazu)

```
<script type="text/template" id="tweet-template">
  <span class="message"><%= message %></span><br>
  erstellt am: <%= timestamp %>
</script>
```

# Template Engine Underscore

## ■ Nutzung des Templates

1. Kompilieren
2. Aufrufen
3. Ergebnis-String (HTML) irgendwo nutzen

```
var compiled = _.template("...");
```

## 2. Aufrufen (sog. Rendern)

```
var result = compiled({name: 'moe'});
```

- Die Funktion `compiled` erwartet ein Objekt als Parameter, welches alle genutzten Variablennamen als Attribute enthalten muss
- Es können auch JSON-Objekte übergeben werden

## ■ Ergebnis-String

```
<div>hello: moe</div>
```

# Template Engine Underscore

## ■ Nutzung des Templates

1. Kompilieren
2. Aufrufen
3. Ergebnis-String (HTML) irgendwo nutzen

```
var compiled = _.template("...");
```

```
var result = compiled({name: 'moe'});
```

## 3. Ergebnis-String nutzen


```
$('#element').html(result);
```

- Der Rückgabewert der vorkompilierten Funktion ist ein String. Darin steckt HTML.

## ■ Fazit:

- **Templates** mit Underscore sind Funktionen, die Variablen nutzen
- **Das Parameterobjekt** zum Rendern des Templates muss alle genutzten Variablen als Attribute enthalten
- **Rückgabewert** ist ein gerenderter HTML-String

## Agenda

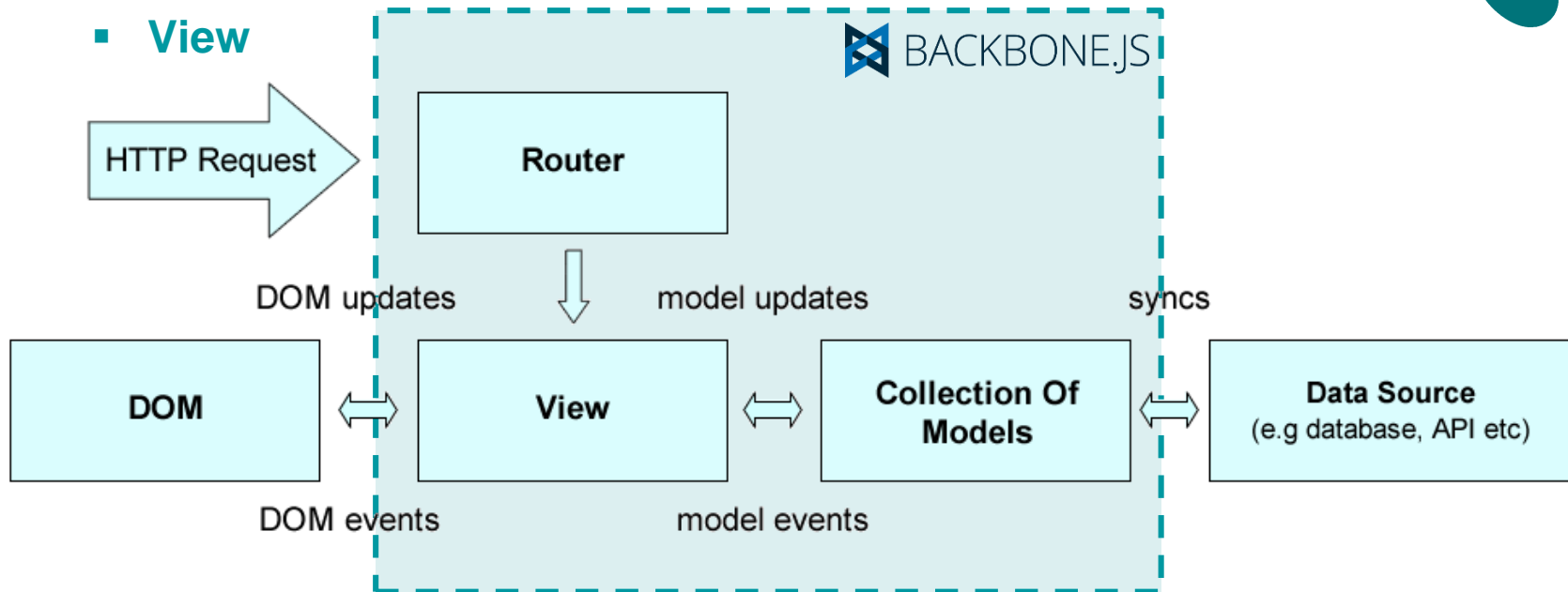
- **Wiederholung NoSQL und mongoDB**
- **(Wdhl) Statische HTML-Seite ausliefern mit require.js darin**
-  **BACKBONE.JS**
  - Router
  - Models
  - Models + REST } Übungsaufgabe
  - Collections
  - Collections + REST
  - Backbone Events mit .on(...)
  - Views
  - DOM-Events } Übungsaufgabe
- **Template Engine** UNDERSCORE.JS
- **Zusammenfassende Fragen**
- **Ausblick auf nächsten Unterricht**

## Zusammenfassende Fragen

### Welche vier Komponenten hat Backbone?



- Router
- Models und Collections (mit REST-Anbindung)
- View

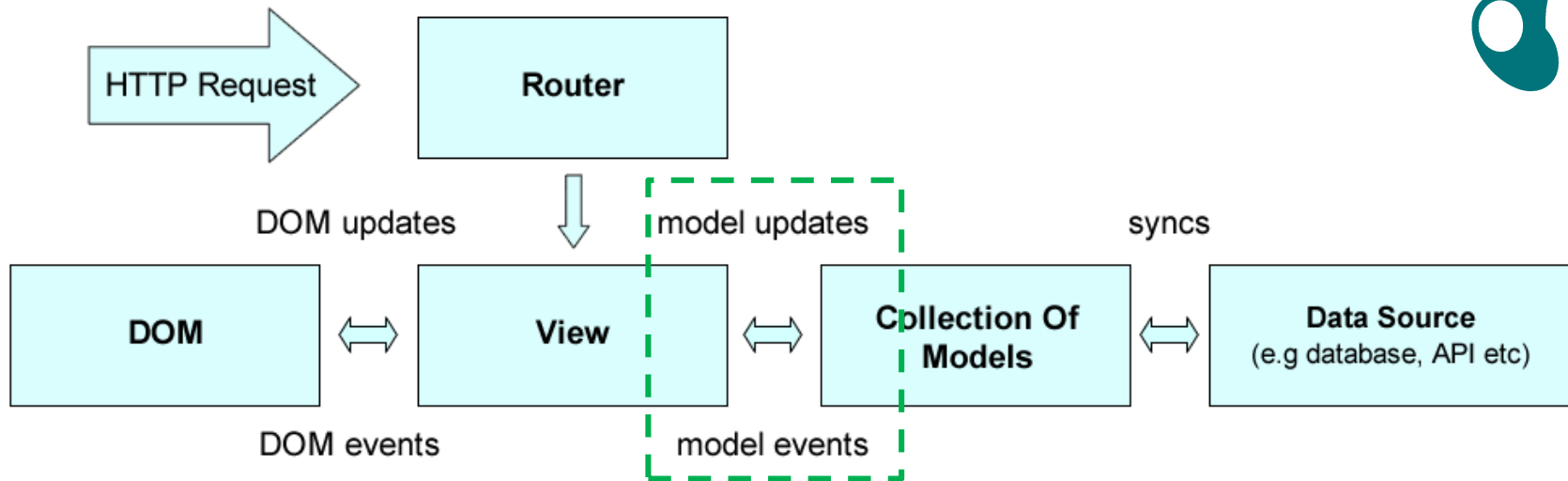


**Keine** Backbone Komponenten sind:

- Templates
- jQuery.js, underscore.js, require.js

## Zusammenfassende Fragen

### Wie gelingt bei Backbone die Verbindung von View und Model?



- Setzen des **model** od. **collection** Attributes im View von Außen (Dependency Injection)

```
var userView = new UserView({model: user});
```

```
var tweetListView = new TweetListView({collection: tweets});
```



## Agenda

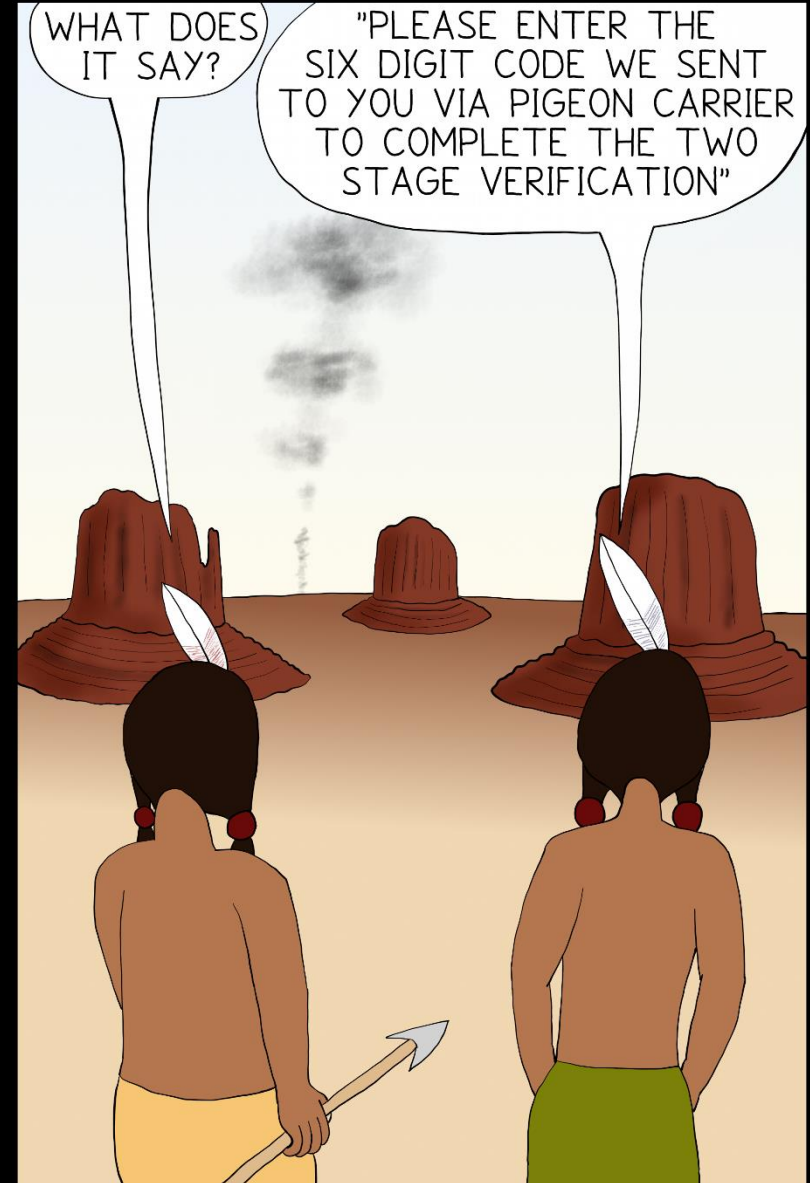
- **Wiederholung NoSQL und mongoDB**
- **(Wdh!) Statische HTML-Seite ausliefern mit require.js darin**
-  **BACKBONE.JS**
  - Router
  - Models
  - Models + REST } Übungsaufgabe
  - Collections
  - Collections + REST
  - Backbone Events mit .on(...)
  - Views
  - DOM-Events } Übungsaufgabe
- **Template Engine** UNDERSCORE.JS
- **Zusammenfassende Fragen**
- **Ausblick auf nächsten Unterricht**

## Ausblick: Nächster Unterricht

- Authentifizierung
- Patterns

**Vielen Dank  
und bis  
zum nächsten  
Mal**

### "Authentication"



<http://randomperspective.com/comic>

## Linksammlung zu Backbone und Clientseitigem MV\*

- <http://addyosmani.github.io/backbone-fundamentals/>
- <http://backbonejs.org/>
- [https://www.youtube.com/watch?v=jM8KE\\_Fa6JI](https://www.youtube.com/watch?v=jM8KE_Fa6JI) (sehr grundlegendes Einführungsvideo zu Backbone, etwas langsam..)
- <http://underscorejs.org/#template>
- <http://www.developerfusion.com/article/8307/aspnet-patterns-every-developer-should-know/>
- [http://schlingel.bplaced.net/mvc\\_mvp\\_mvvm\\_fuer\\_javascript.html](http://schlingel.bplaced.net/mvc_mvp_mvvm_fuer_javascript.html)
- <http://requirejs.org/docs/api.html#define>