

Multimedia Engineering II

10 Authentifizierung, Patterns

Johannes Konert



BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN

University of Applied Sciences



Agenda

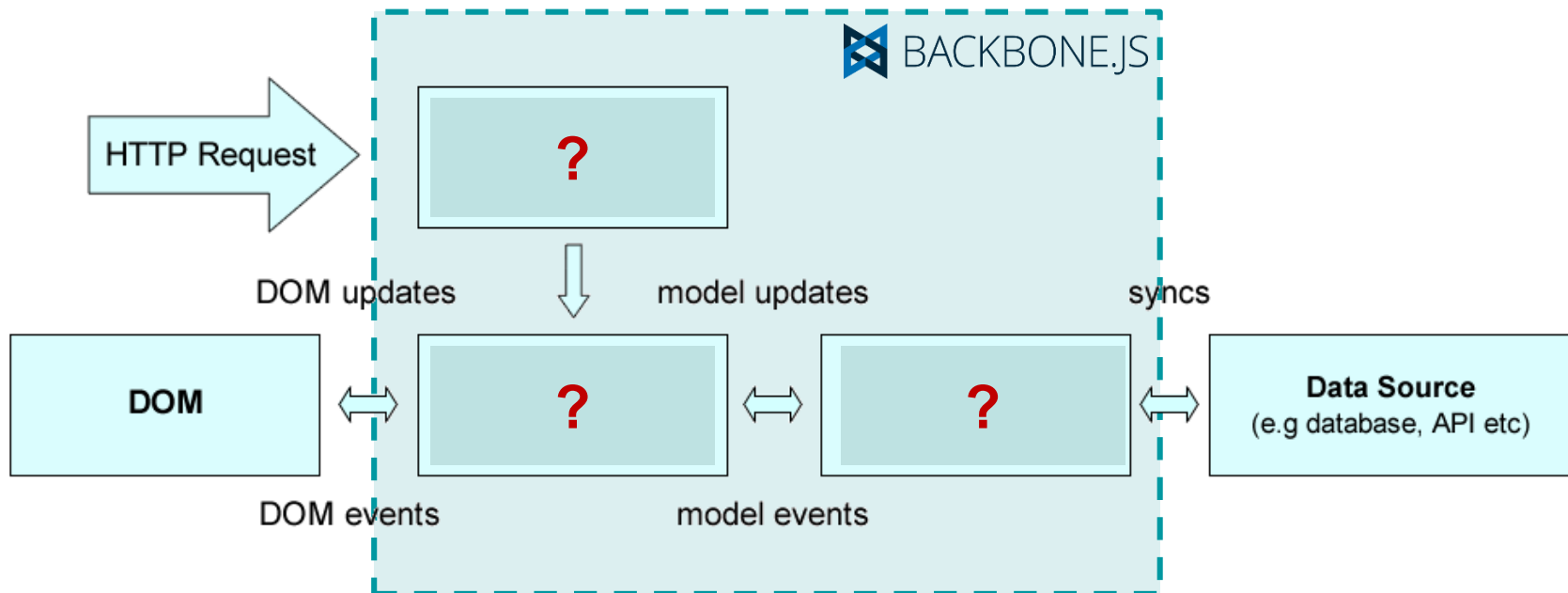
- **Wiederholung zu**  BACKBONE.JS
- **Template-Engine** UNDERSCORE.JS
- **Authentifizierung**
 - **HTTPS**
 - **BASIC Auth mit node.js**
 - **Vergleich mit OAuth 2.0**
 - **Local Auth mit node.js und**  **passport**
- **Patterns: MVC, MVP, MVVM, MVCVM, MV***
- **Zusammenfassung und Ausblick**

Agenda

- **Wiederholung zu**  **BACKBONE.JS**
- **Template-Engine** UNDERSCORE.JS
- **Authentifizierung**
 - **HTTPS**
 - **BASIC Auth mit node.js**
 - **Vergleich mit OAuth 2.0**
 - **Local Auth mit node.js und**  **passport**
- **Patterns: MVC, MVP, MVVM, MVCVM, MV***
- **Zusammenfassung und Ausblick**

Wiederholung: Backbone.js

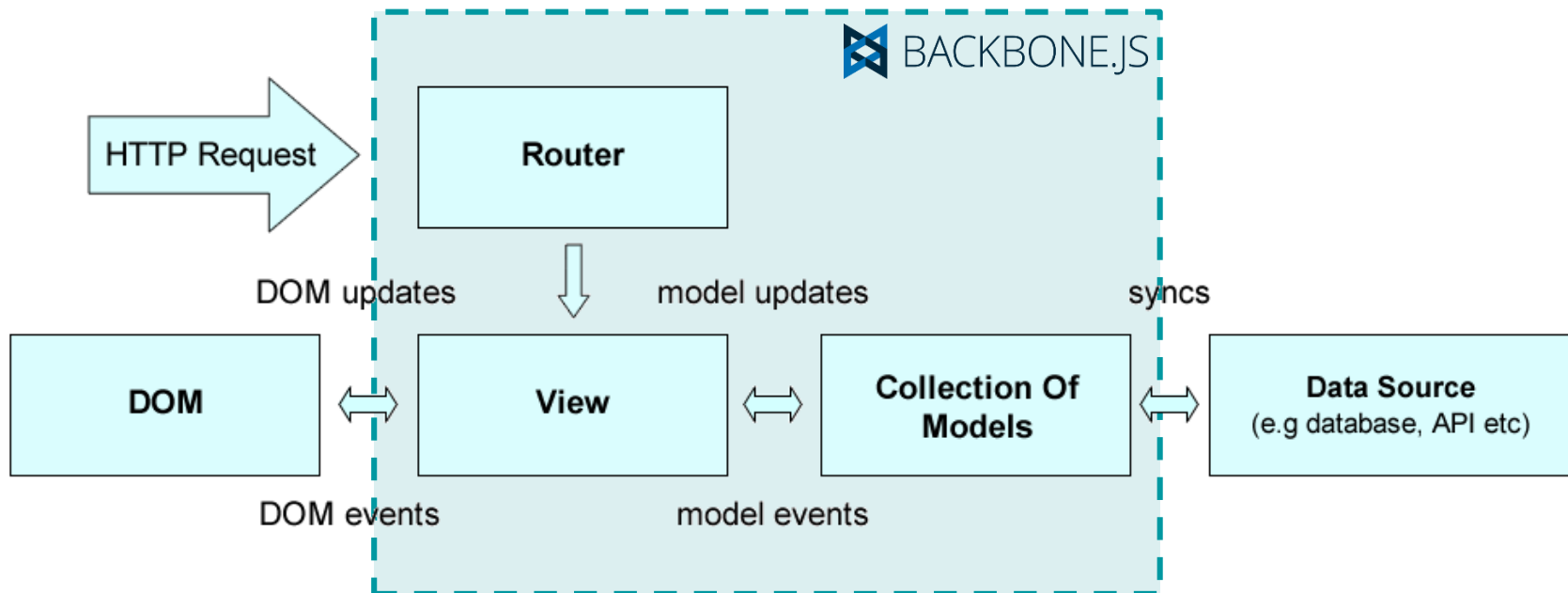
- Vier wesentliche Komponenten



Wiederholung: Backbone.js

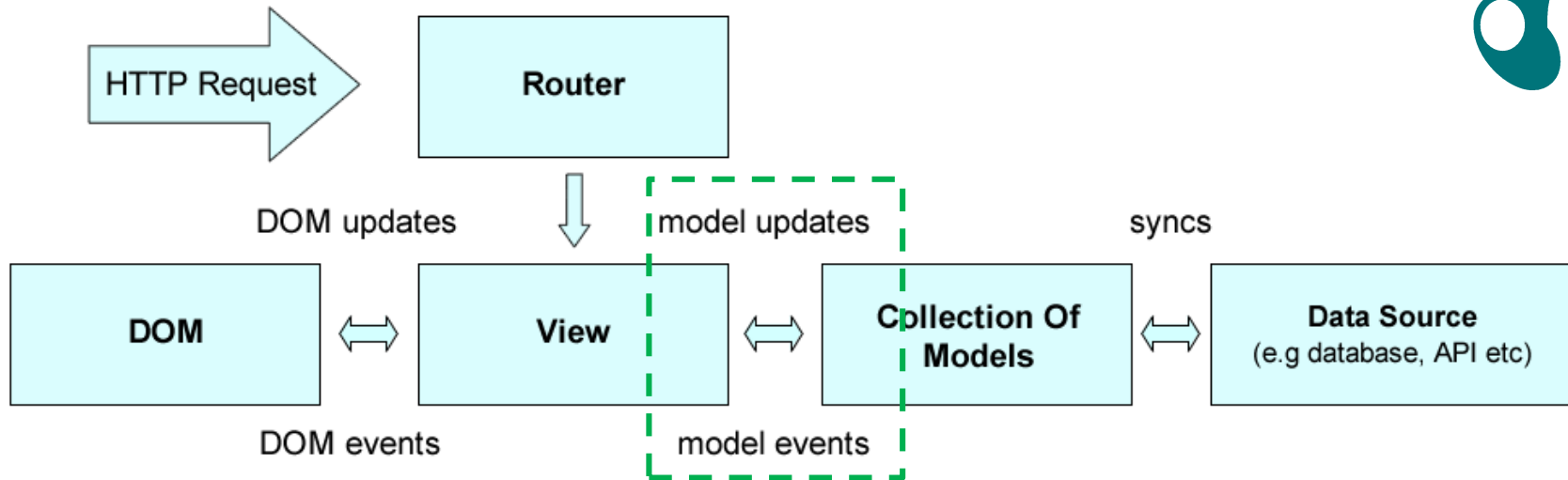
■ Vier wesentliche Komponenten

- Router
- Views
- Models und Collections



Wiederholung: Backbone.js

Wie gelingt bei Backbone die Verbindung von View und Model?



- Setzen des **model** od. **collection** Attributes im View von Außen

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

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

Wiederholung: Backbone.js Views

- Die wesentlichen **neun** Attribute eines Backbone-Views im Überblick.
- Diese können auch von „Außen“ als Konstruktor-Parameter gesetzt werden,
z.B. wenn ein Router neue Views instanziiert und eine Model-Instanz mitgibt.

```
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 myLoginV = new LoginView({ el: $('#login'), model: myUser });
```

Agenda

- Wiederholung zu  BACKBONE.JS
- **Template-Engine**  UNDERSCORE.JS
- **Authentifizierung**
 - HTTPS
 - BASIC Auth mit node.js
 - Vergleich mit OAuth 2.0
 - Local Auth mit node.js und  passport
- **Patterns: MVC, MVP, MVVM, MVCVM, MV***
- **Zusammenfassung und Ausblick**

Template Engines (nochmal langsam)

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

Backbone View (Wiederholung)

■ 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);
```

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.

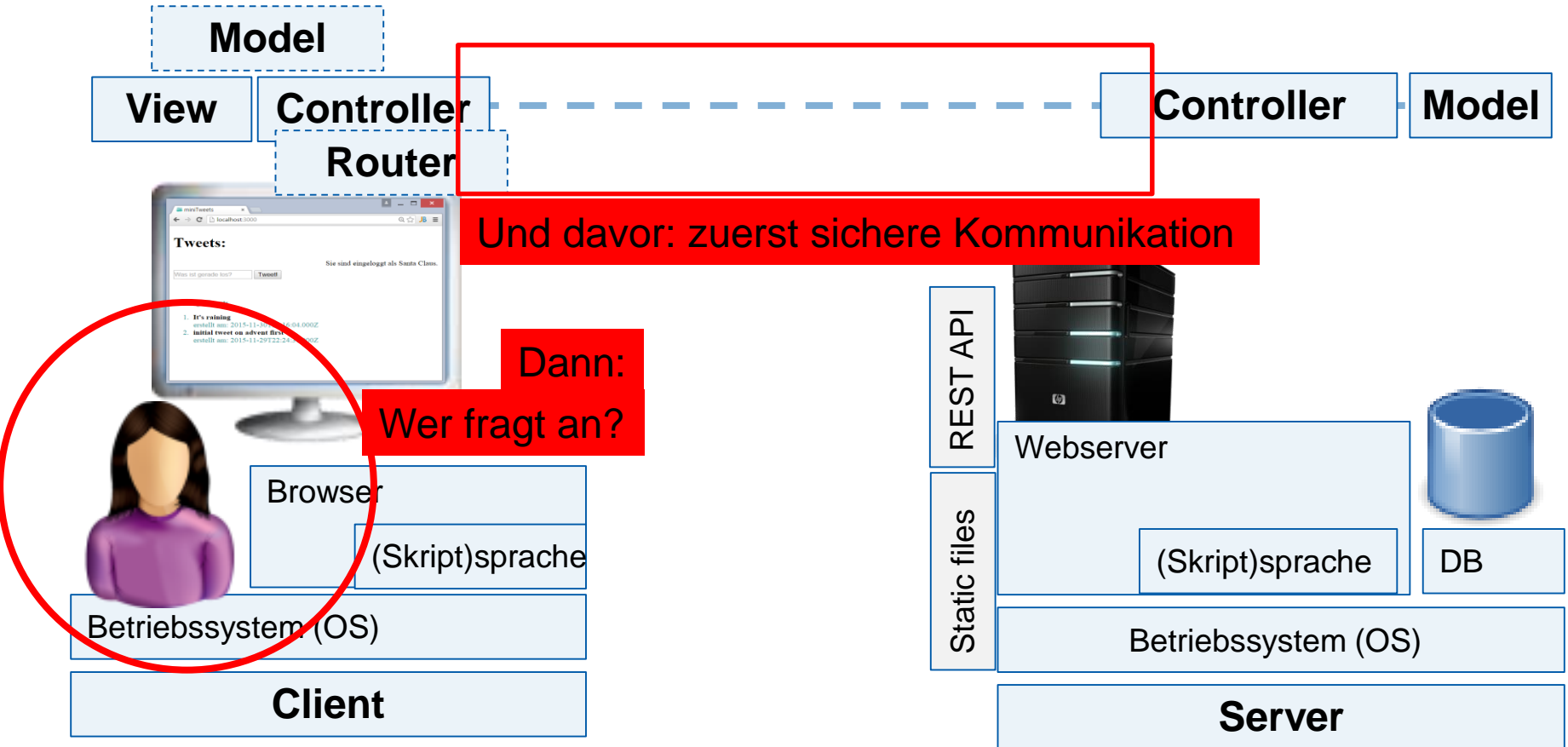
■ Fazit:

- **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 zu  BACKBONE.JS
- Template-Engine  UNDERSCORE.JS
- Authentifizierung
 - HTTPS
 - BASIC Auth mit node.js
 - Vergleich mit OAuth 2.0
 - Local Auth mit node.js und  passport
- Patterns: MVC, MVP, MVVM, MVCVM, MV*
- Zusammenfassung und Ausblick

Authentifizierung



Authentifizierung



■ Ziel:

- Wissen, welche/r Benutzer/in eine Anfrage sendet
- Damit können wir mithilfe von ACLs* prüfen, ob jemand das darf, was sie/er anfragt

■ Voraussetzung:

- Abhörsichere und fälschungssichere Kommunikation

Agenda

- Wiederholung zu  BACKBONE.JS
- Template-Engine UNDERSCORE.JS
- Authentifizierung
 - HTTPS
 - BASIC Auth mit node.js
 - Vergleich mit OAuth 2.0
 - Local Auth mit node.js und  passport
- Patterns: MVC, MVP, MVVM, MVCVM, MV*
- Zusammenfassung und Ausblick

Absicherung mittels HTTPS

- **Standard-Port: 443**
- **HTTP über mit SSL/TLS asymmetrisch verschlüsselten Kanal**
- **Server authentifiziert sich mit einem signierten Zertifikat**
- **Client könnte sich auch mit einem Zertifikat authentifizieren; ist leider nicht verbreitet.**

Fazit:

- **Bei erfolgreichem Aufbau einer HTTPS-Verbindung besteht**
 - **ein gesicherter Kanal**
 - **Client kann sicher sein, dass der Server der richtige ist (signiertes Zertifikat)**
 - **! Server weiß noch nicht(s) sicher über den Client, wer dieser ist**

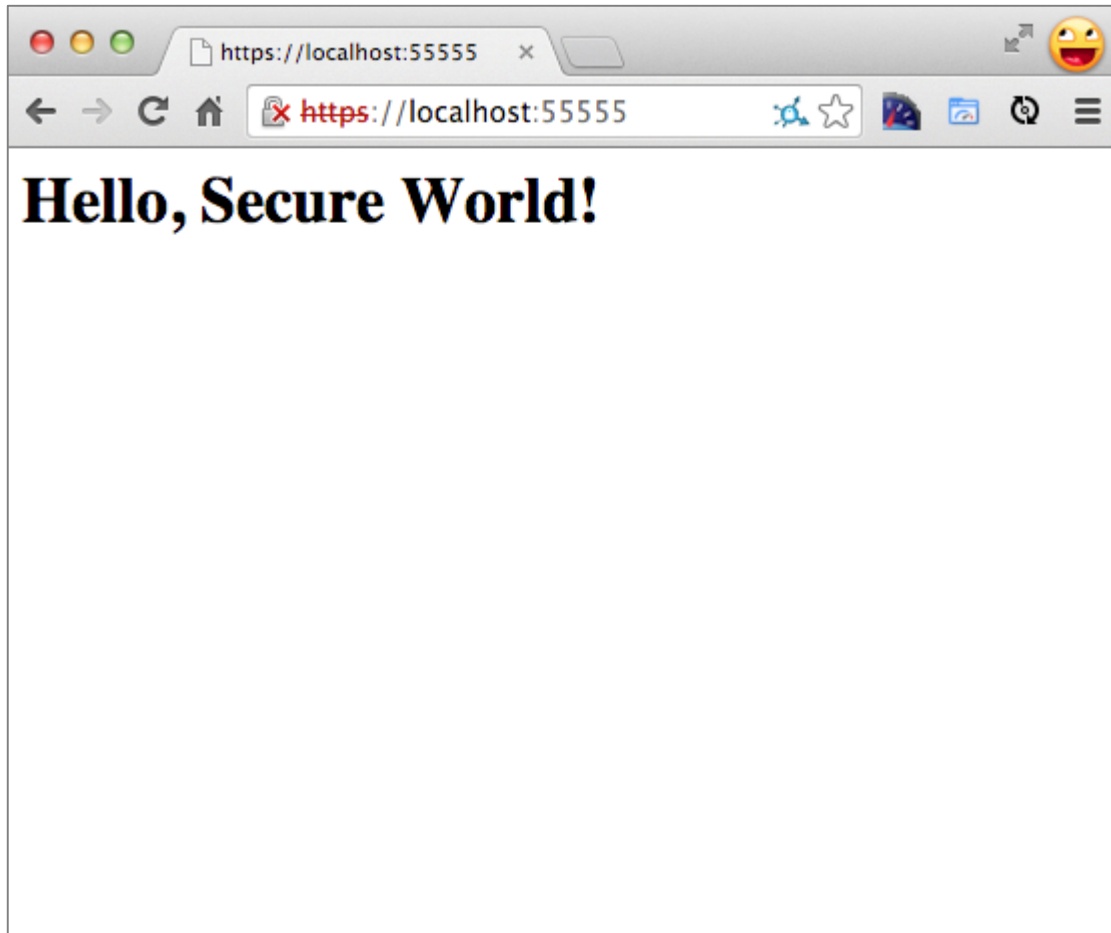
HTTPS mit node.js/express

```
var fs = require('fs');
var https = require('https');
var express = require('express');

var app = express();
app.get('/', function (req, res) {
  res.header('Content-type', 'text/html');
  return res.end('<h1>Hello, Secure World!</h1>');
});

https.createServer({
  key: fs.readFileSync('key.pem'),
  cert: fs.readFileSync('cert.pem')
}, app).listen(5555);
```

HTTPS mit node.js/express



HTTPS mit node.js/express

■ Woher ein gültiges Zertifikat nehmen?

```
var fs = require('fs');  
var https = require('https');  
var express = require('express');  
  
var app = express();  
app.get('/', function (req, res) {  
  res.header('Content-type', 'text/html');  
  return res.end('<h1>Hello, Secure World!</h1>');  
});  
  
https.createServer({  
  key: fs.readFileSync('key.pem'),  
  cert: fs.readFileSync('cert.pem')  
}, app).listen(5555);
```



HTTPS mit node.js/express

- **Woher ein gültiges Zertifikat nehmen?**
- **Idee 1: Ein eigenes Zertifikat erstellen, dann signieren lassen**
 - `openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365`
 - **Anschließende Zertifizierung von einer der großen Stellen...teuer.**
- **Idee 2: Kostenlose Zertifikate**
 - <https://startSSL.com> - kostenloses SSL-Zertifikat (365 Tage max)
 - <https://letsencrypt.org> - seit 03.12.2015 komplett kostenlose SSL-Zertifikate (90 Tage max)

Willkommen zu StartSSL™ PKI

StartSSL™ ist eine Marke der StartCom® Zertifizierungsstelle - ein führendes Unternehmen in der digitalen Zertifizierung. Wir bieten Ihnen alles vom **kostenlosen**, vertrauenswürdigen **SSL-Zertifikat** bis hin zu den modernsten Sicherheits- und PKI-Lösungen für Ihr Unternehmen und den privaten Gebrauch.

StartSSL™ Free (Class 1)	StartSSL™ Verified (Class 2)
128/256-bit Encryption, 1 Year Validity Legitimate SSL/TLS + S/MIME Certificates No Charge, Unlimited + 100 % Free	128/256-bit Encryption, 2 Years Validity Legitimate SSL/TLS + S/MIME + Object Code Wild Cards, Multiple Domain Names (UDC) Unlimited Certificates - US\$ 59.90
StartSSL™ Extended Validation 128/256-bit Encryption, 3 Years Validity Highest Level Third Party Assurance Green Extended Trust Indicator Multiple Domain Names (UDC) Special Offer - US\$ 199.90	High Protection StartSSL™ High Level Protection No MD5 Hashes, Weak Key Scans Minimum 2048-bit Strong RSA Keys
Hardware Aladdin® USB eToken Pro Aladdin® Smart Cards + Reader Original Driver Software + PKI Client Enterprise PKI Customized Solutions	Authentication StartSSL™ Authentication SSL Protected Open Identity Authentication Provider Click here to log into your StartSSL™ Account
Internationally Recognized WebTrust for CAs + WebTrust EV Certified Recognized by major browsers + software vendors	Easy Enrollment Sign-up and you will receive right away an S/MIME client-certificate and a digital StartSSL™ Open Identity without charge during the easy three-step enrollment!

Let's Encrypt LINUX FOUNDATION COLLABORATIVE PROJECTS

Blog · Technology · Contribute · Support · About ·

Let's Encrypt is a new Certificate Authority:
It's free, automated, and open.

Get Started

Lets-Encrypt with node.js/express

`npm install letsencrypt-express --save`

- Automatisches Registrieren der Domain
- Automatische Erneuerung des SSL-Zertifikates bei LetsEncrypt

```
var lex = require('letsencrypt-express').testing();
```

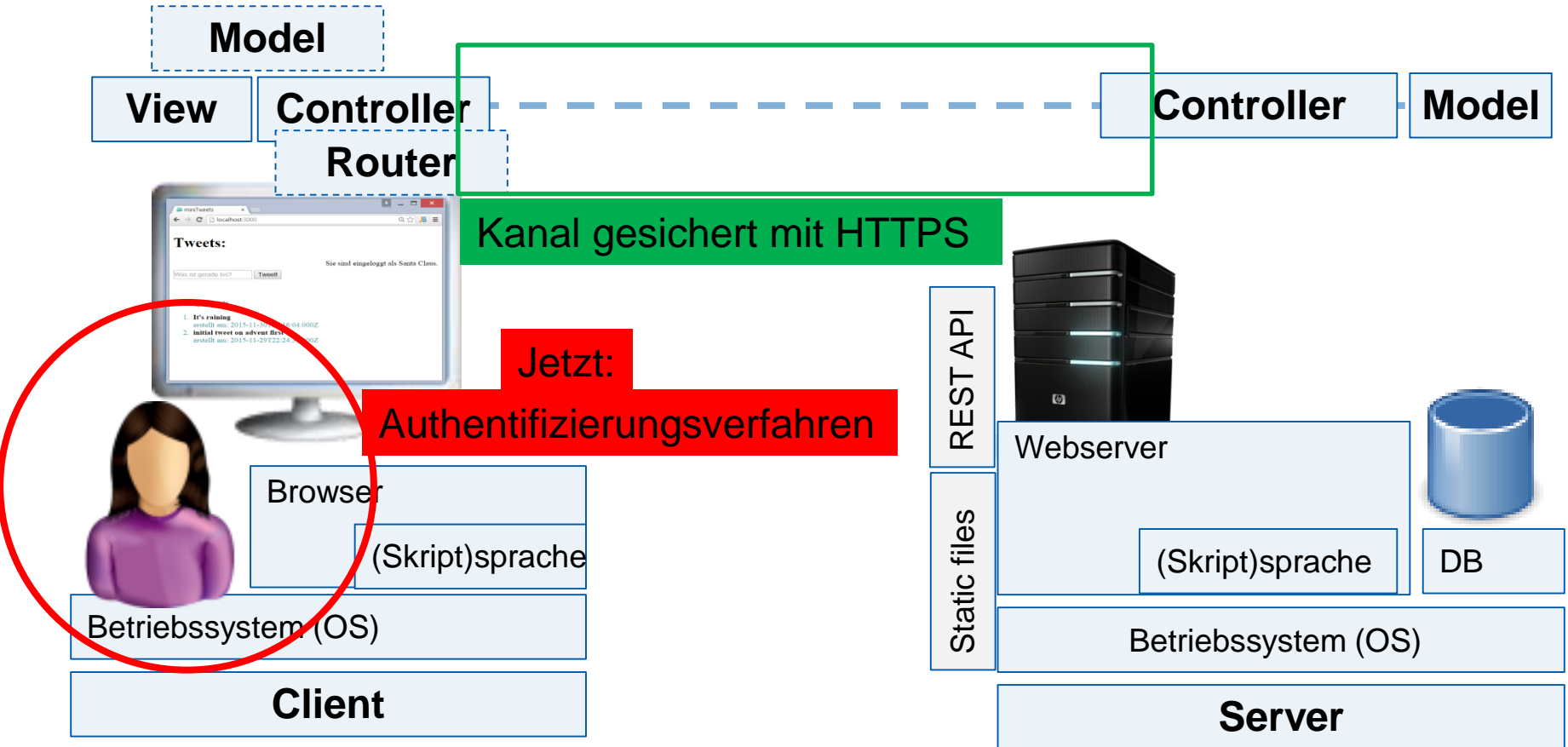
```
var DOMAIN = 'mydom.de'; var EMAIL = 'test@mydom.de';
var lex = lex.create({
  configDir: path.join(__dirname, '/letsencrypt')
, approveRegistration: function (hostname, approve) {
    if (hostname === DOMAIN) {
      approve(null, { domains: [DOMAIN], email: EMAIL, agreeTos: true });
    }
  }
});
lex.onRequest = app;
lex.listen([3000], [443], function () {
  console.log("Listening HTTP:3000 and HTTPS:443");
});
```

Beim Starten folgt die Logger-Ausgabe durch letsencrypt-express



```
[LEX] testing mode turned on
[LEX] default server: https://acme-staging.api.letsencrypt.org/directory
#####
Note: testing certs will be installed because .testing() was called.
      remove .testing() to get live certs.
[LEX] automatic registration handling turned on for testing.
[LEX] creating sniCallback {
  "configDir": "c:\\...\\letsencrypt",
  "debug": true,
  "privkeyPath": "c:\\...\\letsencrypt/live/:hostname/privkey.pem",
  "fullchainPath": "c:\\...\\letsencrypt/live/:hostname/fullchain.pem",
  "certPath": "c:\\...\\letsencrypt/live/:hostname/cert.pem",
  "chainPath": "c:\\...\\letsencrypt/live/:hostname/chain.pem",
  "server": "https://acme-staging.api.letsencrypt.org/directory",
  "letsencrypt": {
    "backend": {}
  }
}
```

Listening HTTP:3000 and HTTPS:443

Authentifizierung



Agenda

- **Wiederholung zu**  BACKBONE.JS
- **Template-Engine** UNDERSCORE.JS
- **Authentifizierung**
 - **HTTPS**
 - **BASIC Auth mit node.js**
 - **Vergleich mit OAuth 2.0**
 - **Local Auth mit node.js und**  **passport**
- **Patterns: MVC, MVP, MVVM, MVCVM, MV***
- **Zusammenfassung und Ausblick**

Authentifizierungsverfahren

HTTP BASIC Auth

- *Header-basiert*
- Server fordert für bestimmten Geltungsbereich (realm) Zugangsdaten an
HTTP Statuscode 401 Unauthorized
Header: WWW-Authenticate: Basic ...
- Client sendet Benutzername und Passwort als Base64-encodierten String
Header: Authorization: Basic ...

Ziel: direkte Authentifizierung beim Server mit Nutzerdaten

OAuth 2.0 Web Profile

- *Header-basiert und Registrierung erforderlich (für Client-ID des Dritten)*
- Drittserver leitet Nutzer/Browser an Server weiter mit eigener Client-ID (dort erfolgt Login)
- Server leitet Nutzer danach zu registriertem Callback mit ?code=
- Drittserver kann nun mit code per POST ein Access-Token beim Server anfordern
- Access-Token wird genutzt für Zugriffe des Drittserver auf Server

Ziel: indirekte Authentifizierung eines Nutzers durch Autorisierung eines Drittanbieters zum Zugriff auf Daten des Servers mit Nutzerdaten

HTTP Basic Auth mit node.js/express

■ `npm install --save basic-auth`

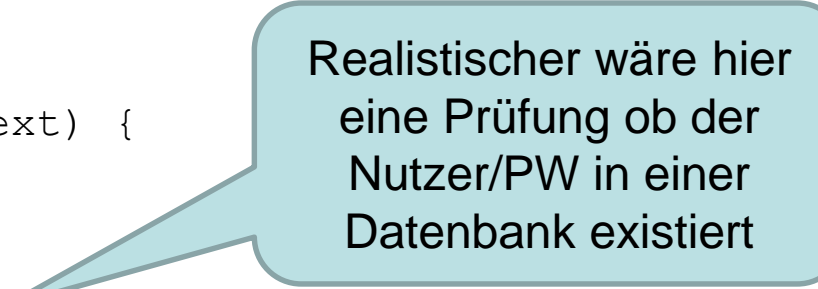
Beispiel: `basicauth.js` (1) Moduldatei schreiben

```
var auth = require('basic-auth');

var admins = {
  'me2': { password: 'geheim!' },
};

module.exports = function(req, res, next) {

  var user = auth(req);
  if (!user || !admins[user.name]
    || admins[user.name].password !== user.pass) {
    res.set('WWW-Authenticate', 'Basic realm="M2 Example page"');
    return res.status(401).end();
  }
  else {
    return next();
  }
};
```



Realistischer wäre hier eine Prüfung ob der Nutzer/PW in einer Datenbank existiert

HTTP Basic Auth mit node.js/express

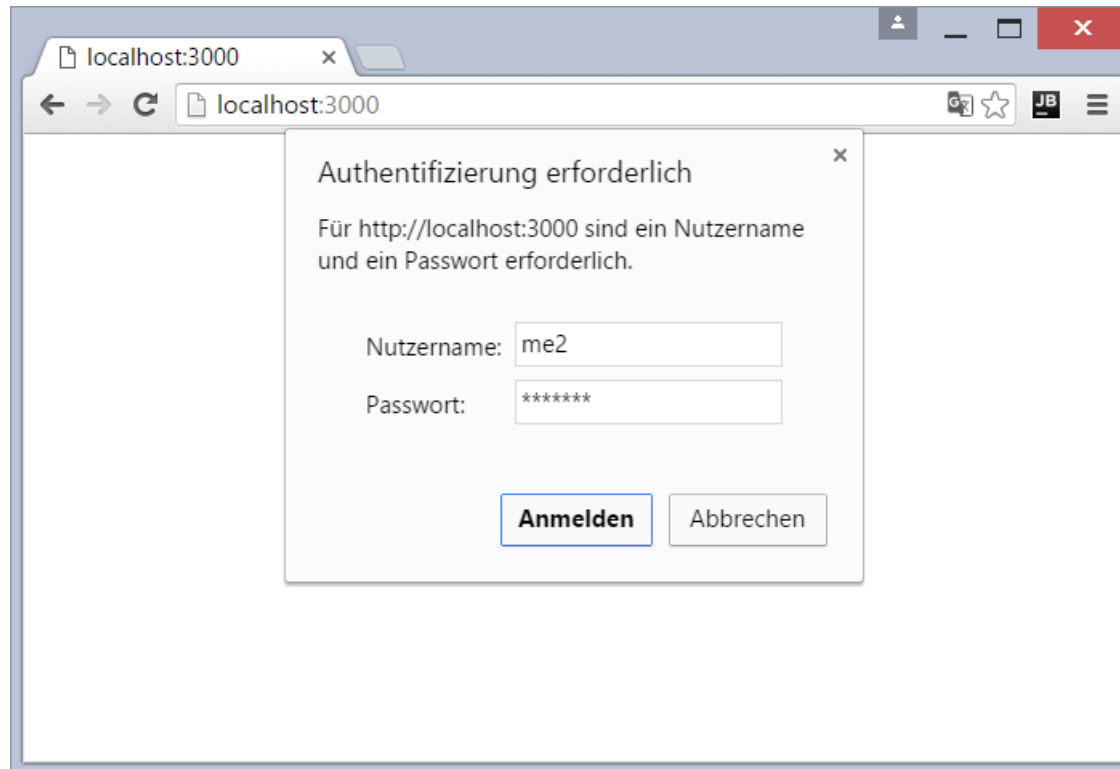
■ `npm install --save basic-auth`

Beispiel: `basicauth.js`

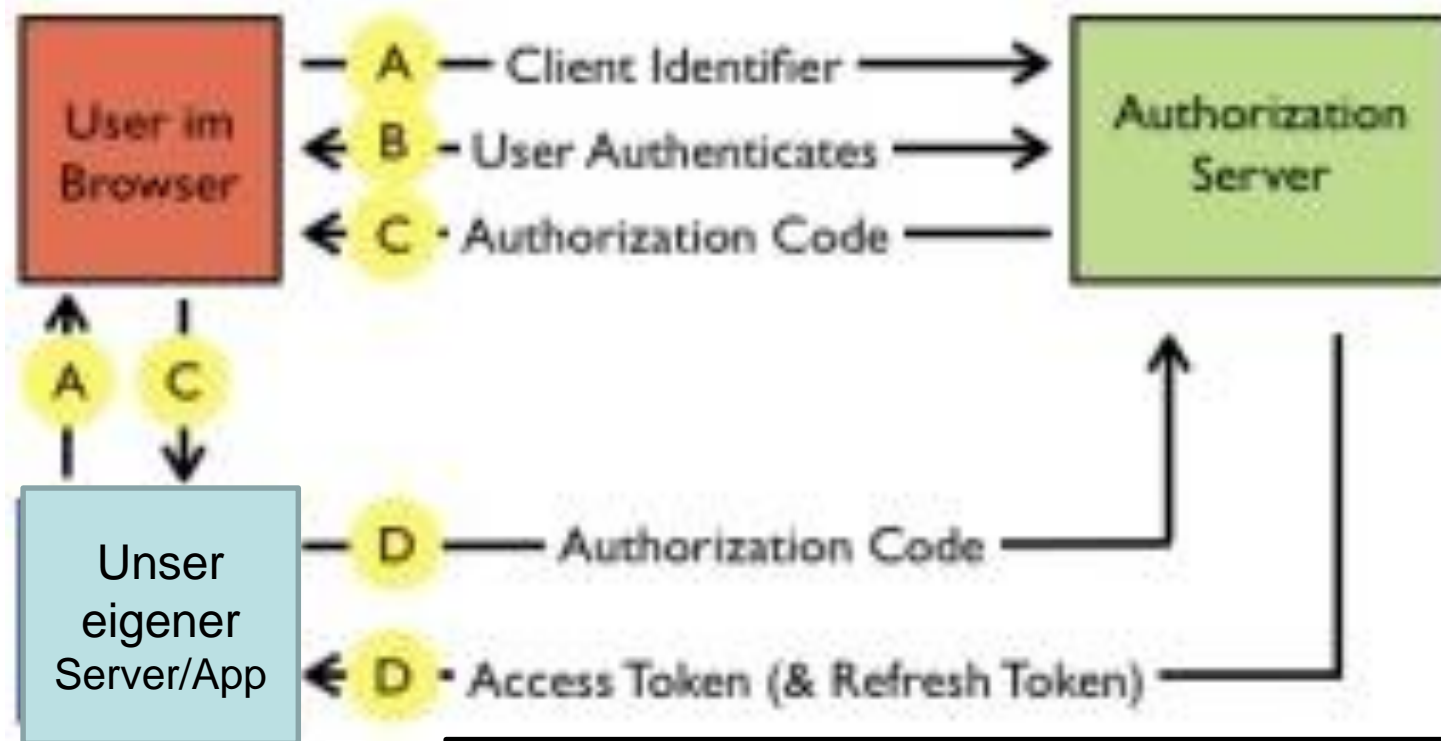
(2) Dann in `app.js` nutzen

```
var auth = require('./auth/basicauth');  
var express = require('express');  
  
var app = express();  
  
// ... Hier nicht authentifizierter Bereich  
  
app.use(auth);  
  
// Routes und Middleware nur mit Authentifizierung zugänglich
```


HTTP Basic Auth mit node.js/express



OAuth 2.0 Web Profile (Konzept)



(E) Eigener Server kann mit AccessToken nun auf Benutzerprofil beim AuthorizationServer zugreifen

■ OAuth 2.0 ermöglicht

- Authentifizierung von Clients (Nutzern) durch Dritte (bspw. Twitter)
- Zugriff über Accesstokens auf Benutzerprofile bei Dritten (bspw. Twitter)

OAuth 2.0 Web Profile in node.js/express nutzen

- **npm install --save passport**
 - **Middleware für** Basic Auth, Google+, OAuth 1.0, 2.0, facebook, ...
(über 300 unterstützte Systeme)
 - **siehe** <http://passportjs.org/docs/authenticate>



Passport



Simple, unobtrusive authentication for Node.js

Passport is authentication middleware for Node.js. Extremely flexible and modular, Passport can be unobtrusively dropped in to any Express-based web application. A comprehensive set of strategies support authentication using a username and password, Facebook, Twitter, and more.



Agenda



- **Wiederholung zu**  BACKBONE.JS
- **Template-Engine** UNDERSCORE.JS
- **Authentifizierung**
 - **HTTPS**
 - **BASIC Auth mit node.js**
 - **Vergleich mit OAuth 2.0**
 - **Local Auth mit node.js und**  **passport**
- **Patterns: MVC, MVP, MVVM, MVCVM, MV***
- **Zusammenfassung und Ausblick**

Passport Middleware - Funktionsweise

- 1. Konfigurieren der zu nutzenden Login-Strategien**
(bsw. BasicAuth oder Lokale Prüfung, Oauth, Facebook)
- 2. Passport-Middleware für zu schützende Routen einklinken**
- 3. (optional) Aktivieren der Session-Unterstützung**
 - **Cookie-basierter Auto-Login bei nächsten Requests**
 - **Abzuschalten für REST-APIs!**



Passport Middleware - Funktionsweise



1. Konfigurieren der zu nutzenden Login-Strategien

```
var passport = require('passport');  
var LocalStrategy = require('passport-local').Strategy;  
  
var UserModel = require('../models/user');  
  
passport.use(new LocalStrategy(  
  function(username, password, done) {  
    UserModel.findOne({username: username,  
                        password: password}, function(err, user) {  
      if (err) {  
        done(err, false, {message: 'Incorrect login!'});  
      }  
      else {  
        done(null, user);  
      }  
    });  
  })  
));  
  
));
```

eigenes Modul: passportlocal.js

Passport Middleware - Funktionsweise



1. Konfigurieren der zu nutzenden Login-Strategien

```
var passport = require('passport');
var LocalStrategy = require('passport-local').Strategy;

var UserModel = require('../models/user');

passport.use(new LocalStrategy(
  function(username, password, done) {
    UserModel.findOne({username: username, password: password}, function(err, user) {
      if (err) {
        done(err, false, {message: 'Invalid username or password'});
      } else {
        done(null, user);
      }
    });
  }
));
```

LocalStrategy Parameter done:

- Aufrufen mit Error Obj., wenn Fehler; Passport sendet 400 „Bad Request“
- Aufrufen mit null, user Obj. bei Erfolg; user wird in req.user zur Verfügung gestellt; Passport ruft next() auf.

eigenes Modul: passportlocal.js

Passport Middleware - Funktionsweise

2. Passport-Middleware für zu schützende Routen einklinken



```
router.use(passport.initialize());  
router.post('/login',  
    passport.authenticate('local', {session: false}),  
    function(req, res, next) {  
        res.json(req.user);  
    });  
router.use('/api/', passport.authenticate('local', {session: false}));
```

eigenes Modul: passportlocal.js

Passport Middleware - Funktionsweise

2. Passport-Middleware für zu schützende Routen einklinken



```
router.use(passport.initialize());  
router.post('/login',  
    passport.authenticate('local', {session: false}),  
    function(req, res, next) {  
        res.json(req.user);  
    });  
router.use('/api/', passport.authenticate('local', {session: false}));
```

eigenes Modul: passportlocal.js

Für /login Prefix sind 2 Handler eingehängt

- Nur wenn Passport durchgeht, liefert der zweite Handler den gefundenen user als JSON zurück.

Passport Middleware - Funktionsweise

2. Passport-Middleware für zu schützende Routen einklinken



```
router.use(passport.initialize());  
router.post('/login',  
    passport.authenticate('local', {session: false}),  
    function(req, res, next) {  
        res.json(req.user);  
    });  
router.use('/api/', passport.authenticate('local', {session: false}));
```

eigenes Modul: passportlocal.js



```
app.use(express.static(path.join(__dirname, 'public')));  
app.use(bodyParser.json());  
app.use(bodyParser.urlencoded({ extended: false }));  
app.use(morgan('tiny'));  
app.use(auth);  
app.use(requestchecks);  
// Routes *****  
app.use('/api/tweets', tweetsRoute);  
app.use('/api/users', usersRoute);  
// Final Middleware handlers / decorators *****  
app.use(limitoffsetWare);  
app.use(filterWare);
```

var auth = require('./auth/passportbasicauth');

app.js

Agenda

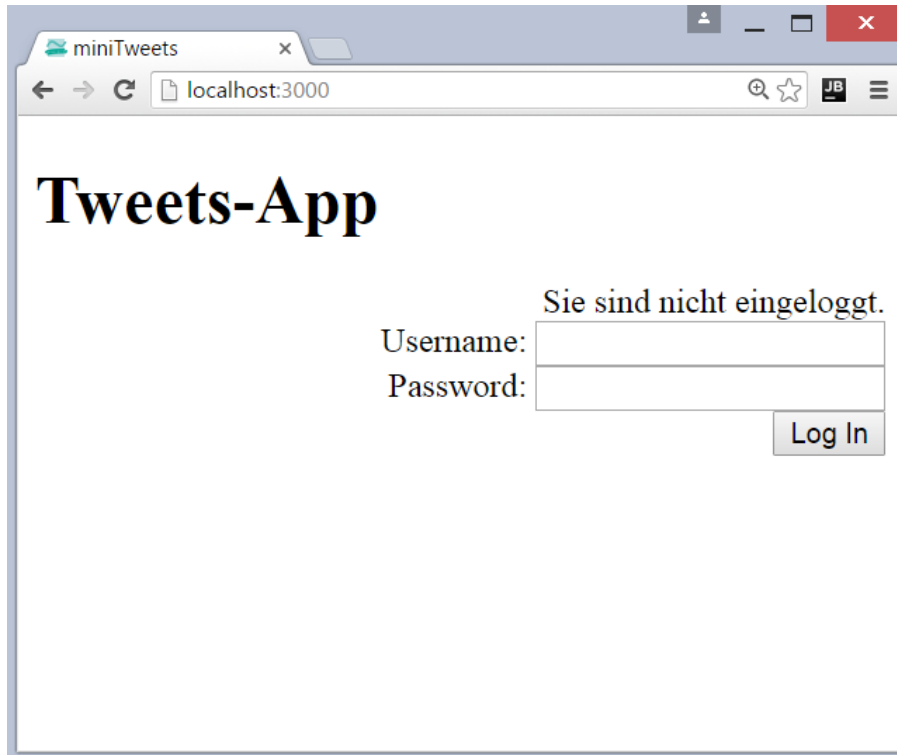


- **Wiederholung zu**  BACKBONE.JS
- **Template-Engine** UNDERSCORE.JS
- **Authentifizierung**
 - **HTTPS**
 - **BASIC Auth mit node.js**
 - **Vergleich mit OAuth 2.0**
 - **Local Auth mit node.js und**  **passport**
- **Patterns: MVC, MVP, MVVM, MVCVM, MV***
- **Zusammenfassung und Ausblick**

 BACKBONE.JS
Client-Seite

Passport Middleware - Client-Seitiger „Login“

1. Backbone-View für Login anpassen
2. Erst bei Login-Erfolg weitere Views anzeigen
3. Bei API-Zugriff dann IMMER Login-Daten mitsenden



Passport Middleware - Client-Seitiger „Login“

1. Backbone-View für Login anpassen

```
<script type="text/template" id="user-login-template">
  Sie sind
  <% if (!firstname && !lastname) {
    %>nicht eingeloggt.
    <div>
      <label>Username:</label>
      <input type="text" name="username"/>
    </div>
    <div>
      <label>Password:</label>
      <input type="password" name="password"/>
    </div>
    <div>
      <button class="submit" value="Log In">Log In</button>
    </div>
  <%
  } else {
    %>eingeloggt als <%= firstname %> <%= lastname %>.
  <% }
  %>
</script>
```

index.html

Passport Middleware - Client-Seitiger „Login“

1. Backbone-View für Login anpassen

```
var UserView = Backbone.View.extend({ ...  
  events: { 'click button': 'loginUser' },  
  render: function() {...},  
  loginUser: function() {  
    var username = this.$el.find("input[name='username']").val().trim()  
    var password = this.$el.find("input[name=password]").val().trim()  
    var that = this;  
    $.ajax({  
      url: '/login/',  
      type: 'post',  
      accepts: 'application/json',  
      data: { username: username, password: password },  
      error: function(xhr, status, errorText) {  
        alert("Login failed: " + status + errorText);  
      },  
      success: function(user) {  
        that.model = new UserModel.Model(user);  
        that.trigger("login", that.model);  
        that.render();  
      }  
    });  
  }  
});
```

../js/views/user.js

Input-Felder auslesen

jQuery AJAX Call zusammenstellen

URL zur Passport-Überprüfung

Wir wollen JSON zurück

Bei Erfolg erstellen wir mit den JSON-Daten eine UserModel-Instanz und lösen ein eigenes Event „login“ aus und aktualisieren die Login-Anzeige

Passport Middleware - Client-Seitiger „Login“

2. Erst bei Login-Erfolg weitere Views anzeigen

Router in ./js/appclient.js

```
...
main: function() {
  var userView = new UserView();
  userView.on('login', function(user) {
    this.user = user;
    ...
    var tweets = new Tweet.Collection();
    var tweetListView = new TweetListView({collection: tweets});
    tweets.fetch();
    var tweetCreateView = new TweetCreateView({collection: tweets,
                                              user: this.user});
  }, this);
}
```

Passport Middleware - Client-Seitiger „Login“

3. Bei API-Zugriff dann IMMER Login-Daten mitsenden



Passport Middleware - Client-Seitiger „Login“

3. Bei API-Zugriff dann IMMER Login-Daten mitsenden

Router in ./js/appclient.js

```
...
main: function() {
  var userView = new UserView();
  userView.on('login', function(user) {
    this.user = user;
    $.ajaxSetup({
      beforeSend: function(xhr, settings) {
        settings.url += '?username=' + user.attributes.username
          + '&password=' + user.attributes.password;
      }
    });
    var tweets = new Tweet.Collection();
    var tweetListView = new TweetListView({collection: tweets});
    tweets.fetch();
    var tweetCreateView = new TweetCreateView({collection: tweets,
      user: this.user});
  }, this);
}
```

Zusammenfassende Frage

- Warum braucht man neben BASIC Auth oder OAuth 2.0 unbedingt auch eine HTTPS-Verschlüsselung?



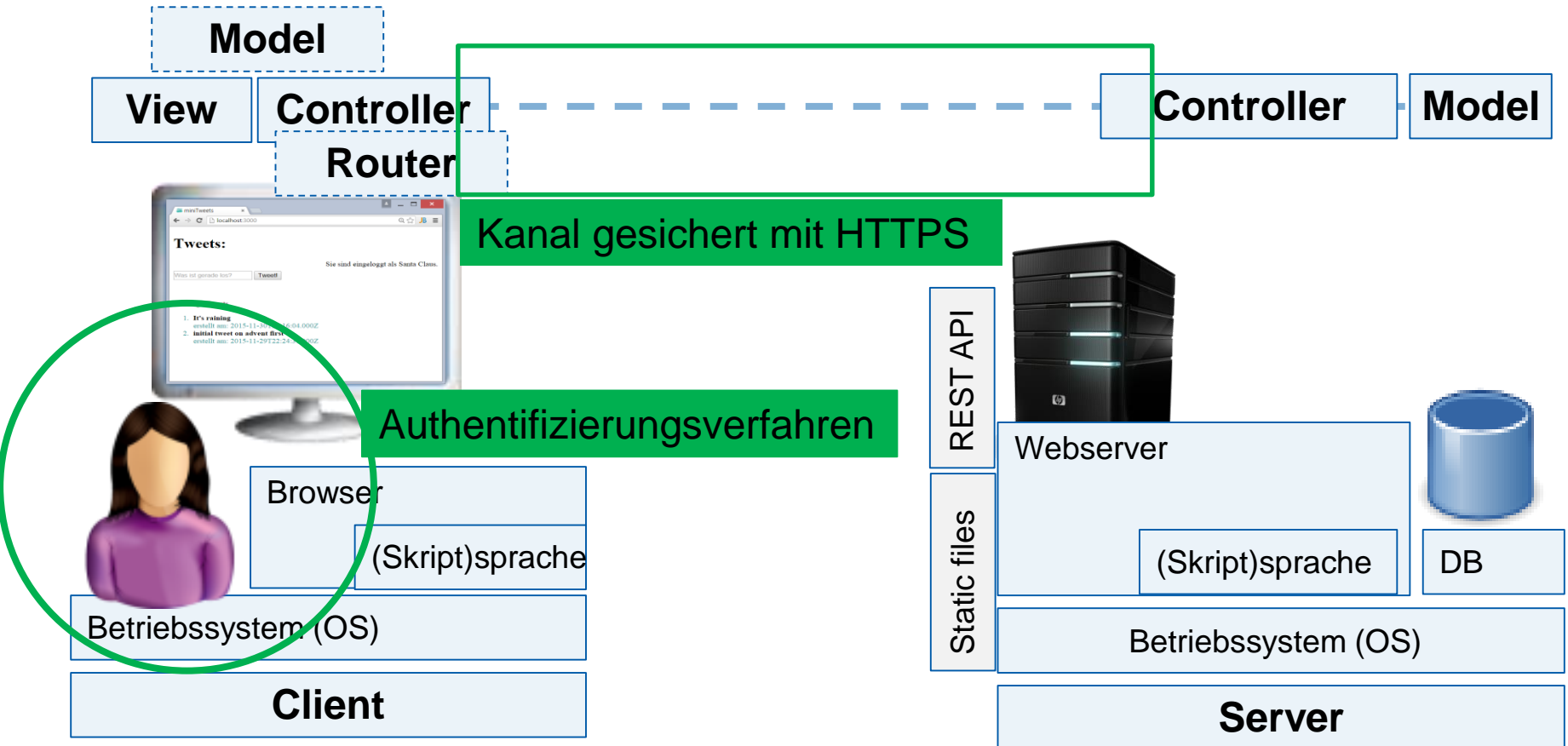
Vorschläge

1. Damit Logindaten verschlüsselt übertragen werden können
2. Damit kein anderer Kommunikation mitlesen kann
3. Damit keine sensiblen Daten gecached werden
4. Damit Client sicher sein kann, dass der Server der korrekte ist
5. Damit Server sicher ist, dass der richtige Client sich verbindet

Lösung: 1,2,3,4 sind richtig.



- HTTPS verschlüsselt die Kommunikation mittels SSL/TLS und
- sichert mittels Zertifikat die Echtheit des Absenders (Servers) zu.

Authentifizierung



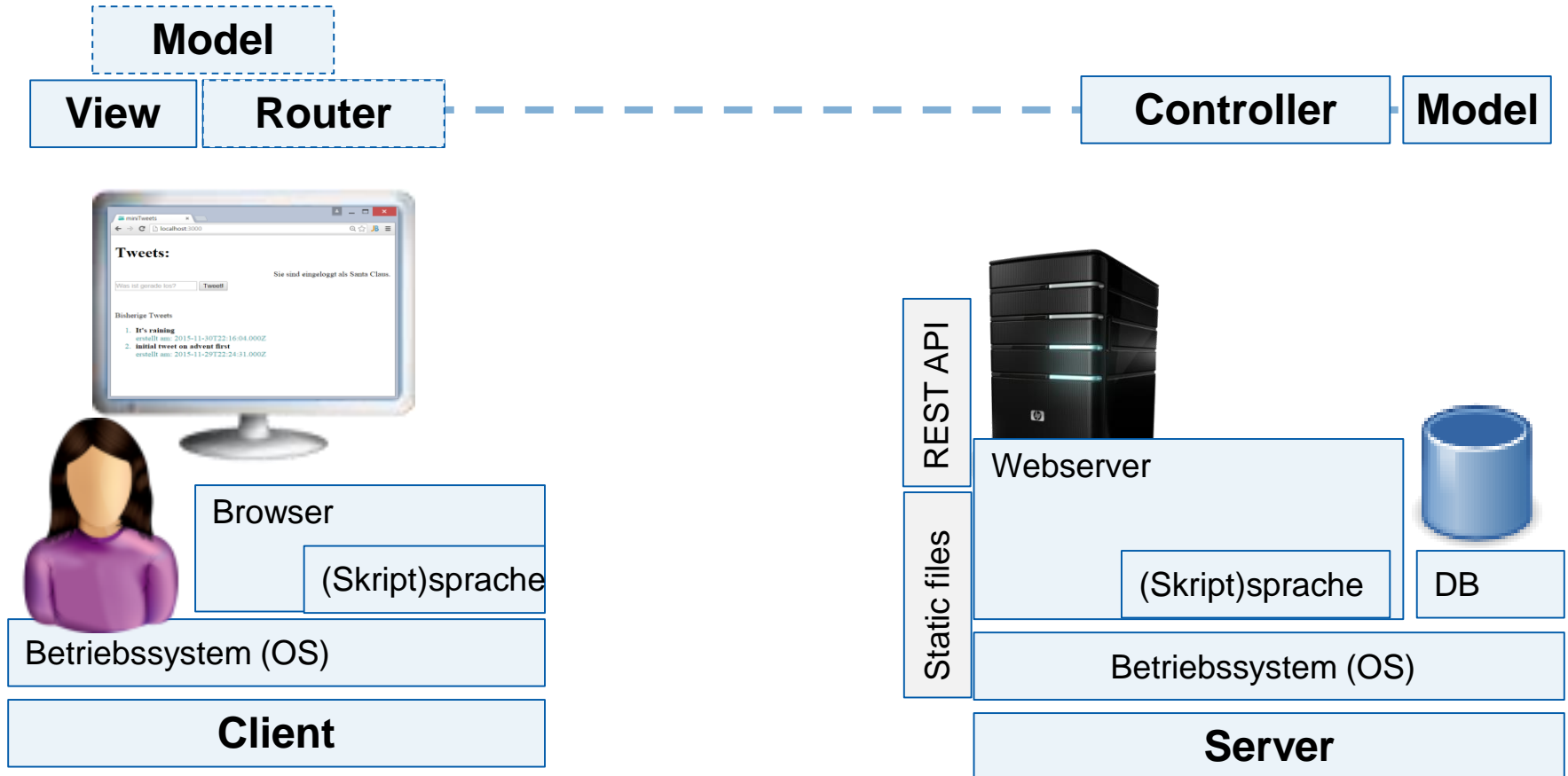
Agenda



- **Wiederholung zu**  BACKBONE.JS
 - **Template-Engine** UNDERSCORE.JS
 - **Authentifizierung**
 - **HTTPS**
 - **BASIC Auth mit node.js**
 - **Vergleich mit OAuth 2.0**
 - **Local Auth mit node.js und**  **passport**
- **Patterns: MVC, MVP, MVVM, MVCVM, MV***
- **Zusammenfassung und Ausblick**

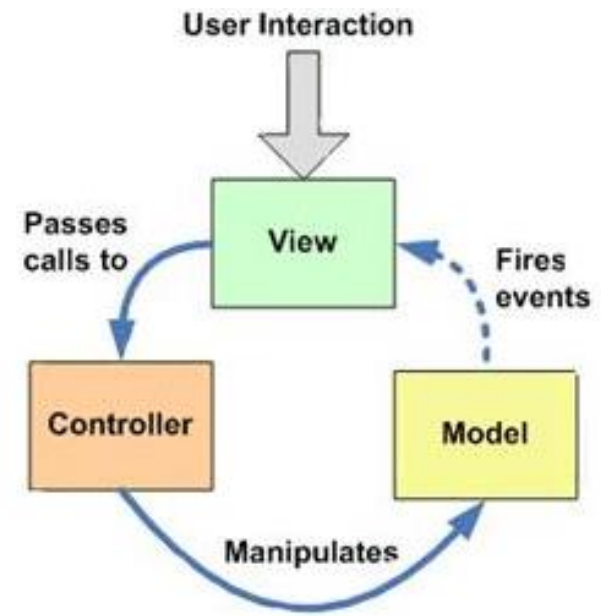
Client-Server Architekturen

■ Rich-Client-Prinzip (MV*)



Der Klassiker: Model View Controller (MVC)

- **Model**
 - Daten einer Anwendung
- **View**
 - Grafische Darstellung
- **Controller**
 - Definition der (erlaubten) Interaktion zwischen Nutzer und Applikation



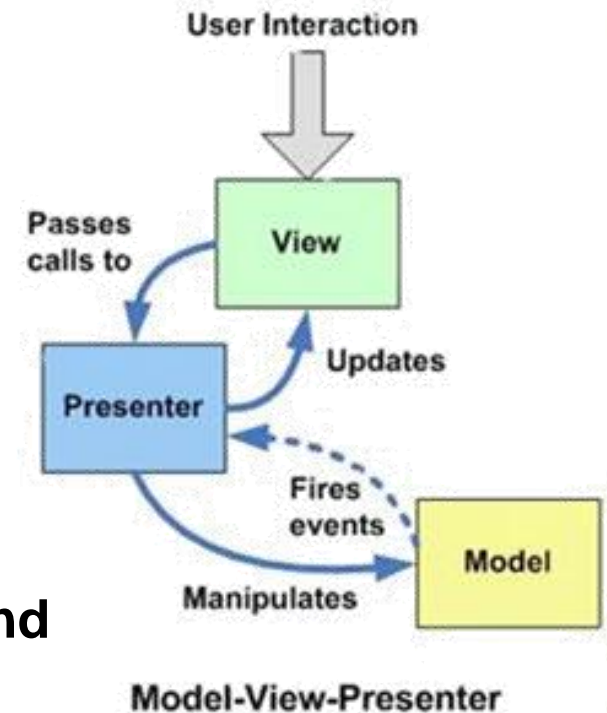
Model-View-Controller

Fazit:

- Wenn Modeldaten dem View ähnlich sind
- Controller fehlt i.d.R. Möglichkeit View-Updates zu koordinieren.

Model View Presenter (MVP)

- **Model**
 - Daten einer Anwendung
- **View**
 - Grafische Darstellung
- **Presenter**
 - Verbindung zwischen Models und View
 - 1:1 von Presenter:View
- **Presenter koordiniert explizit zwischen Model-Zustand und View-Zustand**

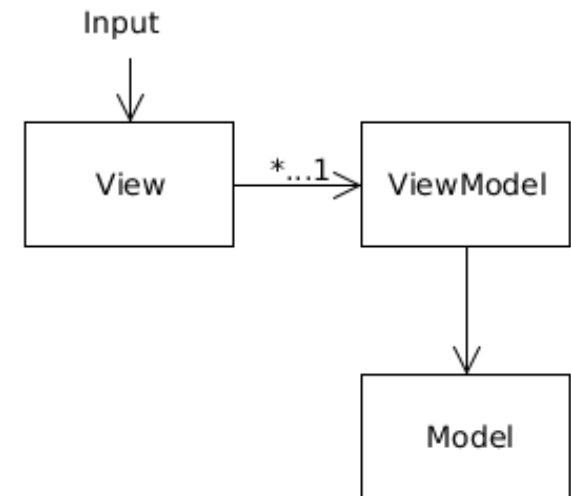


Fazit:

- **Sinnvoll für Model-Daten die stark transformiert werden müssen vom/zum View**

Model View ViewModel (MVVM)

- **Model**
 - Daten einer Anwendung
- **View**
 - Grafische Darstellung
- **ViewModel**
 - Verbindung zwischen Model und Views
- **ViewModel wird von vielen Views benutzt.**

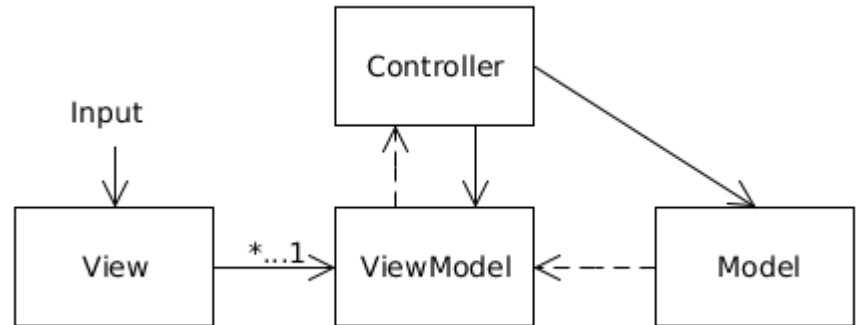


Fazit:

- **Praktisch wenn viele Sichten auf die Daten existieren (Liste, Einzelansicht, Edit-Ansicht usw.)**

Model View Controller ViewModel (MVCVM)

- **Model**
 - Daten einer Anwendung
- **View**
 - Grafische Darstellung
- **ViewModel**
 - Verbindung zwischen Model und Views
- **Controller**
 - Auslagern von Verbindungsmgmt., Setup der Komponenten, REST-Anbindung



Fazit:

- **Controller als „GlueCode“** bietet sich an, wenn ViewModel sonst zu komplex wird
u/o auch für viele ViewModels der Controllercode ähnlich ist.

Zusammenfassung: Model View Whatever (MVW oder auch MV*)

- **MV***
 - Vermischung der Patterns
- **MVW**
 - Model-View-Whatever
 - „whatever works for you“



Fazit:

- Raum für eigene Interpretation
- Weniger starr an einem Pattern festhalten, dafür „sinnvolle“ Lösung etablieren
 - Bspw. mit oder ohne Controller
 - Mehr oder weniger Bindung vom View zum Model

Zusammenfassende Frage

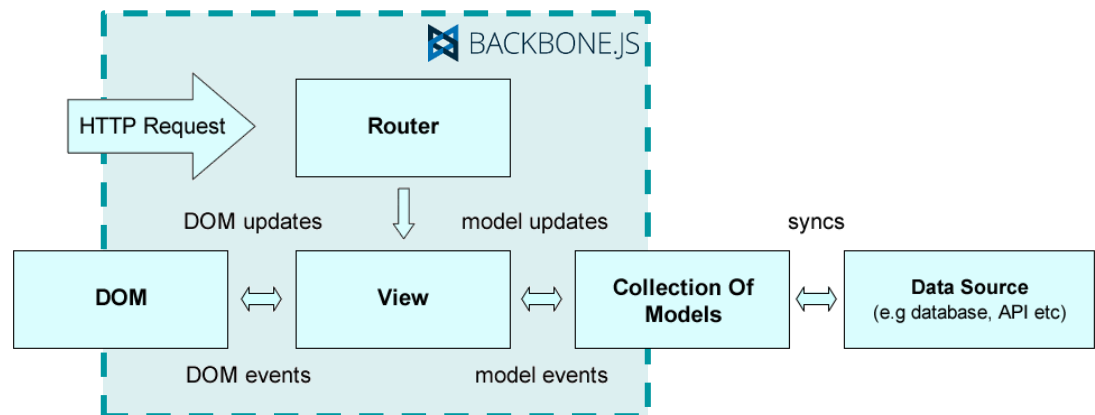
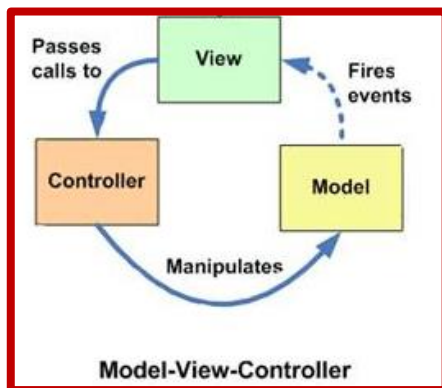
Warum ist Backbone kein MVC?



- Es gibt keinen echten Controller
- Die **Backbone.View** Klasse ähnelt einem **Presenter** aus MVP (strikte Vermittlung zwischen DOM/HTML und Model)
- **Backbone.Router** entspricht einem **Controller** in MVCVM



Was passt ist:

- **Backbone.Model** deckt sich mit dem **Model** aus MVC und MVP
- Die **HTML-Templates** können den **View** aus MVC und MVP zugeordnet werden



Agenda



- Wiederholung zu  BACKBONE.JS
- Template-Engine UNDERSCORE.JS
- Authentifizierung
 - HTTPS
 - BASIC Auth mit node.js
 - Vergleich mit OAuth 2.0
 - Local Auth mit node.js und  passport
- Patterns: MVC, MVP, MVVM, MVCVM, MV*
- Zusammenfassung und Ausblick

Zusammenfassung

- **Underscore-Templates**

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

- **HTTPS mit LetsEncrypt**

```
lex.onRequest = app;
lex.listen([3000], [443], function () {
  console.log("Listening HTTP:3000 and
  HTTPS:443");
});
```

- **Benutzer-Auth**

- **HTTP Basic Auth**

Authentifizierung erforderlich

Für http://localhost:3000 sind ein Nutzname und ein Passwort erforderlich.

Nutzname:

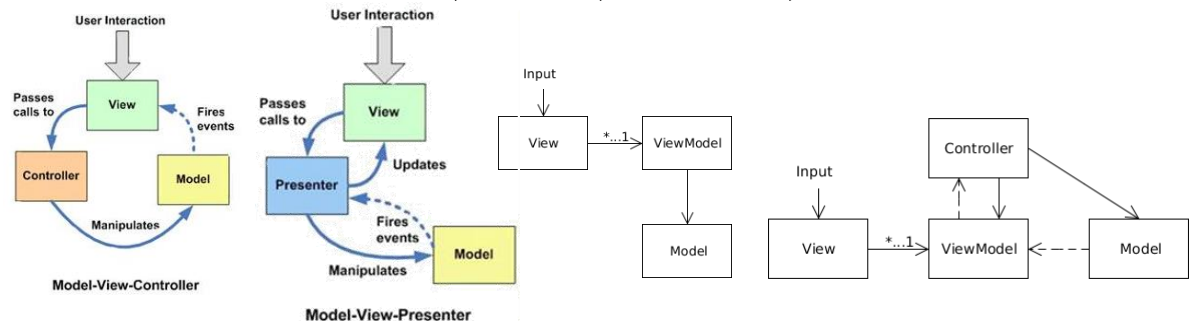
Passwort:

- **Benutzer-Auth**

- **Passport (Bsp: LocalStrategy)**



- **Patterns MVC, MVP, MVVM, MVCVM**



Nächster Unterricht

Nächste Woche:

- **Entwickler und CEO Claudio Bredfeldt** zeigt Beispiele und Lösungen von **mycs**

- **Diskussion/Fragerunde**



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

Darauf:

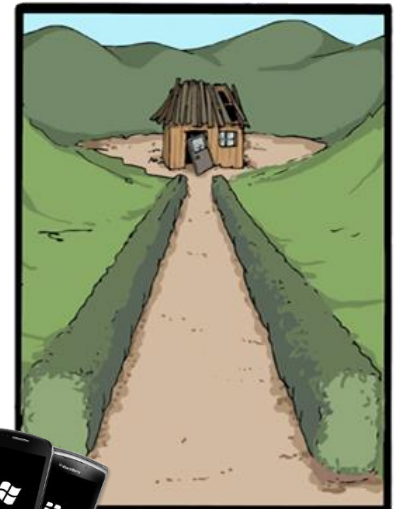
- **Mobile und Cross-Plattform Development mit Web-Technologien**

The dilemma of mobile apps development

Develop a native app for each device and maintain several projects



Use a unique framework (Phonegap, Adobe Air, Appcelerator) and maintain only one project



CommitStrip.com