Musterlösung

Klausur
Multimedia Engineering 2
WiSe 2016/17 – PZR 1

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Fachbereich VI
Fachgebiet Web Engineering
Prof. Dr.-Ing. Johannes Konert
24.01.2017

Hinweise

1. Die Klausur besteht aus 13 Seiten und 8 Aufgaben. Prüfen Sie die Vollständigkeit zu Beginn!
2. Die Heftung der Blätter bitte nicht lösen! Sollten sich Blätter lösen, sind diese unbedingt einzeln mit Namen und Matrikelnummer zu versehen.
3. Die erreichbare Punktzahl für jede Aufgabe ist vermerkt.
   Insgesamt können 100 Punkte erreicht werden.
4. Nur gut lesbare, eindeutige und nachvollziehbare Antworten werden gewertet.
5. Nur dokumentenechte Stifte in dunkler Farbe (kein grün, kein rot) verwenden.
6. Es sind keine Hilfsmittel erlaubt.
7. Jeder Täuschungsversuch führt sofort zum Abbruch der Klausur und zur Bewertung mit Note 5,0.

Viel Erfolg!

Angaben zur Person:

Name: _____

Vorname: _____

Matrikel-Nr.: _____

Handelt es sich um Ihren letzten Versuch diese Prüfung zu bestehen?
(bitte ankreuzen)

☐ ja ☐ nein

Bewertung (wird vom Dozenten ausgefüllt)

Aufgabe	1	2	3	4	5	6	7	8	Gesamt
Max. Punktzahl	13	9	17	10	18	14	14	5	100
Erreichte Punktzahl									

Note: _____

Aufgabe 1: HTML5 und Semantik (13 Punkte)

Um mittels JavaScript auf ein Template zuzugreifen, schlägt Ihnen eine Kollegin die folgenden zwei Varianten (1.) Tag `<script>` und 2.) Tag `<template>` vor. Hier ist identischer Template-Code für die Template-Engine `underscore` eingebettet (zur Veranschaulichung).

1.)
```
<script type="text/template" id="user-login-template">
    Sie sind eingeloggt als <%= firstname %> <%= lastname %>.
</script>
```
2.)
```
<template id="user-login-template">
    Sie sind eingeloggt als <%= firstname %> <%= lastname %>.
</template>
```

a.) Sind diese Tags `<script>` und `<template>` semantische Tags? Begründen Sie kurz Ihre Antwort. (3 Punkte)

Antwort:

<script> ist semantischer Tag, denn er bedeutet, dass hier ausführbarer Programmcode hinterlegt ist, der interpretiert wird, aber nicht durch den Browser zur Anzeige verwendet werden soll.

<template> ist semantischer Tag, denn er bedeutet, dass hier eine Vorlage drin ist, die der Browser nicht im DOM anzeigen soll, sondern dass diese später (durch JS) verwendet wird.

Ebenfalls „ok“: <script> ist kein semantischer Tag, weil er nur bedeutet, dass ein Skript eingebettet ist, aber nicht klar ist, was für eine Art. Erst durch das Attribute type wird das ersichtlich.

Bewertung: Je 1.5P für semantisch und Bedeutung. Noch 1P, wenn die Definition von Semantik korrekt ist.

b.) Sind beide Varianten gleich gut geeignet, um den Template-Code ins HTML-Dokument einzubetten, ohne dass der Browser den Inhalt anzeigt? Begründen Sie Ihre Antwort. (4 Punkte)

Gehen Sie bei Ihrer Antwort davon aus, dass kein CSS die Darstellung manipuliert.

Antwort:

Ja, grundsätzlich führen beide Varianten dazu, dass der Browser den Inhalt nicht anzeigt. <template> wäre semantisch passender, jedoch gibt es ältere Browser, die diesen HTML5-Tag nicht kennen und dann den Inhalt anzeigen würden. Das passiert mit <script> nicht, welches jedoch eigentlich für ausführbaren Code gedacht ist.

Bewertung: 2P für „beide gleich gut geeignet“ . 2P für“<template> in älteren Browsern problematisch“ oder „<script> auch in älteren Browsern unsichtbar“. 2P für <template> ist spezifischer und daher <script> vorzuziehen.

c.) Welche drei wesentlichen
Template mit einer Template-En
Gehen Sie bei ihrer Antwort d
verwendet werden soll. F
var user =
liegt bereit
mögli

c.) Welche drei wesentlichen Schritte müssen Sie stets programmieren, wenn Sie ein Template mit einer Template-Engine verwenden wollen? (6 Punkte)

Gehen Sie bei ihrer Antwort davon aus, das das obige Template in Ihrem JavaScript Code verwendet werden soll. Ein passendes User-Objekt der Form

```
var user = { firstname: 'Grace', lastname: 'Hopper' };
```

liegt bereits vor. Sofern Sie in Ihrer Antwort Pseudo-Code verwenden, halten Sie sich so nah wie möglich an korrekte JavaScript-Schreibweise.

Antwort:

1. Kompilieren: der Template-String wir der Template Engine übergeben. Diese liefert eine Funktion (bspw. var tmpl = _.template($(‚user-login-template'));
2. Rendering: Die Template-Funktion wird mit Daten aufgerufen, um das Template zu rendern. Rückgabe ist ein String. (bspw. var result = tmpl(user);
3. Nutzung: Der Ergebnisstring muss irgendwo eingehängt werden / verwendet werden (bspw. $(‚body').append(result);

Bewertung: je 2P für 1,2,3.

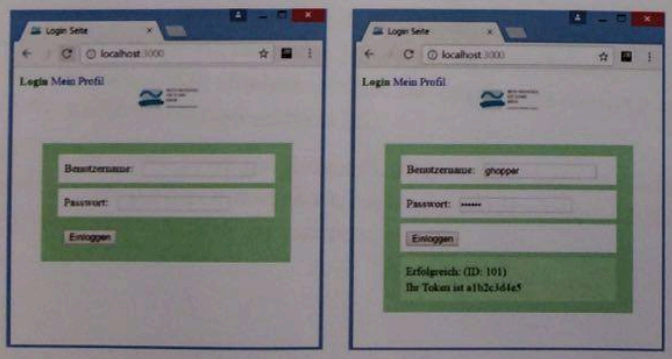
Aufgabe 2: Single Page Applications (9 Punkte)

Folgendes Beispiel einer Web-Anwendung für einen Login ist gegeben. Die angegebene HTML-Seite wurde ausgeliefert bei einem `HTTP GET` auf die URL `http://localhost:3000` . Der Inhalt der `main.js` ist ebenfalls aufgelistet. Der Übersichtlichkeit halber ist die `main.css` nicht abgedruckt. Screenshots geben Ihnen einen Eindruck des Erscheinungsbildes.

HTML (auch für Aufgabe 3):

```
10: <!doctype html>
11: <html lang="de">
12:     <head>
13:         <meta charset="utf-8">
14:         <title>Login Seite</title>
15:         <meta name="description" content="Login für MME2">
16:         <meta name="viewport" content="width=device-width, initial-scale=1">
17:         <link rel="stylesheet" href="css/main.css">
18:         <script src="https://code.jquery.com/jquery-1.12.0.min.js"></script>
19:         <script src="js/main.js"></script>
20:     </head>
21:     <body>
22:         <nav>
23:             <a href="/" class="selected">Login</a>
24:             <a href="/profile">Mein Profil</a>
25:         </nav>
26:         <img src="img/beuth.png">
27:         <form method="POST" action="/login">
28:             <div>
29:                 <label>Benutzername:</label><input type="text"
                                                       name="username"/>
30:             </div>
31:             <div>
32:                 <label>Passwort:</label><input type="password"
                                                   name="password"/>
33:             </div>
34:             <div>
35:                 <button type="submit">Einloggen</button>
36:             </div>
37:         </form>
38:     </body>
39: </html>
```

Screenshot: (1. Nach dem Laden der Webseite, 2. Nach erfolgreichem Login mittels Button-Click)



Datei: js/main.js (auch für Aufgabe 3)

```
50:
51:
52:
53:
54:
55:
56:
57:
58:
59:
60:
61:
```

Datei: js/main.js (auch für Aufgabe 3)

```
50: (function(global, $){
51:     global.addEventListener('DOMContentLoaded', function(){
52:         $('button').on('click', function(event){
53:             event.preventDefault();
54:             var form = $("form");
55:             var username = form.find("input[name='username']").val().trim();
56:             var password = form.find("input[name='password']").val().trim();
57:             $.ajax({
58:                 url: '/login/',
59:                 type: 'POST',
60:                 headers: { Accept: 'application/json',
                               Content-Type: 'application/x-www-form-urlencoded' },
61:                 data: { username: username, password: password },
62:                 error: function(xhr, error, errorText) {
63:                   form.append('<aside class="error">Login fehlgeschlagen: ' +
                                                          errorText + '</aside>');
64:                 },
65:                 success: function(user) {
66:                   form.append('<aside class="success">Erfolgreich: (ID: ' +
                                   user.id+')<br> Ihr Token ist ' + user.token + '</aside>');
67:                 }
68:             });
69:         });
70:     });
71: })(window, $);
```

a.) Was versteht man allgemein unter einer Single Page Application (SPA)? (3 Punkte)

Geben Sie eine passende Definition an.

Antwort:

Eine Single Page Application ist eine nach dem Rich-Client-Prinzip aufgebaute Webanwendung. Diese aktualisiert Ihre Views anhand von Datenaustausch mit dem Server, ohne dass der komplette Zustand der Anwendung im Client verloren geht.

(Dazu wird Client-seitig ein Zustand mittels Controller-Code erhalten, welcher die Verarbeitung von Benutzereingaben, Kommunikation mit dem Server und die Aktualisierung der Views steuert)

Bewertung: 3P Punkte für Zustand Client-Seitig erhalten / oder / 3P Datenaustausch via Controller-Code/AJAX

b.) <u>Begründen Sie</u> passend zu Ihrer Definition, ob es sich bei vorliegender Seite um eine Single Page Application (SPA) handelt. (6 Punkte)

Verwenden Sie in Ihrer Begründung für Belege die Code-Zeilennummern aus HTML und main.js

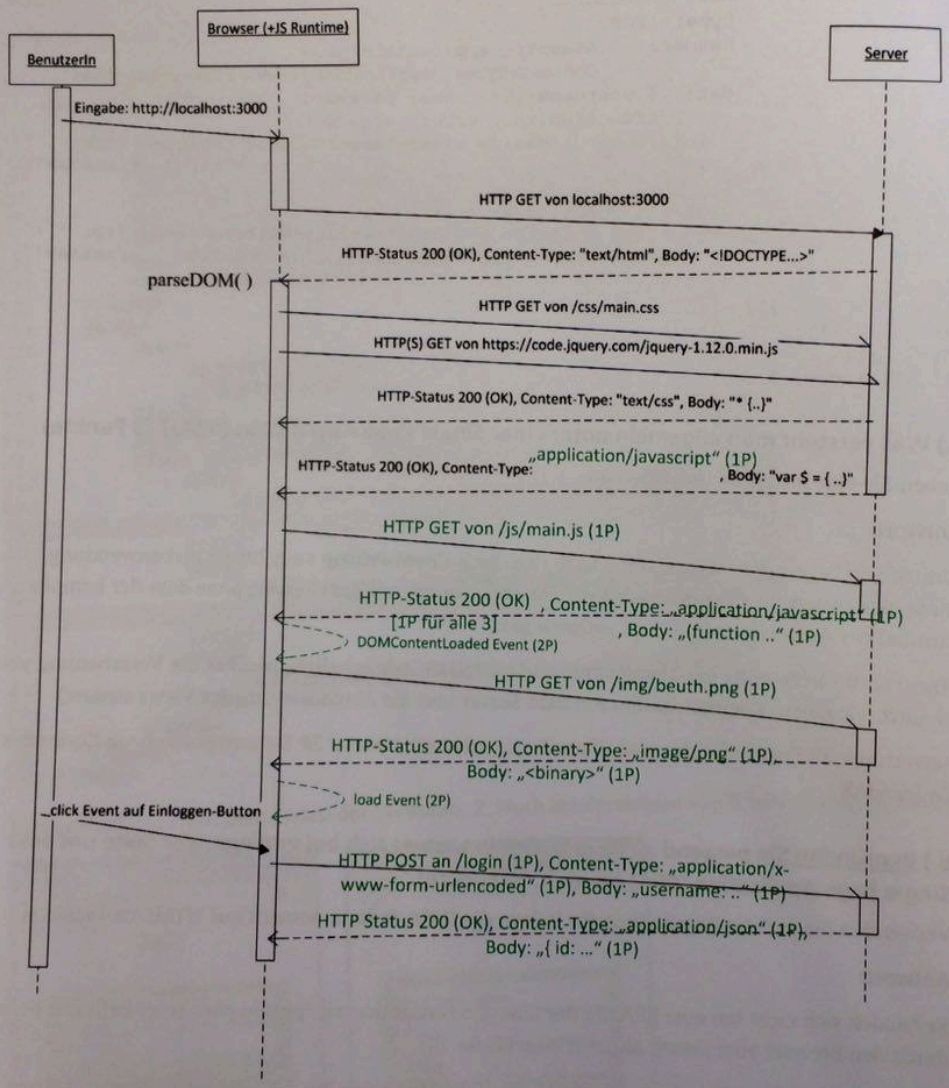
Antwort:

Es handelt sich nicht um eine SPA, da der Link der Navigation zu /profile eine komplett neue Seite durch den Browser vom Server abrufen lässt (Zeile 24).

Bewertung: 6P für die Begründung mittels /profile. Ansonsten 3P, falls das Vorhandensein einer SPA mit den AJAX-Aufrufen in der main.js beim Button-Klick begründet wurde (Zeilen 52 und 53).

3. Aufgabe: Lade-Reihenfolge im Sequenzdiagramm (17 Punkte)

Folgendes Sequenzdiagramm zeigt die Ablaufsequenz bei der Kommunikation zwischen Browser und Webserver beim erstmaligen Laden der minimalistischen HTML-Login-Seite **aus Aufgabe 2** (siehe Quellcode dort).
Aus Gründen der Anschaulichkeit wurde die Interaktion des Benutzers mit dem Browser ebenfalls skizziert. Das Sequenzdiagramm ist noch unvollständig.



a.) Ergänzen Sie alle weiteren fehlenden Daten der HTTP Abfragen und Antworten. Ziehen Sie dafür die Datei-Quellcodes aus Aufgabe 2 zu Rate. Für die Anfragen und Antworten nach dem angegebenen Klick des Benutzers auf den Einloggen-Button gehen Sie von einem erfolgreichen Login aus. (13 Punkte)

Geben Sie jeweils mit an (sofern vorhanden):
1. die HTTP Methode, 2. den Content-Type, 3. den HTTP-Statuscode, 4. Die HTTP Body Daten (nur erste 5-10 Zeichen des Body beispielhaft)

Antwort direkt im Sequenzdiagramm oben.

Bewertung: Punkte im Diagramm jeweils in Klammern (xP)

b.) Während der Browser das HTML-Dokument parst und das `Document Object Model` (`DOM`) aufbaut, werden zwei wesentliche JavaScript-Events ausgelöst. Das `Window load` Event und das `DOMContentLoaded` Event. Zeichnen Sie im Sequenzdiagramm ein, an welcher Stelle diese Events jeweils auftreten oder definieren Sie als Textantwort, wann diese Events auftreten. (4 Punkte)

Antwort direkt im Sequenzdiagramm oben oder hier:

Alternative AW: Das DOMContentLoadedEvent wird ausgelöst, sobald das HTML-Dokument vollständig geparst wurde und alle DOM-Elemente angelegt sind (nach Erreichen des </html> Tags) und die synchron geladenen JavaScripts <script> komplett interpretiert wurden.
Das load-Event wird erst geworfen, wenn auch alle externen Ressourcen wie src von Bildern/Videos geladen wurden.

Bewertung: Punkte im Diagramm jeweils in Klammern (je 2P). Wurden die Namen der Events vertauscht, aber ansonsten korrekt eingezeichnet (-1P).

Aufgabe 4: Authentifizierung (10 Punkte)

Im Rahmen des Unterrichtes haben Sie mehrere Authentifizierungsverfahren kennengelernt. Beispielsweise HTTP Basic Auth, Local Auth und OAuth 2.0.

a.) Um welches Authentifizierungsverfahren handelt es sich bei dem vorhergehenden Login-Beispiel aus Aufgabe 2? Begründen Sie, warum es keines der anderen beiden Authentifizierungs-Verfahren sein kann. (5 Punkte)

Antwort:

Es handelt sich um Local Auth. HTTP Basic Auth arbeitet ausschließlich mit Header-Feldern, es gäbe dort keine <input>-Tags in HTML. OAuth 2.0 benötigt einen Dritt-Server, welcher via JavaScript das Loginformular einbettet und die Authentifizierung durchführt (bspw. Google); hier im Beispiel werden die Daten aber direkt an den eigenen Server (/login) gesendet.

Bewertung: 1P für Local Auth. 2P für Basic Auth nur Header-Felder, 2P für OAuth benötigt Dritt-Server.

b.) Welche wesentliche Sicherheits-Voraussetzung wurde bei der Implementierung des Authentifizierungsverfahrens im Login-Beispiel aus Aufgabe 2 vergessen? Geben Sie auch an, welche Eigenschaften für die Kommunikation zwischen Browser und Server gelten, wenn diese Voraussetzung erfüllt ist. (5 Punkte)

Antwort:

Es wurde versäumt, die Kommunikation auf HTTPS aufzubauen. Erst dann ist die Übertragung sensitiver Daten (Login) sicher und bietet folgende Eigenschaften:

1. Der Client kann sich durch gültiges Zertifikat sicher sein, dass der Server der korrekte ist
2. Die Verbindung ist verschlüsselt und damit abhörsicher

Bewertung: 1P für HTTPS, 2P für Server ausgewiesen durch Zertifikat, 2P für Verschlüsselt=abhörsicher

Aufgabe 6: node.js/express Programmierung (18 Punkte)

Nehmen Sie an, Ihr neuer Kollege hat eine node.js/express Anwendung zur server-seitigen Ausführung implementiert, um folgende Aufgaben zu erfüllen:

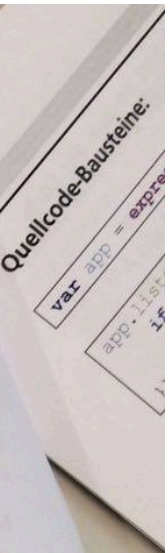
A1. Statische Dateien aus `/public` sollen ausgeliefert werden, wenn der entsprechende Pfad der aufgerufenen URL dazu passt (Bsp.: `http://localhost:3000/index.html` liefert entsprechend die Datei `/public/index.html` aus)

A2. Die Loginseite (index.html) sendet ggf. `HTTP POST Requests` an den Server. Diese Requests enthalten Formulardaten und sollen per AJAX gesendet werden. Für die einzige Benutzerin namens `ghopper` mit Passwort `secret` sollen die `id: 101` und das `token: a1b2c3d4e5` als JSON-Daten zurückgesendet werden. Stimmen die Login-Daten nicht, wird der Statuscode `422 (Unprocessable Entity)` gesendet.

A3. Akzeptiert der anfragende Client keine JSON-Daten, soll einfach auf die Startseite umgeleitet werden.

A4. Alle weiteren Anfragen sollen mit dem Status-Code `404 (Not found)` beantwortet werden.

Sein Quellcode ist durcheinandergeraten und sieht so aus (nächste Seite):



Quellcode-Bausteine:

C1
```
var app = express();
```

C2
```
app.listen(3000, function(err) {
    if (err) { console.error("Whoops! " + err); }
    else { console.log("Listening on Port 3000"); }
});
```

C3
```
app.use( /public , express.static(path.join(__dirname, 'public')));
```

C4
```
var path = require('path');
var express = require('express');
var bodyParser = require('body-parser');
```

C5
```
app.use(function(req,res) {
    res.status(404).end();
});
```

C6
```
app.route('/login')
    .post(function(req, res, next) {
        if (!req.accepts('json')) {
            next(); return;
        }
        if (req.body.username === 'ghopper' && req.body.password === 'secret') {
            // normally DB lookup here
            req.body.token = "a1b2c3d4e5";
            req.body.id = "101";
            res.json(req.body);
        }
        else {
            res.status(422); res.end();
        }
    });
```

C7
```
app.post('/login', function(req, res){
        res.redirect("/");
    });
```

C8
```
app.use(bodyParser.urlencoded({ extended: false }));
```

a.) Geben Sie die korrekte Reihenfolge der Code-Abschnitte C1 bis C8 hier an, damit diese dem typischen Aufbau einer node.js/express Anwendung entspricht. (8 Punkte)

Antwort: Die richtige Reihenfolge ist:

C4	C1	C3	C8	C6	C7	C5	C2

Bewertung: Je 1P für korrekte Position der Code-Bausteine (C3 muss zw. C1 u. C5). Folgefehler werden berücksichtigt, sofern nachvollziehbar. Mehr als 2 Reihenfolgenfehler führen zu 0P (keine Folgefehler mehr).

b.) Zusätzlich haben sich fünf (5) Programmierfehler eingeschlichen, so dass der Code nicht läuft. Korrigieren Sie den Quellcode in zwei der Code-Abschnitte C1-C5 oben direkt durch durchstreichen und dazwischen schreiben. (10 Punkte)
Pseudocode ist erlaubt. Schreiben Sie Syntax, die so nah wie möglich an korrektem JavaScript ist. **Hinweis:** Es handelt sich nicht um Tippfehler oder falsche Namen von API Funktionen, sondern um logische Fehler, vergessene Code-Teile, zu viele Parameter etc.

Antwort: direkt im Code.
Bewertung: je Code-Korrektur 2P (siehe direkt im Code). Es gab 6 potentielle Stellen. Nur 5 mussten gefunden werden.

Aufgabe 7: REST-Level (14 Punkte)

Sie nutzen die neue REST-API eines Anbieters, welcher Ihnen unter Anderem zu einem Städtenamen die Geo-Koordinaten (`lon, lat`) liefern kann.

Die API-Dokumentation enthält folgende Angaben: (auch für Aufgabe 8 zu nutzen)

Resource: geodata
Route-Endpunkt: `/geodata`

API-Basis-Pfad: `rest.geo-cool.my/`

Rest-Level: 3 (vollständiges HATEOAS)

Notwendige Header-Felder:

Feld-Name	Beschreibung erlaubter Werte (values)
`Accept`	nur „`application/json`“
`Content-Type`	nur „`application/json`“ (bei allen HTTP Methoden außer GET)
`X-API-Version`	nur „`3.0.0`“

Unterstütze GET-Parameter (alle optional)

Parametername (key)	Datentyp	Beschreibung erlaubter Werte (values)
`limit`	Number	Beschränkung der Ergebnisse eines HTTP GET einer Ressourcensammlung auf bestimmte Anzahl
`offset`	Number	Überspringen dieser Anzahl an Ergebnissen bei HTTP GET einer Ressourcensammlung
`contains`	String	Liefert bei HTTP GET nur Ergebnisse bei denen im Städtenamen (**cityname**) dieser Teil-String vorkommt.

Es wird empfohlen mittels `HTTP GET` und optionalen GET-Parametern einen Einstieg in die Ressourcensammlung geodata abzurufen und sich anhand der Meta-Daten weiter zu navigieren.
(Ende der API Dokumentation)

Ein HTTP GET auf die URL `http://rest.geo-cool.my/geodata/?contains=Berlin&limit=1` und gesetztem Accept: application/json, sowie X-API-Version: 3.0.0 liefert folgendes Ergebnis:

```
300: {
301:   "version": "3.0.0",
302:   "_self": {
303:     "read": "/geodata/?contains=Berlin",
304:     "next": "/geodata/?contains=Berlin&offset=1&limit=1"
305:   },
306:   "items": [
307:     {
308:       "_self": {
309:         "read": "/geodata/d4c3b2a1",
310:         "update": "/geodata/d4c3b2a1",
311:         "delete": "/geodata/d4c3b2a1"
312:       },
313:       "id": "d4c3b2a1",
314:       "cityname": "Berlin",
315:       "lon": 13.404954,
316:       "lat": 52.52000659999999,
317:       "country": {
318:         "_self": {
319:           "read": "/countries/a2b3c4"
320:         }
321:       },
322:       "pictures": {
323:         "_self": {
324:           "read": "/geodata/d4c3b2a1/pictures"
325:         }
326:       }
327:     }
328:   ]
329: }
```

a.) Begründen Sie anhand der vorliegenden Informationen, inwieweit die vier Eigenschaften einer guten API (GLUE) für diese Schnittstelle erfüllt sind. (6 Punkte)

Antwort:

Gute Dokumentation: **nein**, es fehlt in der API welche Schema-Eigenschaften die Ressource(n) eigentlich haben. Somit müsste man es ausprobieren bei einem update (PUT) was akzeptiert wird.

Langfristig nutzbar: **kann man nicht sagen**, da die Informationen keine Aussage zulassen (die API ist erst neu…vielleicht verschwindet die wieder)

Unabhängig: **ja**, die API ist unabhängig von proprietären Formaten oder irgendwelchen Endgeräten (nutzt nur REST-konforme Spezifikation und das offene JSON-Format)

Erweiterbar: **ja**, da die API-Versionierung mittels Header-Feld erfolgt (X-API-Version)

Bewertung: je 0,5P für die ausgeschriebenen GLUE Eigenschaften. Je 1P für ja/nein/keine Aussage inkl. passender Begründung.

b.) Welche Eigenschaften muss eine REST-API erfüllen, damit Sie REST-Level 3 entspricht. Sofern nötig, geben Sie auch an, welche Eigenschaften anderer REST-Level ebenfalls bei Level 3 erfüllt sein müssen. (8 Punkte)

Antwort:

Alle Eigenschaften der vorherigen Level 0,1,2, müssen auch erfüllt sein, da REST-Level aufeinander aufbauen. (1P)

Aus L0: Es muss mindestens einen Service-Endpunkt geben, an welchen die Anfragen gestellt werden (1P)

Aus L1: Es muss eindeutige Ressourcenzeiger für Sammlungen und einzelne Ressourcen geben (bspw. /geolocation und /countries/a2b3c4) (2P)

Aus L2: Operationen werden über die HTTP-Methoden ausgeführt (GET, POST, PUT, DELETE) und der BODY enthält keine Parameter mehr, nur noch die eigentlichen Daten (bei POST, PUT, PATCH). (2P)

Für L3: Verweise auf alle möglichen Operationen (2P) und Verweise auf alle weitere verbundene Ressourcen (2P) sind angegeben als Meta-Daten zusätzlich zu den abgefragten Daten

Bewertung: siehe Angaben im Text für einzelne Eigenschaften (je 1-2P). Maximal 8P.

Aufgabe 8: Backbone.js Client (14 Punkte)

Ihre Chefin ist begeistert von der neuen REST-API für Geodaten (siehe Dokumentation dazu am Anfang von Aufgabe 7). Sie werden gebeten, eine Backbone-Anbindung (nur Model und Collection) zu schreiben, die diese REST-Schnittstelle nutzt.
Gehen Sie davon aus, dass Asynchronous Module Definition (AMD) mittels `require.js` benutzt wird und die Pfade alle korrekt konfiguriert sind, so dass Sie `require(..)` und `define(..)` benutzen können. Folgender korrekter Code existiert bereits:

Datei: `appclient.js` (diese löst den Abruf mittels fetch(...) aus)

```
require(['backbone', 'models/geodata-collection'],
    function(Backbone, GeodataCollection) {
        var geodata = new GeodataCollection();
        geodata.fetch({
            headers: { 'X-API-Version': '3.0.0',
                       'Accept': 'application/json'
            }
        });
});
```

Ergänzen Sie folgenden JavaScript-Code der Dateien `models/geodata-model.js` und `models/geodata-collection.js`, so dass ein Abrufen der `geodata` Objekte funktioniert. Schreiben Sie Ihre Syntax so gut wie möglich als korrektes JavaScript.

Hinweis: Nutzen Sie folgende **`Backbone.Model`** Eigenschaften, sofern nötig:
`idAttribute, urlRoot, defaults, validate, initialize`

Nutzen Sie folgende **`Backbone.Collection`** Eigenschaften, sofern nötig:
`model, url, initialize, comparator`

Datei: models/geodata-model.js

```
define( ['backbone'] , function(backbone) {
    var GeodataModel = Backbone.Model.extend({
        idAttribute: 'id'
        urlRoot: '/geodata'
    });
    return GeodataModel;
});
```

Datei: models/geodata-collection.js

```
define( ['backbone', 'models/geodata-model'] , function(backbone, GeoModel) {
    var GeodataCollection = Backbone.Collection.extend({
        model: GeoModel ,
        url: '/geodata'
    });
    return GeodataCollection;
});
```

Bewertung: je Code-Baustein innerhalb define 1,5P , ansonsten je 2P. urlRoot bei Model nicht notwendig, wenn es in der Collection eine url: gibt und das model: in Collection gesetzt ist: In dem Falle gibt es die 2P für urlRoot auch, wenn urlRoot fehlt. Nicht notwendige initialize, validate etc. geben keine Punkte.

Aufgabe 9: Idempotenz (5 Punkte)

Geben Sie an, welche HTTP Methoden einer REST-Schnittstelle idempotent sein müssen, welche es sein können und welche es nicht sind.

Idempotenz-Eigenschaft	HTTP-Methoden (ggf. Komma-getrennt)
Ist NICHT vorhanden	POST
MUSS sein	GET, PUT, DELETE
KANN sein	PATCH

Bewertung: je 1P. 1P Abzug bei Mehrfachnennung. 1P Abzug, falls in falscher Zeile einsortiert.