Aufgabe 1: Thin-Client / Rich-Client Architekturen (14 Punkte)

Auf den beiliegenden Seiten sehen Sie ein Beispiel einer Web-Anwendung für das Auflisten und Absenden von miniTweets. Die angegebene HTML-Seite `index.html` wurde ausgeliefert bei einem `HTTP GET` auf die URL `http://localhost:3000` . Der Inhalt der `appclient.js`, `tweet.js` und `tweet-create.js` ist ebenfalls aufgelistet.

Ein Dateibaum gibt Ihnen die Information über die Verzeichnisstruktur. Ein Screenshot gibt Ihnen einen Eindruck des Erscheinungsbildes. (Der Übersichtlichkeit halber sind die Standard-Bibliotheken `require.js`, `jquery.js`, `underscore.js`, `backbone.js` nicht abgedruckt.)

a.) Begründen Sie, ob es sich bei vorliegender Seite um eine Single Page Application (SPA) handelt. (8 Punkte)

Verwenden Sie in Ihrer Begründung die Code-Zeilennummern aus den beiliegenden *.html und *.js Dateien.

Antwort: Ja es handelt sich um eine SPA da Backbone verwendet wird mit seinem definierten Model (Zeile 50-57), seinem Router (Zeile 107 - 118) und seinen Views die immer aktualisiert werden (z. 37 und z. 62) und (z. 32 und 66). Bei einer SPA wird im Gegensatz zum ~~RIA~~ Thin-Client-Prinzip (dort wird die Seite ständig neu geladen), die Seite nicht komplett geladen, sondern nur Teile von ihr. ✓

Passt nicht

✓ POST ✓ keine weiteren Links

3

b.) Benennen Sie eine Zeilennummer, an der eine Closure-Funktion beginnt. Auf welches Element greift diese Funktion zu, damit Sie zur Closure wird? (6 Punkte)

Verwenden Sie in Ihrer Begründung die Code-Zeilennummern aus den beiliegenden *.html und *.js Dateien.

Antwort: Zeilennummer, in der die Definition einer Closure-Funktion beginnt:

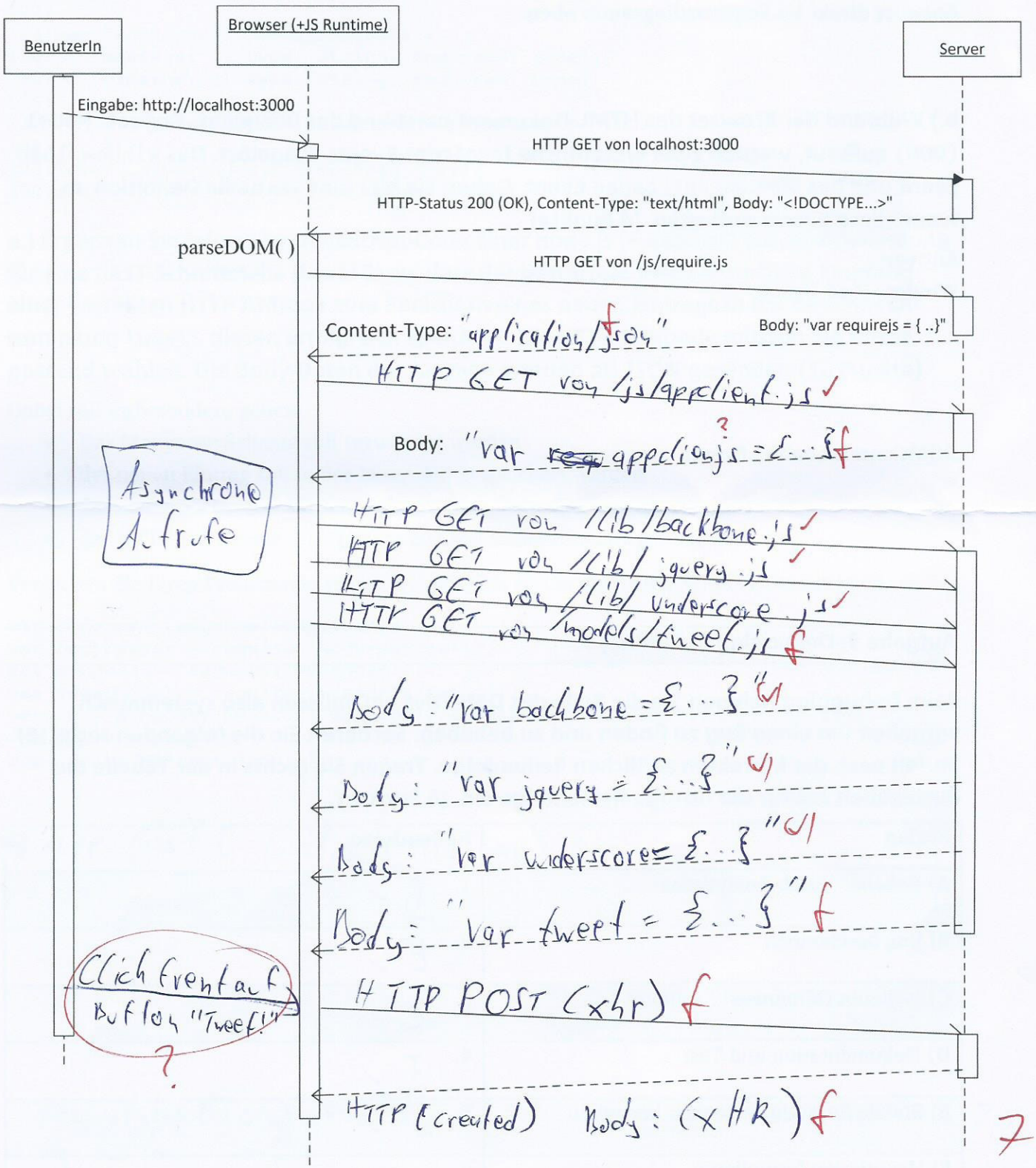
Zugriff auf folgendes Element:

0

1/3

Aufgabe 2: Lade-Reihenfolge im Sequenzdiagramm (17 Punkte)

Folgendes Sequenzdiagramm zeigt die Ablaufsequenz bei der Kommunikation zwischen Browser und Webserver beim erstmaligen Laden der miniTweets-Seite **aus Aufgabe 1** (siehe Quellcode dort). 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 1 zu Rate. (13 Punkte)

Geben Sie nur einmal den Content-Type mit an (danach brauchen Sie den nicht zu wiederholen)
Geben Sie jeweils mit an (sofern vorhanden): 1. die HTTP Methode, 2. Die Ressourcen-URL,
3. die HTTP Body Daten (nur erste 5-10 Zeichen des Body beispielhaft)

Antwort direkt im Sequenzdiagramm oben.

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. Geben Sie hier eine textuelle Definition an, wann diese Events auftreten. (4 Punkte)

Antwort:
Window load Event:

0

DOMContentLoaded Event:

7

Aufgabe 3: Debugging (6 Punkte)

Beim Debugging nehmen Sie die Rolle des Detektivs ein, müssen also systematisch vorgehen um einen Bug zu finden und zu beheben. Sortieren Sie die folgenden sechs (6) Stufen nach der korrekten zeitlichen Reihenfolge. Tragen Sie rechts in der Tabelle die Buchstaben A-E für die richtige Reihenfolge ein. (6 Punkte)

Stufen	Reihenfolge
A) Beheben durch Analysieren	1. B ✓
B) Bug beschreiben	2. E ✓
C) Isolieren (Minimized Example)	3. C ✓
D) Dokumentation und Test	4. F ✓
E) Stabile Reproduzierbarkeit herstellen	5. A ✓
F) Hypothesen formulieren	6. D ✓

6

Aufgabe 4: MongoDB mit Mongoose (12 Punkte)

Sie sehen im Folgenden ein Mongoose Schema für Tweets.

Inhalt der Datei ./models/tweets.js auf dem Server:

```
120:var mongoose = require('mongoose');
121:var Schema = mongoose.Schema;
122:
123:var TweetSchema    = new Schema({
124:    message:  { type: String, required: true},
125:    creator:  { type: String, required: true}
126:    }, {
127:        timestamps: {createdAt: 'timestamp'}
128:    });
129:TweetSchema.index({ timestamp: -1});
130:module.exports = mongoose.model('Tweet', TweetSchema);
```

a.) Ergänzen Sie folgenden JavaScript-Code einer node.js (+ express) Server-Anwendung für eine REST-Schnittstelle (Level 2) so, dass der gestartete Webserver beim Empfang einer korrekten HTTP Anfrage zum Speichern eines neuen Eintrags in der Ressourcen-sammlung `tweets` diesen erfolgreich speichert. Die HTTP Methode müssen Sie selbst passend wählen. Die Body-Daten der Anfrage werden als JSON gesendet. (12 Punkte)

Dabei soll insbesondere gelten:

- Das Mongoose Schema soll verwendet werden
- der neuen Eintrag soll in der MongoDB gespeichert werden
- der neuen Tweet soll zurücksendet werden (inkl. der vergebenen `_id`)
- der HTTP-Statuscode 201 (`Created`) soll in der Antwort gesetzt sein.

Versuchen Sie Ihren Pseudocode so nah wie möglich an der korrekten Syntax zu orientieren.

```
var express = require('express');
var bodyparser = require('body-parser');
var mongoose = require('mongoose');
var TweetModel = require('./models/tweet');
var db = mongoose.connect('mongodb://localhost:27017/me2'); // erfolgreich
var app = express();
app.use(bodyparser.json());

app.post( function (req, res) {
    res.status(200).end();
}
app.listen(3000);
```

2,5

Aufgabe 5: HTTP Operationen und Idempotenz (11 Punkte)

Bei REST-Schnittstellen werden verschiedene Operationen mittels HTTP Methoden durchgeführt.

a.) Ordnen Sie den CRUD-Operationen die passenden HTTP-Methoden zu. (4 Punkte)

CRUD-Operation	HTTP-Methoden (insgesamt fünf)
Create (Erstellen)	POST ✓
Read (Auslesen)	GET ✓
Update (Aktualisieren)	PUT/PATCH ✓
Delete (Löschen)	DELETE ✓

4

b.) Geben Sie an, welche der fünf potentiellen HTTP Methoden einer REST-Schnittstelle idempotent sein müssen, welche es sein können und welche es nicht sind. (5 Punkte)

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

3

c.) Begründen Sie Ihre Einordnung der HTTP Methode für das Erstellen (Create) einer neuen Ressourcen-Instanz aus Aufgabenteil (b.). Wieso kann diese nur hier eingeordnet sein? (2 Punkte)

Antwort: Idempotent heißt, der mehrfache Aufruf ändert den Systemzustand nicht. ~~Wenn man mehrfach GET~~

Wenn man mehrfach POST (Create) aufruft dann hat das immer eine Auswirkung auf den Systemzustand, da immer ein neues Objekt erzeugt und hinzugefügt wird. ✓

2

9

Aufgabe 6: Authentifizierungsverfahren (8 Punkte)

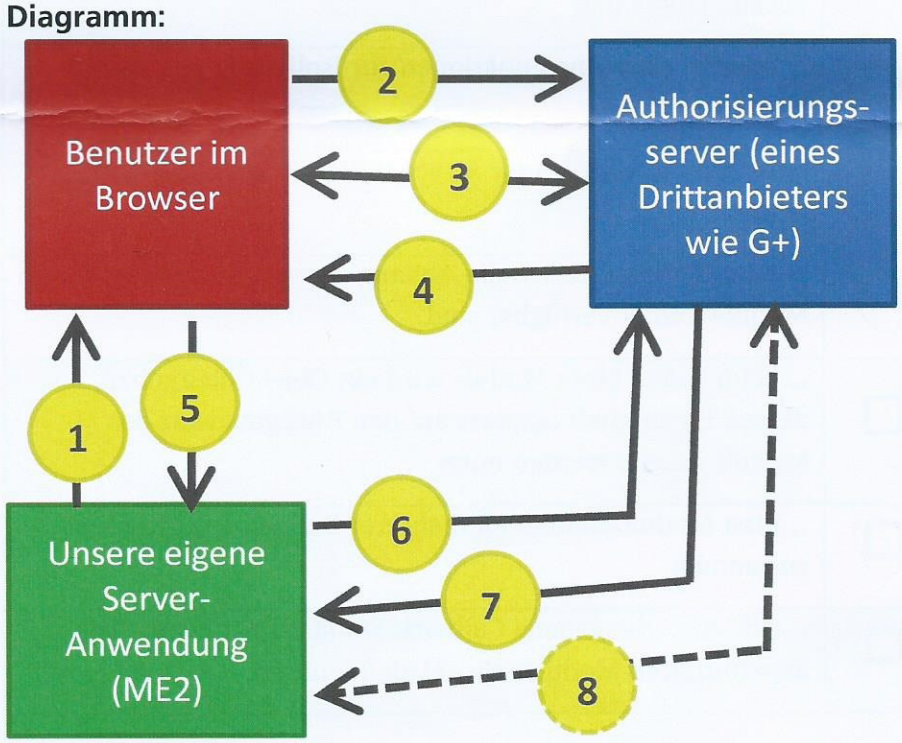
Ordnen Sie die folgenden Aktivitäten (A-F) richtig dem Diagramm für das OAuth-Verfahren zu. Schreiben Sie dazu je einen der Buchstaben (A-F) passend zur Reihenfolge der Schritte (1-8) in die Antwort-Tabelle. Sie dürfen Buchstaben mehrfach verwenden. (8 Punkte)

(Die Aktivitäten sind mit dem Beispiel Google+ als OAuth-Authentifizierungsserver und dem ME2-Server als zugreifenden Server formuliert.)

Aktivitäten (unsortiert):

(A) sendet Access Token
(B) ME2-Server kann nun mit Access Token auf Profil des Benutzers bei G+ zugreifen
(C) stellt im Browser des Nutzers einen Google+-Login-Button dar
(D) sendet Authorization-Code und callback-URL
(E) sendet Authorization-Code
(F) sendet Client-Anwendungs-ID (bspw. „ME2“)
(G) Benutzer authentifiziert sich (in der Regel mit Benutzername/Passwort)

Diagramm:



Antwort:

Schritte (1-8)	Aktivität
1	Gf
2	Bf
3	Ef
4	Df
5	Af
6	D ✓
7	F f
8	C f

1

Aufgabe 7: Modularisierung (9 Punkte)

Für die Modularisierung von JavaScript-Code (ECMAScript 5) haben sich zwei wesentliche Konzepte etabliert: AMD und CommonJS.

Geben Sie zu folgenden Aussagen an, ob diese jeweils zu AMD oder zu CommonJS passen (jeweils drei). Drei Aussagen passen entweder zu beiden oder keinem. Diese bitte als „beide/keine" ankreuzen. Achtung: Falsch gesetzte Kreuze geben 1 Punkt Abzug.

Antwort:

AMD (3x)	CommonJS (3x)	Beide/ keine (3x)	Aussage
☐	☐	☐	…erlaubt das Laden benötigter Module, wenn möglich, parallel unabhängig voneinander
☐	☒ ✓	☐	…erlaubt das Laden benötigter Module nacheinander
☐	☐	☐	…erlaubt die Definition mehrerer Module in einer Datei zusammengefasst.
☐	☐	☐	…benutzt anonyme Funktionen zur Isolierung von Modul-Namensräumen.
☐	☒ ✓	☐	…wird hauptsächlich für server-seitige Modularisierung benutzt.
☐	☐	☐	…beachtet beim Laden die Abhängigkeiten, so dass benötigte Module immer verfügbar sind.
☐	☐	☐	…beim Laden eines Moduls wird ein Objekt übergeben, dessen Eigenschaft .exports auf den Rückgabewert des Moduls gesetzt werden muss
☐	☐	☐	…fasst Moduldefinitionen mehrerer Dateien in einer zusammen
☐	☐	☐	…ruft eine übergebene Callback-Funktion auf, wenn die angeforderten Module alle geladen wurden.

2

Aufgabe 8: Open Web Application Security Project (OWASP) (9 Punkte)

Von den 10 Top Risiken der OWASP Liste für node/express-Web-Anwendungen, wurden die folgenden sechs intensiv im Unterricht besprochen. Wählen sie drei der Risiken durch Ankreuzen und nennen Sie jeweils mindestens eine Gegenmaßnahme, welche Sie ergreifen um dieses Risiko zu adressieren. (Nennen Sie jeweils andere Gegenmaßnahmen – nicht die gleichen).
Lässt sich das Risiko damit ganz entfernen oder nur verringern? Begründen Sie Ihre Antwort.

Liste der 10 Risiken

- ☐ A1 Injection
- ☐ A3 Cross-Side-Scripting (XSS)
- ☐ A4 Insecure Direct Object Reference (DOR)
- ☐ A5 Misconfiguration
- ☐ A8 Cross-Side-Request-Forgery (CSRF)
- ☐ A9 Insecure Components

Welche Gegenmaßnahmen ergreifen Sie? Warum verringern/entfernen diese das Risiko? (9 Punkte)

Antwort:

0

Aufgabe 9: REST APIs (14 Punkte)

Die Firma IcyCreamy ist ein großer Eisproduzent und möchte über eine REST-Schnittstelle Informationen zu den verfügbaren Eissorten anbieten. Nehmen Sie an, die REST-Schnittstelle für `flavours` (Eissorten) wurde durch jemand anderen bereits implementiert. Folgende Angaben stehen in der API-Dokumentation:

Ressource: flavours **API-Basis-Pfad:** `icycreamy.my/rest/v0.1b/`
Route-Endpunkt: `/flavours`

Schema (alle Felder sind erforderlich):

Feldname (key)	Datentyp	Beschreibung
name	String	eindeutiger Kurzname (1-8 Zeichen ohne Sonderzeichen) einer Eissorte, **wird als Identifier verwendet**
price	Number	Preis pro Kilo in Euro (maximal 3 Nachkommastellen)
ingredients	Array[String]	Liste der Inhaltsstoffe
start	Date	Datum des Verkaufsstartes

Operationen und Parameter:

Operation (übliche CRUD HTTP Methoden)	Notwendige Header-Felder	Notwendige Body-Felder	Antwort
CREATE	`Content-Type: application/json`	Alle Schema-Felder mit Werten als JSON-Objekt	Keine Body-Daten. Nur Header-Feld `Status-Code: 201` bei Erfolg oder Fehlercode bei Fehler
READ	`Accept: application/json`	Schema-Feld „`name`“ kann optional als URL-Parameter mitgesendet werden um nur eine und nicht alle Sorten zu lesen	Body-Daten: JSON-Array der Datensätze oder nur einen Datensatz, falls eine bestimmte `flavour` über URL-Parameter angefragt wurde
UPDATE	`Content-Type: application/json` `Accept: application/json`	Alle Schema-Felder des geänderten `flavours` mit Werten als JSON-Objekt	Alle Schema-Felder des aktualisierten Datensatzes als JSON-Objekt
DELETE	`Content-Type: application/json`	Schema-Feld „`name`“ muss als URL-Parameter mitgesendet werden um anzugeben welcher Datensatz entfernt werden soll	Nur Header-Feld `Status-Code: 204` bei Erfolg oder Fehlercode bei Fehler

a) Sie möchten nun die Angaben der Eissorte `vanille` ändern. Wie müssen Sie Ihre HTTP-Anfrage an die REST-Schnittstelle formulieren, um die korrekte Ressource mit der Operation UPDATE zu aktualisieren? (10 Punkte)

Verwenden Sie als neue Werte für `vanille` bspw.:
`price=12, ingredients=[cream,sugar,vanille], start=01.08.2016.`
Füllen Sie die vorgegeben Felder für HTTP Methode, URL der Anfrage, Header-Felder und mitzuschickendem BODY passend in möglichst korrekter Syntax aus.

HTTP-Methode	
URL der Anfrage	
Header-Felder	
Mitzusendender BODY	

0

b.) Begründen Sie anhand der vorliegenden API-Dokumentation, um welches REST-Level es sich maximal handelt (4 Punkte)

Antwort: Es handelt sich um REST-Level 2. → L0

- Bei REST-Level-0 werden die Ressourcen nicht über ~~die~~ eindeutige URLs angesprochen
- Bei REST-Level-1 werden die Daten immernoch in den Body geschrieben, aber ~~ü~~ Ressourcen werden über die URL ermittelt.
- Bei REST-Level-2 wird über die HTTP-Methoden navigiert. Dies trifft hier zu!

0/0