Klausur Webtechnologien

26. Februar 2020

Dies ist die schriftliche Abschlussprüfung der Lehrveranstaltung Webtechnologien. Bitte füllen Sie die Tabelle auf diesem Deckblatt aus und unterschreiben Sie die untenstehenden Hinweise.

Hinweise:

- Sie dürfen kein Blatt herausreißen! Bewertet werden nur die zusammengehefteten Blätter.
- Falls Sie mehr als die vorgesehenen Bereiche zum Beschreiben benötigen, können Sie die leere Seite am Ende der Klausur nutzen. Machen Sie eine Weiterführung der Antwort eindeutig kenntlich.
- Die Verwendung von eigenem Papier ist nicht erlaubt. Sie können uns nach Schmierblättern fragen.
- Bitte achten Sie darauf, dass Sie alle Seiten der Klausur vorliegen haben.
- Auf ihrem Platz darf sich nichts befinden außer: einem Lineal, mehreren Stiften, ihrem Personal- und Studierendenausweis. Sie dürfen keine Unterlagen verwenden. Alle elektronischen Geräte (auch Smartwatches) gehören ausgeschaltet in ihren Rucksack oder Tasche. Diese müssen Sie in der Reihe vor Ihnen abstellen.
- Sie dürfen keine roten und nur dokumentenechte Stifte verwenden! Das bedeutet, Sie dürfen weder einen Füller noch einen Bleistift verwenden.

Matrikelnummer	
Name	
Vorname	
Studiengang	
Hiermit bestätige ich, dass ich die Hinweise verstanden habe und mich in der Lage fühle diese Prüfung durchzuführen. **Unterschrift:**	

	Grundlagen	Styling von Dokumenten	JavaScript im Browser	Server	Sicherheit	Gesamt
Erreichbar	14	21	29	24	12	100
Erreicht						

1 Grundlagen (**14 Punkte**)

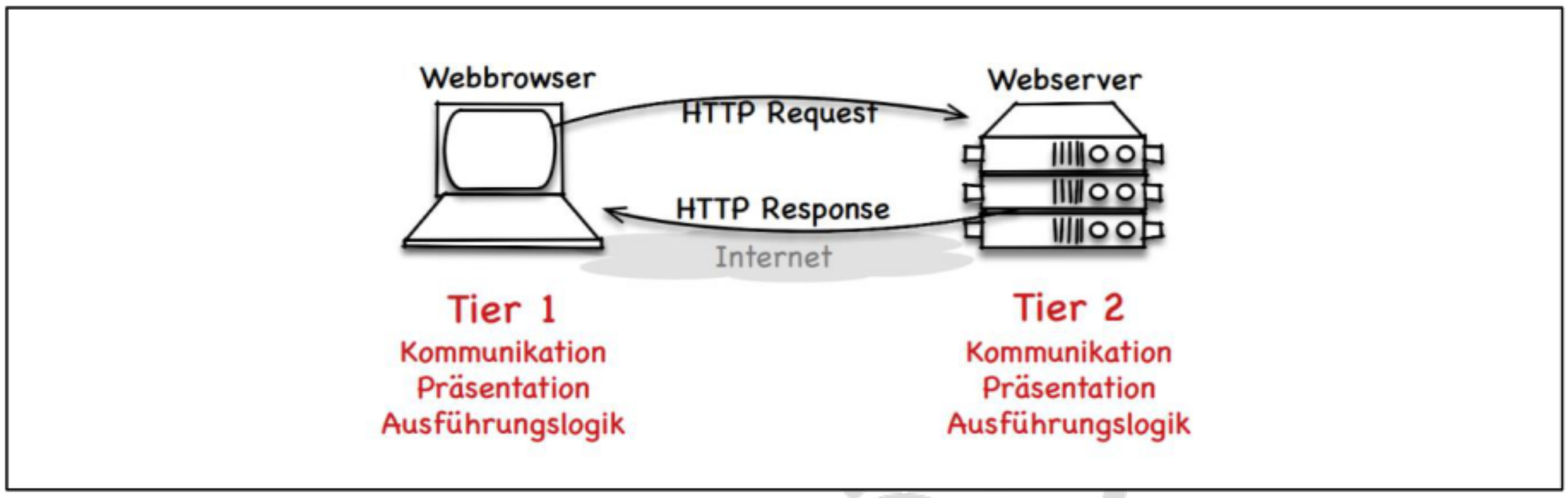
1.1 Multiple-Choice (**5 Punkte**)

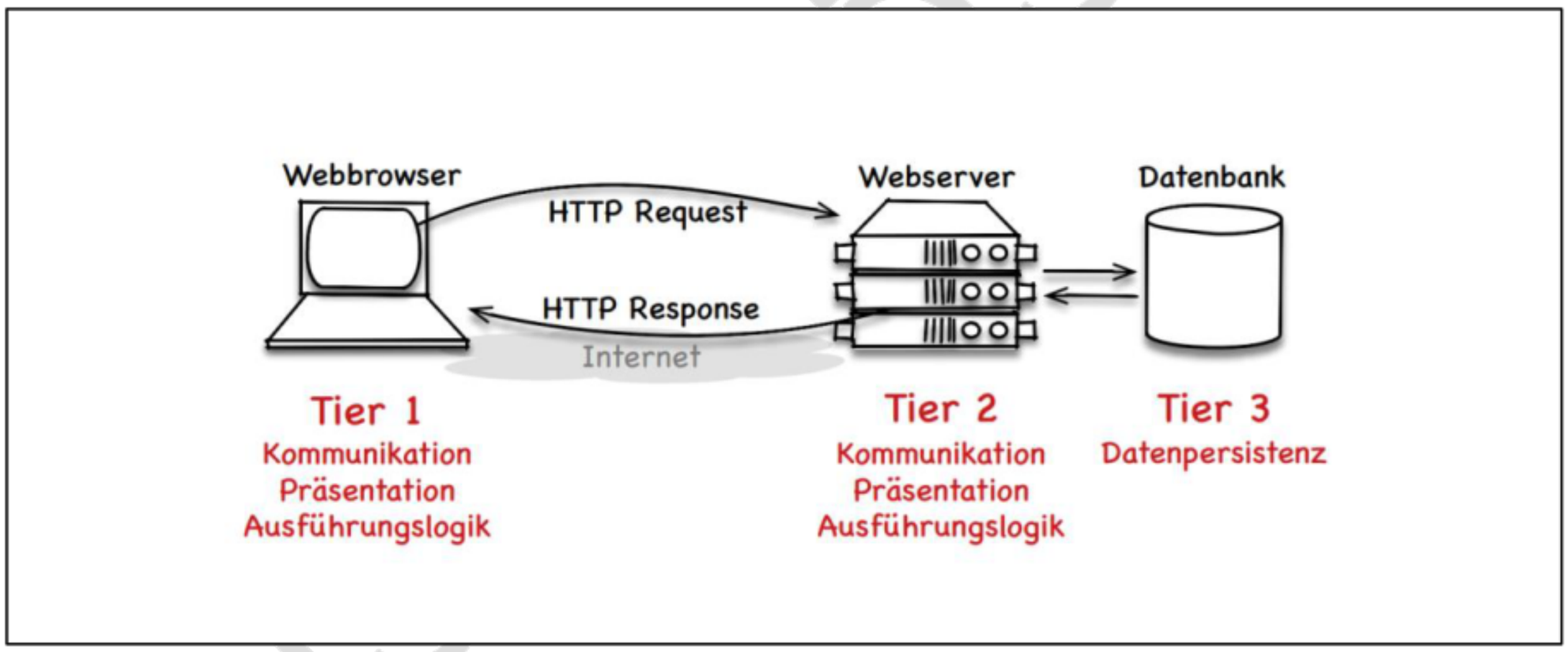
Kreuzen Sie an, ob die aufgeführten Aussagen wahr oder falsch sind. Für jedes richtig gesetzte Kreuz erhalten Sie einen Punkt. Für ein falsch gesetztes Kreuz erhalten Sie keinen Punkt.

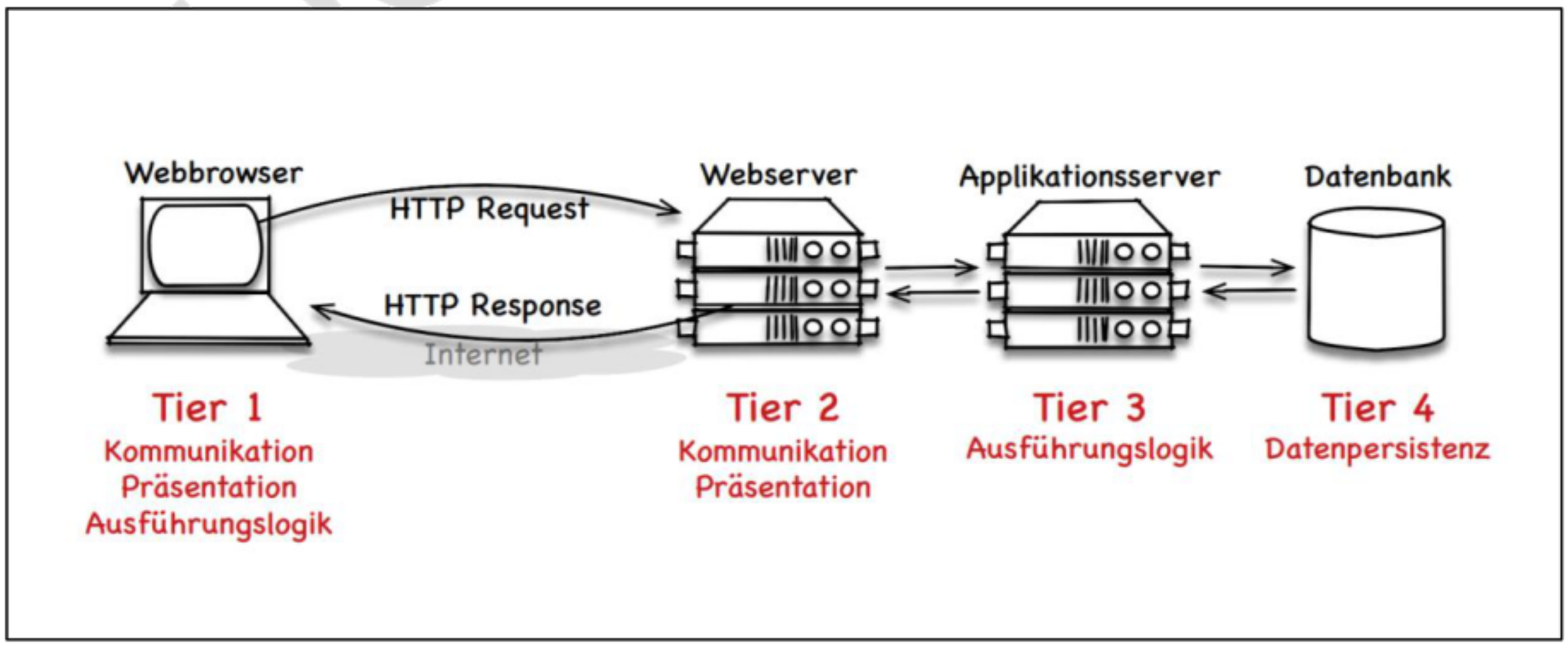
Wahr	Falsch	Aussage
☒	☐	Das Web der Dokumente wurde zu einer Plattform für „Rich Content“ und eine Ausführungsumgebung für Anwendungen weiterentwickelt.
☒	☐	Das Web 3.0 und das Internet der Dinge sieht eine Verlinkung unterschiedlicher Datenquellen vor.
☐	☒	Man spricht nur von einer Webapplikation, wenn diese vollständig im Webbrowser ausgeführt wird.
☒	☐	Ein Webservice stellt überwiegend Application Programming Interfaces (APIs) für Maschinen bereit.
☒	☐	Durch den MEAN-Stack lassen sich Daten auf jeder Ebene des Stacks in einem einheitlichen Format verarbeiten.

1.2 Grundarchitekturen (**9 Punkte**)

Geben Sie für die jeweiligen Komponenten in den unten aufgeführten Architekturen Ihre vorgesehenen Funktionen (Aufgaben) an. Die vorgesehenen Funktionen sind: **Kommunikation**, **Präsentation**, **Ausführungslogik** und **Datenpersistenz**. Beachten Sie, dass eine Komponente mehrere Funktionen haben kann und dass Funktionen auch von mehreren Komponenten übernommen werden können.







2 Styling von Dokumenten (**21 Punkte**)

Gegeben sei die folgende Webseite bestehend aus 3 DIV-Elementen:

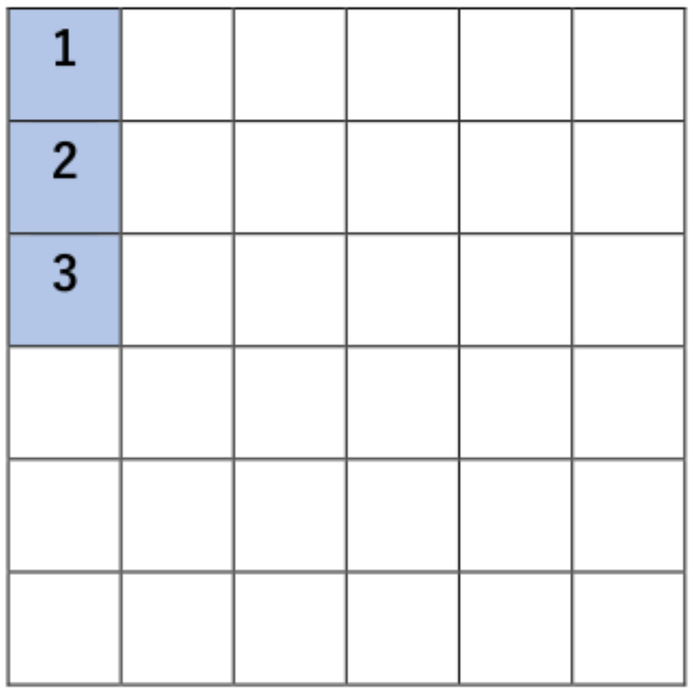
```
<html>
  <head>
    <style>
      div {width: 150px; height: 150px; background: “gray”}
    </style>
  </head>
  <body>
    <div>1</div>
    <div>2</div>
    <div>3</div>
  </body>
</html>
```

In den nächsten Aufgaben sollen Sie entscheiden wie die DIV-Elemente angeordnet werden, wenn es keine weitere CSS-Regel gibt (siehe 2.1.1), wenn eine CSS-Regel (siehe 2.1.2) hinzukommt und wenn zusätzlich dazu eine weitere CSS-Regel (siehe 2.1.3) hinzukommt. Im letzten Aufgabenteil gilt also auch die CSS-Regel aus 2.1.2.

Schraffieren Sie die entsprechenden Kästchen und kennzeichnen Sie diese mit der jeweiligen Nummer des DIV-Elements (1, 2 oder 3). Sie können annehmen, dass ein Kästchen genau 150 Pixel hoch und breit ist.

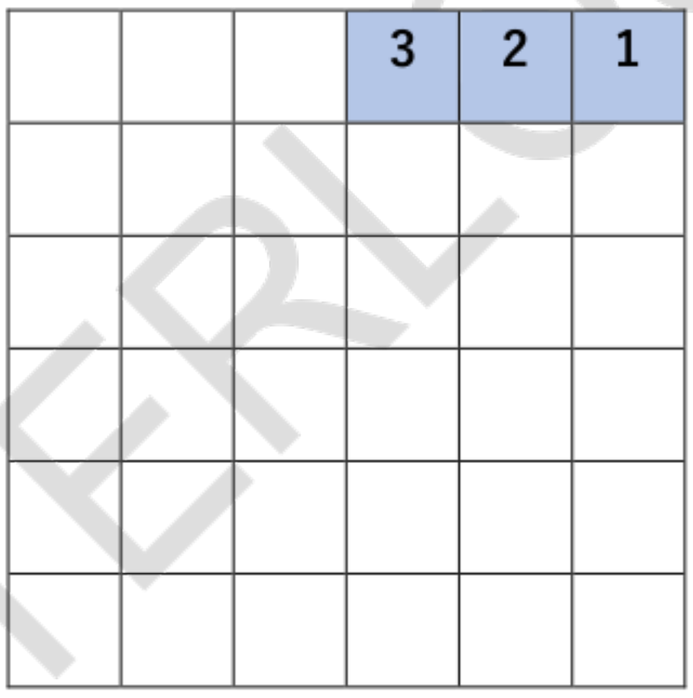
2.1 CSS-Positionierung (**6 Punkte**)

2.1.1 Zeichnen Sie die DIV-Elemente an Ihre entsprechende Stelle. (**2 Punkte**)



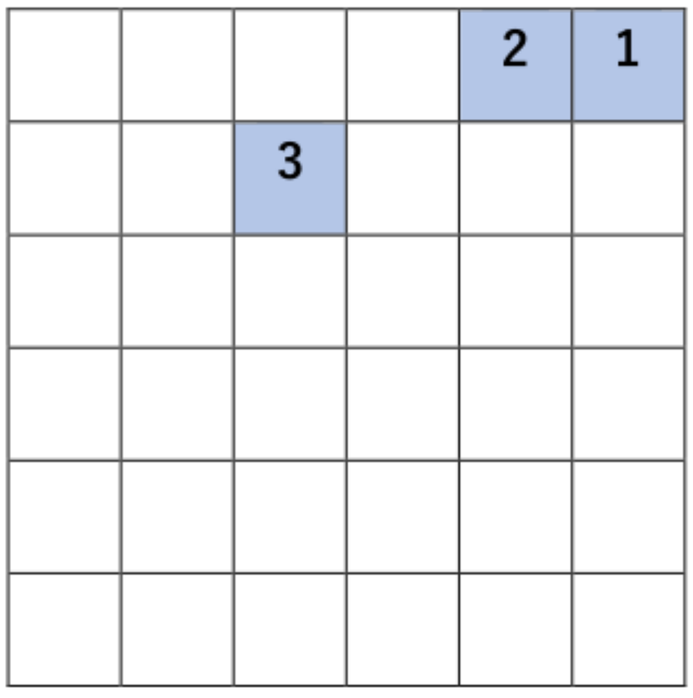
2.1.2 Wir fügen nun die untenstehende CSS-Regel dem Style-Tag im HTML-Head hinzu. Zeichnen Sie die DIV-Elemente an Ihre entsprechende Stelle. (**2 Punkte**)

```
div {float: right;}
```



2.1.3 Wir fügen nun die untenstehende CSS-Regel dem Style-Tag im HTML-Head hinzu. Zeichnen Sie die DIV-Elemente an Ihre entsprechende Stelle. (**2 Punkte**)

```
body div:last-child {position: absolute; margin: 150px 300px;}
```



2.2 CSS-Regeln (**6 Punkte**)

2.2.1 Schreiben Sie eine CSS-Regel, die für jede zweite Spalte einer Tabelle die Hintergrundfarbe „lightgray“ setzt. Der Table-Head soll nicht betrachtet werden. Sie können davon ausgehen, dass ein Table-Body existiert. (**3 Punkte**)

```
tbody tr td:nth-child(2n) {
    background: lightgray;
}
```

2.2.2 Schreiben Sie eine CSS-Regel, die für Bildschirme mit einer Breite unter 1024 Pixel die Schriftgröße im gesamten HTML-Dokument auf 16pt setzt. (**3 Punkte**)

```
@media (max-width: 1024px) {
    html {
        font-size: 16pt;
    }
}
```

2.3 Bootstrap (**9 Punkte**)

Gegeben sei der folgende Code einer HTML-Webseite, die mit Bootstrap realisiert wurde.

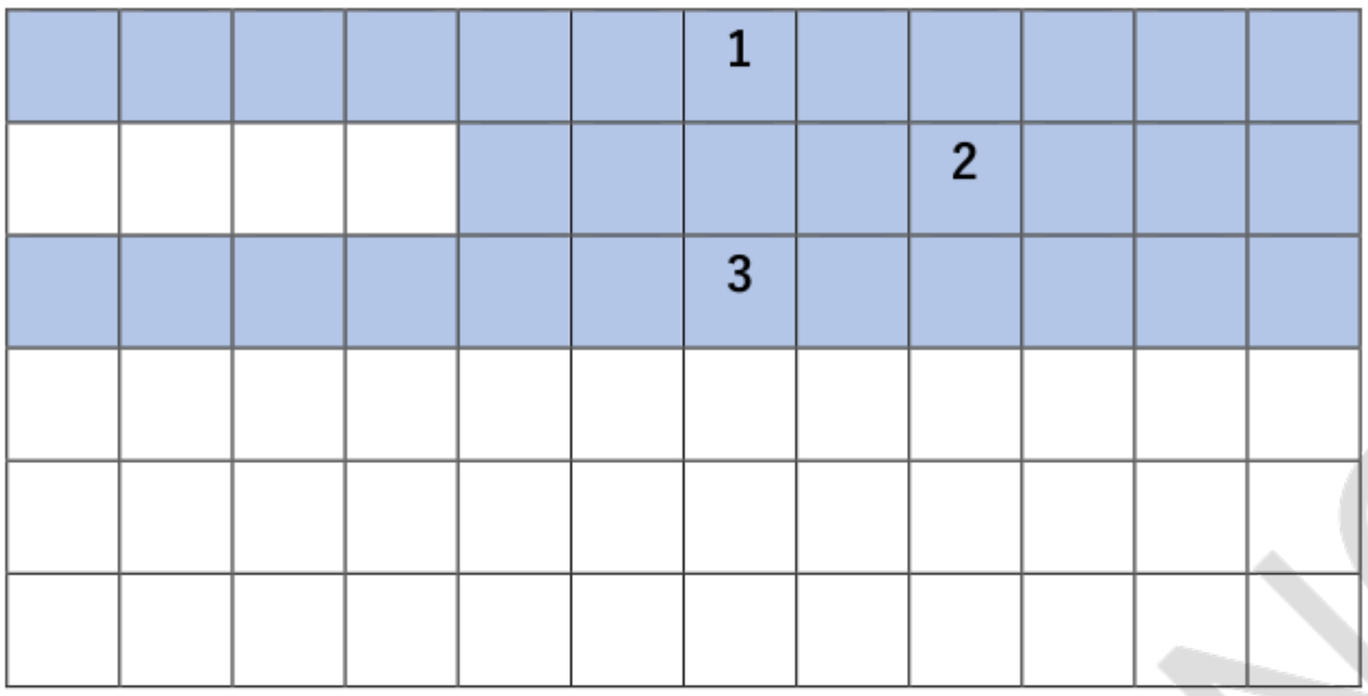
```
<html>
  <head>
    <link ...>
    <style>
        div {background: “gray”;}
    </style>
  </head>
  <body>
    <div class="container">
      <div class="row">
        <div class="col-sm-4">1</div>
        <div class="col-sm-4 offset-4 col-lg-8">2</div>
        <div class="col-sm-4 offset-xl-4 col-lg-8 offset-lg-2">3</div>
      </div>
    </div>
  </body>
</html>
```

In der nächsten Aufgabe sollen Sie entscheiden wie die einzelnen DIV-Elemente angeordnet werden, wenn der Bildschirm eine gewisse Breite hat. Schraffieren Sie dazu die entsprechenden Kästchen und geben Sie an um welches DIV-Element es sich handelt, indem Sie diese mit 1, 2 oder 3 kennzeichnen. Für diese Aufgabe nehmen wir an, dass die DIV-Elemente eine „height“ von genau einem Kästchen haben.

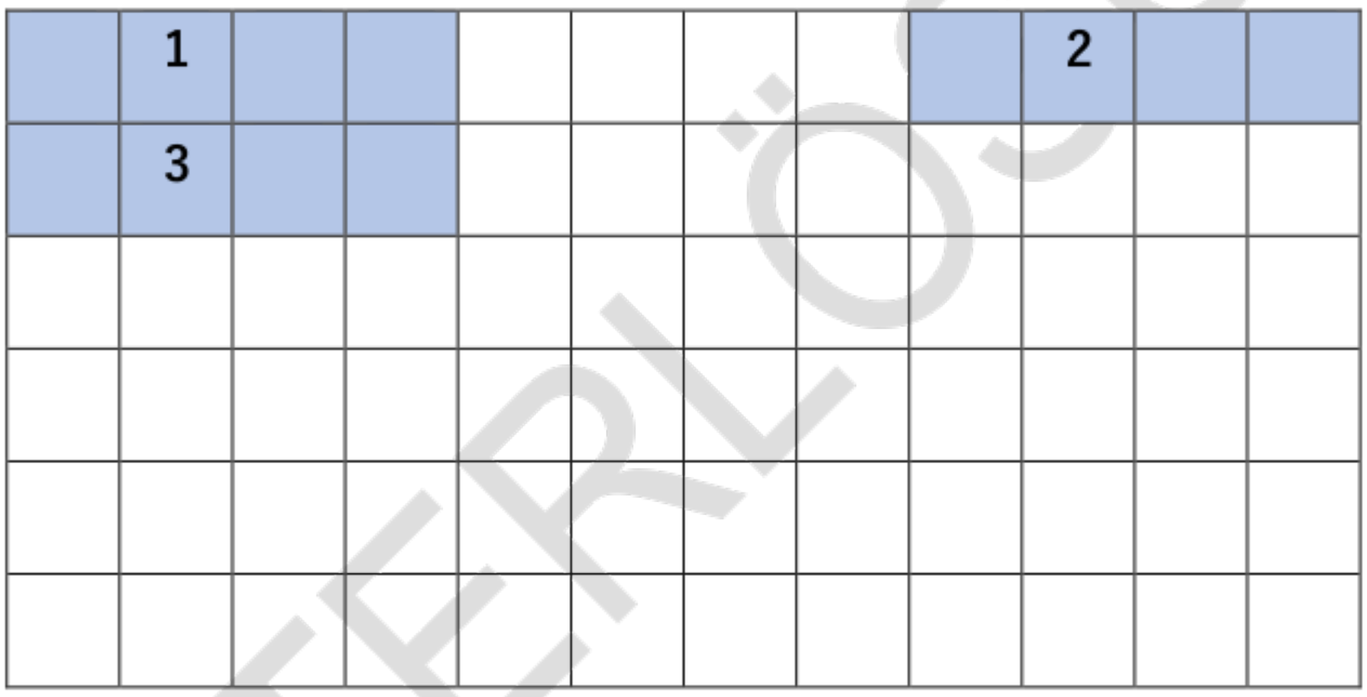
Beachten Sie weiter die sogenannten Breakpoints von Bootstrap, die wie folgt lauten:

- **xs:** < 576 Pixel
- **sm:** >= 576 Pixel und < 768 Pixel
- **md:** >= 768 Pixel und < 992 Pixel
- **lg:** >= 992 Pixel und < 1200 Pixel
- **xl:** > 1200 Pixel

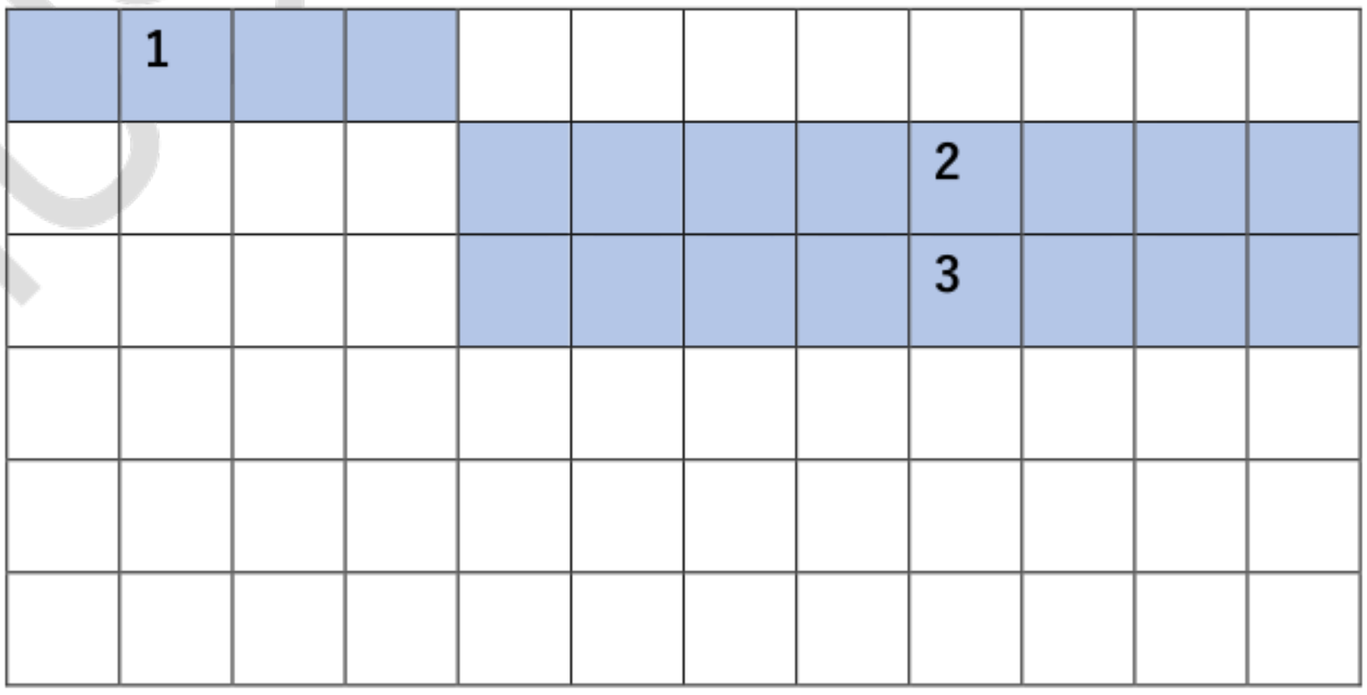
2.3.1 Nehmen Sie an die Bildschirmbreite ist **kleiner** als **576** Pixel. Zeichnen Sie die Boxen an die entsprechende Stelle. (**3 Punkte**)



2.3.2 Nehmen Sie an die Bildschirmbreite ist **größer** als **576** Pixel, aber **kleiner** als **992** Pixel. Zeichnen Sie die Boxen an die entsprechende Stelle. (**3 Punkte**)



2.3.3 Nehmen Sie an die Bildschirmbreite ist **größer** als **1200** Pixel. Zeichnen Sie die Boxen an die entsprechende Stelle. (**3 Punkte**)



3 JavaScript im Browser (**29 Punkte**)

3.1 JavaScript Dynamische Typisierung (**6 Punkte**)

Kreuzen Sie an, ob die aufgeführten JavaScript Ausdrücke zu „true“ oder „false“ evaluiert werden. Für jedes richtig gesetzte Kreuz erhalten Sie einen Punkt. Für ein falsch gesetztes Kreuz erhalten Sie keinen Punkt.

true	false	JavaScript Ausdruck
☒	☐	`true == 1`
☒	☐	`undefined == null`
☒	☐	`typeof NaN == "number"`
☐	☒	`typeof typeof NaN === "String"`
☒	☐	`typeof null === typeof []`
☒	☐	`typeof (() => {}) === typeof console.log`

3.2 Prototypes (**4 Punkte**)

3.2.1 Schreiben Sie eine Prototype-Funktion für den Datentyp Array, die selbst eine Funktion übergeben bekommt, die sie auf jedem Element des Arrays anwendet und das Ergebnis zurückliefert. (**2 Punkte**)

```
Array.prototype.myMap = function(f) {
    for (let i=0; i < this.length; i++) {
        this[i] = f(this[i]);
    }
    return this;
}
```

3.2.2 Schreiben Sie eine Prototype-Funktion für den Datentyp Array, die eine Bedingung übergeben bekommt, die sie für jedes Element des Arrays überprüft und jene zurückliefert für die die Bedingung zu „true“ evaluiert wird. (**2 Punkte**)

```
Array.prototype.myFilter = function(condition) {
    let filteredArray = [];
    for (let i=0; i < this.length; i++) {
        if (condition(this[i])) {
            filteredArray.push(this[i]);
        }
    }
    return filteredArray;
}
```

3.3 Higher-Order-Functions (**6 Punkte**)

Gegeben seien zwei JavaScript-Arrays, die eine undefinierte Anzahl an JavaScript-Objekten enthalten, die jeweils die folgenden Strukturen haben:

```
let examResults = [
  {studentId: 123, note: 3.3, versuch: 1},
  // ...
];

let studentData = [
  {studentName: "Murat Mustermann", studentId: 123, studiengang: "Wi-
   Ing (B. Sc.)", semester: 3},
  // ..
];
```

In der nächsten Aufgabe sollen Sie mit Higher-Order-Functions die Daten nach bestimmten Kriterien durchsuchen. Sie können dafür annehmen, dass die beiden JavaScript-Arrays („`examResults`“ und „`studentData`“) nur JavaScript-Objekte enthalten und dass diese die jeweilige oben aufgeführte Struktur beibehalten. Die Objekte haben immer dieselben Attribute dessen Werte immer vom selben Typ sind. Ferner enthalten beide Arrays zur Vereinfachung keine Duplikate.

3.3.1 Filtern Sie nach allen Studenten, die im zweiten Versuch eine 5.0 geschrieben haben. (**2 Punkte**)

```
examResults.filter((e) => (e.note === 5.0 && e.versuch === 2));
```

3.3.2 Ermitteln Sie die Studiengänge aller Studenten, die eine 2.0 oder besser geschrieben haben. Das resultierende Array kann Duplikate enthalten. (**2 Punkte**)

```
examResults
    .filter((e) => (e.note <= 2.0))
    .map((e) => studentData.find((el) =>
        e.studentId === el.studentId).studiengang);
```

3.3.3 Ermitteln Sie die durchschnittliche Semesterzahl, der Studenten, die eine 5.0 geschrieben haben. (**2 Punkte**)

```
examResults
    .filter((e) => (e.note === 5.0))
    .map((e) => studentData.find((el) =>
        e.studentId === el.studentId).semester)
.reduce((a, b) => a + b, 0)
/ examResults.filter((e) => (e.note === 5.0)).length;
```

3.4 Vue.js (**13 Punkte**)

Gegeben sei die folgende Vue.js-Applikation bestehend aus den Dateien „App.vue“, „ConcertList.vue“, „Concert.vue“ und „Band.vue“.

App.vue

```
<template>
  <div id="app">
    <concert-list />
  </div>
</template>

<script>
  import ConcertList from "./components/ConcertList.vue";
  export default {
    components: { ConcertList }
  };
</script>
```

ConcertList.vue

```
<template>
  <div><!-- Template --></div>
</template>

<script>
  import Concert from "./Concert.vue";
  import Band from "./Band.vue";
  export default {
    components: { Concert, Band },
    data() {
      return {
        concerts: [
          { id: 0, title: "Guns N' Roses", availableTickets: 10595,
            date: 1590514230000, bandId: 21 },
          { id: 0, title: "Iron Maiden", availableTickets: 322,
            date: 1591723830000, bandId: 192 },
          // ...
        ]
      };
    },
    methods: {
      bookTicket() {
        // Daten an Webserver senden
      }
    }
  };
</script>
```

Concert.vue

```
<template>
  <div><!-- Template --></div>
</template>

<script>
  import Band from "./Band.vue";

  export default {
    components: { Band },
    props: [/* Props */],
    methods: {
      handleClick: function() {
        this.$emit("concert-chosen");
      }
    }
  };
</script>
```

Band.vue

```
<template>
  <div><!-- Template --></div>
</template>

<script>
  export default {
    data: function() {
      return {
        name: "",
        mitglieder: [],
        aufTour: false,
      };
    },
    props: [„bandId“],
    created: {
      // Band Informationen vom Server abrufen
    },
  };
</script>

<style scoped>
  /* Styling */
</style>
```

3.4.1 Implementieren Sie das Template für die Komponente „Band.vue“ und beachten Sie folgende Anforderungen: **(3 Punkte)**

- Der Bandname soll in roter Schrift angezeigt werden.
- Die Mitglieder der Band sollen mit Komma getrennt in einem DIV-Element angezeigt werden.
- Sofern „`aufTour`“ „`true`“ ist, soll in einem weiteren DIV-Element ein Text „Jetzt auf Tour“ angezeigt werden.

```
<template>
    <div>
        <span>{{name}}</span>
        <div>{{mitglieder.join(", ")}}</div>
        <span v-if="aufTour">Jetzt auf Tour!</span>
    </div>
</template>

<style scoped>
    span {color: red}
</style>
```

3.4.2 Implementieren Sie das Template für die „Concert.vue“ und beachten Sie folgende Anforderungen: **(6 Punkte)**

- Die Komponente bekommt die folgenden Attribute eines Konzerts übergeben: eine ID, der Titel des Konzerts, die Anzahl noch verfügbarer Tickets, wann das Konzert stattfindet und die ID der Band.
- Der Name des Konzerts wird fett formatiert.
- Wenn weniger als 1000 Tickets noch verfügbar sind, wird die Anzahl Tickets rot gefärbt, sonst schwarz.
- Das Datum wird menschenlesbar dargestellt.
- Die Informationen der Band werden unter den Informationen des Konzerts angezeigt.
- Mittels eines Klicks auf einen Button können Nutzer ein Konzert auswählen. Die `handleClick`-Methode wird beim Klick ausgeführt.

```
<template>
    <div>
        <b>{{this.title}}</b>
        <div v-if="this.availableTickets < 1000"
          class="hot">{{this.availableTickets}}</div>
        <div v-else>{{this.availableTickets}}</div>
        <div>{{new Date(this.date).toString()}}</div>
        <band v-bind:bandId="this.bandId"></band>
        <button @click="handleClick">Auswählen</button>
    </div>
</template>

<style scoped>
    .hot {color: red}
</style>
```

3.4.3 Implementieren Sie das Template für die „ConcertList.vue“ und beachten Sie folgende Anforderungen: **(4 Punkte)**

- Zeigen Sie alle Konzerte aus dem `concerts`-Array in jeweils einem DIV-Element an.
- Übergeben Sie an jede „Concert“-Instanz die geforderten Attribute.
- Wurde ein Konzert vom Nutzer ausgewählt, soll die `bookTicket`-Methode aufgerufen werden.

```
<template>
    <div>
        <div v-bind:key="concert.id" v-for="concert in concerts">
            <concert
              v-bind:id="concert.id"
              v-bind:title="concert.title"
              v-bind:availableTickets="concert.availableTickets"
              v-bind:date="concert.date"
              v-bind:bandId="concert.bandId"
              @concert-chosen="bookTicket">
            </concert>
        </div>
    </div>
</template>
```

4 Server (**24 Punkte**)

4.1 Node.js - „http“-Modul (**5 Punkte**)

Nutzen Sie das Node.js „http“-Modul, um einen Echo-Server zu implementieren, der den Request-Body aus einer Anfrage ausliest und unverändert an den Client zurückschickt. Der Server soll auf Port 8080 laufen.

4.1.1 Importieren Sie das „http“-Modul. (**1 Punkt**)

```
var http = require('http');
```

4.1.2 Implementieren Sie den Server. (**4 Punkte**)

```
http.createServer((req, res) => {
    var body = '';
    req.on('data', (chunk) => {
        body += chunk.toString();
    }).on('end', () => {
        res.end(body);
    });
}).listen(8080);
```

4.2 HTTP, TCP/IP & REST (**6 Punkte**)

Kreuzen Sie an, ob die aufgeführten Aussagen wahr oder falsch sind. Für jedes richtig gesetzte Kreuz erhalten Sie einen Punkt. Für ein falsch gesetztes Kreuz erhalten Sie keinen Punkt.

Wahr	Falsch	Aussage
☐	☒	HTTP steht für Hypertext Transport Protocol und wurde in den 90er-Jahren in Europa entwickelt.
☐	☒	TCP ist ein zustandsloses Protokoll, das vor allem auf Grund seiner Verlässlichkeit für die Übermittlung von Hypertexten gut geeignet ist.
☒	☐	Laut HTTP Spezifikation wird eine HTTP-Anfrage immer durch einen Client und eine HTTP-Antwort durch einen Server gesendet.
☐	☒	Ein Single-Thread-Server arbeitet ausschließlich synchron: Er nimmt HTTP-Anfragen entgegen, bearbeitet diese und versendet Antworten.
☐	☒	Die Antwort auf eine HTTP-OPTIONS Anfrage enthält das Header-Feld ‚Methods', in dem die vom Server unterstützten HTTP-Methoden aufgelistet werden.

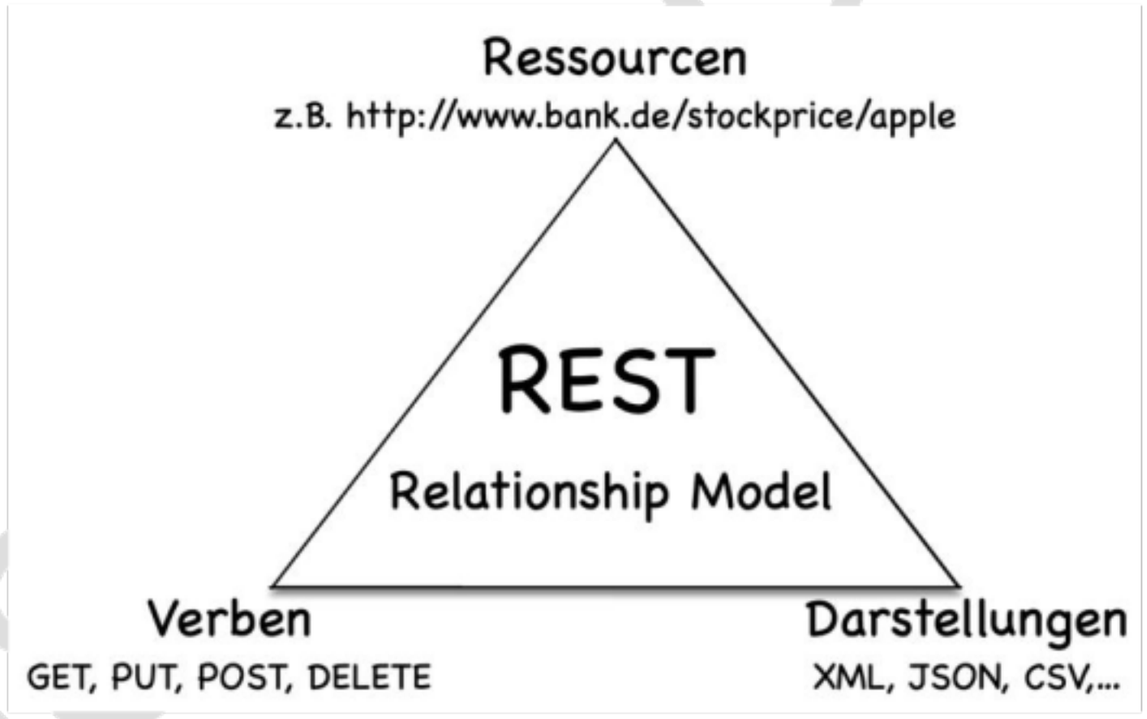
4.3 REST (**5 Punkte**)

4.3.1 Markieren und benennen Sie die Bestandteile einer URL an folgendem Beispiel: (**2 Punkte**)

http://www.tu-berlin.de:1992/snet/module?name=webtechnologien

scheme authority path query

4.3.2 Nennen Sie die Bestandteile des REST-Relationship-Modells und geben Sie für jeden Bestandteil ein Beispiel an. (**3 Punkte**)



4.4 Express,js (**8 Punkte**)

Gegeben ist folgender mit Express implementierte Server:

```
var express = require('express');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var dataRouter = require('./routes/data.js');

var app = express();
var data = {100: {name: 'Peter'}};

app.use(cookieParser());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded());

// Hinterlegt im Response-Objekt eine Referenz auf das "data" Array, damit
   externe Module auf die Daten zugreifen können
app.use((req, res, next) => {
    res.locals.data = data;
    next();
});

// TODO: Middleware

app.listen(80);
```

4.4.1 Implementieren Sie eine Middleware, welche den Session-Cookie "counter" aus dem Request ausliest, den Wert inkrementiert und den Cookie im Response mit neuem Wert anhängt. Sollte dem Request kein entsprechender Cookie anhängen, wird der Wert des Cookies im Response auf 1 gesetzt. **(2 Punkte)**

```
app.use((req, res, next) => {
    var counter = req.cookies.counter ? req.cookies.counter++ : 1;
    res.cookie('counter', counter);
});
```

4.4.2 Binden Sie den in der Datei „./routes/data.js“ implementierten Router an den Pfad „/data“. **(1 Punkt)**

```
app.use('/data', dataRouter);
```

Bisher sieht der Inhalt der Datei „./routes/data.js“ wie folgt aus:

```
var express = require('express');
var router = express.Router();

// TODO: Middleware
```

Ergänzen Sie die Datei entsprechend der folgenden Anweisungen:

4.4.3 Implementieren Sie eine Middleware, die bei GET-Anfragen an die URL „http://localhost/data/<id>“ den Eintrag im "data" Objekt mit der entsprechenden ID zurückgibt. **(2 Punkte)**

```
router.get('/:id', (req, res, next) => {
    res.json(res.locals.data[req.params.id]);
    next();
}
```

4.4.4 Implementieren Sie eine Middleware, die bei POST-Anfragen an die URL „http://localhost/data/“ im "data" Objekt einen neuen Eintrag mit einer neu generierten ID und dem im Request-Body (im Property „name“) angegebenen Namen anlegt. Anschließend wird Statuscode 201 zurückgegeben. **(2 Punkte)**

```
router.post('/', (req, res, next) => {
    res.locals.data[req.params.id] = req.body;
    res.status(201).send();
    next();
});
```

4.4.5 Sorgen Sie dafür, dass der Router in der Datei „server.js“ eingebunden werden kann. (**1 Punkt**)

```
module.exports = router;
```

5 Sicherheit (**12 Punkte**)

5.1 Basis-Konzepte (**3 Punkte**)

Nennen und beschreiben Sie die drei wichtigen Sicherheitsfunktionen. Für jede richtig genannte Sicherheitsfunktion erhalten Sie einen halben Punkt. Für jede richtige Beschreibung einer Sicherheitsfunktion einen weiteren halben Punkt.

Authentisierung und Authentifizierung

„Authentisierung ist der Nachweis einer Person, dass sie tatsächlich diejenige Person ist, die sie vorgibt zu sein. Authentifizierung ist die Überprüfung der behaupteten Authentisierung durch die Person oder Entität gegenüber der die Authentisierung erfolgt ist.“

Vertraulichkeit

„Vertraulichkeit einer Nachricht wird erreicht, wenn nur die autorisierten Personen bzw. Entitäten Zugriff auf die Nachricht haben und das Lesen der Nachricht durch Dritte verhindert wird.“

Integrität

„Integrität bedeutet, dass Daten nicht unbemerkt, z.B. durch Angreifer, verändert werden können.“

5.2 Asymmetrische Verschlüsselung (**4 Punkte**)

Stellen Sie sich für diese Aufgabe vor, dass Alice eine Nachricht an Bob schicken möchte. Um einige der wichtigen Sicherheitsfunktionen zu erzielen, nutzt sie asymmetrische Verschlüsselung. Beantworten Sie die folgenden Fragen, indem Sie die richtige Antwort ankreuzen.

Hierfür gelten folgende Abkürzungen:

- Alice‘ öffentlicher Schlüssel K1
- Alice‘ privater Schlüssel K2
- Bobs öffentlicher Schlüssel K3
- Bobs privater Schlüssel K4

5.2.1 Welchen Schlüssel verwendet Alice zur Verschlüsselung, wenn Sie sicherstellen möchte, dass Bob sicher sein kann, dass nur Alice ihm die Nachricht geschickt haben kann? (**2 Punkte**)

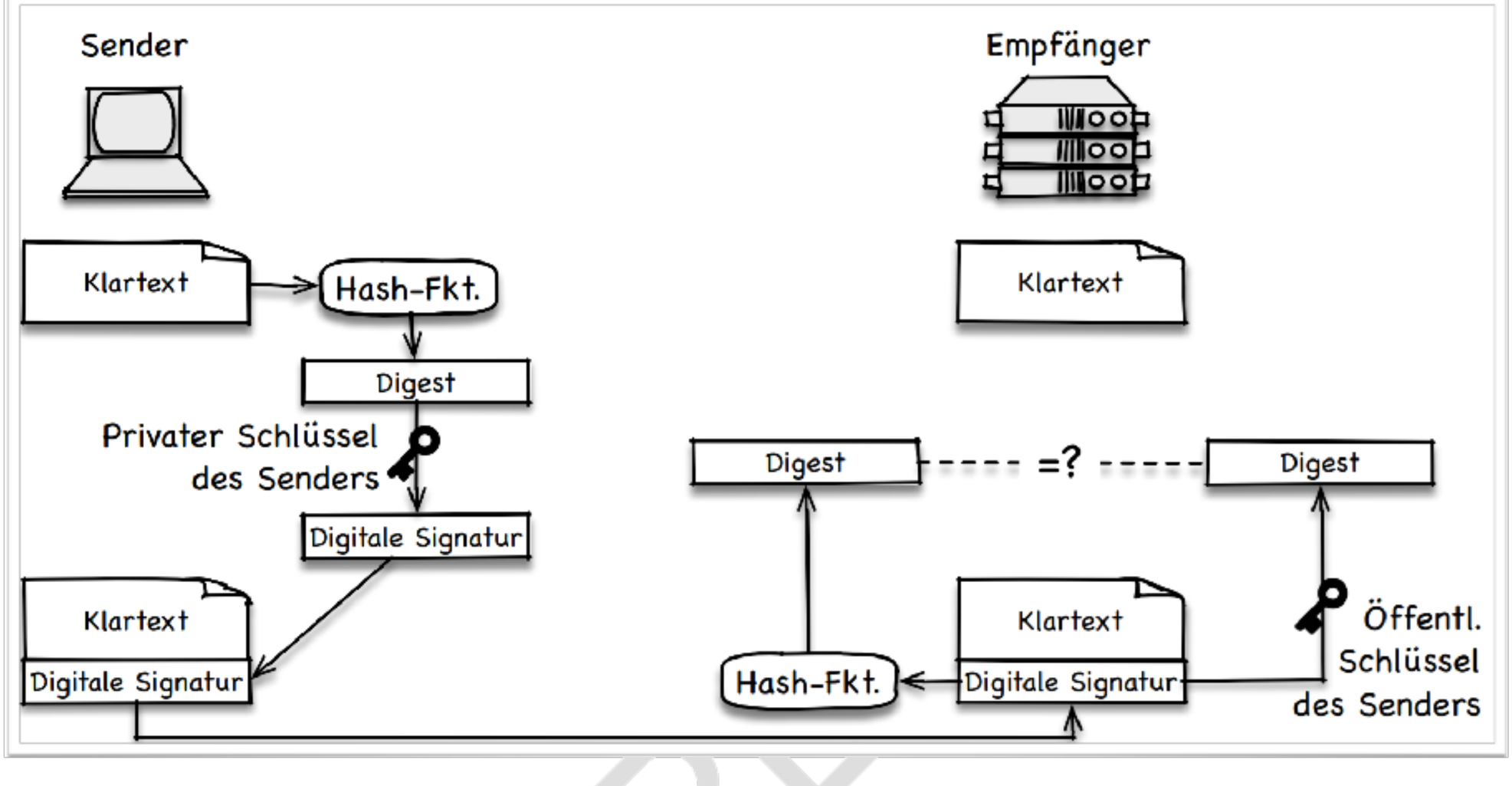
K1	K2	K3	K4
☐	☒	☐	☐

5.2.2 Mit welchem Schlüssel entschlüsselt Bob eine Nachricht von Alice, die diese ihm vor allem vertraulich zukommen ließ? (**2 Punkte**)

K1	K2	K3	K4
☐	☐	☐	☒

5.3 Digitale Signaturen (**5 Punkte**)

Gegeben sei folgendes Schaubild zur Erstellung einer sogenannten digitalen Signatur. Geben Sie an was in den einzelnen Schritten 1-5 passiert. Beschreibung in jeden Schritt die Eingabeparameter, die Operation und dessen Resultat und was mit dem Resultat gemacht wird.



1:

Der Klartext wird gehashed. Wir erhalten einen Digest, der eindeutig für den Klartext ist
Vom Digest lassen sich keinerlei Informationen zum ursprünglichen Klartext herleiten.

2:

Der Digest wird mit dem privaten Schlüssel des Senders verschlüsselt. Wir erhalten die digitale Signatur, die an den Klartext gehangen wird.

3:

Auf Empfängerseite wird der Klartext gehashed, um den Digest zu erhalten. Zu beachten ist, dass dieselbe Hashfunktion genommen werden muss, die der Sender verwendet hat. Das Verhandeln der Hashfunktion findet im TLS-Handshake statt.

4:

Die digitale Signatur wird auf Empfängerseite mit dem öffentlichen Schlüssel des Senders entschlüsselt. Wir erhalten den vom Sender erzeugten Digest.

5:

Abschließend wird geprüft, ob der selbsterzeugte Digest und der vom sendererzeugte Digest identisch sind. Ist dies der Fall, konnte die Integrität der Nachricht sichergestellt werden.