TECHNISCHE UNIVERSITÄT BERLIN

Klausurvorbereitungsaufgaben

Musterlösung

Webtechnologien

Wintersemester 2024/2025

Leonhardt Hollatz

L.hollatz@tu-berlin.de

Beachte:

- Die Inhalte/Aufgaben sind nur eine Auswahl an Themen und beinhalten nicht alles, was klausurrelevant ist!
    - Klausurrelevant ist alles, was in der Vorlesung und den Tutorien besprochen wurde!
- Die angegebenen Minutenzahlen sind ein ungefährer Richtwert und addieren sich insgesamt auf 100 Minuten.

Übersicht der Aufgaben:

1 Grundlagen *(8 Minuten)*

1.1 Ergänze die fehlenden Wörter im folgenden Text. *(5 Minuten)*

Als Erfinder des *World Wide Webs* gilt Sir Tim Berners Lee.
Kerntechnologien des vom ihm entwickelten Web 1.0 sind u.a. *HTTP*, *HTML* und DNS. Das Web 1.0 ist ein weltweites Netz von verlinkten, früher statischen Dokumenten. Es hat sich weiterentwickelt zum Web 2.0, bei dem das Web u.a. auch als *Ausführungsumgebung* für Anwendungen genutzt wird. Das Web 3.0 wird auch als *Web der Maschinen* bezeichnet, da es sich durch eine semantische Strukturierung der Daten und der Verlinkung unterschiedlicher Datenquellen auszeichnet.
Neben dem World Wide Web existieren noch weitere Anwendungen auf der Infrastruktur des *Internets*.

1.2 Kreuze an, ob die aufgeführten Aussagen wahr oder falsch sind. *(3 Minuten)*

Wahr	Falsch	Aussage
X		Eine Website enthält mehrere Webseiten. *(siehe Kap. 1, Folie 8.2 und 8.3)*
	X	Die Kommunikation zwischen Webbrowser und Webserver geschieht u.a. mittels der Hypertext Markup Language. *Die Hypertext Markup Language (HTML) ist für die Strukturierung von Webseiten entwickelt worden. Das Kommunikationsprotokoll für die Kommunikation zwischen Webbrowser und Webserver heißt Hypertext Transfer Protocol (HTTP). (siehe Kap. 1, Folie 9.2)*
	X	Ein Webservice wird vorrangig für menschliche Interaktionen programmiert. *Ein Webservice wird vorrangig für die Maschine-zu-Maschine Kommunikation programmiert. (siehe Kap. 1, Folie 9.3)*

2 HTML & CSS *(20 Minuten)*

2.1 HTML-Formulare *(4 Minuten)*

Gegeben ist eine HTML-Datei. Ergänze die Datei um die beschriebenen Elemente oder Attribute.

2.1.1 Der im Dokument verwendete Zeichensatz soll utf-8 sein. *(1 Minute)*
2.1.2 Das Formular soll mittels eines Buttons abgesendet werden können. *(1 Minute)*
2.1.3 Die Daten des Formulars sollen bei Bestätigung als Query Parameter an zielseite.html gesendet werden. *(1 Minute)*
2.1.4 E-Mail und Passwort müssen eingegeben werden, bevor das Formular bestätigt werden kann. *(0,5 Minuten)*
2.1.5 Beim (Neu-)laden der Seite, soll das E-Mail-Textfeld den Fokus erhalten. *(0,5 Minuten)*

```
<!DOCTYPE html>
<html lang="de">
<head>

    <meta charset="utf-8">

</head>
<body>
    <header><h1>Log-In</h1></header>

        <form   action="zielseite.html" method="get"    >

            <label for="mail">E-Mail</label>
            <input id="mail" type="email" name="mail"
                  required autofocus                >


            <br>
            <label for="password">Passwort</label>
            <input id="password" type="password" name="pw"
                 required                           >

            <br>

              <input type="submit">

        </form>
</body>
</html>
```

2.2 Beschreibe den Unterschied zwischen den HTML-Elementen <header> und <head>. *(2 Minuten)*

header ist ein Element zur Seitenstrukturierung. Es definiert einen Bereich als Kopf eines Seitenbereichs.

head gehört zum Grundaufbau/Grundgerüst jeder HTML-Datei. Es beinhaltet die Steuerelemente einer Webseite.

2.3 CSS *(8 Minuten)*

Gegeben ist der body-Teil einer HTML-Datei:

```
<body>
    <nav>
        <ul>
            <li>Start</li>
            <li>Kategorien
                <ol>
                    <li>Smart Home</li>
                    <li>Technik für <span>unterwegs</span></li>
                    <li>Garten</li>
                </ol>
            </li>
            <li>Neuheiten</li>
        </ul>
    </nav>
    <main id="app"></main>
    <footer>
        <p>Bitte gebe uns alle deine Daten damit wir dir ein
einzigartiges Benutzererlebnis kreieren können!</p>
    </footer>
</body>
```

2.3.1 Schreibe eine CSS-Regel, sodass nur das <main>-Element einen Außenabstand von 15 Pixel, sowie einen Innenabstand von 10 Pixel in alle Richtungen erhält. *(2 Minuten)*

```
main {                          ODER        #app {...}
    margin: 15px;
    padding: 10px;
}
```

2.3.2 Schreibe eine CSS-Regel, sodass nur der Listenpunkt „Neuheiten“ eine rote Schriftfarbe erhält. Es sollen keine anderen Elemente eingefärbt werden. *(2 Minuten)*

```
ul > li:last-child {
    color: red;
}
```

2.3.3 Schreibe eine CSS-Regel, sodass das <footer>-Element fest am unteren Rand des Bildschirms positioniert wird, ohne Rücksicht auf eventuell positionierte Elternelemente, und nicht mit dem Inhalt mitscrollt. *(2 Minuten)*

```
footer {
    position: fixed;
    bottom: 0px;
}
```

2.3.4 Im Folgenden sind zwei CSS-Regeln gegeben. Kreuze jeweils an, welche Elemente (mehrere möglich) im HTML durch den beschriebenen Selektor selektiert werden. Die HTML-Elemente sind jeweils durch ihren Inhalt beschrieben. Wenn kein Element selektiert wird, kreuze „Kein Element“ an. *(2 Minuten)*

```css
ul > li li:first-child {
    color: green;
}
```

- [ ] Start
- [ ] Kategorien
- [x] Smart Home
- [ ] Technik für unterwegs
- [ ] Garten
- [ ] Neuheiten
- [ ] Kein Element

```css
.span {
    color: aqua;
}
```

- [ ] Start
- [ ] Kategorien
- [ ] Smart Home
- [ ] Technik für unterwegs
- [ ] Garten
- [ ] Neuheiten
- [x] Kein Element

2.4 **Bootstrap** *(6 Minuten)*

Gegeben sind die Umbruchpunkte für das Bootstrap Rastersystem:

Breakpoint	**Class infix**	**Dimensions**
Extra small	*None*	<576px
Small	sm	≥576px
Medium	md	≥768px
Large	lg	≥992px
Extra large	xl	≥1200px
Extra extra large	xxl	≥1400px

Ergänze im untenstehenden HTML die Klassen der div-Elemente, sodass das Ergebnis wie folgt aussieht (in Klammern steht das Verhältnis der Elemente zueinander):

Für Bildschirme mit einer Breite ≥ 1200px:

div 1 (1/3)	div 2 (1/3)	div 3 (1/3)

Für Bildschirme mit einer Breite ≥ 768px und < 1200px:

div 1 (1/2)	div 2 (1/4)	div 3 (1/4)

Für Bildschirme mit einer Breite < 768px:

div 1 (1/2)	div 2 (1/2)
div 3 (1/1)	

```
<div class="container">
   <div class="row">


      <div class="col-6 col-md-6 col-xl-4">div 1</div>

      <div class="col-6 col-md-3 col-xl-4">div 2</div>

      <div class="col-12 col-md-3 col-xl-4">div 3</div>

   </div>
</div>
```

3 JavaScript *(40 Minuten)*

3.1 Listenfunktionen *(9 Minuten)*

Gegeben sind zwei Variablen: `products` zur Beschreibung von Produkten und `inventory` zu dessen Verfügbarkeiten. Schreibe zu den jeweiligen Aufgabenstellungen jeweils einen Ausdruck, wobei nur Array-Methoden erlaubt sind welche eine Callback-Funktion als Argument nehmen (z.B. map, filter, reduce).

```
let products = [
    {
        id: 0,
        name: "Fahrrad E-416",
        price: 199.99
    },
    {
        id: 1,
        name: "Helm",
        price: 50.49
    }
];
let inventory = [
    {
        id: 0,
        stock: 30
    },
    {
        id: 1,
        stock: 0
    }
];
```

3.1.1 Der Gesamtpreis aller angebotenen Produkte (egal ob verfügbar oder nicht). *(2 Minuten)*

```
products.reduce((acc, cur) => acc + cur.price, 0)
```

3.1.2 Alle Produkte welche nicht mehr verfügbar sind (stock = 0). *(3 Minuten)*

```
products.filter(e => inventory.find(e2 => e2.id == e.id).stock == 0)
```

3.1.3 Der Name des Produktes mit der höchsten Verfügbarkeit. *(4 Minuten)*

```
products.find(e => e.id == inventory.reduce((acc, cur) => acc.stock
> cur.stock ? acc : cur).id).name
```

3.2 Asynchrone Programmierung *(6 Minuten)*

3.2.1 Beschreibe, was eine mit `async` gekennzeichnete Funktion auszeichnet und was in diesem Zusammenhang das Schlüsselwort `await` bedeutet.
(2 Minuten)

async gibt an, dass die gekennzeichnete Funktion ein Promise als Rückgabeargument liefert.

await bedeutet, dass innerhalb einer async-Funktion auf die Finalisierung eines Promise gewartet wird.

3.2.2 async/await und Promises *(4 Minuten)*

Gegeben ist die asynchrone Funktion `fetchDataAsync`, welche Daten von einem lokal laufenden Webserver abfragt und auf der Konsole ausgibt.

```
async function fetchDataAsync() {
    let res = await fetch("http://localhost:3000/")
    let data = await res.json();
    console.log(data);
}
```

Schreibe dieselbe Geschäftslogik der Funktion `fetchDataAsync` mittels Promises im Funktionsrumpf der Funktion `fetchDataPromise`. Benutze für verwendete Variablen dieselben Bezeichnungen wie in der asynchronen Funktion `fetchDataAsync`.

```
function fetchDataPromise() {



fetch("http://localhost:3000/")
    .then(res => res.json())
    .then(data => console.log(data))









}
```

3.3 DOM-Manipulationen *(9 Minuten)*

Gegeben ist folgende HTML-Datei:

```
<!DOCTYPE html>
<html>
    <head>
        <title>DOM Manipulationen</title>
    </head>
    <body>
        <div id="app">

        </div>
        <script src="domManipulation.js"></script>
    </body>
</html>
```

Zudem sind Ausschnitte aus der Datei domManipulation.js gegeben. Ergänze die Funktion `addDOMElements`, sodass folgende Punkt erfüllt werden:

- Die Texte jedes Elements im Array `instruments` sollen als Listenelemente in einer ungeordneten Liste im div-Element der HTML-Datei eingefügt werden. *(6 Minuten)*
- Auf jedes Listenelement kann geklickt werden. Daraufhin soll der Text des Listenelements auf der Konsole ausgegeben werden. *(3 Minuten)*

```
const instruments = ['Violine', 'Viola', 'Violoncello',
'Kontrabass'];

addDOMElements();
function addDOMElements() {


    let ul = document.createElement('ul');

    for (let instrument of instruments) {
        let li = document.createElement('li');
        li.innerText = instrument;
        li.addEventListener('click', (evt) =>
console.log(evt.target.innerText));

        // ODER: li.addEventListener('click', () =>
console.log(instrument));

        ul.appendChild(li);
    }

    document.getElementById('app').appendChild(ul);

}
```

3.4 Vue *(16 Minuten)*

Gegeben sind Teile einer Vue-Anwendung, welche insgesamt aus vier Komponenten besteht. Die Komponente RoomOverview ist bereits vollständig gegeben und darf nicht geändert oder erweitert werden. Füge den anderen drei Komponenten die im Folgenden beschriebene Logik hinzu. Alle dazu notwendigen imports sind bereits gegeben. Beachte zudem, dass auch einige Ereignis- und Variablennamen bereits gegeben sind, die verwendet werden sollen.

App.vue

Die App-Komponente ist die hierarchisch höchste Komponente der Anwendung und beinhaltet bereits ein Array mit Raumobjekten (siehe Variable `rooms`). Es sollen die Komponenten Filter und RoomOverview eingebunden werden, sowie die Such- und Löschfunktionalität für Räume bereitgestellt werden. Es soll nach der Eigenschaft `fullRoomName` gesucht werden können. Die Sucheingabe wird von der Filter-Komponente bereitgestellt. Die Suche darf case sensitive sein. Die, der Zeichenkette nach, gefilterten Räume sollen der RoomOverview-Komponente übergeben werden. Die RoomOverview-Komponente stellt zudem auch die Information bereit, ob ein bestimmter Raum gelöscht werden soll.

Filter.vue

Die Filter-Komponente soll ein input-Element beinhalten, welches ein Suchfeld für Räume darstellt. Die eingegebene Zeichenkette in dieses Suchfeld soll der App-Komponente übergeben werden.

Room.vue

Die Room-Komponente soll die Darstellung der konkreten Räume in der Tabelle (siehe RoomOverview-Komponente) übernehmen. Es sollen nur die Eigenschaften `building` und `room` eines Raumes dargestellt werden. Zudem soll jeder in der Tabelle dargestellte Raum ein button anzeigen, der die Löschung dieses Raumes veranlasst. Das wird jeweils beim Klicken dieses Buttons der App-Komponente mitgeteilt.

Hinweis: Die einzelnen Komponenten sind mit horizontalen Linien voneinander abgegrenzt. Die Überschrift gibt den Namen der jeweiligen Komponente an.

App.vue

```
<script setup>
import { ref, reactive, computed } from 'vue';
import Filter from './components/Filter.vue';
import RoomOverview from './components/RoomOverview.vue';

const rooms = ref([{
  "id": 0,
  "building": "FH",
  "room": 314,
  "fullRoomName": "FH 314"
},
{
  "id": 1,
  "building": "EW",
  "room": 203,
  "fullRoomName": "EW 203"
}]);

const searchVal = ref("");

const filteredRooms = computed(() => {
  return rooms.value.filter(e =>
e.fullRoomName.includes(searchVal.value));
});

function deleteRoom(id) {
  delete rooms.value[id];
}

</script>
<template>



<Filter @search="searchVal = $event"></Filter>
<RoomOverview :rooms="filteredRooms"
@delete="deleteRoom($event)"></RoomOverview>



</template>
```

RoomOverview.vue

```
<template>
    <table>
        <thead>
            <tr>
                <th>Gebäude</th>
                <th>Raumnummer</th>
                <th>Einstellungen</th>
            </tr>
        </thead>
        <tbody>
            <Room v-for="room in rooms"
            :room="room"
            @delete="$emit('delete', $event)"></Room>
        </tbody>
    </table>
</template>
<script setup>
import Room from './Room.vue';
const props = defineProps(['rooms']);
</script>
```

Filter.vue

```
<template>
    <label for="search">Suche nach einem Raum: </label>


     <input type="text" id="search"
         @input="$emit('search', searchVal)"
         v-model="searchVal">



</template>

<script setup>
import { ref, reactive, computed } from 'vue';


const searchVal = ref("");
defineEmits(['search']);

</script>
```

Room.vue

```
<template>



     <tr>
        <td>{{ room.building }}</td>
        <td>{{ room.room }}</td>
        <td>
              <button
              @click="$emit('delete',room.id)">
                   Löschen
              </button>
        </td>
    </tr>









</template>

<script setup>
import { ref, reactive, computed } from 'vue';


const props = defineProps(['room']);









</script>
```

4 Server *(32 Minuten)*

4.1 Node.JS http-Server *(10 Minuten)*

Gegeben ist ein Teil eines Node.JS-Servers mittels des http-Moduls. Ergänze im Folgenden den Server unter Beachtung folgender Punkte:

- Es dürfen keine weiteren Module importiert werden
- Clients senden HTTP-Bodys nur im `x-www-form-urlencoded` Format
- Der Server unterstützt nur die HTTP POST-Methode
  - Anfragen vom Client werden aber immer mit einem passenden Statuscode beantwortet
- Bei einer passenden Anfrage setzt der Server den Body zusammen und fügt den Wert der `data`-Eigenschaft vom Body als Objekt mit `id` der `db` Variable hinzu
  - Das Objekt soll nur hinzugefügt werden, wenn die `data`-Eigenschaft nicht `undefined` ist
  - Die ids der Objekte in der `db` Variable sind fortlaufend
  - Die id des neu erstellten Objekts wird dem Client zurücksendet

```
const http = require('http');
const qs = require('querystring');
let body = '';
let db = [{id: 0, data: ''}];

http.createServer((req, res) => {



if (req.method == "POST") {
        body = '';
        req.on('data', chunk => {
            body += chunk;
        });
        req.on('end', () => {
            let newData = qs.parse(body);
            if (newData.data) {
                let newId = db[db.length-1].id + 1;
                db.push({id: newId, data: newData.data});
                res.writeHead(201);
                res.end(JSON.stringify({id: newId}));
            } else {
                res.writeHead(400);
                res.end();
            }
        });
```

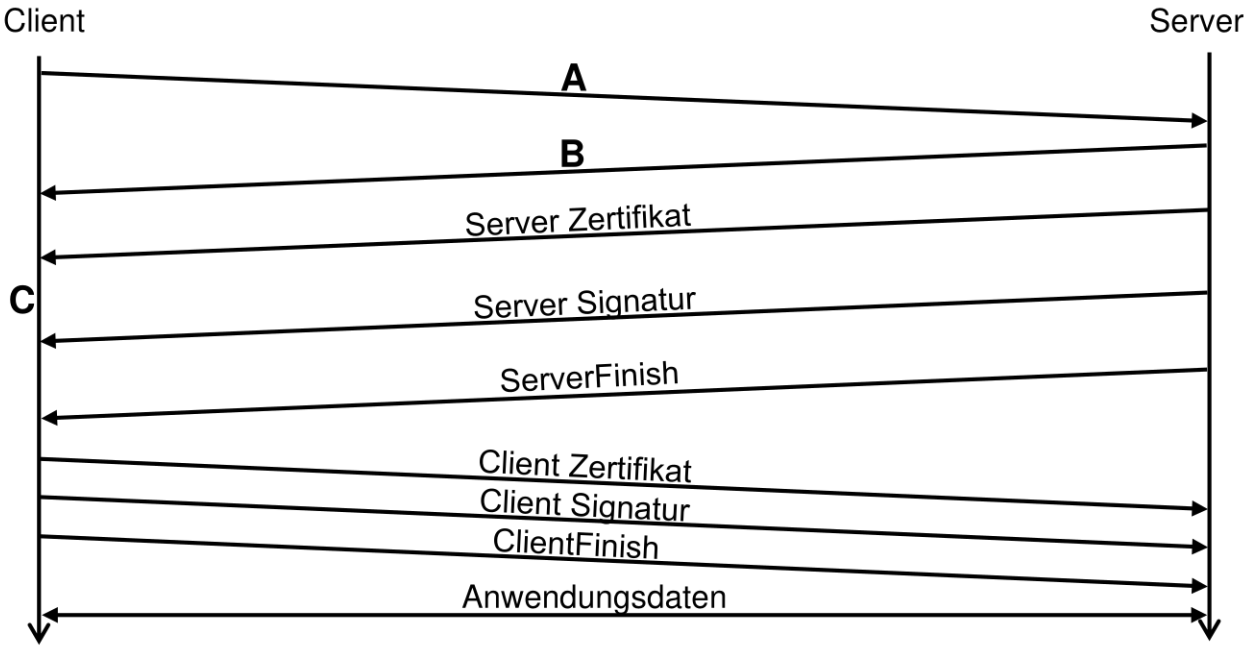
```
    } else {
        res.writeHead(405);
        res.end();
    }
```

```
}).listen(3000);
```

4.2 Sicherheit *(8 Minuten)*

4.2.1 TLS-Handshake *(4 Minuten)*

Gegeben ist eine grobe Darstellung des TLS-Handshakes in der Version 1.3. Beantworte die untenstehenden Fragen, basierend auf der Position des Buchstabens.



A und B: Wie heißen die versendeten Nachrichten und welche Inhalte werden ausgetauscht? *(2 Minuten)*

- *A: ClientHello*
- *B: ServerHello*
- *Austausch ephemeraler Schlüssel und Einigung auf Cipher Suite*

C: Was prüft der Client und was kann er damit sicherstellen? *(2 Minuten)*

- *Überprüfen des Server-Zertifikats*
- *Client kann damit sicherstellen, dass der öffentliche Schlüssel des Servers auch wirklich dem Server gehört*

4.2.2 Beschreibe kurz, wozu ephemerale Schlüssel im Kontext von TLS dienen. *(4 Minuten)*

Ephemerale Schlüssel sind kurzfristige Schlüssel und sorgen für Perfect Forward Secrecy. Damit wird sichergestellt, dass die Kommunikation sicher bleibt, selbst wenn die langfristigen privaten Schlüssel kompromittiert werden.

4.3 Express *(6 Minuten)*

Nutze das Express-Framework in Node.JS um einen Echo-Server auf Port 3005 zu implementieren. Der Express-Server soll den Body und die query-Parameter der Anfrage vom Client auslesen und diese im Antwort-Body mit dem Statuscode 200 als JSON-Objekt zurücksenden:

`{"body": <Body der Anfrage>, "query": <Query-Parameter der Anfrage>}`

Der Server soll alle HTTP-Methoden unterstützen sowie Anfrage-Bodys in den Formaten `JSON` und `x-www-form-urlencoded`. Alle notwendigen imports sind bereits gegeben.

```
const express = require('express');
const bodyParser = require('body-parser');



let app = express();

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: true}));


app.use((req, res) => {
    res.send({
        body: req.body,
        query: req.query
    });
})

app.listen(3005);
```

4.4 Verschiedenes *(8 Minuten)*

4.4.1 Liste die Ebenen des MVC-Patterns auf und gebe jeweils deren Hauptzweck an. *(3 Minuten)*

MVC steht für Model-View-Controller

Model: Datenstrukturen (z.B. Datenbank)

View: Nutzerinterface (z.B. Browser)

Controller: Geschäftslogik (z.B. Anwendungsserver)

4.4.2 Beschreibe kurz die Haupttechnologie die Sitzungsmanagement ermöglicht. *(2 Minuten)*

Sitzungsmanagement wird hauptsächlich durch Cookies realisiert. Diese transportieren eine SessionId oder den gesamten Sitzungszustand zwischen Client und Server.

4.4.3 Kreuze an, ob die aufgeführten Aussagen wahr oder falsch sind. *(3 Minuten)*

Wahr	Falsch	Aussage
	X	MongoDB ist eine dokumentenorientierte Datenbank, welche Dokumente nach festen Schemen ordnet und speichert. *MongoDB ist eine dokumentenorientierte Datenbank und ist schemafrei. D.h. Dokumente müssen nicht einem festen Schema folgen (Kap. 8, Folie 40.2)*
X		Eine MongoDB-Datenbank kann auf verschiedene Instanzen aufgeteilt werden, sodass u.a. horizontale Skalierbarkeit umgesetzt werden kann. *(siehe Kap. 8, Folien 48-49)*
	X	Passwörter in einer Datenbank werden ausschließlich verschlüsselt gespeichert, um sie sicher zu speichern. *Es reicht nicht Passwörter verschlüsselt zu speichern. Um sie sicher zu speichern, müssen Passwörter gehasht und gesalted werden (siehe Kap. 8, Folie 57.2)*