Klausurvorbereitungsaufgaben Webtechnologien

Wintersemester 2023/2024

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 Vorlesungsfolien, Tutorien und Hausaufgaben besprochen wurde!**
- **Diese Klausurvorbereitungsaufgaben sind ein freiwilliger zusätzlicher Service. Es können Fehler in der Aufgabenstellung sowie in der Musterlösung vorkommen.**
- **Die angegebene Minutenzahl ist ein ungefährer Richtwert und addieren sich insgesamt auf 100 Minuten.**

1 HTML und CSS *(25 Minuten)*

1.1 HTML-Formulare *(6 Minuten)*

Gegeben ist eine HTML-Datei mit dem HTML-Grundgerüst, zwei label-Elementen und einem input-Element vom Typ submit in einem Formular.

1.1.1 Ergänze das form-Element um Attribute, sodass beim Absenden des Formulars eine HTTP GET-Anfrage an zielseite.html versendet wird. *(2 Minuten)*

1.1.2 Ergänze die fehlenden HTML-Elemente im form-Element, sodass eine E-Mail-Adresse sowie ein Passwort eingegeben werden kann und diese mit dem jeweiligen label-Element verbunden sind. Alle Elemente sollen untereinander im Browser dargestellt werden. *(4 Minuten)*

```
<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>HTML-Aufgabe</title>
</head>
<body>
  <form                                                    >
```

```
        <label for="email">E-Mail</label>
```

```
        <label for="password">Passwort</label>
```

```
        <input type="submit" value="Absenden">
```

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

1.2 **CSS** *(13 Minuten)*

Gegeben ist eine CSS-Regel. Nutze im Folgenden ausschließlich CSS damit…

1.2.1 die CSS-Regel nur angewendet wird, wenn der Viewport <= 800 px ist. *(3 Minuten)*

```
label {
     font-size: smaller;
}
```

Gegeben ist folgender Ausschnitt aus einer HTML-Datei:

```
<div>
   <header>
       <nav>
           <ul>
               <li>Startseite</li>
               <li>News</li>
           </ul>
       </nav>
   </header>
   <main id="articles">
       <p>TUB baut neues <span>Mathegebäude</span></p>
       <p>Neue Tramlinie am Ernst-Reuter-Platz</p>
       <p>TUB und TUM schließen sich zusammen</p>
       <p>Neues aus dem Präsidium</p>
       <p>Das neue <span>Sommersemester</span></p>
   </main>
   <footer>
       <a href="#" class="footer">Impressum</a>
       <a href="#" class="footer">Datenschutz</a>
   </footer>
</div>
```

1.2.2 Schreibe eine CSS-Regel, sodass das footer-Element einen schwarzen durchgezogenen border um sich herum hat, welcher 2px dick ist. *(2 Minuten)*

1.2.3 Das Wort „Sommersemester“ im untersten p-Element soll rot gefärbt werden. Schreibe eine passende CSS-Regel dafür, die keine anderen Wörter färbt. *(2 Minuten)*

1.2.4 Das nav-Element soll einen Außenabstand nach unten von genau 35px haben. Schreibe eine CSS-Regel, die diese Anforderung erfüllt. *(2 Minuten)*

Gegeben sind zwei CSS-Regeln.

1.2.5 Kreuze jeweils an, welche Elemente im HTML durch den Selektor selektiert werden. Die HTML-Elemente sind jeweils durch ihren Inhalt beschrieben. Wenn kein Element selektiert wird, kreuze „Kein Element“ an. *(4 Minuten)*

```
p ~ p:nth-last-child(2) {
    background-color: yellow;
}
```

- ☐ TUB baut neues Mathegebäude
- ☐ Neue Tramlinie am Ernst-Reuter-Platz
- ☐ TUB und TUM schließen sich zusammen
- ☐ Neues aus dem Präsidium
- ☐ Das neue Sommersemester
- ☐ Kein Element

```
div > #articles p:first-of-type {
    background-color: green;
}
```

- ☐ TUB baut neues Mathegebäude
- ☐ Neue Tramlinie am Ernst-Reuter-Platz
- ☐ TUB und TUM schließen sich zusammen
- ☐ Neues aus dem Präsidium
- ☐ Das neue Sommersemester
- ☐ Kein Element

1.3 Bootstrap *(6 Minuten)*

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

Wahr	Falsch	Aussage
		Bootstrap kann als CSS-Framework gesehen werden, welche auf HTML und CSS basierende Gestaltungsvorlagen enthält.
		Bootstrap beinhaltet Styles für alle HTML-Elemente mit dem Ziel ihrer einheitlichen Darstellung in allen Web-Browsern und auf allen Plattformen.
		Eine Standard-Reihe im Bootstrap Rastersystem hat sechs Spalten.

1.3.2 Gegeben sind zwei HTML div-Elemente. Verwende zusätzliche geeignete HTML-Elemente sowie Klassen aus dem Bootstrap Rastersystem, sodass beide div-Elemente in gleicher Größe horizontal nebeneinander dargestellt werden. Das Rastersystem soll eingebettet sein in einem flexiblen Container. *(3 Minuten)*

```
        <div class="                                    ">1</div>

        <div class="                                    ">2</div>
```

2 JavaScript *(34 Minuten)*

2.1 Allgemeines *(11 Minuten)*

2.1.1 Nenne drei wichtige Merkmale von JavaScript. *(3 Minuten)*

2.1.2 Nenne einen der Hauptunterschiede von JavaScript zu TypeScript. *(1 Minute)*

2.1.3 Gegeben ist ein JavaScript-Programm. Kreuze an, welche Buchstaben (mehrere möglich) in der Konsole ausgegeben werden. *(2 Minuten)*

```
new Promise((resolve, reject) => {
    reject("A");
})
.then(wert => {console.log(wert); return "B";})
.catch(err => {console.log(err); return "C";})
.then(wert => {console.log(wert); return "D";})
.catch(wert => {console.log(wert); return "E";})
.finally(() => console.log("F"));
```

☐ A ☐ B ☐ C ☐ D ☐ E ☐ F

2.1.4 Gegeben sind zwei JavaScript-Programme. Beurteile, inwieweit sich die Programme voneinander unterscheiden. *(5 Minuten)*

Programm 1

```
1 async function sendRequest() {
2     let res = await
  fetch("localhost:5500");
3     let data = await res.json();
4     console.log(data);
5 }
```

Programm 2

```
1 function sendRequest() {
2     fetch("localhost:5500")
3     .then(res => res.json())
4     .then(data =>
  console.log(data));
5 }
```

2.2 Higher-Order-Functions *(13 Minuten)*

Gegeben ist ein JavaScript-Array, welches Wetterdaten beinhaltet. Beantworte im Folgenden die Fragen ausschließlich mit den Array-Eigenschaften sowie Array-Methoden welche eine Callback-Funktion als Argument nehmen (z.B. map, filter, reduce, sort). Schreibe dazu jeweils einen Ausdruck.

```
let weather = [
    {
        date_time: "2024-02-05T16:00Z",
        temp: 16.48,
        feels_like: 12.74,
        temp_min: 12.56,
        temp_max: 20.05,
    },
    {
        date_time: "2024-02-06T16:00Z",
        temp: 18.52,
        feels_like: 14.31,
        temp_min: 14.16,
        temp_max: 19.55,
    },
    {
        date_time: "2024-02-07T16:00Z",
        temp: 18.95,
        feels_like: 16.29,
        temp_min: 15.05,
        temp_max: 22.64,
    }
];
```

2.2.1 Alle Daten (date_time) in denen die minimale Temperatur (temp_min) größer als 15 war. *(2 Minuten)*

2.2.2 Das Datum (date_time) mit der höchsten aktuell gemessenen Temperatur (temp). *(3 Minuten)*

2.2.3 Der Durchschnitt der höchsten Temperaturen die am Tag gemessen wurden (temp_max). *(3 Minuten)*

2.2.4 Welchen Wert gibt der folgenden JavaScript Ausdruck zurück? *(1 Minute)*

```
weather.map(e => e.feels_like).sort((a,b) => a - b)[0];
```

- ☐ 12.74
- ☐ 14.31
- ☐ 16.29
- ☐ undefined

2.2.5 Gegeben ist ein JavaScript Ausdruck. Beschreibe grob, was dieser Ausdruck tut und welche Werte es im Allgemeinen zurückliefert. *(4 Minuten)*

```
1  weather.filter(e =>
2      weather.filter((e2, i, a) =>
3          a.map(e3 => e3.date_time).indexOf(e2.date_time) !== i
4      )
5      .some(e2 => e2.date_time == e.date_time)
6  );
```

2.3 DOM-Manipulation *(10 Minuten)*

Gegeben ist der body-Teil eines HTML-Dokuments. Die Aktivität für Sonntagmorgen soll in der Tabelle vervollständigt werden, mittels DOM-Manipulationen. Schreibe dafür in den Aufgaben JavaScript-Code, als wenn dieser in der Datei „domManipulation.js“ stehen würden.

```
<body>
    <table>
        <thead>
            <tr>
                <th>Tageszeit</th>
                <th>Samstag</th>
                <th>Sonntag</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>Morgens</td>
                <td>Spazieren</td>
            </tr>
            <tr>
                <td>Mittags</td>
                <td>Essen gehen</td>
                <td>Essen gehen</td>
            </tr>
            <tr>
                <td>Abends</td>
                <td>Film schauen</td>
                <td>Brettspiel spielen</td>
            </tr>
        </tbody>
    </table>
    <input type="text">
    <button id="submitBtn">Hinzufügen</button>

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

2.3.1 Füge dem button mit der id „submitBtn“ einen EventListener hinzu, sodass beim click-Event auf den button eine Funktion namens `sonntagMorgenAktivitaet` aufgerufen wird. *(2 Minuten)*

2.3.2 Die Funktion `sonntagMorgenAktivitaet` soll nun implementiert werden. Der Funktionsrumpf ist bereits gegeben. Beachte die folgenden Punkte bei der Implementierung: *(6 Minuten)*

- Der Wert, der im input-Element steht, soll ausgelesen werden und als Text in einem neuen td-Element eingefügt werden
  - **Hinweis:** Der Wert des input-Elements steht in dessen value-Attribut
- Das neue td-Element mit Text soll im DOM direkt hinter bzw. unter dem bestehenden td-Element mit dem Text „Spazieren“ eingefügt werden
- Der EventListener aus Aufgabe 2.3.1 soll wieder entfernt werden, sodass der button nur einmal geklickt werden kann

```
function sonntagMorgenAktivitaet() {









}
```

2.3.3 Erkläre, warum die JavaScript-Datei „domManipulation.js“ erst am Ende des body-Teils des HTML-Dokuments geladen wird. *(2 Minuten)*

3 Vue *(12 Minuten)*

Gegeben sei die folgende Vue.js-Applikation bestehend aus den Dateien „App.vue“, „OrchesterEinstellungen.vue“, „OrchesterVioline.vue“, „OrchesterViola.vue“, „OrchesterCello.vue“ und „OrchesterBass.vue“.

„App.vue“ und „OrchesterEinstellungen.vue“ müssen an den markierten Stellen vervollständigt werden. Siehe dazu die Aufgabenstellungen weiter unten. Für die Bearbeitung der Aufgabenstellung darf nur an den vorgegebenen Stellen Code geschrieben werden.

OrchesterVioline.vue

```
<template>
    <span id="violine">Violine</span>
</template>

<style scoped>
#violine {
    height: 50px;
    width: 50px;
    border: 2px solid red;
}
</style>
```

OrchesterViola.vue

```
<template>
    <span id="viola">Viola</span>
</template>

<style scoped>
#viola {
    height: 50px;
    width: 50px;
    border: 2px solid green;
}
</style>
```

OrchesterCello.vue

```
<template>
    <span id="cello">Cello</span>
</template>

<style scoped>
#cello {
    height: 50px;
    width: 50px;
    border: 2px solid blue;
}
</style>
```

OrchesterBass.vue

```
<template>
    <span id="bass">Bass</span>
</template>

<style scoped>
#bass {
    height: 50px;
    width: 50px;
    border: 2px solid orange;
}
</style>
```

OrchesterEinstellungen.vue

```
<template>
    <select  /* Aufgabe 3.2 */  >
        <option selected>Violine</option>
        <option>Viola</option>
        <option>Cello</option>
        <option>Bass</option>
    </select>
    <input id="eingabe" type="number"
    @input="sendData"
    min="0" :value="props.count[instrument]">
</template>

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

    const emit = defineEmits(['countVioline', 'countViola', 'countCello',
'countBass']);
    const props = defineProps(['count', 'countMax']);

    const instrument = ref('Violine');

    function sendData(evt) {
        /* Aufgabe 3.3 */

    }
</script>
```

App.vue

```
<template>
  <OrchesterEinstellungen
  @countVioline="count.Violine=$event" @countViola="count.Viola=$event"
  @countCello="count.Cello=$event" @countBass="count.Bass=$event"
  /* Aufgabe 3.1 */
  ></OrchesterEinstellungen>

<div>
    <OrchesterVioline id="violine" v-for="number in count.Violine"
:key="number"></OrchesterVioline>
    <OrchesterViola id="viola" v-for="number in count.Viola"
:key="number"></OrchesterViola>
    <OrchesterCello id="cello" v-for="number in count.Cello"
:key="number"></OrchesterCello>
    <OrchesterBass id="bass" v-for="number in count.Bass"
:key="number"></OrchesterBass>
  </div>
</template>

<script setup>
import OrchesterEinstellungen from './components/OrchesterEinstellungen.vue';
import OrchesterVioline from './components/OrchesterVioline.vue';
import OrchesterViola from './components/OrchesterViola.vue';
import OrchesterCello from './components/OrchesterCello.vue';
import OrchesterBass from './components/OrchesterBass.vue';
import { reactive } from 'vue';

const count = reactive({
  Violine: 5,
  Viola: 2,
  Cello: 4,
  Bass: 2
});
const countMax = reactive({
  Violine: 10,
  Viola: 4,
  Cello: 6,
  Bass: 3
});
</script>

<style scoped>

/* Aufgabe 3.4 */

</style>
```

3.1 Der nachfolgende Code zeigt einen Ausschnitt aus „App.vue“. Ergänze die fehlenden Vue-Direktiven, um der OrchesterEinstellungen-Komponente zwei properties mitzugeben. Entnehme die Namen der props aus „OrchesterEinstellungen.vue“. Es sollen jeweils die gleichnamigen Variablen aus „App.vue“ mitgegeben werden.
(2 Minuten)

```
<OrchesterEinstellungen
@countVioline="count.Violine=$event"
@countViola="count.Viola=$event"
@countCello="count.Cello=$event"
@countBass="count.Bass=$event"









></OrchesterEinstellungen>
```

3.2 Der nachfolgende Code zeigt einen Ausschnitt aus „OrchesterEinstellungen.vue“. Ergänze die fehlende Vue-Direktive, um die Eingabe mit der reactive state Variable „instrument“ zu verbinden und andersherum (sog. Two-way data binding). *(1 Minute)*

```
<select                                                    >
    <option selected>Violine</option>
    <option>Viola</option>
    <option>Cello</option>
    <option>Bass</option>
</select>
```

3.3 Der nachfolgende Code zeigt einen Ausschnitt aus „OrchesterEinstellungen.vue“. Die Funktion „sendData“ wird immer aufgerufen, wenn eine Eingabe im input-Feld gemacht wird. Sie soll die getätigte Eingabe im input-Feld an „App.vue“ übertragen. Die EventListener in App.vue sind bereits implementiert. Sie erwarten unter den angegebenen Namen jeweils eine Ganzzahl vom Typ Number. Übertrage die Daten jedoch nur, wenn die getätigte Eingabe kleiner als die in countMax mitgegebene Anzahl ist. Sollte die Zahl größer sein, setze die Maximalzahl (aus countMax) in das input-Feld (in das value-Attribut), sodass keine höheren Zahlen angezeigt werden können. *(6 Minuten)*

```
function sendData(evt) {




}
```

3.4 Der nachfolgende Code zeigt einen Ausschnitt aus „App.vue“. Das span-Element der Viola- und Cello-Komponente soll von der aktuellen Position aus, 80px nach unten verschoben werden. *(3 Minuten)*

```
<style scoped>




</style>
```

4 Server *(25 Minuten)*

4.1 NodeJS *(6 Minuten)*

Gegeben sind folgende zwei Server.

```
const http = require('http');
const url = require('url');

const instruments = [{_id: 0, name: 'Violine'}, {_id: 1,name: 'Viola'},
 {_id: 2,name: 'Violoncello'}, {_id: 3,name: 'Kontrabass'}];

http.createServer(function(req, res) {
    let query = url.parse(req.url, true).query;

    if (instruments.filter(e => e._id == query.id).length > 0) {
        res.writeHead(200);
        res.end(JSON.stringify(instruments[query.id]));
    } else if (query.id) {
        res.writeHead(404);
        res.end();
    } else {
        res.writeHead(200);
        res.end(JSON.stringify(instruments.map(e => e._id)));
    }
}).listen(3000);
```

```
const http = require('http');
const url = require('url');

const instruments = [{_id: 0,name: 'Gitarre'}, {_id: 1,name: 'Cembalo'},
{_id: 2,name: 'Ukulele'}, {_id: 3,name: 'Banjo'}];

http.createServer(function(req, res) {
    let query = url.parse(req.url, true).query;

    if (instruments.filter(e => e._id == query.id).length > 0) {
        res.writeHead(200);
        res.end(JSON.stringify(instruments[query.id]));
    } else if (query.id) {
        res.writeHead(404);
        res.end();
    } else {
        res.writeHead(200);
        res.end(JSON.stringify(instruments.map(e => e._id)));
    }
}).listen(3001);
```

Zudem ist ein Ausschnitt aus der Konfiguration eines Nginx-Servers gezeigt, welcher als Reverse Proxy agiert.

```
...
server {
        listen 8080;
        location /stringInstrument/ {
            proxy_pass http://localhost:3000;
        }
        location /pluckedInstrument/ {
            proxy_pass http://localhost:3001;
        }
    }
...
```

Gebe in den nachfolgenden Aufgaben an, welcher Statuscode und welcher Response-Body vom jeweiligen Server zurückgegeben wird, wenn eine HTTP-Anfrage mit der GET-Methode an die jeweilige URL gesendet wird. Wenn kein Body zurückgegeben wird, kreuze das Kästchen „Kein Body“ an.

4.1.1 http://localhost:8080/stringInstrument?id=0 *(2 Minuten)*

☐ Kein body

Status Code:

Body:

4.1.2 http://localhost:8080/pluckedInstrument *(2 Minuten)*

☐ Kein body

Status Code:

Body:

4.1.3 http://localhost:3001/pluckedInstrument?id=4 *(2 Minuten)*

☐ Kein body

Status Code:

Body:

4.2 REST *(6 Minuten)*

4.2.1 Beschreibe, was unter der Abkürzung REST verstanden wird und wofür es im Allgemeinen genutzt wird. *(2 Minuten)*

4.2.2 Kreuze an, ob die aufgeführten Aussagen wahr oder falsch sind. *(4 Minuten)*

Wahr	Falsch	Aussage
		Eine Ressource im Sinne von REST ist jede Art von kohärenter und sinnvoller Information, die über eine Adresse zur Verfügung gestellt werden kann.
		RESTful-Dienste basieren auf HTTP und verwenden HTTP-Methoden als Standardvokabular der API.
		Die HTTP-Methoden GET, POST und DELETE sind idempotent.
		Die HTTP-Methoden GET, OPTIONS und HEAD sind safe.

4.3 **Express** *(13 Minuten)*

Gegeben ist folgendes JavaScript-Programm, das eine HTTP GET-Anfrage an einen lokal laufenden Server schickt.

```
function request() {
    fetch('http://localhost:3000/train')
    .then(res => {
        if (res.status == 204) {
            request();
        } else {
            return "Daten erhalten";
        }
    })
    .then(data => console.log(data))
}
request();
```

Die Aufgabe ist es, den entsprechenden Server mithilfe des Express-Frameworks zu implementieren. Der folgende Servercode ist dabei bereits gegeben.

```
const express = require('express');
const server = express();
var savedResponse;
```

4.3.1 Implementiere eine Middleware die bei einer HTTP GET-Anfrage auf den Pfad `/train` aufgerufen wird und das Antwort-Objekt in der savedResponse Variable speichert. Nach 10000 Millisekunden wird die Anfrage mit dem Statuscode 204 beantwortet, wenn sich die Antwort noch in der Variable savedResponse befindet, ansonsten passiert nichts. *(4 Minuten)*
Hinweis: `setTimeout(functionRef, delay)`

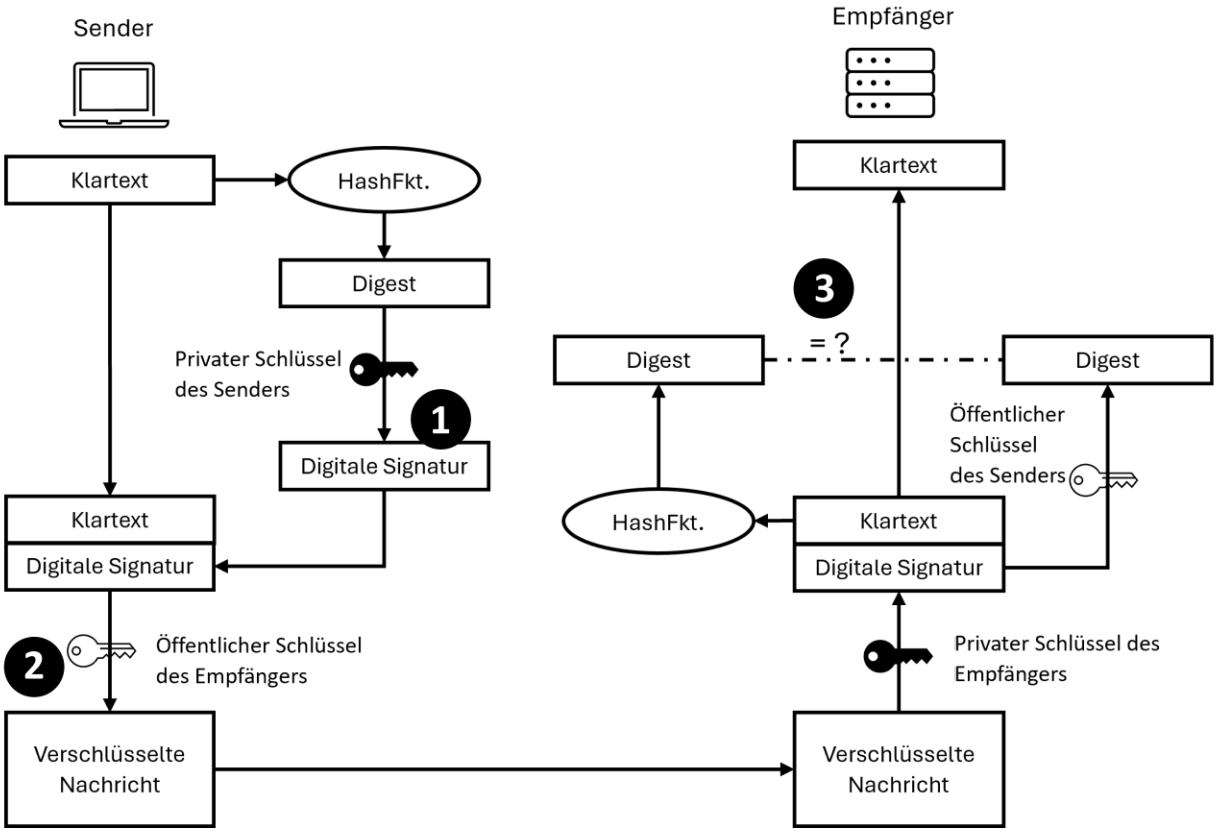
4.3.2 Implementiere nun eine Middleware die bei einer HTTP GET-Anfrage an den Pfad `/trainUpdate` aufgerufen wird. Wenn sich ein Antwort-Objekt in der savedResponse Variable befindet, wird die dort gespeicherte Antwort mit dem Statuscode 200 beantwortet und danach die Variable auf den Wert `undefined` gesetzt. Zuletzt wird die Anfrage an diese Middleware, unabhängig von der savedResponse Variable, mit dem Statuscode 200 beantwortet. *(4 Minuten)*

4.3.3 Die HTTP-Anfrage vom JavaScript-Programm (siehe oben) soll den Server auch tatsächlich erreichen. Ergänze den fehlenden Methodenaufruf im Server. *(1 Minute)*

4.3.4 Erkläre welche Anfragetechnik mit dem hier implementierten Server umgesetzt wird und welche Vorteile diese bringt.
Hinweis: Wenn die Middleware nicht implementiert wurde, nehme den Text als Beschreibung der Funktionalität. *(4 Minuten)*

5 Sicherheit *(4 Minuten)*

Gegeben sei folgendes Schaubild zum Ablauf einer sicheren Kommunikation mit asymmetrischer Verschlüsselung und digitaler Signatur.



Die drei mit schwarzem Kreis markierten Punkte markieren wichtige Stellen im Ablauf einer sicheren Kommunikation. Gebe für jeden der drei Punkte an, welche der drei wichtigen Sicherheitsfunktionen es erfüllt. Unterscheide beim Punkt der Authentifizierung zwischen Authentisierung und Authentifizierung. *(4 Minuten)*

1:

2:

3: