

## UE3 – Serverseitiges JavaScript, Node.js, JWT (25 Punkte)

Ziel dieses Übungsbeispiels ist es, serverseitige Technologien kennenzulernen und einzusetzen. Dazu soll das Beispiel aus Übung 2 um serverseitige Funktionalität erweitert und clientseitig entsprechend angepasst werden. Zusätzlich soll eine Zugriffskontrolle mit Hilfe von JSON Web Token realisiert werden.

Deadline der Abgabe via TUWEL:

**Sonntag, 21.05.2017 23:55 Uhr**

**Nur ein Gruppenmitglied muss die Lösung abgeben.**

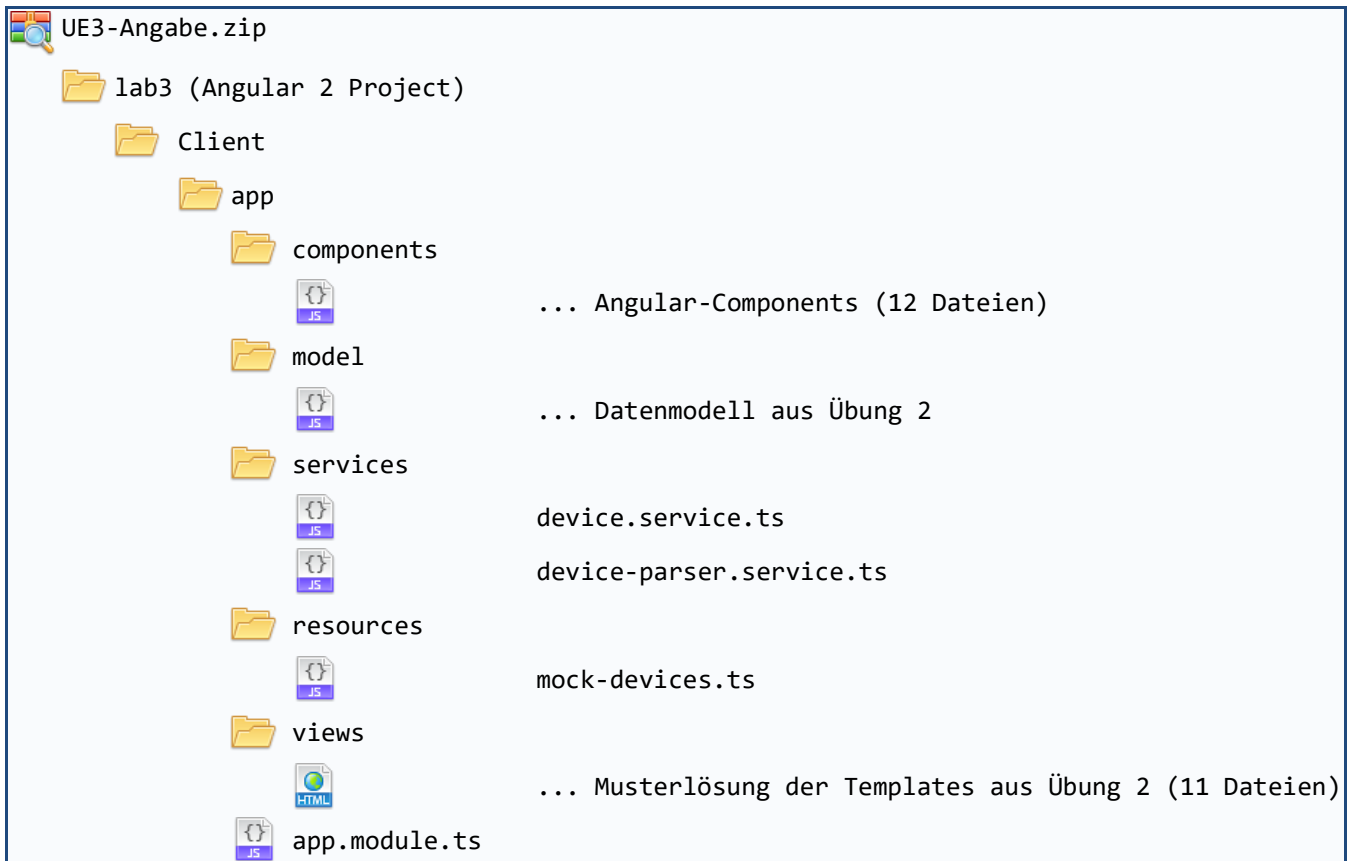
### BIG Smart Home

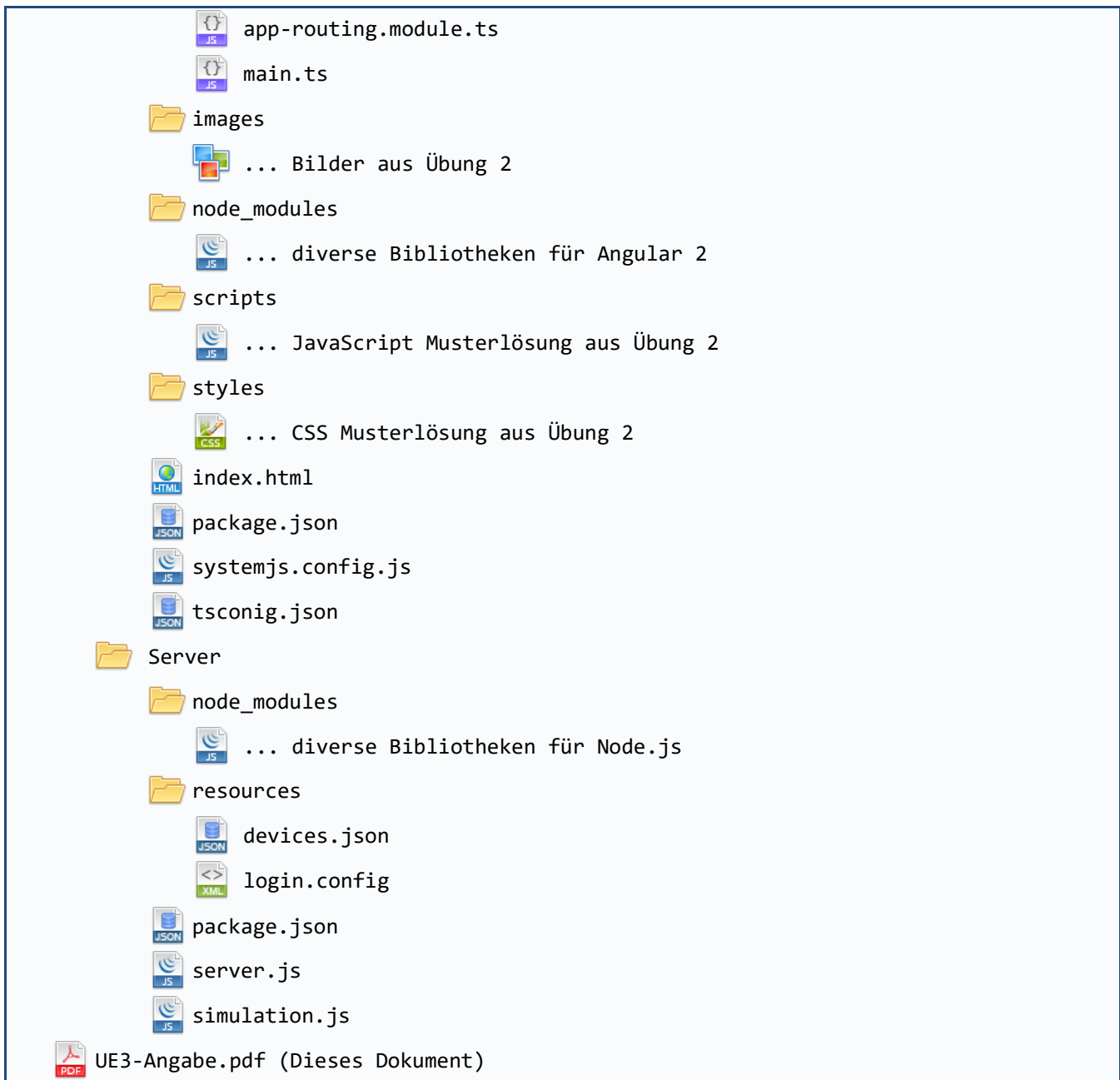
Bei BIG Smart Home handelt es sich um eine visuelle Schnittstelle zur Steuerung und Überwachung von Smart Devices eines vernetzten Haushalts. Autorisierte Personen dürfen dabei die aktuellen Zustände und Werte der Geräte einsehen, so wie auch Einfluss auf diese nehmen. Bei Neuanschaffungen können Geräte einfach ins System eingegliedert werden oder, wenn Geräte ausgemustert werden, ist auch eine Entfernung aus dem System möglich. Über simple Grafiken werden zur besseren Übersicht die aktuellen Zustände der Geräte dargestellt.

Für alle Geräte werden zusätzlich Diagramme angeboten, welche Aufschluss über den Verlauf der bisherigen Gerätezustände und Werte bieten.

### Angabe

Diese Angabe umfasst folgende Dateien:





Bitte beachten Sie, dass nicht alle der bereitgestellten Dateien bearbeitet werden sollen (siehe [Abgabemodalität](#)). Nehmen Sie jedoch keinesfalls Änderungen an den verwendeten Bibliotheken vor. Alle für diese Aufgabe benötigten Bibliotheken sind bereits enthalten und entsprechend eingebunden und bedürfen daher kein Einschreiten Ihrerseits.

Implementieren Sie eine REST-Schnittstelle mit Node.js und nutzen Sie diese um die Daten für Ihre Angular 2 Applikation zu laden und zu verändern. Verwenden Sie als Basis für Ihre Implementierung die bereitgestellten Dateien und erweitern Sie diese entsprechend den Anforderungen. Der Großteil aller Stellen, die von Ihnen angepasst werden müssen, wurden zur besseren Erkennung mit „*TODOs*“ markiert und dort erneut kurz (aber nicht vollständig) beschrieben. Achten Sie bitte darauf, dass bei manchen „*TODOs*“ gewisse Anweisungen zu beachten und einzuhalten sind. Um realistischeres Verhalten der Applikation zu erreichen, wird serverseitig eine Simulation bereitgestellt, welche periodisch Daten der Geräte verändert und diese an alle Clients überträgt.

## Anmerkungen zum Datenmodell

Das Datenmodell zur Darstellung der Geräte ist vorgegeben und soll nicht verändert werden. Die smarten Geräte werden innerhalb des Programms als Devices (definiert in *device.ts*) dargestellt. Ein Gerät kann dabei mehrere Interaktionsmöglichkeiten besitzen, im weiteren auch Steuerungselemente genannt (definiert in *controlUnit.ts*). Über diese können Einstellungen an den Geräten vorgenommen werden. Pro Gerät können mehrere Steuerungselemente existieren, wobei drei unterschiedliche Arten von Steuerungselementen vorhanden sind, welche für die dazugehörigen Datentypen verwendet werden. Folgende Datentypen sind vorhanden (definiert in *controlType.ts*): kontinuierliche Werte (bspw. für Temperatur), diskrete Werte/Enum (bspw. für Rollläden) und boolesche Werte (bspw. für Beleuchtung).

## Hauptanforderungen an Ihre Implementierung

- **REST-Schnittstelle:** Implementieren Sie mit Node.js unter der Verwendung des `express`<sup>1</sup> Modules eine REST-Schnittstelle. Über diese sollen alle Daten der Geräte abgerufen und verändert werden können. Stellen Sie daher folgende Funktionen über diese Schnittstelle zur Verfügung: Abrufen aller Geräte als Liste, Hinzufügen eines neuen Gerätes, Löschen eines vorhandenen Gerätes, Bearbeiten eines vorhandenen Gerätes (Verändern des Gerätezustandes und Anpassen des Anzeigenamens), Log-in und Log-out des Benutzers, Ändern des Passworts und das Abrufen des Serverstatus (Startdatum, fehlgeschlagene Log-ins). Binden Sie diese Schnittstelle in weiterer Folge auch vollständig in die clientseitige Applikation ein. Das heißt alle oben erwähnten Funktionen (Abrufen der Geräteliste, Hinzufügen eines Gerätes, ...) müssen in der clientseitigen Applikation über die REST-Schnittstelle durchgeführt werden. Verwenden Sie zum Ansprechen der REST-Schnittstelle die HTTP-Funktionalität<sup>2</sup> von Angular 2 - mit einer Ausnahme, siehe *Ändern des Gerätezustands*.
- **Websocket:** Um auf allen Clients denselben Datenzustand zu erreichen, implementieren Sie einen Websocket, der diese Aufgabe übernimmt. Verwenden Sie serverseitig dafür das `express-ws`<sup>3</sup> Modul. Immer wenn ein Gerät hinzugefügt, verändert oder entfernt wird, sollen alle verbundenen Clients über diese Aktion informiert werden. Dabei sollen nur so viele Daten wie benötigt übertragen werden. Verwenden Sie diesen Websocket weiters um periodisch simulierte Gerätedaten - die Simulation wird vorgegeben - an alle Clients zu übertragen.
- **Benutzer Log-in:** Lesen Sie serverseitig die bereitgestellte Datei mit den Benutzerdaten (*login.config*) ein und überprüfen Sie beim Log-in Versuch eines Benutzers, ob die erhaltenen Daten korrekt sind. Der Benutzer muss zur Identifikation den Benutzernamen und das entsprechende Passwort angeben. Falls die Daten nicht korrekt oder unvollständig übertragen wurden, geben Sie eine entsprechende Fehlermeldung an den Absender zurück und erhöhen Sie auch den Counter für fehlgeschlagene Log-ins.
- **Zugriffskontrolle:** Realisieren Sie mit Hilfe von JSON Web Tokens (JWT) und Angular 2 Guards eine Zugriffskontrolle sowohl für die serverseitige wie auch die clientseitige Funktion. Verwenden Sie JWT um den unbefugten Zugriff auf Ihre REST-Schnittstelle zu verhindern. Erstellen Sie dazu bei einem erfolgreichen Log-in ein neues Token, welches bei jedem weiteren Request im Header

---

<sup>1</sup> <http://expressjs.com/>

<sup>2</sup> <https://angular.io/docs/ts/latest/guide/server-communication.html>

<sup>3</sup> <https://www.npmjs.com/package/express-ws>

mitgeschickt werden muss, damit die entsprechende Funktion ausgeführt werden kann. Verwenden Sie dieses Token gemeinsam mit Angular 2 Guards<sup>4</sup> um einen unbefugten Zugriff auch clientseitig zu verhindern. Stellen Sie hier sicher, dass sich der User zuerst einloggen muss, bevor dieser Zugriff auf konkrete Daten erhält. Speichern Sie dieses Token im LocalStorage des Browsers, damit dieses auch nach einem Neuöffnen der Applikation verwendet werden kann. Senden Sie anschließend bei jedem weiteren Zugriff auf die Schnittstelle das gespeicherte Token im http-Header mit. Bei einem Log-out durch den User soll dieses Token in weiterer Folge als abgelaufen markiert werden.

- *Ändern des Gerätezustands:* Das Ändern des Gerätezustandes erfolgt weiterhin über die Detailseite des Gerätes. Um Änderungen des Zustands durchzuführen, soll die REST-Schnittstelle nicht mit Hilfe der http-Funktionalität von Angular 2 angesprochen werden. Die Schnittstelle soll in diesem Fall direkt über einen nativen JavaScript Ajax-Request aufgerufen werden. Setzen Sie dafür einen *XMLHttpRequest* auf Ihre Schnittstelle ab und reagieren Sie entsprechend auf die Statusänderungen. Diese Änderung am Gerätezustand muss wiederum an alle verbundenen Clients mittels Websocket übertragen werden. Dabei ist auch darauf zu achten, dass sich die SVG-Grafiken auf der Overview-Seite der verbundenen Clients entsprechend aktualisieren sollen.
- *Diagramme:* Um einen Verlauf der letzten Gerätezustände zu erhalten, speichern Sie alle bisherigen Zustände im SessionStorage Ihres Browsers. Das heißt, jede erhaltene Änderung an den Gerätezuständen soll entsprechend gespeichert und in weiterer Folge beim Anzeigen des dazugehörigen Diagramms geladen und dargestellt werden.
- *Geräte hinzufügen:* Implementieren Sie client- sowie serverseitig die Möglichkeit zum Hinzufügen eines Gerätes. Die grafische Darstellung zur Anlage eines Gerätes ist dabei bereits vorgegeben. Clientseitig müssen daher nur die Daten aus den entsprechenden Feldern ausgelesen und auf ihre Vollständigkeit - für jeden Steuerungstypen müssen alle dazugehörigen Datenfelder ausgefüllt sein - überprüft werden. Serverseitig soll die Neuanlage eines Gerätes entsprechend an alle verbundenen Clients übermittelt werden. Das neue Gerät soll jedoch nicht persistent, sondern auch weiterhin in einer JavaScript Variable gespeichert werden.
- *Passwort ändern:* Über die Optionsseite soll das Passwort für das Log-in geändert werden können. Sprechen Sie dafür erneut die REST-Schnittstelle an, welche die Korrektheit des alten Passworts und die Übereinstimmung des neuen Passworts und dessen Wiederholung sicherstellen soll. Falls diese Vorgaben erfüllt sind, soll das neue Passwort in das *login.config* File geschrieben werden und eine entsprechende Antwort an den Absender zurückgegeben werden. Falls die Eingabe nicht korrekt erfolgt ist oder ein anderer Fehler aufgetreten ist, soll der Absender ebenfalls eine passende Fehlermeldung bekommen. Beachten Sie dabei auch, dass der Benutzername statisch ist und daher nicht verändert werden soll.
- *Dynamische Inhalte:* Die Gerätedaten (beispielsweise Gerätename, aktueller Gerätezustand, der Gerätetyp, usw.) müssen dynamisch ausgegeben werden.
- *Speichern von Daten:* Da Sie in dieser Übung keine Datenbankbindung implementieren müssen, speichern Sie die Daten in JavaScript Variablen am Server. Das heißt, die Daten gehen verloren, sobald der Server neu gestartet wird. Dies gilt für fast alle Operationen, wie etwa Hinzufügen, Löschen und Verändern eines Gerätes. Ausgenommen von dieser Regelung sind nur die Benutzerdaten, welche persistent im *login.config* File gespeichert werden sollen.

---

<sup>4</sup> <https://angular.io/docs/ts/latest/guide/router.html#!#guards>

## Testdaten

Verwenden Sie zum Testen Ihrer Implementierung die von uns zur Verfügung gestellten Geräte in der *devices.json* Datei. Lesen Sie die Geräte aus dieser Datei beim Starten des Servers ein und speichern Sie diese in einer JavaScript Variable. Verändern Sie den Inhalt und die Struktur dieser Datei bitte nicht.

## Hinweise

### Validierung

Der von Angular 2 generierte Code muss nicht auf Validität geprüft werden, sollte jedoch so weit wie möglich valide gehalten werden und den Vorgaben von WAI Conformance Level Double-A entsprechen. Ausgenommen von dieser Regelung sind die Diagramme auf der Detailseite, welche bezüglich WAI-Tauglichkeit nicht berücksichtigt werden müssen.

### Entwicklungsumgebung

Es ist Ihnen freigestellt, welche Entwicklungsumgebung Sie für diese Übung verwenden. Beispielsweise bieten sich IntelliJ IDEA Ultimate<sup>5</sup> oder WebStorm<sup>6</sup> an, welche mit den entsprechenden Plugins eine sehr gute Basis für die Entwicklung dieser Aufgabe darstellen.

### Hilfsmittel

Um Ihre REST-Schnittstelle zu testen, können Sie eine Vielzahl an REST-Clients verwenden. Eine gute Wahl stellt dabei postman<sup>7</sup> dar, welcher alle in dieser Übung benötigten Funktionen beherrscht.

### npm Projekt

Das von uns zur Verfügung gestellte Projekt wird mittels npm<sup>8</sup> administriert, um eine leichtere Verwaltung der Bibliotheken zu ermöglichen. Gleichzeitig wird npm verwendet, um Ihre Angular 2 sowie auch die Node.js Applikation zu starten. Um npm verwenden zu können, installieren Sie am besten Node.js<sup>9</sup>, welches Sie in dieser Übung ebenfalls benötigen und eine entsprechende Installation für npm mitbringt. Mit dem Befehl „*npm start*“ können Sie in weiterer Folge Ihre clientseitige sowie auch die serverseitige Implementierung starten und ausführen, wobei dieser Vorgang für jeden Teil separat zu erfolgen hat. Sie können diesen Befehl entweder über die Kommandozeile in Ihrem Projektverzeichnis oder direkt aus Ihrer Entwicklungsumgebung ausführen. Stellen Sie vor der Abgabe dieser Aufgabe sicher, dass Ihre fertige Applikation weiterhin auf diese Art gestartet werden kann.

---

<sup>5</sup> <https://www.jetbrains.com/idea/>

<sup>6</sup> <https://www.jetbrains.com/webstorm/>

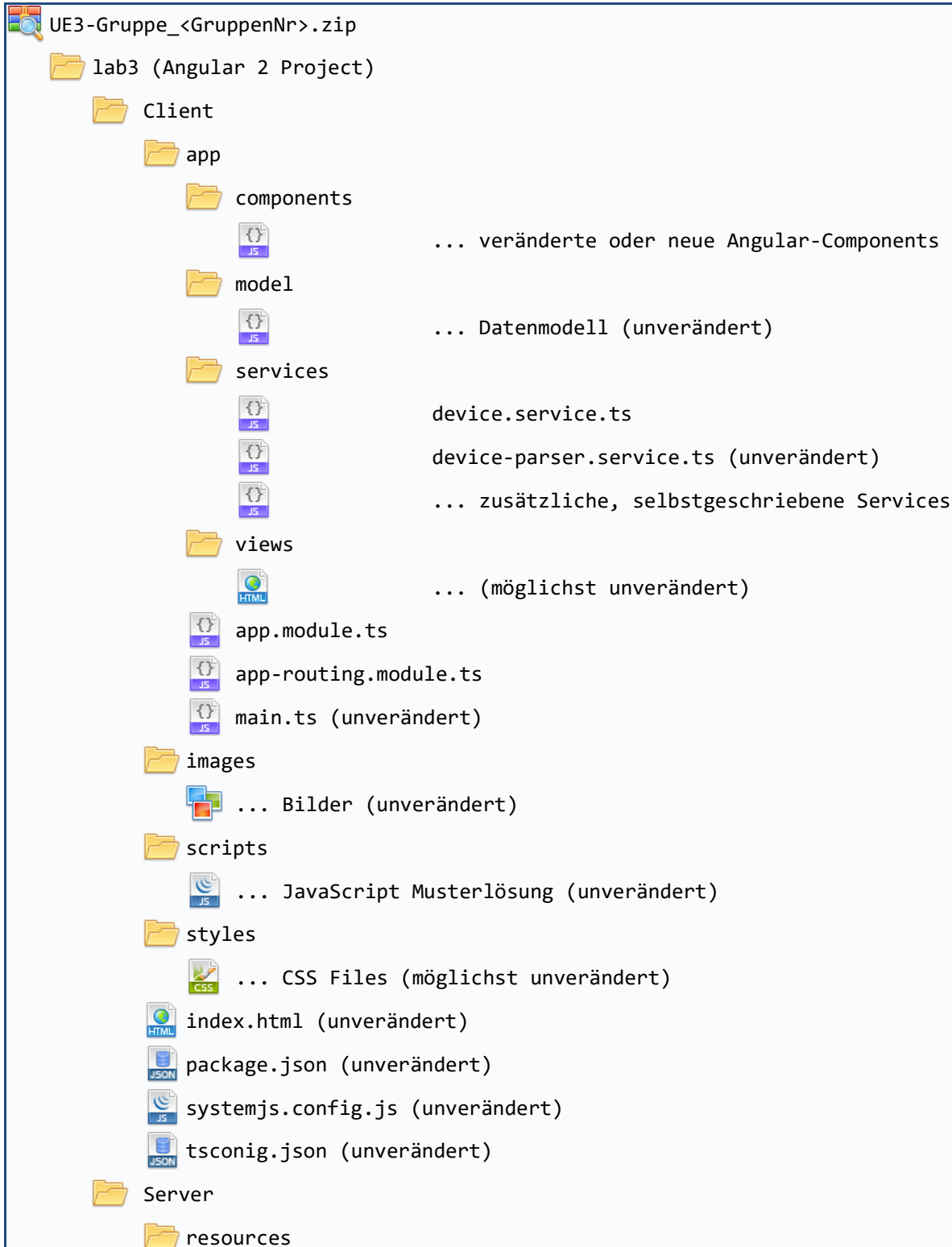
<sup>7</sup> <https://www.getpostman.com/>

<sup>8</sup> <https://www.npmjs.com/>

<sup>9</sup> <https://nodejs.org/en/>


## Abgabemodalität

Beachten Sie die allgemeinen Abgabemodalitäten des TUWEL-Kurses<sup>10</sup>. Zippen Sie Ihre Abgabe, sodass sie die folgende Struktur aufweist, beachten Sie dabei besonders, dass Sie die Bibliotheken im Verzeichnis `node_modules` **nicht** abgeben müssen:




<sup>10</sup> <https://tuwel.tuwien.ac.at/course/view.php?id=7423>

 devices.json (unverändert)

 login.config

 package.json (unverändert)

 server.js

 simulation.js (unverändert)

Alle Dateien müssen UTF-8 codiert sein!

ACHTUNG: Wird das Abgabeschema nicht eingehalten, so kann es zu Punkteabzügen kommen!