UE3 – Angular, Serverseitiges JavaScript (25 Punkte)

Ziel dieses Übungsbeispiels ist es, serverseitige Technologien kennenzulernen und die clientseitige Applikation daran anzubinden. Dazu soll das Beispiel aus Übung 2 als Grundlage dienen. Clientseitig wird dafür eine Angular-App erstellt und an einen ExpressJS-Server angebunden. Alle Funktionalitäten der Übung 2 sollten dabei weiter noch zur Verfügung stehen, wobei die Kontrollelemente für Gerätezustände in eigene Detailseiten ausgelagert werden sollen. Weiters sollen auf den Detailseiten die Zustandsänderungen geloggt werden und alle Logeinträge in Diagrammen visualisiert werden.

Deadline der Abgabe via TUWEL: **Sonntag, 20.05.2018 23:55 Uhr**

Nur ein Gruppenmitglied muss die Lösung abgeben!

Angabe

```
UE3-Angabe.zip
    client
        src
            app
                components      (von Ihnen zu erweiternde Komponenten)
                models          (von Ihnen zu erweiternde Model-Klassen)
                services        (von Ihnen zu erweiternde Services)
                validators      (von Ihnen zu erweiternde Validators)
                app.module.ts   (von Ihnen zu erweiterndes App-Modul)
            ...                 (mehrere Ressourcen-Verzeichnisse)
            ...                 (mehrere Dateien)
        resources               (HTML als Basis für Komponenten, Screenshots)
        .gitignore              (u.a. wegen node_modules)
        README.txt              (Informationen zu Angular CLI)
        ...                     (mehrere weitere Dateien)
    server
        resources
            devices.json        (verfügbare Gerätetypen)
            login.config        (Login-Daten für den Server)
        .gitignore              (u.a. wegen node_modules)
        server.js               (Server-Applikation)
        ...                     (mehrere weitere 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 keines Einschreitens Ihrerseits.

Node.js und npm

Für diese (und die nächste) Übung wird *Node.js* verwendet. Node.js ist eine JS-Laufzeitumgebung, um JS-Code direkt (ohne Browser) ausführen zu können. Node.js beinhaltet mit *npm* einen Package Manager, über den die Abhängigkeiten heruntergeladen und in weiterer Folge auch die erstellte Applikation gestartet werden kann.

Sie müssen daher zunächst sicherstellen, dass Sie eine (aktuelle) Version von Node.js inklusive npm installiert haben. Für die Übung ist die LTS-Version (derzeit 8.11.1) ausreichend, Sie können aber auch die aktuellste Version 9.11.1 verwenden. Die fertigen Pakete[1] von Node.js enthalten auch npm. Auf Linux-Systemen bietet sich alternativ eine Installation aus den Repositories von NodeSource[2] an. Beachten Sie, dass die inkludierten Repositories von Linux-Distributionen oft veraltete Node.js-Versionen enthalten, die dann nicht geeignet sind!

Nach der Installation müssen mit npm die Abhängigkeiten heruntergeladen werden. Dazu müssen Sie in den Unterverzeichnissen client und server jeweils einmal `npm install` ausführen. Dieser Prozess kann einige Minuten dauern und lädt insgesamt einige hundert MB herunter – führen Sie die Installation daher möglichst nicht mit getakteten Internetverbindungen aus.

Verändern Sie die Versionen der Abhängigkeiten nicht und fügen Sie das node_modules-Verzeichnis keinesfalls zum Git-Repository hinzu (gitignore in den Angaberessourcen ist entsprechend konfiguriert).

1 https://nodejs.org/en/download/

2 https://github.com/nodesource/distributions – es handelt sich jedoch um eine Fremdquelle!

Client (Angular 5)

Implementieren Sie eine auf Angular 5 basierende clientseitige Web-Applikation, welche die BIG Smart Production Plattform realisiert. Die Grundstruktur der Angular-App ist in der Angabe bereits vorgegeben, Sie müssen sie daher nicht selbst erstellen. Die Applikation verwendet *Angular CLI*[3], dessen Funktionalität Ihnen daher zur Verfügung steht – die wichtigsten Befehle finden Sie in der README-Datei. Für den Einstieg können außerdem einige Abschnitte von *Tutorial* und *Fundamentals* der *Angular Docs*[4] hilfreich sein.

Mittels `npm start` (im Verzeichnis `client`) können Sie einen Testserver unter localhost:4200 starten.

In Ihrer Angular-App sollen folgende Funktionen umgesetzt werden:

- Die Navigation zwischen den in weiterer Folge beschriebenen Seiten soll mit dem **Routing-Modul**[5] von Angular umgesetzt werden.
- Mit Ausnahme der Login-Seite soll ein **Guard**[6] den Zugriff auf eingeloggte User beschränken. Setzen Sie dazu nach einem erfolgreichen Login eine entsprechende Variable bzw. löschen Sie sie beim Logout. Der Login-Status soll erhalten bleiben, wenn die Applikation neu geladen wird.
- Wandeln Sie die bisher statische **Login**-Seite in eine Angular-Komponente um. Beim Abschicken des Formulars werden die eingegebenen Login-Daten an den Server geschickt. Wenn die Login-Daten gültig waren erfolgt eine Weiterleitung zur Übersichtsseite, ansonsten wird eine Fehlermeldung angezeigt.
- Wandeln Sie die bisher statische **Optionen**-Seite in eine Angular-Komponente um. Die Validierung, ob das neue Passwort und die Passwort-Wiederholung übereinstimmen, soll im Client erfolgen. Außerdem soll auch weiterhin überprüft werden, ob das neue Passwort den Vorgaben entspricht (siehe Übung 1). Beim Abschicken des Formulars werden das alte und neue Passwort an den Server übertragen. Anschließend soll eine Meldung angezeigt werden, ob das Ändern des Passworts erfolgreich war oder nicht (beispielsweise, weil das alte Passwort ungültig war).
- Für die **Übersicht**-Seite soll eine Angular-Komponente erstellt werden, die den bereits vorhanden JS-Code weiterverwendet:
  - Die verfügbaren **Gerätetypen** (Sidebar) sollen vom Server geladen werden. Dabei können Sie für jeden Listeneintrag die zur Verfügung gestellte Komponente `available-device` verwenden, in der die aus Übung 2 bekannten Draggables aktiviert werden. Sie müssen dafür (in einer Schleife) den folgenden Code verwenden:
    ```
    <li app-available-device [data]="availableDevice"></li>
    ```
  - Für die **Diagrammfläche** können Sie die zur Verfügung gestellte Komponente `diagram` verwenden, in der das aus Übung 2 bekannte Diagram-Objekt initialisiert wird. Sie müssen der

3 https://github.com/angular/angular-cli/wiki

4 https://angular.io/docs

5 https://angular.io/guide/router

6 https://angular.io/guide/router#milestone-5-route-guards

Komponente als Parameter eine Referenz zum Arrow-Button der Sidebar übergeben (im bisherigen Code das Element mit der ID `arrow-sidebar-add`):
`<app-diagram [arrowAdd]="arrowAddRef"></app-diagram>`

- Aus dem Diagramm-Code der vorherigen Übung werden nun die **Callbacks** in der Service-Klasse `DiagramService` aufgerufen. Hier müssen Sie die erstellten Geräte und Pfeile in einer Liste speichern, sodass die Diagramm-Inhalte bei der Navigation zwischen den Unterseiten (z.B. Übersicht → Optionen → Übersicht) erhalten bleiben. Details finden Sie in Kommentaren im Code.

  Hinweis: Eine persistierende Speicherung muss derzeit nicht erfolgen, d.h. beim Reload der Angular-App müssen keine Geräte im Diagramm vorhanden sein.

- Für die **Counter** oberhalb des Diagramms wird der Code aus Übung 2 nicht weiterverwendet, stattdessen sollen diese Teile in Angular umgesetzt werden. Die Werte sollen auf Basis der Callbacks in `DiagramService` angepasst werden.

- Die **Controls** sind auf der Übersicht-Seite gar nicht mehr vorhanden und werden durch die Detail-Seite ersetzt (siehe unten).

- **Detail-Seiten:** Wie zu Beginn erwähnt, sollen die Kontrollelemente für die Gerätezustände innerhalb des Diagramms in eigenen Detail-Seiten ausgelagert werden. Jedes Gerät, das sich innerhalb des Diagramms befindet, soll dabei eine eigene Detail-Seite erhalten, in welcher nur der Zustand für das eine ausgewählte Gerät angezeigt und verändert werden kann.

  Hinweis: Sehen Sie sich für die Detailseite insbesondere das Property `Device.control` an!

  - Die Detailseite kann über Doppelklick auf das jeweilige Gerät UND über den Button „Detailseite“ im Kontext-Menü des Gerätes aufgerufen werden. Diese Events existieren bereits und rufen die Methode `DiagramService.onDeviceDetails()` auf. In dieser Methode ist nun von Ihnen eine Weiterleitung zur Detailseite durch Angular Routing zu implementieren. Übergeben Sie dafür den Index des Gerätes in der URL.

  - Da die Zustände der Geräte unterschiedliche Typen (*enumeration*, *continuous*, *boolean*) aufweisen, sind drei unterschiedliche Detailseiten zu erstellen (siehe Screenshots im Angabeordner). Für jeden Datentyp existiert ebenfalls ein dazugehöriges Diagramm. Für kontinuierliche Änderungen in den Zuständen (wie es bei der Maschine, dem Mülllager und dem Endlager der Fall ist) wird ein Line-Chart verwendet. Für Enumerations (Maschine) werden Pie-Charts verwendet. Für Boolean-Änderungen wird eine spezielle Form des Pie-Charts verwendet, ein Doughnut-Chart.

    Für die Diagramme ist das sehr einfache *ngx-charts* Plugin[7] zu verwenden. Achten Sie hierbei auf die Datenstruktur der Daten, die im Chart angezeigt werden, die für die jeweiligen Chart-Typen verwendet werden müssen.

  - Wie in Übung 2 sollen die Zustände auf den einzelnen Detailseiten anhand des Datentyps verändert werden können. Erstellen Sie daher Angular-Komponenten für die Detailseiten,

7 https://swimlane.gitbooks.io/ngx-charts/content

die eine Schnittstelle für die Zustands-Formularfelder zur Verfügung stellen und aktualisieren Sie nach jeder Zustandsänderung sowohl die Logliste (siehe Log-Einträge in den Screenshots) also auch das jeweilige Diagramm. Alle Werteänderungen sollen sich auf die ganze Applikation auswirken (das jeweilige Gerät im Diagramm auf der Overview-Seite soll also ebenfalls angepasst werden). Nach einem Browser-Refresh der Applikation gehen sämtliche Log-Einträge und damit auch alle Einträge in den Diagrammen verloren.

- Der HTML Code ist bereits für die Boolean-Detailseite vorgegeben. Im Code finden Sie ein TODO, wo das Chart-Element eingefügt werden muss sowie den Log-Container und das Formular für die Boolean-Zustandsänderung. Achten Sie darauf, dass bei den Detailseiten für kontinuierliche Veränderungen nur Werte erlaubt sind, die durch *min* und *max* als Unter- und Obergrenze für das jeweilige Gerät zulässig sind.

Weitere Vorgaben:

Für alle Formulare soll der Submit-Button nun tatsächlich (nicht nur optisch) deaktiviert werden, solange das Formular ungültige Daten enthält. Bei fehlerhaften Formulareingaben sollen entsprechende Hinweise angezeigt werden (die exakte Gestaltung bleibt Ihnen überlassen, solange hilfreiche Informationen gegeben werden).

Erstellen Sie die Angular-Komponenten jeweils in einem Unterverzeichnis von components und verwenden Sie als Basis für den HTML-Code die zur Verfügung gestellten HTML-Ressourcen (= Musterlösung für Übung 2). Trennen Sie View (Angular-Komponenten) und Business-Logic (Service-Layer). Außerdem ist es sinnvoll (aber hier nicht verpflichtend), einen eigenen Layer für den Zugriff auf REST-Schnittstellen zu verwenden. Verwenden Sie die in Angular zur Verfügung stehende *Dependency Injection*[8], um aus den Komponenten auf Ihre Service-Klassen zuzugreifen.

Versuchen Sie Code-Duplizierung zu vermeiden, indem Sie beispielsweise eigene Komponenten für mehrfach vorhandene Blöcke erstellen.

Im Code-Gerüst sind bereits einige Klassen und Komponenten vorgegeben:

- `models/arrow.model.ts`, `models/device.model.ts`, `models/diagram.model.ts`

  Diese Dateien enthalten Deklarationen, sodass die JS-Models aus dem *global namespace* auch in TypeScript aufgerufen werden können. Die tatsächlichen Definitionen befinden sich in den bereits aus Übung 2 bekannten JS-Dateien. Diese Dateien werden ohne Benennung der Symbole importiert: `import 'abc/models/xyz.model.ts';`

- `models/authentication.request.ts`, `models/password.change.request.ts`, `models/control.ts`, `models/device.available.ts`

  Diese Dateien enthalten Interfaces, mit denen die Struktur verschiedener Objekte beschrieben wird. Beachten Sie, dass ein TypeScript-Objekt implizit jedes Interface implementiert, sobald die Schnittstellen übereinstimmen (siehe Foliensatz M5, Slide 15 oben) – Sie müssen daher nicht unbedingt explizit implementierende Klassen dazu schreiben!

- `services/diagram.service.ts`

8 https://angular.io/guide/architecture-services

Diese Klasse enthält (von Ihnen zu implementierende) Callbacks für den Legacy-JS-Code. Beachten Sie die Kommentare in der Datei selbst!

Anmerkung: Eines der Design-Goals von TypeScript ist die einfache, schrittweise Transformation von Legacy-JS-Applikationen. Daher muss am bestehenden Code nur wenig geändert werden. Um Ihnen den Umstieg zu erleichtern ist die Einbindung des Legacy-Code bereits vorgegeben – versuchen Sie jedoch dennoch, die Funktionsweise dieser Klassen zu verstehen!

Server (ExpressJS)

Mittels ExpressJS[9] sollen Sie nun einen einfachen Server implementieren, der einige REST-Schnittstellen bereitstellt. Der Server wird vollständig in der Datei `server.js` implementiert. Mittels `npm start` (im Verzeichnis `server`) können Sie die Server-Applikation starten.

In Ihrer Server-Applikation sollen folgende Funktionen umgesetzt werden:

- Lesen Sie die Datei `resources/login.config` aus und legen Sie die enthaltenen Login-Daten in einer globalen Variable ab. Verwenden Sie dafür die Filesystem-Methoden[10] von Node.js. Beachten Sie, dass die Datei betriebssystembedingt verschiedene Line-Endings haben kann (CRLF oder LF) – Ihre Lösung soll dynamisch mit beiden Varianten umgehen können.
- Lesen Sie die Datei `resources/devices.json` aus und legen Sie die enthaltenen Daten in einer globalen Variable ab.
- Implementieren Sie eine REST-Schnittstelle für die Authentifizierung. Diese Schnittstelle bekommt die Logindaten übergeben (siehe Interface AuthenticationRequest vom Client) und soll sie mit dem serverseitigen User abgleichen.
- Implementieren Sie eine REST-Schnittstelle für das Ändern des Passworts. Diese Schnittstelle bekommt das alte und das neue Passwort übergeben. Wenn das alte Passwort korrekt ist, soll sowohl es in der globalen User-Variable aktualisiert werden. Außerdem soll das neue Passwort in die Config-Datei geschrieben werden, sodass beim nächsten Server-Neustart das neue Passwort verwendet wird.

  Anmerkung: Eingabedaten müssen aus Sicherheitsgründen immer serverseitig validiert werden (eine clientseitige Validierung ist immer nur eine Hilfestellung für die User). Sie müssen in diesem Fall jedoch keine Validierung des neu gesetzten Passworts implementieren.
- Implementieren Sie eine REST-Schnittstelle für das Laden der verfügbaren Gerätetypen. Die Schnittstelle soll die Liste wie in devices.json zurückgeben.

Verwenden Sie für alle REST-Schnittstellen passende Namen und die korrekten HTTP-Methoden. Verwenden Sie für Fehlerfälle außerdem die passenden Statuscodes[11]. Im Rahmen dieser Übung ist eine Authentifizierung vor dem Aufruf der REST Schnittstellen, die nicht den Login betreffen, nicht nötig. Nach dem Login bestimmt also nur die Client-Applikation, ob auf die REST Schnittstellen am Server zugegriffen werden darf oder nicht.

9 http://expressjs.com

10 https://nodejs.org/api/fs.html

11 https://de.wikipedia.org/wiki/HTTP-Statuscode

Hinweise

Ausführen der Applikation

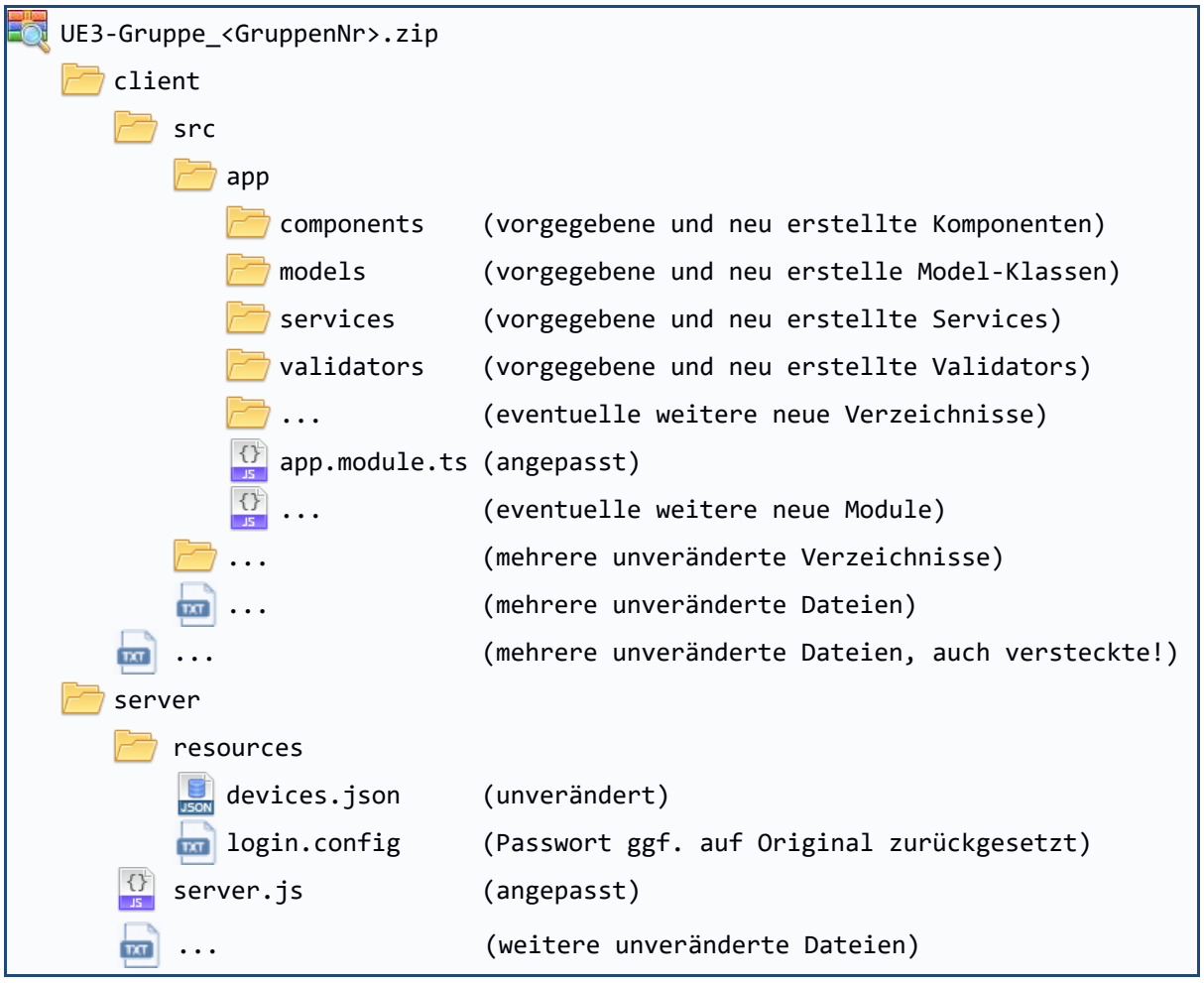
Sowohl Client- als auch Server-Applikation müssen mittels npm start ausgeführt werden können. Verändern Sie die mitgelieferten Bibliotheken nicht (auch nicht deren Versionen).

Validierung

Die Validität des von Angular generierten Codes muss nicht überprüft werden. Achten Sie aber darauf, möglichst validen Code zu erstellen und behalten Sie die Accessibility-Features der letzten Übung bei.

Abgabemodalität

Beachten Sie die allgemeinen Abgabemodalitäten des TUWEL-Kurses. Zippen Sie Ihre Abgabe, sodass sie die folgende Struktur aufweist:



- Alle Dateien müssen UTF-8 codiert sein!
- Nur ein Gruppenmitglied muss die Lösung abgeben!
- Geben Sie keinesfalls `node_modules` ab!
- Wird das Abgabeschema bzw. der Name der Zip-Datei (UE3-Gruppe_<GruppenNr>.zip) nicht eingehalten, kommt es zu Punkteabzügen!