Dienstplan

About

Diese Anwendung ist ein eigenständiger Webserver der zusammen mit einer MySQL-Datenbank betrieben wird.

Bei der ersten Ausführung wird der Server sich mit dem eingestellten Datenbank-Server verbinden, die Datenbank auswählen und wenn nicht vorhanden selbst erstellen, genauso wie jede Tabelle die benötigt wird.

Außerdem werden zum Testen Mitarbeiter direkt hinzugefügt, damit man sich gleich einloggen kann.
Für den Adminstratorzugang:

- Username: admin
- Passwort: admin

Das Webpanel ist über den jeweiligen Port verfügbar, standardmäßig: http://localhost:8080

Es empfiehlt sich allerdings für externe Verbindungen https mithilfe eines Proxy einzurichten (wie z.b. nginx)

Decisions

Wir haben in diesem Prototypen die Anwendungsfälle Login, Mitarbeiteransicht und Mitarbeiter hinzufügen umgesetzt, da ein einzelner Anwendungsfall bei unserem Beispiel zu trivial wäre und alleine auch nicht umsetzbar.

Für die Implementation haben wir uns für TypeScript in einer Node.js Umgebung entschieden, da es sowohl als static fileserver, als auch als API Server funktioniert und eine gute Modellierung unseres Szenarios ermöglicht.

Die meisten Abweichungen von der Planung sind durch vorherige Fehler in der Planungsphase oder Implementationslimitation der gewählten Sprache entstanden (z.B. die Node.js Umgebung erlaubt keine Circular Dependencies).

Wichtige Änderungen die daraus resultierten:

- Wir haben die Address-Klasse entfernt und deren Felder der Employee-Klasse zugewiesen, da durch die 1:1-Beziehung der Vorteil einer seperaten Klasse dafür nicht mehr gegeben ist.
- In der Employee- und Event-Klasse haben wir die statische add() und nicht statischen edit() und delete() Methoden entfernt. Datenbank-Interaktionen sollten möglichst nur über den DatabaseController laufen. Die Methoden werden im Prinzip jetzt aber durch die REST-API trotzdem dargestellt, nur über HTTP.

- `DatabaseController.getEmployeeByAnyInfo(info): Employee[]` ist jetzt `DatabaseController.getEmployees(where?: {key: string, value: string}, limit?: number): Employee[]`
  Da die spezifizierte Aufrufsignatur ungenau war und somit genauer und allgemeiner zu verwenden ist.

- Die Constructor von Employee und Event nehmen jetzt nicht mehr einfach die Properties als parameter, sondern stattdessen das DatenbankController Objekt mit dem gearbeitert werden soll, ein Data Objekt, welches die Properties enthält und optional die ID, unter der das jeweilige Objekt in der Datenbank referenziert wird. Hier wird Implementationsbedingt benötigt, dass der Controller bei der Instanziierung mit übergeben wird, und die ID sollte optional sein, da DatabaseController.addEmployeeToDb/addEventToDb fertige Employee/Event Objekte benötigen.

Getting Started

Zur Installation und zum Ausführen wird Node (https://nodejs.org) und der dazugehörige Node Package Manager (https://www.npmjs.com/) benötigt, welcher mit Node mitgeliefert wird. Beide Tools sollten über die Pfadvariable des Systems verfügbar sein.

Setup

```
$ git clone https://github.com/IZEDx/SWT-17-18.git
$ cd Prototyp/
$ npm install
```

NPM installiert nun alle Dependencies für die Entwicklung, inklusive dem Typescript Compiler.

Beachte: Alle npm Commands müssen im "Prototyp" Ordner ausgeführt werden.

Build

```
Prototyp$ npm run build
```

Dies führt nun den Typescript Compiler aus, welcher das Projekt anhand der tsconfig.json zu Javascript übersetzt. Die fertigen Dateien sind im "dist" Ordner.

Außerdem kann man eine PDF aus dieser README.md erstellen mithilfe von:

```
Prototyp$ npm run build-pdf
```

Run

```
Prototyp$ npm start
```

Startet den Server mit den Einstellungen aus der config.json, sollte keine vorhanden sein werden die Standardeinstellungen aus der config-sample.json verwendet.

Configuration

Es empfiehlt sich die config-sample.json nicht direkt zu ändern sondern umzubenennen in config.json, damit die gespeicherten Einstellungen bei einem Update nicht überschrieben werden.