ZHAW Modul WebEngineering 2 – David Elsener

Projektbeschreibung

TodoApp

Dieses Dokument beschreibt die Umsetzung der Todo-Applikation – respektive des simplen Scrum Boards – mit nodejs. Die Aufgabenstellung findet sich unter https://github.com/delsener/webengineering.todoapp.

Infrastruktur / URLs

- **Versionskontrollsystem: Git**
  Repository: https://github.com/delsener/webengineering.todoapp
  Beinhaltet Source Code, Konfigurationen, Dokumentation, Aufgabenstellung (README)
- **Deploy-Umgebung: Heroku**
  Die Applikation wurde auf Heroku unter http://hidden-ravine-7841.herokuapp.com/ deployed.
- **Liste Titel**
  Liste Text

Installation

Möchte man die Applikation lokal auschecken und installieren, muss man folgende Schritte tun:

- nodejs und Paket **express** (npm install express) installieren
- Gesamtes Git-Repo auschecken
- Mittels **node server** den Server starten

  Standardmässig hört der Server auf den Port 8002, man erreicht ihn also unter http://localhost:8002/. Möchte man einen anderen Port konfigurieren, so kann man diesen über die Umgebungsvariable PORT übersteuern. Auf einem Linux-Rechner zum Beispiel mittels **export PORT=8080**.

Erreichte und nicht erreichte Punkte aus der Aufgabenstellung

- **Erreicht**
  - Server bietet **REST-API** an und unterstützt folgende Ressourcen / URLs:
    - Alle Todos: http://hidden-ravine-7841.herokuapp.com/todos
    - Ein Todo: http://hidden-ravine-7841.herokuapp.com/todos/0
    (wobei PUT = update, POST = erstellen, GET = lesen, DELETE = löschen)
  - Frontend wurde mittels Backbone entwickelt
  - Es können Todos erfasst, bearbeitet und gelöscht werden
  - Status von Todos kann mittels dem "Next"-Button geändert werden. Bemerkung: Von DONE geht's einfachheitshalber auf TODO zurück.
- **Nicht erreicht**
  - Keine Websockets eingebunden
  - Kein aufwendiges / schönes GUI

Komponenten

Server
Wie im vorherigen Kapitel gesehen, wurde ein minimalistischer nodejs-Server entwickelt. Dafür habe ich zusätzlich das npm Paket express verwendet, welches die Implementierung des Servers vereinfacht.

Webseite / Frontend

- **Backbone**
  Vereinfachung der Frontend (Arbeit mit Model, etc.) Implementierung. Folgende Backbone-Komponenten habe ich verwendet:
  - Backbone.Model: View-Model für Todo's
  - Backbone.Collection: Liste der View-Models, über eine URL synchronisiert mit den Todo's auf dem Server
  - Backbone.View:
    - Abstrahierte View eines einzelnen Todo's
    - Haupt-View, stellt im Prinzip die Frontend-App dar

  Alle Backbone-Komponenten sind in **public\js\todo_backbone.js** implementiert und werden im **public\index.html** inkludiert.

- **Bootstrap**
  Für das erforderliche Spalten-Layout (Scrum-Karten sollen in Spalten dargestellt werden) habe ich Bootstrap eingesetzt, welches alle damit verbundenen Schwierigkeiten behebt.

- **JQuery**
  Wird von Backbone verwendet. Ich selbst habe keine / nur indirekt JQuery-Funktionen eingesetzt.

Funktionsweise

Server-Seitig gibt es ein Model (**models\todo.js**), welches ein Todo definiert gleichzeitig auch den gesamten Persistenzlayer darstellt (Todos werden nur im Memory des Server behalten, keine DB). Es bietet also auch Funktionen an, um Todo's zu speichern / laden / etc. Das Arbeiten mit diesen Model übernimmt ein Controller (**controllers\todo_controller.js**).

Die Kommunikation mit dem Server funktioniert im Prinzip im **RESTful** Prinzip. Über ein im **routes\todo\index.js** definierte Routing sind die "REST-URLs" gemappt auf die entsprechenden Funktionen des Controllers. So liefert zum Beispiel localhost:8002/todos auf die Ressource zurück, die alle Todos beinhaltet, und zwar im gängigen JSON-Format:


```
[
  {
    "id": 0,
    "titel": "title",
    "description": "description     ",
    "estimation": "3",
    "assignee": "unassigned",
    "state": 1,
    "title": "title"
  },
  {
    "title": "example title",
    "description": "description",
    "estimation": "5",
    "assignee": "Developer",
    "id": 1,
    "state": 1
  }
]
```


Im Frontend wird nun in Kombination mit einem Backbone Model und einer Collection, welches vom Typ dieses Models und gemappt auf die URL **localhost:8002/todos** die Synchronisation der Todo's zwischen Client und Server übernimmt. Updates auf dem Server werden über RESTful URLs und den entsprechenden HTTP-Methoden GET, PUT, POST und DELETE gemacht und vom Controller verarbeitet.

Screenshots (falls Heroku aussteigt ;))