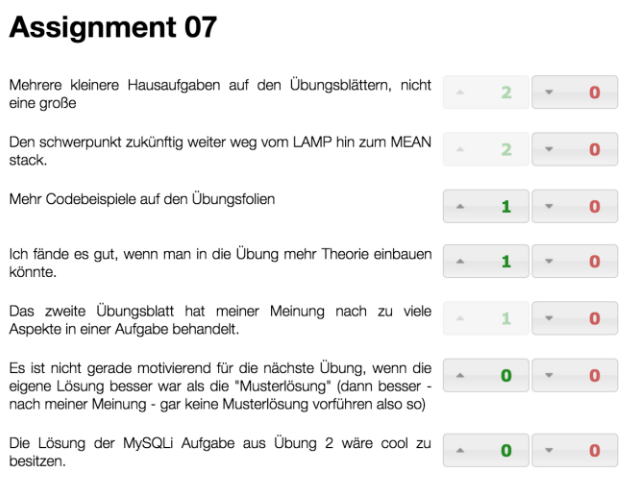
Assignment 7 (HF)

Due: Mon 15.12.2014; 16:00h (2 Weeks)

Please note: Although the due date is two weeks from now, there will be another assignment next week!

Task: Feedback Voting Web Application



Assignment 07

Mehrere kleinere Hausaufgaben auf den Übungsblättern, nicht eine große	▲ 2	▼ 0
Den schwerpunkt zukünftig weiter weg vom LAMP hin zum MEAN stack.	▲ 2	▼ 0
Mehr Codebeispiele auf den Übungsfolien	▲ 1	▼ 0
Ich fände es gut, wenn man in die Übung mehr Theorie einbauen könnte.	▲ 1	▼ 0
Das zweite Übungsblatt hat meiner Meinung nach zu viele Aspekte in einer Aufgabe behandelt.	▲ 1	▼ 0
Es ist nicht gerade motivierend für die nächste Übung, wenn die eigene Lösung besser war als die "Musterlösung" (dann besser - nach meiner Meinung - gar keine Musterlösung vorführen also so)	▲ 0	▼ 0
Die Lösung der MySQLi Aufgabe aus Übung 2 wäre cool zu besitzen.	▲ 0	▼ 0

Imagine a website, where students can submit suggestions for improvement regarding a lecture. Already submitted suggestions are listed and all students can up- or down-vote a suggestion. The voting is done with buttons that show distinguishable icons and the number of respective votes. The suggestions are ordered according to their average score. As people submit suggestions and vote for them, the views refresh automatically.

Watch the demo here: https://www.youtube.com/watch?v=4VjWNc54FOk

Your task this week is to write such a web application. However, submitting a suggestion is not a required feature – simply take the list of suggestions that you find in the material for this assignment.

Although it might be difficult to make a clear cut, you can tackle the task by distributing the coding between the frontend and the backend. The requirements are described in the following.

Task 1: Frontend

The front end is 'what the user sees' or, in other words, an interface between the user and the back end. For our project, it's everything included in the "public" folder (get the template here: https://gitlab.cip.ifi.lmu.de/stockinger/mmn-nodejs-template/tree/master) plus the templates in the "view" folder.

Include the following in your user interface:

- *List of suggestions*. Each suggestion has its own ID. Use the `<div>`'s id or a data attribute to store the corresponding information.
- Each suggestion is accompanied by *two buttons* to vote it up or down.
  - If the user has already voted, the button should be deactivated.
  - The users should be able to change their minds. Hence, only deactivate the button that they have already clicked. For example, if the user voted 'up', the 'up' button is deactivated but once the user clicks 'down' for the same suggestion, 'down' is deactivated and 'up' is re-activated.
  - We suggest using a jQuery-UI button widget for this functionality, because it has handy methods to enable/disable buttons.
- If the user clicks a button, send a POST request to `'/service'` in order to update the model in the back end. The request should be a JSON Object, which includes
  - The ID of the suggestion
  - The vote: 'up' or 'down'
- Once the voting request has finished, the list of suggestions is rearranged according to the score of a suggestion.
  - The easiest scoring algorithm could be
    $$score(suggestion) = suggestion.upVoteCount - suggestion.downVoteCount$$
  - `array.sort(function(a,b){})` lets you pass you own sorting function – a & b are the comparable objects.
  - Use `$(element).after(succeedingElement)` to generate the order.
- Write a class API that encapsulates all API calls:
  - `voteUp(suggestionID)`
  - `voteDown(suggestion)`
  - `getSuggestions()`

Task 2: Backend

The backend is the part of the web app that the users do not see. It holds the data model and provides interfaces to modify the model. There should be two main routes to the back end that you define in app.js.

- Entry point: `'/'`
  - Create the file index.js which renders views/index.jade for all GET requests. Use the Express Router to achieve this.
  - Retrieve a list of suggestions from the service.js module and sort the suggestions by score if necessary.
  - Export the router object, that other modules can import with require().
- Suggestion API: `'/service'`
  - Create the file service.js and use the Express router to accept POST requests for its root path. Export the router object, that other modules can import with require().

- Require the module (data.js) that is included in the material for this assignment. The object can be modified, e.g. by
  `data[0].someNewAttribute = 'Some new value'`
- POST requests update the model. A user can only vote 'up' or 'down' once for each suggestion. You can use the user's IP to block duplicate votes. The user IP can be retrieved from the request object:
  ```
  var getIpForRequest = function(request){
      return request.headers['x-forwarded-for'] ||
          request.connection.remoteAddress ||
          request.socket.remoteAddress ||
          request.connection.socket.remoteAddress;
  };
  ```
  Make sure the users can change their mind and swap their vote.
- Create another route `'/service/suggestions'` which supports GET and delivers the current data as JSON object. Handle the IP from which the request was sent to add a key **`hasVoted`** that indicates if this IP has voted for the suggestion.

Further Tips and Tricks

- *No database*. For this exercise, we make use of NodeJS capability to store data across multiple requests (this is one of the biggest advantages over Apache/PHP – it trumps sessions because the data can be shared across requests from different users!). Therefore, we do not need a database to store the suggestions, but the data is lost once we restart the app.
- *Suggestion data structure*. A suggestion is represented with the following model:
  ```
  {
      id: 1,
      text: 'Suggestion Text',
      votes: {
          up: 0,
          down: 0
      },
      votesByUser: {}
  }
  ```
  votesByUser is a map of IPs to votes. It may look like this, after a couple of requests:
  ```
  votesByUser: {
          '127.0.0.1': 'up',
          '192.168.2.100': 'down'
  }
  ```
- *Use Socket.io (optional)*. Push updates in the data model to the clients. This will not be part of the solution that's shown in the tutorial.

Further notes:

- Please make sure to comment your code sufficiently to facilitate correction of your submission.
- Incomplete submissions are welcome. However, please make sure to include a "README" text file to tell us how far you have come.