Abschlussbeleg
„adViz“ – Frontend und Backend mit DB

Letzter Commit ins git: 1.PZR – 30.07.2022, 23:59 Uhr **ODER** 2. PZR – 30.09.2022, 23:59 Uhr

Zur Bestätigung, dass Sie den Beleg als beendet und abgegeben betrachten, bitte ich Sie, mir eine Email nach Ihrem finalen Commit in git zu schicken. Diese Email bestätigt die Abgabe des Belegs und muss folgendes enthalten:
Betreff: WAD-Abschlussbeleg

- Name, Vorname
- Matrikelnummer
- gitHub-Repo
- den git commit hash Ihres finalen Commits für diesen Beleg

User Story: Als Product Owner des „AdViz“-Projekt möchte ich eine letzte Version der adViz-Webapp dem Management vorstellen. Diese Version soll die Kontaktdaten aus einer Datenbank beziehen, neue Kontakte in dieser Datenbank speichern und Kontakte in der DB updaten und löschen. In die „adViz“-Webapp kann ich mich als Admin-User, admina, oder als ein:e „normale:r“ Nutzer:in, normalo, einloggen. Es gelten die Rollenrechte für admina und normalo aus Beleg 2.

Voraussetzung: Die Anforderungen aus Beleg 2 sind umgesetzt. Sollte das nicht der Fall sein, dann muss das nachgebessert werden. Soll heißen: welche Screens, wann und wie angezeigt werden sollen, wird hier nicht noch einmal beschrieben. Diese Anforderungen finden Sie in Moodle.

AdViz mit Backend muss so funktionieren wie Adviz in Beleg 2 + Update & Delete von Kontakten, bloß dass die Klicks auf entsprechende Buttons jetzt mit Ihrem Backend kommunizieren.

Teil 1: Dieser Beleg muss in Einzelarbeit angefertig werden: gitHub Änderungen!!

Das Teammitglied, welches zum git-Repository eingeladen wurde, richtet sich in gitHub sein eigenes, privates WAD-Repository ein und übernimmt den Stand des gemeinsamen Repos vor Beleg2-Abgabe. Das Teammitglied, welches jetzt sein eigenes Repo hat, muss aus dem gemeinsamen Repo wieder ausgeladen werden, so dass jetzt alle Studierenden ihre eigenen, privaten WAD-Repos haben. **Bitte mich, eschuler22, als Contributer/Collaborator zu Ihrem privaten WAD-Repo einladen!**

Teil 2: Node.js-Backend (erst einmal ohne DB)

- Wenn die PO-Person die URL, http://localhost:3000/, im Browser öffnet, wird der Login-Screen im Browser angezeigt.
- Die zwei User, admina und normalo, aus Beleg 2 müssen übernommen werden und sind verfügbar, können erst einmal im JavaScript-Code hart kodiert sein:

```
let admina = {username: "admina", password: "password", role:"admin"};
let normalo = {username: "normalo", password: "password", role:"normal"};
```

- Die vier Kontakte, die für Beleg 2 angelegt wurden müssen übernommen werden und sind verfügbar, , können erst einmal im JavaScript-Code hart kodiert sein.
- Ihr Node.js-Backend stellt zwei Endpoints zur Verfügung:
    - **/users**: der Endpoint für den Login, er akzeptiert nur POSTs mit username/password in der Payload. Bei ungültiger username/password-Kombo, wird der HTTP-Statuscode 401 („Unauthorized“) zurückgeschickt. Bei gültiger username /password-Kombination, wird der HTTP-Statuscode 200 und entsprechendes User-POJO als JSON in der Payload zurückgeschickt.
    - **/contacts**: der REST-Endpoint für die CRUDs von Kontakten - Dieser Endpoint handhabt alle 4 HTTP-Methoden:

HTTP-Request	RESTful Webservice	Successful HTTP-Response
POST /contacts mit neuem Kontakt in der Payload, **(WICHTIG: Kontakt hat noch keine ID!)**	**C**reate resource: Legt neuen Kontakt an, schickt Id des neuen Kontaktes an den Client zurück	Status code: 201 & HTTP-Header: Location: /contacts/newId
GET /contacts	**R**ead resources: Schickt alle Kontakte	Status code: 200 & HTTP-Header: Content-Type: application/json mit Payload = {Array von Kontakten}
PUT /contacts/id mit aktualisiertem Kontakt in der Payload	**U**pdate resource: Aktualisiert Kontakt	Status code: 204, keine Payload
DELETE /contacts/id	**D**elete resource: Löscht den Kontakt	Status code: 204, keine Payload

- Beispiele für die http-CRUDs von contacts:
  - POST http://localhost:3000/contacts mit Payload

```
{
        "firstname":"Garfield",
        "lastname":"Kater",
        "street":"Finowstr 10",
        …
        "owner":"normalo",
        "lat":"52.5",
        "lng":"13.4"
}
```

    Server legt diesen neuen Kontakt an und schickt die neue ID an den Client im HTTP-Header „Location“ zurück
    Beispiel: newId = 7,
    Response: HTTP-Status=201 Location: /contacts/7

  - GET http://localhost:3000/contacts: schickt alle Kontakte zurück

  - PUT http://localhost:3000/contacts/7 mit Payload:

```
{
        "id": 7,
        "firstname":"Garfield",
        "lastname":"Kater",
        "street":"Neue Straße 22",
        …
        "owner":"normalo",
        "lat":"52.5",
        "lng":"10.2"
}
```

    Server führt das Update des Kontaktes durch und schickt, falls erfolgreich, nur den Statuscode 204 zurück.

  - DELETE http://localhost:3000/contacts/7 ohne Payload: Server löscht den Kontakt mit Id 7 und schickt, falls erfolgreich, nur den Statuscode 204 zurück

Teil 3: Einbinden der Datenbank

- MongoDB**,** eine lokale Instanz at localhost:27017 muss genutzt werden

- Ihr lokal laufender MongoDB-Server muss ohne Authentifizierung nutzbar sein. D.h., der Befehl „mongo“ im Terminal ohne user/password-Argumente muss funktionieren:

```
~ $ mongo
MongoDB shell version v4.4.3
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
…
> show databases
admin    0.000GB
advizDB  0.000GB
…
> use advizDB
switched to db advizDB
> show collections
contacts
users
```

- Es muss die Datenbank ‚advizDB‘ mit zwei Collections angelegt werden:
  - eine Collection ‚users‘ mit den Feldern: username, password, role
  - eine Collection ‚contacts‘ mit den Felder aus der „Add Contact“-Form und den zusätzlichen Feldern owner, lat und lng.
- Die zwei Nutzer:innen aus Beleg 2 müssen genau so:

```
{username: "admina", password: "password", role:"admin"};
{username: "normalo", password: "password", role:"normal"};
```

  in die users-Collection übernommen werden und dürfen nicht mehr im JavaScript hart kodiert sein.
- Die vier Kontakte aus Beleg 2 müssen in die contacts-Collection übernommen werden und dürfen nicht mehr im JavaScript hart kodiert sein.
- Ihr Node.js-Backend ist mit der Datenbank verbunden und nutzt diese für die Validierung der Logins und die CRUD-Operationen für Kontakte.

Teil 4: Belegabgabe

Ich werde Ihren Code aus Ihrem gitHub-Repo aus dem **main**-branch, Ihren **letzten Commit vor dem Due-Date** auschecken und auf meinem Laptop zum Laufen bringen. Damit dies erfolgreich durchgeführt werden kann, werden präzise Anweisungen gebraucht. Diese Anweisungen müssen in einer README.md-Datei im Projektverzeichnis dokumentiert werden. Die Erstellung dieser README-Datei ist Ihre Aufgabe und Teil der Bewertung.

Da ich meine lokale MongoDB nutzen werde, aber Ihre Webanwendung mit Ihren Kontakten testen muss, müssen Ihre Daten irgendwie in meine lokale MongoDB kommen. Das kann man mit Hilfe der mongo-shell und JS-Sciripten bewerkstelligen. Wie genau, finden Sie hier https://www.mongodb.com/docs/manual/tutorial/write-scripts-for-the-mongo-shell/ . Ihre Aufgabe ist es, ein JS-Script zu schreiben, welches die Collection ‚contacts‘ in der Database ‚advizDB‘ erstellt und diese mit den Daten aus Ihrer contacts-Collection befüllt. Dieses Script muss ins git, damit ich es finden und ausführen kann. Wo dieses Script zu finden ist und wie es auszuführen ist, gehört auch in README.md Datei. Der Befehl zum Befüllen meiner advizDB mit Ihren Daten, den ich ausführen werde:

```
mongo localhost:27017/advizDB yourScriptfile.js
```

Die README.md ist wichtig! Sie beschreibt anderen Entwickler:innen Ihr Projekt und wie man es erfolgreich zum Laufen bringt. Was muss in Ihre README?
Name
Matrikel-Nr
Project name
Project description
Instructions for

- checkout of code
- installation of code and packages
- creating db-collections and importing data

- starting the server
- URL to the login-Page of your webapp