Prof. Dr. Sebastian von Klinski, klinski@beuth-hochschule.de Raum B230a

| ÜBUNG 1 | | | |
|---|---|---|---|
| LV | **WEB-ENGINEERING II** | | |
| THEMA | **REST-SERVER** | FÄLLIGKEIT | **SIEHE MOODLE** |

# UMSETZUNG DES REST-SERVERS FÜR EIN STUDIERENDENBEWERBER-PORTAL

Im Rahmen der ersten Übungsaufgabe soll das Backend für ein Studierendenbewerberportal umgesetzt werden. Das Backend soll als REST-Service-Server umgesetzt werden. Der REST-Server soll als Message-Protokoll JSON verwenden.

Die Abgabe der Übung 1 unterteilt sich in die folgenden Meilensteine:

- **Meilenstein 1:** In diesem ersten Meilenstein müssen erste Funktionen eines REST-Servers umgesetzt werden. Hierzu wird ein einfacher REST-Service zum Verwalten der User-Objekte umgesetzt.
- **Meilenstein 2**: Zum Meilenstein 2 soll die Authentifizierung und ein zweiter Endpoint umgesetzt sein.
- **Meilenstein 3 (finale Abgabe)**: Bei der finalen Abgabe sollen alle nachfolgend beschrieben Anforderungen umgesetzt sein.

Die Termine für die Abgabe der Meilensteine sowie der finalen Abgabe werden im Moodle angegeben.

In den nachfolgenden Abschnitten wird der REST-Server beschrieben, der zur finalen Abgabe umgesetzt werden soll. Für die Meilensteine sowie die finale Abgabe gibt es in Moodle darüber hinaus noch Vorgabebeschreiben, in denen auch Hilfestellungen und Hinweise enthalten sind.

Bitte beachten Sie, dass entsprechend den allgemeinen Vorgaben der REST-Server mit TypeScript umgesetzt werden soll.

## BESCHREIBUNG DER FACHDOMÄNE

Das Studierendenbewerberportal erlaubt es, dass sich Studierenden für die Zulassung zu einem Studium an einer Hochschule bewerben. Hierzu können Studierende in der Web-Anwendung den Studiengang auswählen, für den sie sich bewerben wollen und anschließend eine Bewerbung anlegen.

Mit der Web-Anwendung und dementsprechend mit dem REST-Server können User, Studiengänge und Studienbewerbungen verwaltet werden. Es können beliebig viele User, Studiengänge und Bewerbungen in der Web-Anwendung verwaltet werden.

Dementsprechend kann der REST-Server die folgenden Entitäten mit den aufgeführten Attributen verwalten.

- **User**: Die User ermöglichen die Umsetzung einer Authentifizierung und erlauben es zu verfolgen, wer welche Daten angelegt und editiert hat. Der User umfasst
  - **userID**: (z.B. manfred), ist die eindeutige ID, mit dem man sich beispielsweise einloggen kann. Dieses Feld ist obligatorisch.
  - **firstName**: Das ist der Vorname der Person. Dieses Feld ist optional.
  - **lastName**: Die ist der Nachname der Person. Dieses Feld ist optional.
  - **isAdministrator**: Über diesen boolschen Wert wird angegeben, ob der User Administratorrechte hat oder ein normaler Studierender ist. Standardwert ist false.

    - **password**: In diesem Attribut wird das gehashte Passwort des Users abgelegt. Dieses Feld ist obligatorisch und darf nicht leer sein.
- **DegreeCourse**: Der „Degree Course“ ist ein Studiengang, für den sich Studierende bewerben können. Der Studiengang umfasst die folgenden Attribute, die alle obligatorisch sind:
    - **id**: Das ist der Unique-Identifier des Studiengangs.
    - **name**: Das ist der vollständige Name des Studiengangs (z.B. „Medieninformatik Bachelor“)
    - **shortName**: Das ist die Kurzbezeichnung des Studiengangs, über die schneller nach dem Studiengang gesucht werden kann (z.B. MB-B)
    - **universityName**: Das ist der Name der Hochschule, in der der Studiengang angeboten wird (z.B. „Berliner Hochschule für Technik Berlin“).
    - **universityShortName**: Das ist die Kurzbezeichnung der Hochschule (z.B. BHT).
    - **departmentName:** Das ist der Name des Fachbereichs, in dem der Studiengang angeboten wird (z.B. „Informatik und Medien“)
    - **departmentShortName:** Das ist die Kurzbezeichnung des Fachbereichs (z.B. „FB VI“)
- **DegreeCourseApplication:** Die Studienbewerbung führt die beiden anderen Entitäten zusammen. Sie umfasst den User, der sich für einen Studiengang bewirbt, sowie den Studiengang. Darüber hinaus wird das Bewerbungssemester angegeben. Alle folgende Attribute sind obligatorisch.
    - **id:** Der Unique-Identifier von der Studienbewerbung.
    - **applicantUserID**: Das ist die User-ID des Studierenden.
    - **degreeCourseID**: Das ist der Unique-Identifier des Studiengangs.
    - **targetPeriodYear**: Das ist die Angabe des Jahres, für das sich der Studierende bewirbt.
    - **targetPeriodShortName**: Das ist die Kurzbezeichnung des Semesters ( „WiSe“ oder „SoSe“)

Es kann beliebig viele Studiengänge geben. Für einen Studiengang kann es beliebig viele Bewerbungen geben. Ein Studierender kann beliebig viele Bewerbungen für unterschiedliche Studiengänge anlegen. Es dürfen aber nicht mehrere Bewerbungen für den gleichen Studiengang im gleichen Semester geben. Für unterschiedliche Semester ist es aber möglich.

**Hinweise:**

- Verwenden Sie exakt die oben angegebenen Bezeichnungen sowohl für die Entitäten als auch die Attribute.
- MongoDB erstellt automatisch das Attribut „_id“ als Unique Identifier. Sie können dieses Attribut beibehalten, müssen aber bei allen Anfragen und Antworten des REST-Servers ein entsprechendes Mapping der Attribute vornehmen. Bei Antworten des REST-Servers sollte immer das Attribut „id“ und nicht „_id“ zurückgegeben werden.

## SOFTWARE-KOMPONENTEN

Für die Umsetzung des REST-Servers sollten Sie die folgenden Software-Komponenten Node.js, Express, Mongoose und MongoDB verwendet werden.

### *Hinweis zur Anbindung von MongoDB*

Für den Zugriff auf MongoDB sollten Sie Mongoose verwenden (siehe Vorlesungsfolien). Implementieren Sie den REST-Service mit einer **lokalen** Datenbank. Diese Datenbank müssen Sie zunächst installieren! Installieren Sie sich hierzu die MongoDB Community Edition.

Die Verwendung einer Cloud-Datenbank-Instanz ist nicht zulässig. Für die Tests verwende ich eine lokale Datenbank auf meinem Rechner. Verwenden Sie daher für die Anbindung der Datenbank ausschließlich Standardwerte (z.B. Port) und verwenden Sie keine Login-Daten.

### *Verwendung von TypeScript*

Der REST-Server muss in TypeScript umgesetzt werden. Die Verwendung von JavaScript ist im Gegensatz zu den vergangenen Semestern nicht mehr möglich.

## UMZUSETZENDE ENDPOINTS

Es sollen im REST-Server 5 Endpoints umgesetzt werden:

- **/publicUsers:** Das ist ein User-Test-Endpoint für den ersten Meilenstein. Dieser Endpoint ist als erste Programmierübung zum Anlegen, Editieren und Abrufen von Usern gedacht. Bei diesem Endpoint wird keine Authentifizierung durchgeführt und auch keine Autorisierung.
- **/users**: Das ist der eigentliche Endpoint zum Verwalten der User. Er wird erst im zweiten Meilenstein umgesetzt. Wichtig: /publicUsers und /users sollen die gleichen Entitäten verwalten!
- **/authenticate**: Über diesen Endpoint kann man sich als User authentifizieren und erhält für alle weiteren Anfragen einen Access-Token als Rückmeldung, falls die Authentifizierung erfolgreich war.
- **/degreeCourses**: Dieser REST-Service umfasst alle Funktionen zum Verwalten von Studiengängen (Anlegen, Abrufen, Ändern, Löschen, Suchen)
- **/degreeCourseApplications**: Dieser REST-Service umfasst alle Funktionen zum Verwalten von Studienbewerbungen (Anlegen, Abrufen, Ändern, Löschen, Suchen)

Bitte beachten Sie, dass bei der Umsetzung der Endpoints die Vorgaben der Vorlesungsfolien zu Best-Practices bei REST-Services einzuhalten sind. Bitte sprechen Sie die Lehrkraft an, sollte Sie dazu Fragen haben oder einzelne Aspekte der Aufgabenstellung nicht zu den Vorgaben der Vorlesung passen.

### *Public-User-Endpoint*

Der erste Endpoint, den Sie zum Meilenstein 1 umsetzen sollen, ist unter der URL „/publicUsers“ abrufbar und umfasst alle Funktionen zum Anlegen, Abrufen und Ändern von Usern. Dieser Endpoint soll zum Einstieg in die Umsetzung von REST-Services genutzt werden. Er sollte unter anderem das Abrufen aller User, das Anlegen eines neuen Users, das Aktualisieren und Löschen eines Users sowie das Abrufen von einzelnen Usern anhand der User-ID umsetzen.

Dieser Endpoint ist ohne Authentifizierung und Access-Token nutzbar, damit die Umsetzung möglichst einfach ist. Dieser Endpoint wird unter anderem für das automatisierte Testen des REST-Servers benötigt.

In diesem Endpoint sollen bei der Rückgabe von Usern alle Daten, also auch die Passwörter enthalten sein. Das würde man bei einem produktiven Server nie machen! Aber mit diesem Endpoint wird das Hashing des Passwortes überprüft.

Die Passwörter der User sollen in der Datenbank entsprechend den Vorlesungsfolien als Hash-Wert mit Salt abgelegt werden. Diese Besonderheit ist beim Aktualisieren von User-Daten wichtig. Wenn bei einem User nur der User-Name geändert wird, muss der Hashwert gleichbleiben. Wenn jedoch das Passwort geändert wird, muss für das neue Passwort erneut ein Hashwert berechnet werden.

Die ausführliche Beschreibung des Endpoints finden Sie in der Aufgabenbeschreibung von Meilenstein 1. Beachten Sie, dass der publicUsers-Endpoint auch nach dem Meilenstein 1 erhalten bleibt. Er wird bei allen automatischen Tests genutzt.

### *User-Endpoint*

Der User-Endpoint ist unter der URL „/users“ abrufbar und umfasst alle Funktionen zum Anlegen, Abrufen, Löschen und Ändern von Usern für die Web-Seite. Dieser Endpoint entspricht funktional dem Public-User-Endpoint. Jedoch werden in diesem Endpoint alle Zugriffe auf die Daten über eine Authentifizierung und Autorisierung geschützt.

Sowohl der User-Endpoint als auch der PublicUser-Endpoint greifen auf die gleichen Daten zu!

### *Authentifizierungs-Endpoint*

Der Authentifizierungs-Endpoint ist unter der URL „/authenticate“ erreichbar und erlaubt die Authentifizierung des Users. Er gibt bei erfolgreicher Authentifizierung einen Access-Token zurück, im Fehlerfalle eine Fehlernachricht.

### *DegreeCourse-Endpoint*

Der DegreeCourse-Endpoint ist unter der URL „/degreeCourses“ erreichbar. Er umfasst alle notwendigen Funktionen zum Anlegen, Abrufen, Ändern, Löschen und Suchen von Studiengängen.

### *degreeCourseApplication-Endpoint*

Der Endpoint für Studienbewerbungen ist unter der URL „/degreeCourseApplications“ erreichbar. Er umfasst alle notwendigen Funktionen zum Anlegen, Abrufen, Ändern, Löschen und Suchen von Forum-Nachrichten.

## VERZEICHNIS- UND MODULSTRUKTUR

In der nachfolgenden Abbildung wird beispielhaft an einem alten Projekt dargestellt, wie ihr Projekt strukturiert sein sollte. Das zentrale Server-Modul sollte httpServer.js heißen. Alle Endpoints sollten im Verzeichnis „endpoints“ sein. Für jeden Endpoint sollte es ein entsprechendes Unterverzeichnis geben.

Die Endpoints sollten im Wesentlichen wie dargestellt strukturiert sein: Es gibt eine Express-Route, diese greift auf ein Service-Modul zu und das verwendet ein Mongoose-Modell. Im Meilenstein 3 wird geprüft, ob diese Vorgaben eingehalten wurden. Falls Ihre Umsetzung von den Vorgaben abweicht, werden dafür entsprechende Abzugspunkte berechnet.

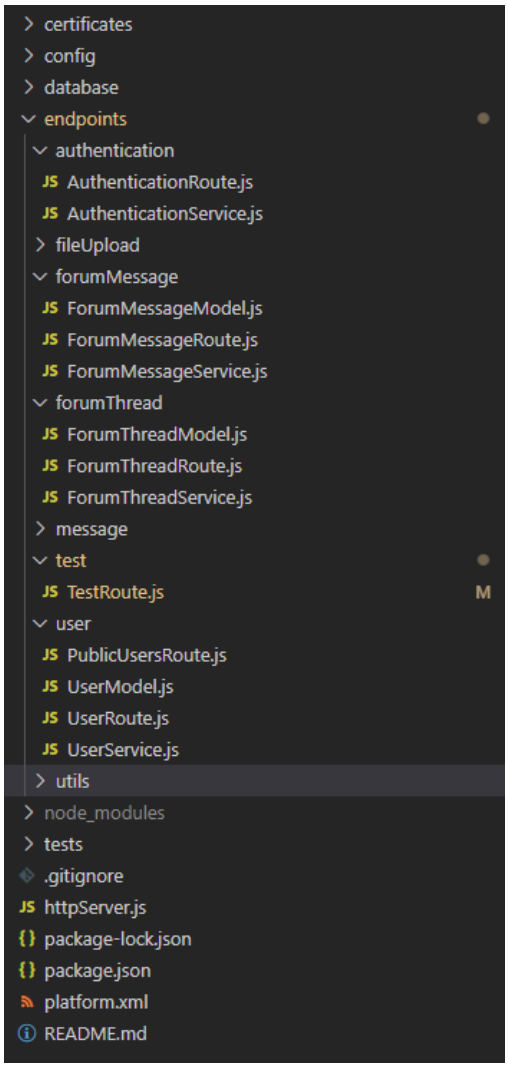


**Abbildung 1:** Einzuhaltende Verzeichnisstruktur

Durch die Trennung von Routen und Service wird eine funktionale Kapselung umgesetzt. Das bedeutet, dass Routen und Services unterschiedliche Funktionen umsetzen, die zur Verbesserung der Wiederverwendung getrennt bleiben sollen.

Routen haben die folgenden Aufgaben:

- Routen setzen die protokollspezifischen Aspekte der Endpoints um. Das bedeutet, dass das Empfangen von http-Requests und senden von http-Reponses ausschließlich in Routen vorgenommen werden sollte. Daraus folgt, dass in Service-Modulen nicht mehr http-Response-Objekte oder http-Request-Objekte sein sollten!
- Routen kapseln außerdem alle Aspekte der Authentifizierung, indem sie beispielsweise den Access-Token prüfen. Darüber hinaus wird dort auch festgelegt, bei welchen Routen eine Authentifizierung notwendig ist und bei welchen nicht.
- Bei Fehlern wird in der Route entschieden, welche http-Status-Codes zurückgegeben werden.

Im Gegenzug sollten die Services die folgenden Aufgaben umsetzen:

- Der Service-Layer bildet die Anwendungslogik um. Das umfasst unter anderem die Verknüpfung der verschiedenen Entitäten und die Validierung von Daten beim Anlegen und Ändern.

- Die Autorisierung (Prüfung, ob der User bestimmte Rechte hat) wird auf der Ebene der Services umgesetzt. Anmerkung: Auf der Ebene der Route können aber die Routen als Ganzes geschützt werden, indem nur User mit Admin-Rechten bestimmte Routen benutzen dürfen.
- Der Service kapselt außerdem den Zugriff auf das Persistenz-Medium, also in diesem Projekt den Zugriff auf die Datenbank über Mongoose.

Diese funktionale Trennung ist bei der Umsetzung einzuhalten! Sie wird bei der finalen Abgabe geprüft. Für jeden Endpoint, der nicht korrekt umgesetzt wurde, werden Abzugspunkte berechnet.

Im Verzeichnis „certificates“ sollten die Zertifikate für die Umsetzung von HTTPS enthalten sein. Im Verzeichnis „config“ sollten die Konfigurationsdatei(en) sein. Im Verzeichnis „database“ sollte das Modul zum Laden der Datenbankanbindung enthalten sein.

**Hinweis:**

Verwenden Sie in der Konfigurationsdatei besser nicht localhost, beispielsweise um den Pfad zur lokalen MongoDB-Datenbank zu verwenden. Nutzen Sie stattdessen 127.0.0.1. Die Auflösung des Namens „localhost“ funktioniert nicht auf allen Rechnern korrekt.

```
{
    "session": {
        "tokenKey": "djghhhhuuwiwuewieuwieuriwu",
        "timeout": 300
    },
    "db": {
        "dbConfigOptions": {
            "useNewUrlParser": true,
            "useUnifiedTopology": true
        },
        "connectionString": "mongodb://127.0.0.1/WE2"
    }
}
```

Abbildung 2: Beispiel für eine Konfigurationsdatei zum Auslagern von Daten wie die URL zur Datenbank

## VORGABEN FÜR ALLE ENDPOINTS

Bei der Umsetzung von allen Endpoints sind die Vorgaben aus der Vorlesung „Best-Practices bei der Umsetzung von REST-Services“ einzuhalten. Insbesondere sollten Sie die folgenden Punkte beachten:

- Setzen Sie „einfache REST-Antworten“ (vergleiche Vorlesungsfolien) um. Es sollte immer nur die abgefragten Objekte als JSON-Objekte zurückgegeben werden. Es sollte keine Wrapped-REST-Antworten und nie Strings (z.B. einfache Fehlermeldungen) zurückgegeben werden.
- Verwenden Sie die empfohlenen HTTP-Status-Codes

Alle Endpoints müssen die gängigen CRUD-Methoden sowie notwendige Suchfunktionen umfassen. Hierzu gehören die folgenden Methoden:

- Alle Entitäten abrufen
- Entität anlegen
- Entität aktualisieren
- Entität löschen
- Entität über die ID abrufen

Verwenden Sie für die Meilensteine 1 und 2 den Standard-http-Port 80. Beim Meilenstein 3 sollen Sie auf HTTPS wechseln. Da sollte dann der Standard-HTTPS-Port 443 verwendet werden.

Je nach Endpoint können zusätzliche Routen erforderlich sein. Beispielsweise wird es bei der DegreeCourseApplication-Route notwendig sein, die Bewerbungen des aktuellen Users abzurufen. Solche zusätzlichen Routen und Anforderungen werden in den Aufgabenstellungen zu den Meilensteinen erläutert.

In den folgenden Abschnitten werden die Anforderungen an die Anfragen und Antworten exemplarisch am Beispiel des publicUser-Endpoints beschrieben.

**Wichtiger Hinweis**: Nach der Server-ID und vor der Resource wird immer „api“ in der URL eingefügt. Das eine gängige Konvention, um REST-Schnittstellen von Web-Anwendungen abzugrenzen.

```
GET http://localhost/api/users
```

### *Route für das Abrufen aller Objekte*

Für das Abrufen von Entitäten von einem Endpoint wird http-GET verwendet. Beispiel: Der Request zum Abrufen aller User über den Public-User-Endpoint sieht beispielsweise wie folgt aus:

```
GET http://localhost/api/publicUsers
```

Bei diesem Request-Beispiel wird keine Authentifizierung vorgenommen. Der Request mit gleichzeitiger Authentifizierung per JWT-Token sieht wie folgt aus:

```
GET http://localhost/api/users
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey-
Jib2R5Ijoic3R1ZmYiLCJ1c2VySUQiOiJhZG1pbiIsImlhdCI6MTYzMDI1NTAwMX0.bwvnOHZMloU6ol-
FOsMoQOL6O5y7fKQ65z05yNPu81Rg
```

Die Routen zum Abrufen der anderen Entitäten sehen entsprechend aus, nur dass die URLs entsprechend angepasst sind. Die Antworten auf diese Anfragen sollten, falls die Anfrage korrekt verarbeitet wurde, einen JSON-Array zurückgeben. Wenn es beispielsweise keine User in der Datenbank gibt, sollte ein leerer Array zurückgegeben werden.

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept
Access-Control-Expose-Headers: Authorization
Content-Type: application/json; charset=utf-8
Content-Length: 2
ETag: W/"2-l9Fw4VUO7kr8CvBlt4zaMCqXZ0w"
Date: Wed, 02 Feb 2022 16:38:35 GMT
Connection: close

[]
```

Wenn es User gibt, sollten die im Array zurückgegeben werden. Auch wenn es nur einen User gibt, sollte dieser in einem Array zurückgegeben werden.


```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept
Access-Control-Expose-Headers: Authorization
Content-Type: application/json; charset=utf-8
Content-Length: 360
ETag: W/"168-FLCEMIxfncjJ+1Sxl8IgTxIXjwg"
Date: Tue, 05 Sep 2023 16:45:41 GMT
Connection: close

[
  {
    "id": "64b64b04622a547c0bf4a567",
    "userID": "admin",
    "firstName": "Udo",
    "lastName": "Müller",
    "isAdministrator": true
  },
  {
    "id": "64b64b1c622a547c0bf4a56b",
    "userID": "manfred",
    "firstName": "Manfred",
    "lastName": "Schreiber",
    "isAdministrator": false
  },
  {
    "id": "64f736c0223c6ab41e360b9a",
    "userID": "manfred2",
    "firstName": "Manfred",
    "lastName": "Mustermann",
    "isAdministrator": false
  }
]
```


Abbildung 3: Response beim Zurückgeben von allen Usern in einem Array

Bei der Umsetzung von allen Responses sollten Sie darauf achten, dass der Unique-Identifier von MongoDB „_id“ ist. Der REST-Server sollte aber die ID als Attribute mit dem Namen „id“ zurückgeben. Daher können die Mongoose-Objekte nicht direkt zurückgegeben werden. Stattdessen muss das Attribut „_id“ nach „id“ umgeschrieben werden, sowohl bei Anfragen als auch Antworten.

**Hinweis:** Beim Anlegen von Entitäten wird der Unique Identifier „_id“ von der Datenbank erzeugt!

### *Route für das Anlegen von Objekten*

Zum Anlegen von Entitäten werden http-Post-Requests verwendet. Im Gegensatz zu Get-Requests kann bei Post-Requests auch ein Request-Body mitgeschickt werden. Der Request-Body enthält dabei die notwendigen Daten im JSON-Format.

Der Request zum Anlegen eines Users sieht beispielsweise wie folgt aus:

```
POST http://localhost/api/publicUsers
Content-Type: application/json

{
    "userID": "admin",
    "firstName": "Udo",
    "lastName": "Müller",
    "password": "123",
    "isAdministrator": true
}
```

Die Antwort sollte dabei wie folgt aussehen:

```
HTTP/1.1 201 Created
X-Powered-By: Express
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept
Access-Control-Expose-Headers: Authorization
Content-Type: application/json; charset=utf-8
Content-Length: 123
ETag: W/"7b-m4DZQWszrfQY41kwLPC6dnW6MJM"
Date: Tue, 05 Sep 2023 14:10:08 GMT
Connection: close

{
  "id": "64f736c0223c6ab41e360b9a",
  "userID": "manfred2",
  "firstName": "Manfred",
  "lastName": "Mustermann",
  "isAdministrator": false
}
```

Generell sollte bei allen Requests, bei denen ein Objekt angelegt wird, das angelegte Objekt zurückgegeben werden! Bitte beachten Sie, dass als http-Status-Code 201 zurückgegeben werden sollte. Beim Anlegen von Usern sollte auf keinen Fall das Passwort zurückgegeben werden, außer bei dem publicUsers-Endpoint.

### *Routen zum Suchen einer Entität*

Routen zum Suchen einer konkreten Entität werden über einen einfachen GET-Request umgesetzt, wobei an den Endpoint die ID des Objektes angehangen wird. **Bei den User-Routen ist das die User-ID**, bei allen anderen Routen ist das der Unique Identifier des jeweiligen Objektes

```
GET https://localhost/api/publicUsers/manfred

GET https://localhost/api/degreeCourses/612bbd4ac5826400c858b5f1
```

Wenn der Request erfolgreich ist, sollte das Objekt als JSON-Objekt zurückgegeben werden:

```
HTTP/1.1 200 OK
Connection: close
Date: Fri, 16 Sep 2022 17:14:52 GMT
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept
Access-Control-Allow-Credentials: *
Access-Control-Expose-Headers: *
Content-Type: application/json
Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJEQUlGLUFjY2Vzcy1Ub2tlbiIsImV4cCI
6MTY2MzM1MjA1OSwidXNlcklEIjoiYWRtaW4iLCJzY29wZSI6IlNlcnZpY2UtUmVxdWVzdHMifQ.QoqxoQJfit
dPZf4EU4CFih1w_3OFa4JTeWiDKbitauo
Content-Length: 225
Server: Jetty(9.4.35.v20201120)

{
  "firstName": "Sebastian",
  "lastName": "von Klinski",
  "userStatus": "Active",
  "gender": "männlich",
  "id": "noid-698b7fe0-8e07-4041-8622-51501fb8b0f7",
  "title": "Prof. Dr.",
  "userID": "admin"
}
```

Wenn der Request nicht erfolgreich war, sollte ein entsprechender http-Error-Code und Fehlermeldung im JSON-Format zurückgegeben werden. Wenn es keine Entität mit der ID gibt, sollte der http-Status-Code 404 zurückgegeben werden.

```
HTTP/1.1 404 Not Found
X-Powered-By: Express
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept
Access-Control-Expose-Headers: Authorization
Content-Type: application/json; charset=utf-8
Content-Length: 66
ETag: W/"42-LKSQNmylQXrWSwf1hbx9sD7+dko"
Date: Wed, 02 Feb 2022 19:56:59 GMT
Connection: close

{
  "Error": "Es konnte kein User mit der ID admin2 gefunden werden."
}
```

Wenn es bei Antworten im Body einen Text gibt, sollte der immer im JSON-Format sein. Niemals sollte einfacher Text im Body sein!

### *Route zum (selektiven) Ändern von Objekten*

Die Routen zum Aktualisieren von Entitäten sind über http-PUT umzusetzen. Als Ergebnis wird das geänderte Objekt mit allen Daten zurückgegeben. Falls ein Fehler auftritt, wird eine entsprechende Fehlernachricht zurückgegeben.

Die Identifikation der Entität, die aktualisiert werden soll, wird wie bei der Suche nach einer Entität über die URL umgesetzt. Die ID des zu aktualisierenden Objektes wird dabei an die URL des Endpoints angehangen. Beim Public-User-Endpoint und dem User-Endpoint wird die Identifikation über die User-ID umgesetzt. Der Request zum Ändern des Users „manfred“ sollte wie folgt aussehen:

```
PUT http://localhost/api/publicUsers/manfred
Content-Type: application/json

{
    "firstName": "Udo"
}
```

Bei den anderen Entitäten wird zur Identifikation der Unique-Identifier verwendet (z.B. 61fab3c2d156bc4d8083327a). Die zu ändernde Attribute sind im JSON-Body enthalten. Der oben aufgeführte Request ändert beispielsweise das Attribut „firstName“. Es können auch mehrere Attribute gleichzeitig geändert werden. **Beim User darf jedoch nicht die User-ID geändert werden!**

Wenn der Request scheitert, beispielsweise weil es den User nicht gibt, sollte eine Antwort mit einem http-Fehler-Code (z.B. 400) zurückgegeben werden. Im Body sollte dann ein JSON-Objekt mit einem Error-Attribut und der Fehlernachricht enthalten sein. Der http-Code sollte entsprechend den Best-Practices gewählt werden.

```
HTTP/1.1 400 Bad Request
X-Powered-By: Express
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept
Access-Control-Expose-Headers: Authorization
Content-Type: application/json; charset=utf-8
Content-Length: 55
ETag: W/"37-umzivL4PUFETKb1ZGcyQoAK82wE"
Date: Wed, 02 Feb 2022 16:59:35 GMT
Connection: close

{
  "Error": "Update failed: Could not find user: manfred"
}
```

**Hinweis**: Korrekter wäre es, einen 404 zurückzugeben, wenn es den User nicht gibt und 400, wenn beispielsweise Attribute geändert werden sollen, die es nicht gibt. Eine solche Validierung ist keine Pflichtaufgaben. Sie können eine solche Validierung jedoch umsetzen und sich als Zusatzleistung anerkennen lassen.

Bei allen anderen Endpoints wird zur Identifikation der Unique-Identifier verwendet. Der Request zum Ändern eines Studiengangs würde beispielsweise wie folgt aussehen.

```
PUT https://localhost/api/degreeCourses/612bbd4ac5826400c858b5f1
Content-Type: application/json
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJib2R5Ijoic3R1ZmYiLCJ1c2VySUQiO-
iJhZG1pbiIsImlhdCI6MTYzMDI1NTYzNn0.XBPO9bqTHzu5dHVyxMfJjhA5bVT1u5Ik4FsHJYEO_V0

{
    "name": "Ein sehr kurzer Text"
}
```

### *Routen zum Suchen von assoziierten Entitäten*

Neben den grundlegenden CRUD-Methoden (Create, Read, Update, Delete) werden immer auch Suchfunktionen benötigt. Beispielsweise wird für Studienbewerbungen eine Suchfunktion benötigt, um alle Bewerbungen des aktuell eingeloggten Users abzurufen. Grundsätzlich sollten Suchfunktionen so umgesetzt werden, wie es in der Vorlesung „Best-Practices bei der Umsetzung von REST-Services“ im Abschnitt „Suchfunktionen“ beschrieben ist. Für die jeweiligen Meilensteine werden solche Suchfunktionen explizit beschrieben.

### *HTTP-Statuscode bei Antworten des REST-Servers*

Bei allen Antworten muss ein korrekter HTTP-Statuscode zurückgegeben werden. In der Vorlesung „Best-Practices bei der Umsetzung von REST-Services“ wird beschrieben, welche http-Statuscodes zu verwenden sind.

Wenn eine Anfrage korrekt bearbeitet werden konnte, muss ein Statuscode im 200er-Bereich zurückgegeben werden. Wenn eine Anfrage fehlerhaft war, muss ein Status-Code zwischen 400 und 511 zurückgegeben werden.

Die Status-Codes werden beispielsweise wie folgt umgesetzt:

```
    if (result) {
      res.status(200).json(Object.values(result));
    }
    else {
      res.status(404).json({ "Error": "Es konnte kein User mit der ID " + userID
                              + " gefunden werden." });
    }
```

In jedem Fall sollte es auf jede Anfrage auch eine Antwort geben. Beachten Sie dabei insbesondere alle möglichen Fehlerfälle und fangen Sie eventuelle Exceptions.

### *Fehlermeldungen*

Wenn bei einer Anfrage ein Fehlerauftritt, sollte stets ein Json-Objekt mit einer Fehlermeldung zurückgegeben werden. Die Antwort könnte beispielsweise wie folgt aussehen.

```
HTTP/1.1 404 Not Found
X-Powered-By: Express
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept
Access-Control-Expose-Headers: Authorization
Content-Type: application/json; charset=utf-8
Content-Length: 66
ETag: W/"42-LKSQNmylQXrWSwf1hbx9sD7+dko"
Date: Wed, 02 Feb 2022 19:56:59 GMT
Connection: close

{
  "Error": "Es konnte kein User mit der ID admin2 gefunden werden."
}
```

In jedem Fall sollte es bei Fehlermeldungen das Attribut „Error“ geben. Als Wert sollte dann die Fehlermeldung zurückgegeben werden, die dem User oder Entwickler angezeigt werden kann. Darüber hinaus sollte der http-Status-Code entsprechend gesetzt sein.

### *Antwort, wenn Route nicht vorhanden*

Falls es eine Route nicht gibt, sollte eine entsprechend Fehlermeldung und der http-Status-Code 404 zurückgegeben werden.

```
HTTP/1.1 404 Not Found
X-Powered-By: Express
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept
Access-Control-Expose-Headers: Authorization
Content-Type: application/json; charset=utf-8
Content-Length: 33
ETag: W/"21-ewJBLd1XoXnevjPb1+5HZ3ysCxM"
Date: Fri, 04 Feb 2022 13:50:19 GMT
Connection: close

{
  "Error": "Endpoint not existing"
}
```

Abbildung 4: Fehlermeldung, wenn die angefragte Route nicht existiert.

### *User-Identifikation in den Endpoints*

Beim Anlegen oder Ändern von Entitäten wird häufig hinterlegt, welcher User die Entität angelegt, bzw. geändert hat. Diese Identität des Users sollte nicht im JSON-Body übergeben werden, sondern muss aus dem Token bestimmt werden! Nur so kann sichergestellt werden, dass diese Angabe auch korrekt ist. Das heißt, dass in dem Token die User-ID abgelegt werden muss, wenn er angelegt wird, um sich später bei den folgenden Anfragen diese Information wieder rauszuholen.

Für die Autorisierung bei den REST-Services in dieser Übung ist es auch empfehlenswert, im Token zu hinterlegen, ob der User ein Administrator ist. Auf diese Weise muss der User nicht aus der Datenbank

abgerufen werden, um die Rechte zu prüfen. Diese Umsetzung ist nicht uneingeschränkt empfehlenswert, spart Ihnen jedoch für dieses Projekt etwas Arbeit.

In professionellen REST-Servern ist eine solche Vorgehensweise bei komplexen Rechterollen eher nicht empfehlenswert. Stattdessen würde bei jeder Anfrage zunächst das Rechteprofil des Users abgerufen werden, bevor die Anfrage in der Route bearbeitet wird.

## IT-SICHERHEIT

Der REST-Server soll zur finalen Abgabe alle Aspekte der IT-Sicherheit umsetzen, die auch für professionelle REST-Server notwendig sind. Das umfasst:

- Als Kommunikationsprotokoll sollte HTTPS verwendet werden. Erzeugen Sie sich hierfür ein selbst erstelltes Zertifikat.
- Für die Authentifizierung sollten Sie **Basic-Authentication** nutzen.
- Verwenden Sie JWT (Json-Web-Tokens) für die Kommunikation nach der Authentifizierung.
- Die Passwörter für die User sollten als Hashwerte mit Salt abgelegt werden. Verwenden Sie hierzu Bcrypt.
- In allen geschützten Routen sollte das Prüfen des Tokens über eine Middleware-Funktion umgesetzt werden, die einfach in den Funktionskopf der Routen eingefügt werden kann. Die Routen sollten die Middleware-Funktionen aus einem Utility-Modul verwenden, damit sie nur einmal implementiert werden.
- Es soll eine einfache Autorisierung umgesetzt werden, mit der sichergestellt wird, dass bestimmte Operationen (z.B. das Bearbeiten von Usern) nur durch einen Administrator durchgeführt werden können. Hierzu sollte einfach der boolsche Wert „isAdministrator“ im User-Objekt verwendet werden.

### *Filtern der Daten, die zurückgegeben werden*

Bei der Umsetzung der REST-Services ist der Schutz vertraulicher Daten zu beachten. Die REST-Services sollten nie sicherheitskritische Daten zurückgeben. Dazu gehören insbesondere Passwörter. Systemdaten wie die letzte Änderung/ Speicherung oder Unique-Identifier sollten nur zurückgegeben werden, wenn es wirklich nötig ist. Die einzige Ausnahme zu diesen Vorgaben ist der Public-User-Endpoint, der nur zur Einarbeitung und für die automatisierten Tests umgesetzt wird.

### *Authentifizierung*

Bei der Umsetzung der Authentifizierung sind die folgenden Punkte einzuhalten:

- Bei der Authentifizierung sollen die Credentials (User-Name und Passwort) per **Basic-Authentication** übergeben werden. Sie dürfen nicht per JSON-Body an den Server übertragen werden.
- Die Implementierung der Authentifizierung über Basic-Authentication muss dem Standard entsprechen (siehe Vorlesungsfolien). Das bedeutet, dass der Header so aussehen muss, wie es in den Vorlesungsfolien erläutert wird.
- Der Access-Token soll im Response-Header entsprechend den Vorlesungsfolien und den Programmierbeispielen zurückgegeben werden.
- Der Token ist dem Standard entsprechend im Header zu übergeben (siehe Vorlesungsfolien). Eine Übergabe per Cookie oder im JSON-Body ist nicht zulässig.

### *Vorgaben für die Autorisierung*

Der REST-Server und bei Übung 2 auch die Web-Anwendung sollen die folgenden Vorgaben für die Autorisierung umsetzen:

User-Management allgemein

- Nur Administratoren können alle User abrufen.
- Nur Administratoren können User anlegen, editieren und löschen.
- User können ihre eigenen User-Daten abrufen.
- User können ihr eigenes Passwort sowie Vor- und Nachnamen ändern. Alle anderen Daten können sie nicht ändern.
- Nicht authentifizierte User können keine User-Daten abrufen.

Studiengangverwaltung

- Nur Administratoren können alle Studiengänge bearbeiten (anlegen, ändern, löschen).
- Alle User können alle Studiengänge abrufen (lesen). Das Abrufen alle Studiengänge ist auch ohne Login möglich.

Studiengangbewerbungen

- Administratoren können für alle User Studienbewerbungen bearbeiten (anlegen, ändern, löschen) und abrufen.
- Nicht-Administratoren können nur ihre eigenen Studienbewerbungen abrufen. Sie können das Jahr und das Semester editieren.
- Nicht-Administratoren können ihre eigenen Bewerbungen löschen.
- Es kann pro User, Studiengang und Semester nur eine Bewerbung angelegt werden.
- Nicht authentifizierte User können keine Studienbewerbungen abrufen.

Diese Vorgaben beschreiben die wesentlichen Nutzerrechte. Bitte schreiben Sie mir eine E-Mail, sollte es noch Fragen zu diesen oder anderen Rechten geben.

## ALLGEMEINE ANFORDERUNGEN AN DIE UMSETZUNG

Bei der Umsetzung des REST-Servers sollten auch die folgenden Aspekte beachtet werden.

### *Starten des Projekts*

Die automatisierten Tests verwenden „npm start“ zum Starten der Web-Anwendung. Funktioniert diese Start-Methode nicht, werden mindestens 5 Abzugspunkte berechnet.

### *Erster User*

Wenn der REST-Server zum ersten Mal startet und die Datenbank leer ist, gibt es noch keine Nutzer, die sich einloggen könnten. Ab Meilenstein 2 dürfen User aber nur noch durch authentifizierte Administratoren angelegt werden. Daher muss ab Meilenstein 2 (nicht vorher!) beim Hochfahren zwingend sichergestellt werden, dass es zumindest den Standardadministrator gibt.
Der Server sollte beim Hochfahren schauen, ob es schon User in der Datenbank gibt. Falls es noch keine User gibt, sollt ein Standardadministrator mit einem Standardpasswort angelegt werden. Der Standard-Administrator sollte „admin“ heißen, das Passwort „123“ sein.

### *Formatierung*

Der Code muss bei der finalen Abgabe korrekt formatiert und ohne Code-Leichen sein. Überflüssige Leerzeilen und auskommentierter Code sind zu entfernen.

### *Konfiguration*

Entsprechend den Empfehlungen in der MongoDB-Vorlesung sollten Sie Angaben zu URLs (z.B. Datenbank-URL) oder anderen Konfigurationsparametern auslagern. Pfade zur Datenbank und ähnliche Aspekte sollten nicht im Code hart programmiert sein. Verwenden Sie hierzu ein Konfigurationsmodul. Hier sollte es im Verzeichnis „./config“ eine JSON-Datei geben, in der die Konfigurationsangaben enthalten sind.

### *Umgebungsvariablen*

Passwörter (z.B. für eine E-Mail-Registrierung) sollten als Umgebungsvariablen ausgelagert werden und nicht Teil der Anwendung sein.

## HINWEISE ZU ABGABEN

### *Starten des Servers*

Die Anwendung muss bei mir auf dem Rechner laufen. Dementsprechend sollten Konfigurationseinstellungen und Umgebungsvariablen so ausgelegt werden, dass der REST-Server auf einem anderen Rechner ohne Anpassung läuft. Testen Sie das bitte vor der Abgabe.

Sie können davon ausgehen, dass MongoDB unter dem Standard-Port läuft.

### *Abhängigkeiten prüfen*

Über den Befehl „npm install“ werden alle Abhängigkeiten zu Drittanbietermodulen aufgelöst und die Module im Verzeichnis „node_modules“ abgelegt. Achten Sie darauf, dass alle erforderlichen Abhängigkeiten in der package.json eingetragen sind. Wenn Sie Module als global in Ihrer Entwicklungsumgebung eingetragen hatten, kann es sein, dass die Abhängigkeit nicht drinsteht! Testen Sie das vor Ihrer Abgabe. Falls nicht alle Abhängigkeiten korrekt aufgelöst werden, werden Abzugspunkte berechnet.

### *Start-Skript anlegen*

Durch den Befehl „npm start“ kann die Anwendung gestartet werden. Dazu muss aber das Start-Skript in der package.json definiert werden. Definieren Sie in der package.json-Datei das Start-Skript. Der Server muss mit dem Befehl „npm start“ gestartet werden können. Das kann beispielsweise wie folgt gemacht werden:

```
"scripts": {
    "start": "node ./HttpServer.js"
  },
```

Wenn das Startskript nicht definiert wurde, werden dafür Abzugspunkte berechnet.

## ÜBUNGSLEISTUNGEN

Zum Bestehen der Übung müssen die folgenden Übungsleistungen erbracht werden:

- Meilenstein 1
- Meilenstein 2
- Meilenstein 3 (finale Abgabe)
- Abnahmetest

Für jede dieser Übungsleistung gibt es im Moodle ein weiteres Dokument, in dem die Vorgaben weiter beschrieben und Hilfestellungen gegeben werden.

# Viel Erfolg bei der Umsetzung!