FH MÜNSTER
University of Applied Sciences

MSB FB Wirtschaft
Münster School of Business

Web-Entwicklung II

Wintersemester 2017/18

Übungen zum Kapitel 5

Prof. Dr. Norman Lahme-Hütig
Informatik/Wirtschaftsinformatik, Schwerpunkt Web Engineering

Corrensstraße 25
D-48149 Münster

fon +49 (0)251.83 65-190
fax +49 (0)251.83 65-525

norman.lahme-huetig@fh-muenster.de
www.fh-muenster.de

Übung 1

Aufgaben: Hashing von Passwörtern

1. Erweitern Sie die von Ihrem Dozenten zur Verfügung gestellte Version der Anwendung `express-tm` wie folgt
   - Extrahieren Sie die Eigenschaften `id` und `createdAt` des Task-Interfaces in ein Oberinterface Entity (Datei: `src/models/entity`)
   - Erstellen Sie in der Datei `src/models/user.ts` ein Interface `User` als Subinterface von Entity mit den Eigenschaften `name` (`string`), `email` (string) und `password` (`string`)
   - Fügen Sie in der Datei `src/app.ts` dem Objekt `app.locals` die Eigenschaft `userDAO` hinzu und initialisieren Sie diese mit einem geeignetem Objekt der Klasse `GenericDAO`
   - Erweitern Sie die Datei `src/routes/users.ts`, sodass bei einem POST an die URL `/users` ein neues Objekt, das konform zum Interface User ist, in der DB abgelegt wird und anschließend ein Redirect zu `/tasks` erfolgt

Tipp
Orientieren Sie sich an der Datei `src/routes/tasks.ts`

Übung 1

Aufgaben: Hashing von Passwörtern

2. Testen Sie die Anwendung, indem Sie einen Benutzer registrieren und anschließend mit der MongoDB Shell überprüfen, ob ein passendes Dokument in der `users`-Collection angelegt wurde
3. Ändern Sie die Datei `src/routes/users.ts`, sodass beim Anlegen eines neuen Benutzers dessen Passwort mit bcrypt gehasht wird
   - Hierzu sind die Node.js-Paket `bcryptjs` und `@types/bcryptjs` zu installieren
   - Verwenden Sie bitte die asynchrone Variante zur Hash-Erstellung
4. Löschen Sie die DB und testen Sie die Anwendung erneut. Ermitteln Sie anschließend mit der MongoDB Shell, wie das Passwort nun in der DB abgelegt wird.

Übung 1

Aufgaben: Hashing von Passwörtern

5. Erweitern Sie die Datei src/routes/users.ts, sodass bei einem POST an die URL /users/signin folgendes passiert
   - Es wird ermittelt, ob ein User mit der angegebenen E-Mail in der DB existiert. Dazu ist die Klasse GenericDAO um folgende Methode zu erweitern

```
findOne(entityFilter: Partial<T>, cb: DAOCallback<T>) {
   this.db.collection(this.collection).findOne(
     entityFilter as object,
     (err, result) => { cb(err, result); });
 }
```

   - Falls der User existiert und das Passwort mit dem aus der DB übereinstimmt, soll zur Seite `/tasks` weitergeleitet werden. In allen anderen Fällen soll eine Weiterleitung an die URL `/users/signin` erfolgen. Verwenden Sie zur Überprüfung des Passworts die asynchrone Variante (`bcrypt.compare`)

Übung 2

Aufgaben: JSON Web Token

1. Erweitern Sie die Anwendung um die Generierung von JSON Web Tokens
   - Bei einer erfolgreichen Anwendung soll ein JWT erzeugt und im Cookie mit dem Namen jwt-token abgelegt werden

     ```
     res.cookie('jwt-token', token);
     ```

     – Das JWT soll die ID, den Namen und die E-Mail des Benutzers enthalten
     – Es soll asynchron erzeugt werden
   - Bei einer erfolglosen Anmeldung soll hingegen ein ggf. vorhandenes jwt-token-Cookie wieder entfernt werden

     ```
     res.clearCookie('name');
     ```

Übung 2

Aufgaben: JSON Web Token

2. Erweitern Sie die Anwendung um die Validierung von JSON Web Tokens, indem Sie in der Datei `src/app.ts` eine Middleware hinzufügen

```
app.use(cookieParser());
app.get('/', (req, res) => { res.redirect('/tasks'); });
app.use('/users', users);
app.use((req, res, next) => {
  // Hier bitte die Middleware implementieren
});
app.use('/tasks', tasks);
```

Achtung
Beachten Sie die Reihenfolge der `app.get()`- und `app.use()`-Aufrufe

- Falls das Cookie `jwt-token` nicht gesetzt ist oder kein gültiges Token enthält, dann soll eine Weiterleitung an `/users/signin` erfolgen
- Um das Cookie auslesen zu können, sind die Node.js-Paket `cookie-parser` und `@types/cookie-parser` zu installieren

Übung 3

Aufgaben: Promises

1. Stellen Sie Ihre Klasse GenericDAO* auf Promises um
   - Ändern Sie zur Unterstützung von Promises zunächst in der Datei `tsconfig.json` den Wert der Eigenschaft `target` auf es6 (oder höher)
   - Bei jeder Methode ist der jeweils letzte Parameter (Callback-Funktion) zu entfernen und stattdessen ein Promise-Objekt zurückzuliefern
   - Bei den Methoden `findOne` und `findAll` können Sie ausnutzen, dass der MongoDB-Treiber schon Promises unterstützt

```
findOne(entityFilter: Partial<T>) {
  return this.db.collection(this.collection)
    .findOne(entityFilter as object) as Promise<T>;
}
```

1. Passen Sie nun die Aufrufe der GenericDAO-Methoden an (Datei `src/routes/tasks.ts` und `src/routes/users.ts`)

*Datei src/models/generic.dao.ts

Übung 3

Aufgaben: Promises

3. Integrieren Sie die von Ihrem Dozenten bereitgestellte Datei `auth.service.ts`
   - Legen Sie diese im Verzeichnis `src/services` ab
   - Passen Sie die Datei `src/routes/users.ts` so an, dass die Methoden des `authService`-Objekts verwendet werden
4. Modifizieren Sie die Datei `src/routes/users.ts` wie folgt
   - Markieren Sie die Callbacks der beiden `router.post()`-Aufrufe mit dem Schlüsselwort `async`
   - Verwenden Sie innerhalb der Callback-Implementierungen den `await`-Operator

Übung 4
Vorwort

- Was würden Sie an der Anwendung express-tm als nächstes ändern?
  - Was fehlt funktional?
  - Welche Änderungen sind dafür nötig?

Übung 4

Aufgaben: Erweiterungen

1. Erweitern Sie die Anwendung, sodass ein Benutzer sich abmelden kann
2. Erweitern Sie den Registrierprozess, sodass nach erfolgreicher Registrierung der Benutzer direkt angemeldet ist
3. Erweitern Sie die Anwendung, sodass jeder Benutzer nur auf die von ihm erstellten Aufgaben zugreifen kann

Übung 5

Aufgaben: WebSockets

1. Übernehmen Sie die von Ihrem Dozenten bereitgestellten Dateien in Ihr Projekt und betrachten Sie diese
   - `src/app.ts`
   - `src/ws-server.ts`
   - `src/routes/tasks.ts`
2. Erweitern Sie die Anwendung, sodass der Client eine WebSocket-Verbindung zum Server aufbaut und als Reaktion auf eine WebSocket-Nachricht die aktuelle Seite neu lädt
   - Dies soll jedoch nur geschehen, wenn die URL `/tasks` aufgerufen wird
   - Testen Sie die Anwendung mit zwei Browsern und demselben Benutzerkonto