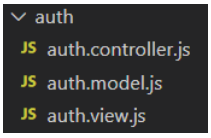
Exkurs: Authentifizierung in Node.js durch Passport

In Express kann die *Passport-Middleware* (`npm install passport`) verwendet werden, um eine Web-Anwendung abzusichern und nur authentifizierten Benutzern gewisse Bereiche der Anwendung zugänglich zu machen. Der eigentliche Authentifizierungsmechanismus von Passport muss als Plugin (`npm install passport-local`) zusätzlich installiert werden. Dabei kann z.B. eine Authentifizierung mittels Facebook, Twitter oder lokaler Datenbank gewählt werden.

Da die Benutzerinformationen automatisch Session-bezogen von Request zu Request mitgeführt werden, muss in Express das *Sessionmanagement* (`npm install express-session`) aktiviert werden.

Anhand von Passport kann für jede Route kontrolliert werden, ob der Benutzer sich authentifiziert hat, und bei fehlender Authentifizierung kann das Login-Formular aufgeworfen werden. So können passwortgeschützte Routen realisiert werden. Dazu muss ebenfalls mit `npm install connect-ensure-login` die entsprechende Middleware installiert werden.

Die Authentifizierungskomponente wird nach dem MVC-Pattern erstellt. Dabei übernimmt die View die Ausgabe des Login-Formulars und das Modell die Bereitstellung der Benutzerinformationen.



Der Controller wird in `index.js` eingebunden:

index.js

```
const authController = require('./auth/auth.controller');
authController(app);
```

WICHTIG: Der `authController` muss vor dem `movieRouter` in `index.js` und nach dem *Body-Parser* importiert werden.

Der Controller wird folgendermaßen realisiert:

/auth/auth.controller.js

```
const passport = require('passport');
const expressSession = require('express-session');
const LocalStrategy = require('passport-local');

const authView = require('./auth.view');
const authModel = require('./auth.model');

module.exports = app => {

  app.use(
    expressSession({
      secret: 'top secret',
      resave: false,
      saveUninitialized: false
    })
  );
```

Dadurch wird in Express das Sessionmanagement über Cookies aktiviert. Das Session-Cookie wird über das Passwort in `secret` signiert. Mit `resave = false` wird verhindert, dass unveränderte Sessions neu gespeichert werden. Mit `saveUninitialized = false` wird verhindert, dass neue Sessions bei denen sich der Benutzer noch nicht eingeloggt hat, am Server nicht gespeichert werden.

```
  passport.serializeUser((user, done) => done(null, user.username));

  passport.deserializeUser((username, done) => {
    const user = authModel.get(username);
    if (user) {
      user.password = '';
      done(null, user);
    } else {
      done(null, false);
    }
  });
```

Diese beiden Methoden werden aufgerufen, wenn der Benutzer dem Session-Cookie zugeordnet werden soll (`serializeUser`) bzw. wenn vom Session-Cookie der Benutzer wiederhergestellt werden soll (`deserializeUser`). Beim Serialisieren wird für den Benutzer eine eindeutige Kennung (`username`) verwendet und

diese in das Session-Cookie eingetragen. Beim Deserialisieren werden aus dieser Id die Benutzerdaten geholt. Die *Callback-Funktion* done() verarbeitet Daten indem sie diese an die Session weitergibt. Wird ihr als zweiter Parameter `false` übergeben, bedeutet dies, dass die Benutzerdaten nicht gefunden werden konnten. In diesem Fall reagiert done() darauf.

WICHTIG: Bei erfolgreicher Deserialisierung enthält der Request das Objekt `user` mit den geholten Eigenschaften. Auf diese kann dann in der Web-Anwendung über `request.user.username`, `request.user.firstname`, `request.user.lastname` zugegriffen werden.

```
app.use(passport.initialize());
app.use(passport.session());
```

`initialize()` initialisiert Passport und schafft so die Verbindung zwischen Express und Passport. `session()` sorgt dafür, dass die Anmeldesessions mithilfe der Express-Session-Middleware gespeichert werden.

```
passport.use(
  new LocalStrategy((username, password, done) => {
    const user = authModel.get(username);
    if (user && user.password === password) {
      user.password = '';
      done(null, user);
    } else {
      done(null, false);
    }
  })
);
```

Der Kontruktor der Klasse `LoacalStrategy` erhält als Parameter eine *Callback-Funktion*, die den übergebenen Benutzernamen und Passwort validiert und das Ergebnis wiederum an eine weitere *Callback-Funktion* (done()) weiterreicht, welche diese dann verarbeitet.

```
app.get(
  '/login',
  (request, response) => response.send(authView.login(request.query.error))
);
```

Wird die Route `'/login'` über `GET` aufgerufen, so wird das Login-Formular zur Eingabe von Benutzername und Passwort an den Browser geschickt (siehe `auth.view.js`). Bei Falscheingaben wird das Login-Formular nochmals angesprungen und dabei eine Fehlermeldung (`request.query.error`) angezeigt.

```
app.post(
  '/login',
  passport.authenticate(
    'local',
    { keepSessionInfo: true, failureRedirect: '/login?error=Notallowed' }),
    (request, response) =>
      response.redirect(request.session.returnTo ? request.session.returnTo : '/'
  )
);
```

Wird die Route `'/login'` über `POST` aufgerufen, bedeutet dies, dass das Formular mit eingegebenem Benutzernamen und Passwort ausgewertet werden soll. Bei fehlerhafter Authentifizierung wird das Login-Formular nochmals aufgerufen wobei als Parameter der Fehler übergeben wird. Bei erfolgreicher Authentifizierung wird auf die Route gesprungen, welche der Benutzer ursprünglich anspringen wollte (`request.session.returnTo`).

```
app.get(
  '/logout',
  (request, response) => {
    request.session.returnTo = undefined;
    request.logout(error => error ? next(error) : response.redirect('/'));
  }
);
};
```

Wird die Route `'/logout'` aufgerufen, so wird die Session zerstört und auf die Hauptseite weitergeleitet.

/auth/auth.model.js

```
const users = [
  {username: 'sepp', password: 'sepp', firstname: 'Sepp', lastname: 'Hintner' },
  {username: 'resi', password: 'resi', firstname: 'Resi', lastname: 'Rettich' },
```

```
  {username: 'rudi', password: 'rudi', firstname: 'Rudi', lastname: 'Rüpel' }
];

function get(username) {
  const user = users.find(user => user.username === username);
  return user ? Object.assign({}, user) : null;
}

module.exports = { get };
```

Der Benutzer wird geklont.

/auth/auth.view.js

```
function login(error) {
  return `
    <!DOCTYPE html>
    <html lang="en">
      <head>
        <meta charset="UTF-8">
        <title>Filmliste: Login</title>
      </head>
      <body>
        <p>
          ${error ? 'Benutzername und/oder Passwort falsch' : ''}
        </p>
        <form action="/login" method="POST">
          <div>
            <label for="username">Benutzername:</label>
            <input type="text" id="username" name="username" value="sepp" autofocus>
          </div>
          <div>
            <label for="password">Passwort:</label>
            <input type="password" id="password" name="password" value="sepp">
          </div>
          <input type="submit" value="Anmelden">
        </form>
      </body>
    </html>`;
}

module.exports = { login };
```

Definition passwortgeschützter Routen

Beim Aufruf bestimmter Routen kann über die *Middleware* `connect-ensure-login` kontrolliert werden, ob der Benutzer sich authentifiziert hat und bei Bedarf automatisch das Login-Formular aufgerufen werden. Hier wird beispielsweise der Movie-Router folgendermaßen angepasst, damit nur authentifizierte Benutzer Filme neu anlegen, ändern und löschen können:

/movie/movie.router.js

```
…
const { ensureLoggedIn } = require('connect-ensure-login');

…
router.get('/', listAction);
router.get('/remove/:id', ensureLoggedIn('/login'), removeAction);
router.get('/edit/:id?', ensureLoggedIn('/login'), editAction);
router.post('/save', ensureLoggedIn('/login'), saveAction);

…
```