LU06c - Umsetzung als Beispiel

Das Beispiel zeigt eine Umsetzung der Autorisation in einer RESTful Flask Applikation.

Frontend

Token speichern

Das Token wird vom Webserver als Text gesendet. Falls mein Login-Request erfolgreich war, speichere ich diesen Test im Session Storage.

```
fetch("./login",
    {
        ...
    .then(function (response) {
        if (response.ok) {
            return response.text();
        } else {
            ...
        }
    })
    .then(function (data) {
        sessionStorage.setItem("token", data);
    })
```

Token senden

```
async function httpFetch(
    url,
    httpMethod = "GET",
    token
) {
    response = await fetch(url, {
                method: httpMethod,
                headers: {
                    "Authorization": "Bearer " + token
                }
            });
    ...
}
```

Dieses Beispiel zeigt einen asynchronen Aufruf eines Webservices mit GET.

Backend

Das Backend der Nachprüfungsapplikation kennt drei unterschiedliche Rollen:

- Keine gültige Anmeldung: Der Client hat kein gültiges Token.
- Lernender: Ein gültiges Token ist vorhanden, die Rolle ist „user“
- Lehrperson: Ein gültiges Token ist vorhanden, die Rolle ist „teacher“

ExamService

Die einzelnen Services werden mit eigenen Decorators gekennzeichnet.

- @token_required: Prüft ob ein gültiges Token vorhanden ist.
- @teacher_required: Prüft zusätzlich ob der Benutzer die Rolle „teacher“ hat.

```
class ExamService(Resource):
    @token_required
    def get(self, exam_uuid):
        ...

    @token_required
    @teacher_required
    def post(self):
        ...
```

In diesem Beispiel darf jeder angemeldete Benutzer einen einzelnen Eintrag lesen (GET). Um einen Eintrag zu speichern (POST) muss der Benutzer die Lehrer-Rolle haben.

authorization.py

In diesem Modul sind die beiden Decorators programmiert.

```
def token_required(func):
    """
    checks if the authorization token is valid
    :param func: callback function
    :return:
    """

    @wraps(func)
    def decorator(*args, **kwargs):
        token = None
        if 'Authorization' in request.headers:
            token = request.headers['Authorization']
```

```
        if not token:
            return make_response(jsonify({"message": "A valid token is
missing!"}), 401)
        try:
            data = jwt.decode(token[7:],
current_app.config['ACCESS_TOKEN_KEY'], algorithms=["HS256"])
            email = data['email']
            person_dao = PersonDAO()
            g.user = person_dao.read_person(email)
        except Exception:
            return make_response(jsonify({"message": "EXAM/auth: Invalid
token!"}), 401)

        return func(*args, **kwargs)

    return decorator


def teacher_required(func):
    @wraps(func)
    def wrap(*args, **kwargs):
        if not g.user.role == 'teacher':
            return make_response(jsonify({"message": "not allowed for
students"}), 401)
        return func(*args, **kwargs)

    return wrap
```

Die Funktion `token_required` prüft zunächst, ob `Authorization`-Header im Request vorhanden ist. Die Daten in diesem Header werden mit `jwt.decode` entschlüsselt und die Signatur geprüft. Falls das Token gültig ist, wird der entsprechende User gelesen und in der globalen Collection g gespeichert.

Die Funktion `teacher_required` prüft lediglich die Rolle des angemeldeten Benutzers.

g steht für global

Die Variable g steht für „global“. Dabei handelt es sich um eine Collection, die während eines Requests in allen Services verfügbar ist. Dadurch können relativ einfach zentrale Daten, wie der angemeldete Benutzer, zwischen verschiedenen Funktionen ausgetauscht werden.

M321-LU06