LU08.L03 - Bookshelf mit Authentifikation

Musterlösung zu LU08.A03 - Bookshelf mit Authentifikation (Postman). Die Collection wird im Runner mit **3 Iterationen** ausgeführt. In jeder Iteration bestimmt `01 Initialize` die nächste Rolle (`admin` → `user` → `guest`) und springt zum passenden Login. Danach laufen alle Bücher-Requests ab `10 Read Book`.

Ablauf

Iteration	role	Login-Request	token	erwartet bei read/list	erwartet bei save/delete
1	admin	02 Login admin	gesetzt	200	200/201/400/404/422 wie bisher
2	user	03 Login user	gesetzt	200	403
3	guest	04 Login invalid (401)	leer	403	403

01 Initialize (Pre-request Script)

```
// Nächste Rolle bestimmen: admin -> user -> guest -> admin ...
const order = ["admin", "user", "guest"];
const current = pm.environment.get("role");
const index = order.indexOf(current);
const role = order[(index + 1) % order.length];   // unbekannt/leer -> admin

pm.environment.set("role", role);
pm.environment.unset("token");                     // kein Token aus der letzten Iteration

const login = {
    admin: "02 Login admin",
    user: "03 Login user",
    guest: "04 Login invalid"
};
pm.execution.setNextRequest(login[role]);
```

Vor dem ersten Durchlauf muss `role` leer sein (oder `guest`), damit die erste Iteration mit `admin` beginnt. In älteren Postman-Versionen heisst der Aufruf `postman.setNextRequest(…)` und funktioniert nur im Reiter *Tests*.

02 Login admin / 03 Login user (Tests)

```
pm.test("Login erfolgreich (200)", function () {
    pm.response.to.have.status(200);
});
```

```
const data = pm.response.json();
pm.environment.set("token", data.token);

pm.execution.setNextRequest("10 Read Book");
```

04 Login invalid (Tests)

```
pm.test("Login abgelehnt (401)", function () {
    pm.response.to.have.status(401);
});

pm.execution.setNextRequest("10 Read Book");
```

Bücher-Requests: Tests nach Rolle

Im Header jedes Requests steht `Authorization: Bearer {{token}}`.

10 Read Book

```
const role = pm.environment.get("role");

if (role === "admin" || role === "user") {
    pm.test("[" + role + "] Status 200", function () {
        pm.response.to.have.status(200);
    });
    pm.test("[" + role + "] richtiges Buch", function () {
        const book = pm.response.json();
pm.expect(book.book_uuid).to.eql("c746a291-0ef9-4b2a-8268-392b12d636bd");
    });
} else {
    pm.test("[guest] Status 403", function () {
        pm.response.to.have.status(403);
    });
    pm.test("[guest] leere Antwort", function () {
        pm.expect(pm.response.json()).to.eql([]);
    });
}
```

List Books

Gleich aufgebaut wie Read Book: `admin` und `user` erwarten 200 und ein Array mit 3 Büchern, `guest` 403.

Save / Delete

Nur `admin` darf schreiben. Für alle anderen Rollen ist 403 richtig, unabhängig von den gesendeten Daten:

```
const role = pm.environment.get("role");
const expected = 201;            // je nach Request: 200, 201, 400, 404, 422

if (role === "admin") {
    pm.test("[admin] Status " + expected, function () {
        pm.response.to.have.status(expected);
    });
} else {
    pm.test("[" + role + "] Status 403", function () {
        pm.response.to.have.status(403);
    });
}
```

Die Berechtigung muss vor der Validierung geprüft werden. Liefert die API einem `user` bei ungültigen Daten 422 statt 403, ist das ein Fehler in der API: Sie verrät einem Benutzer ohne Schreibrecht, welche Daten gültig wären.

Ausführen

Collection Runner → **Iterations: 3** → Run. In jeder Iteration erscheint genau ein Login-Request, die beiden anderen werden übersprungen. Alle Tests sind grün.

Da `Delete` in der ersten Iteration (admin) das Buch löscht, erwartet der zweite Durchlauf beim `admin` 404. Im Runner läuft `admin` aber nur einmal, darum stimmt die Reihenfolge `admin` → `user` → `guest`.

M450-LU08, M450-C3I, M450-B1I

Marcel Suter

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/modul/m450/learningunits/lu08/loesungen/automation

Last update: **2026/11/17 17:53**