

Kontakt: Sabrina.Bayer@bibliothek.uni-regensburg.de

FOLIO Praxistage in Bamberg

POSTMAN und FOLIO-API: Hands-on Lab für Bibliothekar*innen (Dienstag, 01.10.2024)

Vor dem Hands-on Lab durch die Teilnehmenden zu erledigen:

Installieren Sie sich die Postman-App auf Ihrem Laptop und erstellen Sie sich einen kostenlosen Postman-Account. Installationsanleitung:

<https://www.postman.com/downloads/>

Bringen Sie Ihren Laptop mit zum Hands-on Lab (vorzugsweise mit genügend Akku 😊)

Zur Info: Postman kann man nicht nur mit der Desktop-App, sondern auch im Browser verwenden. Sollte die Installation der App nicht funktionieren, sollte es auch möglich sein, die Übungen über den Browser zu machen.

Hilfreiche Links

- FOLIO Wiki: Entwicklerwerkzeuge im Browser (Chrome): <https://folio-org.atlassian.net/wiki/spaces/FOLIOTips/pages/5673541/Intro+to+Developer+Tools+in+Google+Chrome>

- FOLIO Wiki: Working with FOLIO APIs: <https://folio-org.atlassian.net/wiki/spaces/FOLIOTips/pages/5672571/Working+with+FOLIO+APIs>

- FOLIO Wiki: Getting started with Postman: <https://folio-org.atlassian.net/wiki/spaces/FOLIOTips/pages/5672500/Getting+started+with+Postman>

- FOLIO: API Reference: <https://dev.folio.org/reference/api/>

- FOLIO Slack-Channel: Learning APIs: <https://folio-project.slack.com/archives/CQ7EK52LB>

- Learning Center von Postman:
<https://learning.postman.com/docs/introduction/overview/>

Verwendete FOLIO-Installation:

FOLIO Snapshot: <https://folio-snapshot.dev.folio.org/>

Username: diku_admin

Passwort: admin

1. Was ist Postman?

2. Was braucht man, um mit der FOLIO API zu arbeiten und wo findet man diese Informationen?

- a. x-okapi-tenant -> Einstellungen-App / Systeminformation / Installationsdetails / Okapi
- b. x-okapi-token -> über Request anfragen
- c. Okapi-URL -> Einstellungen-App / Systeminformation / Installationsdetails / Okapi
- d. API endpoint -> verschiedene Möglichkeiten

3. Eine erste API Anfrage in Postman: Token abrufen

POST

URL inkl. Endpoint: <https://folio-snapshot-okapi.dev.folio.org/authn/login>

Headers: x-okapi-tenant -> diku

Body (JSON):

```
{  
  "username": "diku_admin",  
  "password": "admin"  
}
```

4. Wo findet man Informationen zu den FOLIO endpoints?

Möglichkeit 1: Im Browser Entwicklerwerkzeuge mit F12 aufrufen

In Chrome: auf Netzwerk klicken (evtl. auf deutsch umstellen) und "Fetch/XHR" auswählen (macht es übersichtlicher).

In Firefox: auf Netzwerkanalyse klicken

Nun die gewünschte Aktion ausführen (z.B. Kostenart neu anlegen).

Anschließend auf die entsprechende Zeile in der Spalte "Name" klicken, dann geht ein weiteres Fenster auf. Hier hat man unter Allgemein / Anfrage URL die entsprechenden Informationen.

Kontakt: Sabrina.Bayer@bibliothek.uni-regensburg.de

Möglichkeit 2: In der FOLIO API Dokumentation nachschauen:

<https://dev.folio.org/reference/api/>

Möglichkeit 3: Okapi Path mapper unter Einstellungen / Entwickler / Okapi Pfad-Mapper

5. API Anfrage: Kostenart in FOLIO anlegen

Body (JSON):

```
{  
  "code" : "{{code}}",  
  "externalAccountNumberExt" : "{{externalAccountNumberExt}}",  
  "name" : "{{name}}"  
}
```

6. Tags löschen

Um Tags zu löschen, wird zuerst die UUID des Tags benötigt, der gelöscht werden soll.

Hierzu muss folgende Abfrage erfolgen:

GET

URL inkl Endpoint: <https://folio-snapshot-okapi.dev.folio.org/tags>

Headers: x-okapi-token und x-okapi-tenant

Als Ergebnisliste erhält man hier nun alle Tags, die in diesem Mandanten vorhanden sind. Hier sucht man sich nun den Tag, der gelöscht werden soll, inkl. Seiner UUID.

Die UUID wird für die nächste Abfrage benötigt:

DELETE

URL inkl. Endpoint und UUID des Beispiel-Tags: <https://folio-snapshot-okapi.dev.folio.org/tags/1fafa36e-226b-4ff8-9b0c-e412571b528e>

Headers: x-okapi-token und x-okapi-tenant

Schickt man diese Abfrage ab, wird der Tag in diesem Mandanten gelöscht.

Allgemeines ACHTUNG:

Man kann mit API-Abfragen (sowohl mit POST, PUT und DELETE) auch seinen FOLIO-Mandanten schrotten.

Kontakt: Sabrina.Bayer@bibliothek.uni-regensburg.de

Es gibt Business-Logic-Endpoints (hier wird die Logik überprüft bzw. auch weitere Aktionen erzeugt) und es gibt Storage-Endpoints (hier greift man direkt auf die Daten zu). Bitte dies bei der Arbeit mit den API-Endpoints beachten.

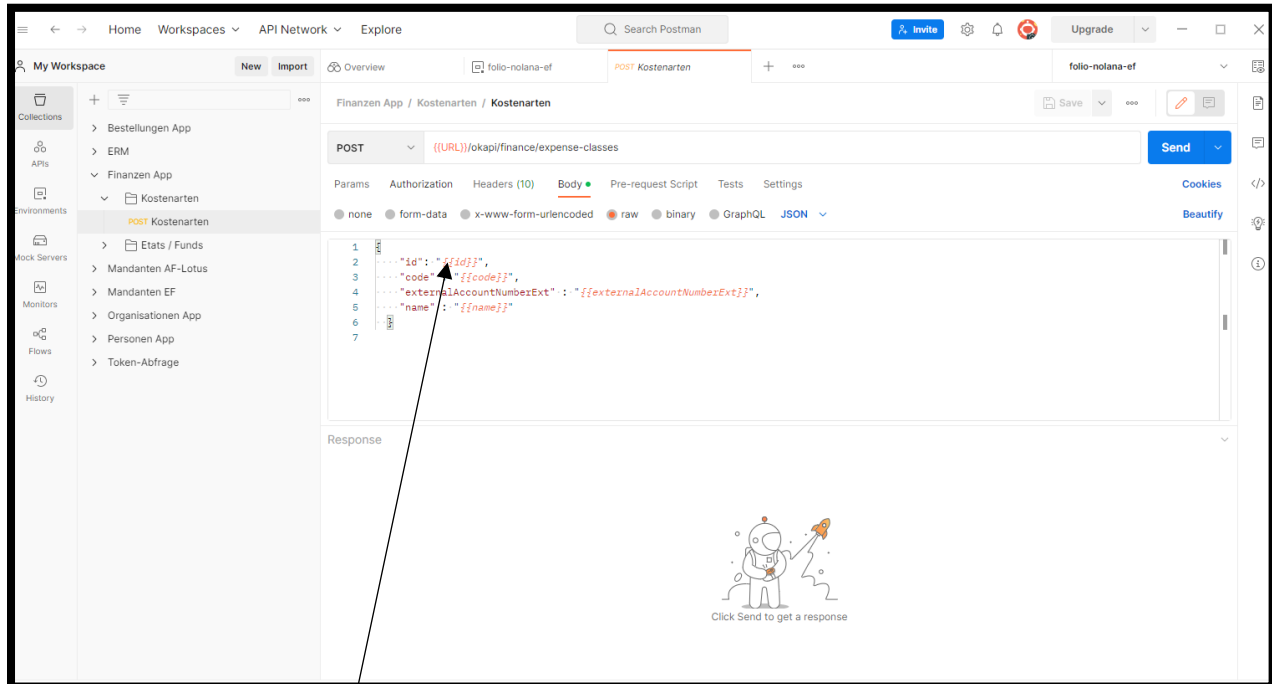
Postman: Collection Runner

Um mehrere Datensätze/Objekte, z.B. eine JSON-Datei mit verschiedenen Kostenarten, in einem Schritt über Postman in FOLIO zu laden, hat man mehrere Möglichkeiten mit dem Collection Runner.

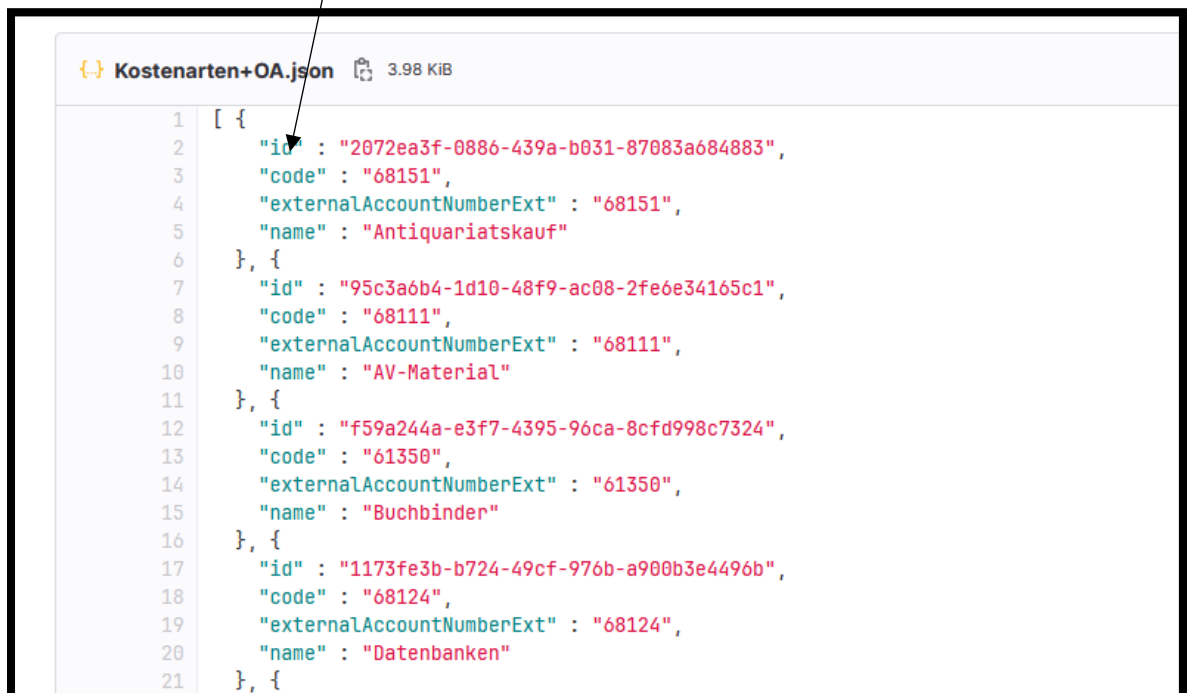
Möglichkeit 1: Collection Runner mit Templates

1. Vorbereitung

Man legt den Request (mit Headern+URL) an und trägt im Body ein Template ein. Speichern nicht vergessen.

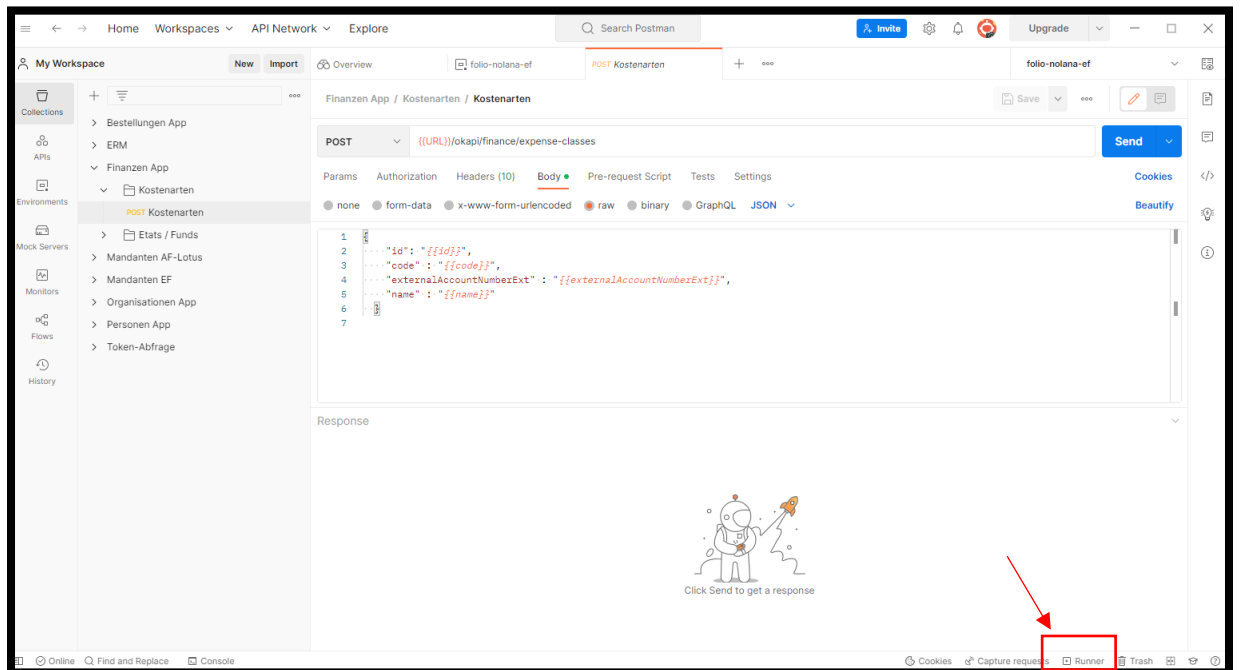


Die Variablen in den geschweiften Klammern entsprechen den Schlüsseln der JSON-Datei (bzw. den Spaltenüberschriften eines Tabellenformats, bevor man diese in eine JSON-Datei umwandelt).



2. Runner aufrufen

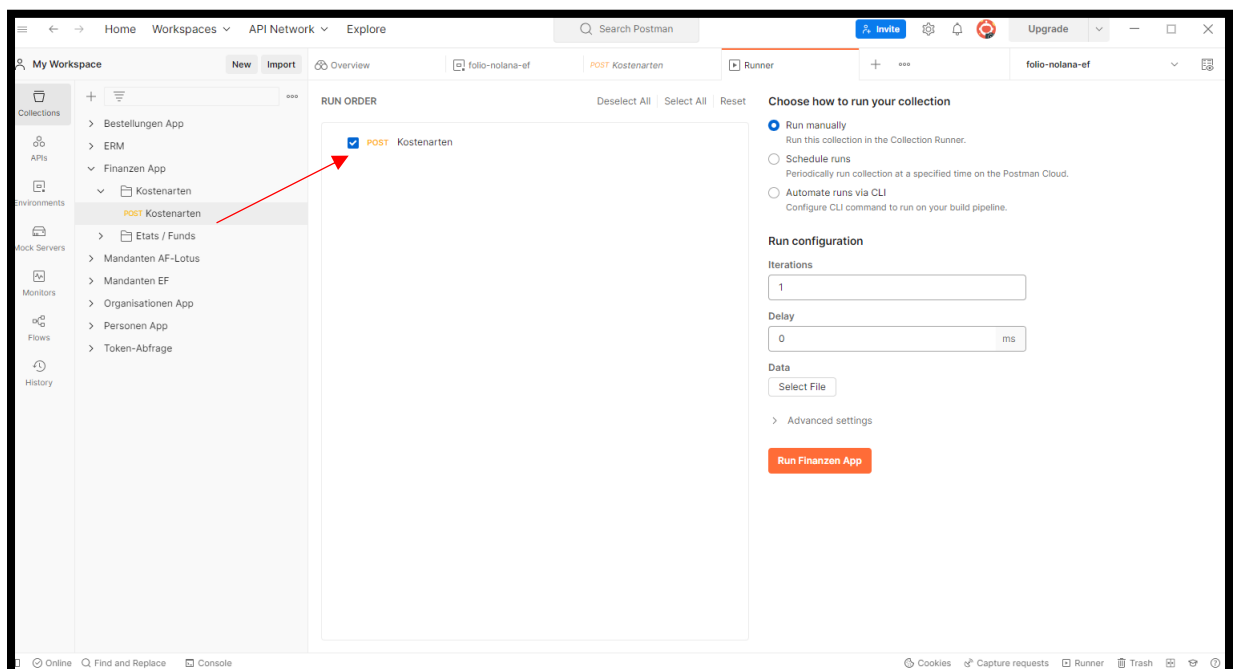
Um den Runner zu starten, klickt man rechts unten in der Leiste auf „Runner“.



Anschließend geht ein neuer Tab in Postman auf.

3. Runner: Request auswählen

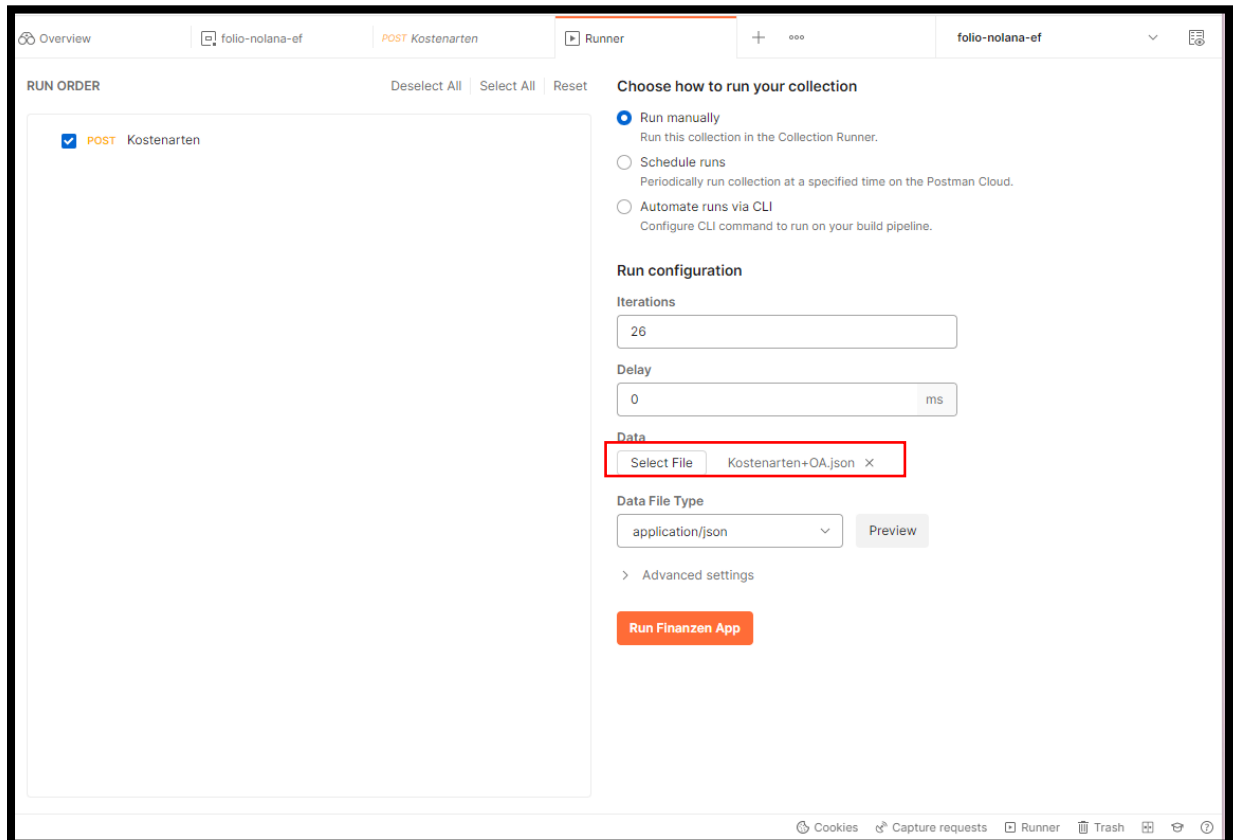
In dem neuen Tab zieht man nun mittels Drag&Drop die Collection/Ordner in das linke Feld, in dem sich der gewünschte Request befindet. Bei mehreren Requests, hakt man nur den Request an, den man ausgeführt haben möchte.



4. Runner: Datei auswählen

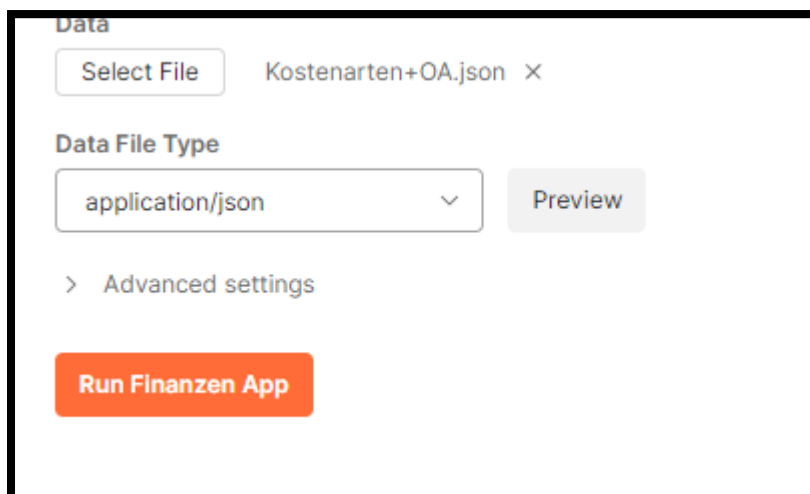
Anschließend wählt man über „Select file“ die gewünschte Datei aus. Postman zeigt auch gleich an, wie viele Iterations er mit diesem Request machen wird (d.h. wie oft das Request ausgeführt wird um alle Datensätze/Objekte aus der JSON-Datei hochzuladen). In unserem Fall wird Postman 26 Kostenarten hochladen.

Hinweis: Sollte etwas mit der JSON-Datei formal nicht stimmen, zeigt Postman in manchen Fällen hier bereits eine Fehlermeldung an.



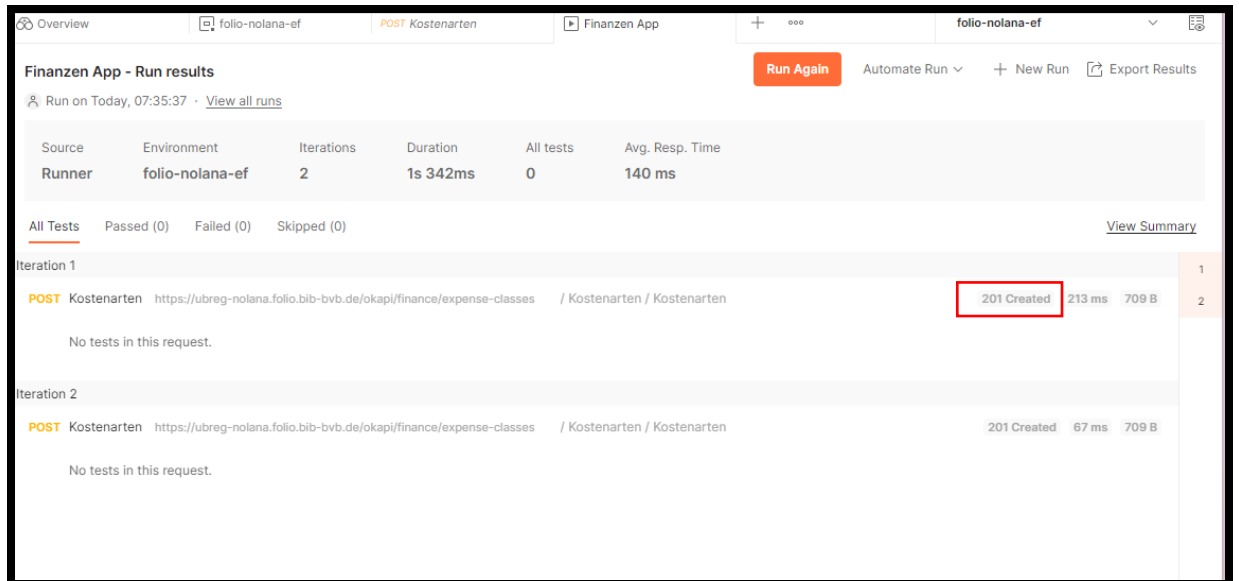
5. Anschließend auf „Run“ klicken

(bei mir heißt die Collection „Finanzen App“, deshalb zeigt Postman „Run Finanzen App“ in diesem Screenshot an).



6. Ergebnis:

Wenn alles gut läuft, wird für jeden der 26 Requests das Ergebnis „201 created“ angezeigt und in FOLIO wurden die Kostenarten unter Einstellungen / Finanzen / Kostenarten erzeugt. (für die diese Anleitung habe ich nur zwei Test-Kostenarten hochgeladen, deshalb werden hier auch nur 2 iterations angezeigt).



The screenshot shows the 'Run results' for the 'Finanzen App' in Postman. The interface includes a top navigation bar with 'Overview', 'folio-nolana-ef', 'POST Kostenarten', and 'Finanzen App'. Below this, there's a 'Run Again' button and options for 'Automate Run', 'New Run', and 'Export Results'. A summary table shows the following data:

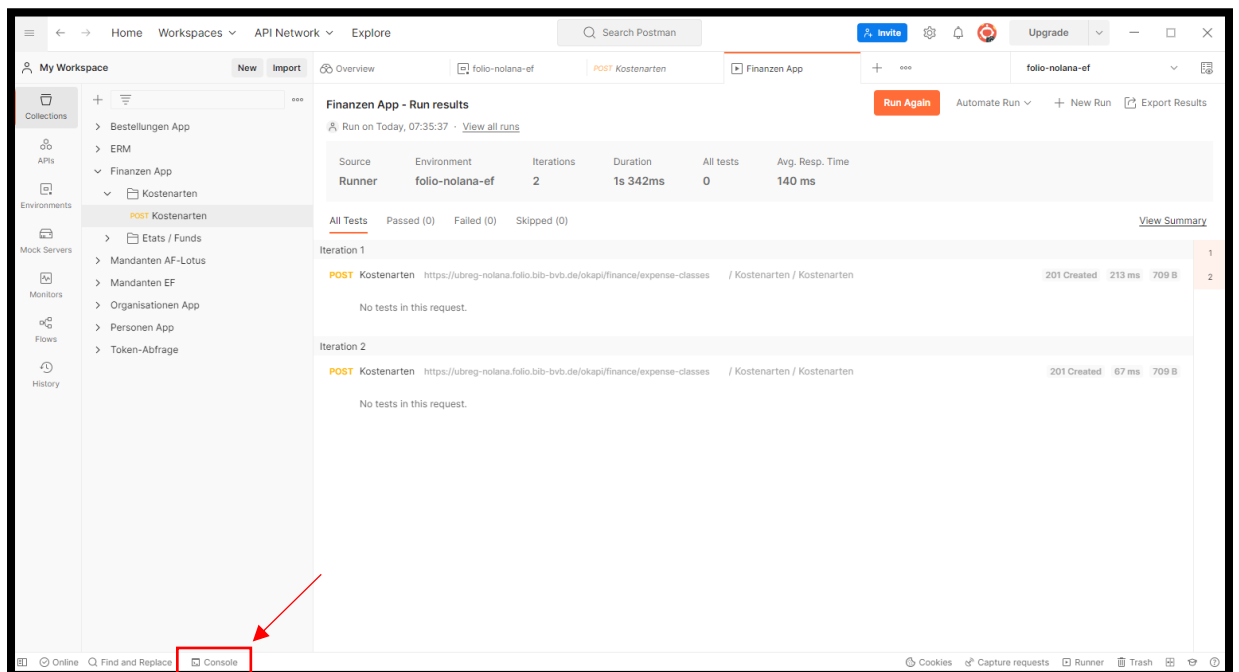
Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	folio-nolana-ef	2	1s 342ms	0	140 ms

Below the summary, there are tabs for 'All Tests', 'Passed (0)', 'Failed (0)', and 'Skipped (0)'. The main content area displays two iterations:

- Iteration 1:** A POST request to 'https://ubreg-nolana.folio.bib-bvb.de/okapi/finance/expense-classes / Kostenarten / Kostenarten' resulted in '201 Created' (highlighted with a red box), 213 ms, and 709 B. Below the request, it says 'No tests in this request.'
- Iteration 2:** A POST request to the same URL resulted in '201 Created', 67 ms, and 709 B. Below the request, it says 'No tests in this request.'

7. Problembehandlung

Unter „Console“ kann man sich die Requests anzeigen lassen und sieht dann vielleicht schon, wo das Problem liegen könnte.



The screenshot shows the Postman interface with the 'Console' tab highlighted by a red arrow. The interface includes a top navigation bar with 'Home', 'Workspaces', 'API Network', and 'Explore'. Below this, there's a 'Search Postman' bar and options for 'Invite', 'Upgrade', and window controls. The main content area displays the 'Run results' for the 'Finanzen App' in the 'Console' view, showing the same summary table and iteration details as in the previous screenshot. The 'Console' tab is highlighted in the bottom navigation bar.

Wenn man auf „Console“ klickt, werden die Requests angezeigt und man kann die einzelnen Elemente aufklappen.

The screenshot shows the Postman interface for a workspace named 'folio-nolana-ef'. The main view displays the 'Run results' for a collection named 'Finanzen App'. The results table shows two iterations, both successful (All Tests Passed). The first iteration took 1s 342ms and the second took 140ms. Below the table, the console view is open, showing two POST requests to the URL 'https://ubreg-nolana.folio.bib-bvb.de/okapi/finance/expense-classes'. The first request has a status of 201 and a response size of 213 bytes. The second request has a status of 201 and a response size of 67 bytes. The console view is currently collapsed, and the two POST requests are highlighted with a red box.

In diesem Fall hatte alles gepasst:

The screenshot shows the Postman console view for a POST request to the URL 'https://ubreg-nolana.folio.bib-bvb.de/okapi/finance/expense-classes'. The request headers include 'x-okapi-tenant: ubreg', 'x-okapi-token: [REDACTED]', 'Content-Type: application/json', 'User-Agent: PostmanRuntime/7.31.1', 'Accept: */*', 'Postman-Token: 93972832-965e-4208-a89a-e219289e29fe', 'Host: ubreg-nolana.folio.bib-bvb.de', 'Accept-Encoding: gzip, deflate, br', 'Connection: keep-alive', and 'Content-Length: 147'. The request body is a JSON object:

```
{ "id": "b6d6c9d1-ad45-4304-9c76-80683923ae39", "code": "test1", "externalAccountNumberExt": "test1", "name": "test1" }
```

. The response headers include 'Date: Fri, 03 Mar 2023 06:35:38 GMT', 'Content-Type: application/json', 'Transfer-Encoding: chunked', 'Connection: keep-alive', 'Location: http://okapi:9130/finance/expense-classes/b6d6c9d1-ad45-4304-9c76-80683923ae39', and 'content-encoding: gzip'. The response body is a JSON object:

```
{ "id": "b6d6c9d1-ad45-4304-9c76-80683923ae39", "code": "test1", "externalAccountNumberExt": "test1", "name": "test1", "metadata": { "createdAt": "2023-03-03T06:35:38.774+00:00", "createdBy/UserId": "1dafece5-8924-4725-bf8c-50f64441ae45", "updatedAt": "2023-03-03T06:35:38.774+00:00", "updatedBy/UserId": "1dafece5-8924-4725-bf8c-50f64441ae45" } }
```

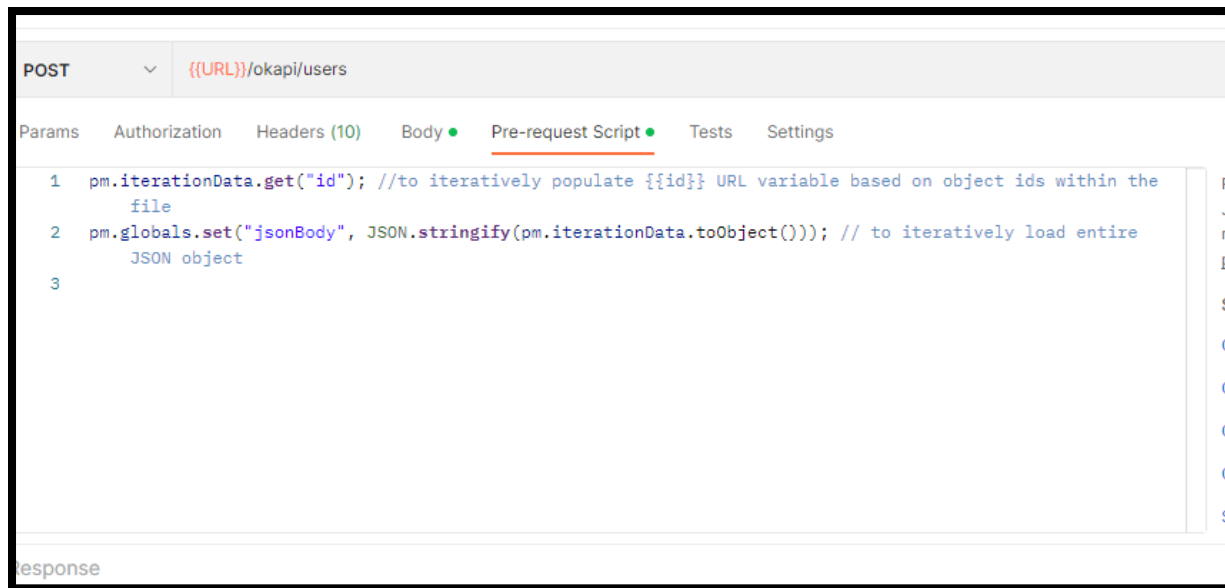
Möglichkeit 1: Collection Runner mit „pre-request scripting“

Diese Anleitung stammt netterweise von deutschen Kollegen.

1. Request anlegen (mit Headern und Endpoints)

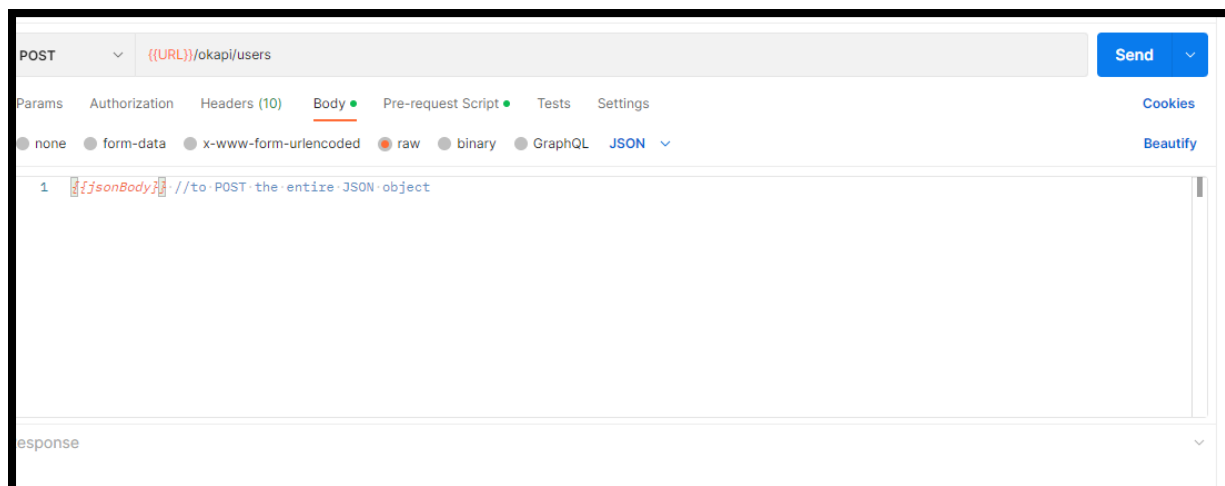
2. Bei "pre-request scripting" folgendes eintragen:

```
pm.iterationData.get("id"); //to iteratively populate {{id}} URL
variable based on object ids within the file
pm.globals.set("jsonBody",
JSON.stringify(pm.iterationData.toObject())); // to iteratively
load entire JSON object
```



3. Bei "body" eintragen:

```
{{jsonBody}} //to POST the entire JSON object
```



4. Request abspeichern

5. Collection Runner (weiteres Vorgehen wie bei Möglichkeit 1, Punkt 2): benötigten POST Request öffnen, JSON-Datei auswählen, „save responses“ und „keep variable values“ anhaken

6. Auf „Run ...“ klicken

Hinweise:

Sollen bereits vorhandene Daten überschrieben werden:
bei einem PUT muss zusätzlich im Pfad /`{{id}}` als Variable stehen.