

LU16b - UPDATE (PUT) & DELETE (DELETE) mit Express und Postman

Lernziele

- Sie können einen bestehenden Post über PUT /api/posts/:id aktualisieren (Update).
- Sie können einen bestehenden Post über DELETE /api/posts/:id löschen (Delete).
- Sie wissen, wie Sie req.params.id (Route-Parameter) und req.body (Body-Daten) verwenden.
- Sie verwenden passende **HTTP-Statuscodes** (200, 400, 404).
- Sie können die Endpoints mit **Postman** testen.

Voraussetzungen

Damit PUT und DELETE funktionieren, muss in Ihrem index.js weiterhin Folgendes vorhanden sein:

- app.use(express.json()); (damit JSON aus dem Request-Body in req.body landet)
- das Array posts mit Beispiel-Daten
- die bereits erstellten Routes GET /api/posts, GET /api/posts/:id und POST /api/posts



Fügen Sie die folgenden Routen **unterhalb** Ihrer bisherigen Routen in index.js ein (die Reihenfolge ist nicht entscheidend, aber Übersicht hilft).

UPDATE - Post aktualisieren (PUT /api/posts/:id)

Was bedeutet UPDATE mit PUT?

Mit PUT aktualisieren Sie einen bestehenden Datensatz. Der Client schickt dazu einen Request an:

- /api/posts/:id → id ist die post_id des Posts, der geändert werden soll.
- Die neuen Werte werden als JSON im Request-Body mitgeschickt.

Code: PUT-Route

```
// UPDATE – vorhandenen Post aktualisieren
app.put('/api/posts/:id', (req, res) => {
  const id = Number(req.params.id);
```

```
// Post suchen nach der ID aus dem Request im posts-Array  
(Liste)  
const post = posts.find(p => p.post_id === id);  
  
if (!post) {  
    return res.status(404).send('Post nicht gefunden');  
}  
  
// Alle Felder werden erwartet (PUT ersetzt alles)  
const user_id = req.body.user_id;  
const title = req.body.title;  
const image_url = req.body.image_url;  
const description = req.body.description;  
const likes = req.body.likes;  
  
// Validierung: sind wirklich alle Felder vorhanden?  
// likes kann 0 sein -> deshalb auf undefined prüfen  
if (  
    user_id === undefined ||  
    title === undefined ||  
    image_url === undefined ||  
    description === undefined ||  
    likes === undefined  
) {  
    return res.status(400).send('Bitte user_id, title,  
image_url, description und likes mitsenden (PUT ersetzt  
alles).');  
}  
  
// Post überschreiben  
post.user_id = user_id;  
post.title = title;  
post.image_url = image_url;  
post.description = description;  
post.likes = likes;  
  
// Antwort: 200 OK + aktualisiertes Objekt  
res.status(200).json(post);  
});
```

Test mit Postman (PUT)

The screenshot shows the Postman interface for a PUT request. The URL is `localhost:3000/api/posts/2`. The method is set to `PUT`. The ID `2` is highlighted with a green arrow pointing from the URL to the method field. The Body tab is selected, showing a raw JSON payload:

```

1 {
2   "title": "Sunset Vibes (updated)",
3   "likes": 99
4 }

```

A green box highlights the JSON body. A green arrow points from the text "Body: Werte, die im Post geändert werden sollen." to the JSON payload. Below the request, the response is shown with status `200 OK` and the message "Resultat: Aktualisierter Post wird zurückgegeben". The response body is also highlighted with a blue box and contains the updated post data:

```

1 {
2   "post_id": 2,
3   "user_id": 2,
4   "title": "Sunset Vibes (updated)",
5   "image_url": "https://picsum.photos/800/450?random=2",
6   "description": "Caught this beautiful sunset today! Nature never disappoints.",
7   "likes": 99
8 }

```

1. Methode: PUT
2. URL: <http://localhost:3000/api/posts/2>
3. Tab Body → raw → JSON
4. Beispiel-Body:

```
{
  "title": "Sunset Vibes (updated)",
  "likes": 99
}
```

Erwartung:

- Status 200 OK
- JSON-Objekt des aktualisierten Posts (`post_id: 2`) mit neuem `title` und `likes`
- Danach GET `/api/posts/2` testen → Änderungen sollten sichtbar sein

DELETE - Post löschen (DELETE /api/posts/:id)

Was bedeutet DELETE?

Mit DELETE entfernen Sie einen Datensatz. Der Client schickt dazu einen Request an:

- DELETE /api/posts/:id

Der Server sucht den passenden Post und entfernt ihn aus der Liste.

Code: DELETE-Route

```
// DELETE – Post löschen
app.delete('/api/posts/:id', (req, res) => {
  const id = Number(req.params.id);

  // Index suchen (praktisch fürs Löschen)
  const index = posts.findIndex(p => p.post_id === id);

  if (index === -1) {
    return res.status(404).send('Post nicht gefunden');
  }

  // Löschen: splice gibt ein Array zurück -> [0] ist das
  // gelöschte Objekt
  const deletedPost = posts.splice(index, 1)[0];

  // Antwort: 200 OK + gelöschter Post (zur Kontrolle)
  res.status(200).json(deletedPost);
});
```

Test mit Postman (DELETE)

HTTP <http://localhost:3000/api/posts/1> DELETE HTTP-Methode – löscht den Post

DELETE <http://localhost:3000/api/posts/1> ID (=post_id) – bestimmt welcher Post aus der Liste gelöscht wird.

Save Share Send

Docs Params Authorization Headers (7) Body Scripts Settings Cookies

(none) form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body Cookies Headers (7) Test Results Resultat: gelöschter Post 200 OK 60 ms 424 B

```
{
  "post_id": 1,
  "user_id": 1,
  "title": "Morning Coffee",
  "image_url": "https://picsum.photos/800/450?random=1",
  "description": "Nothing beats starting the day with a warm cup of coffee.",
  "likes": 10
}
```

1. Methode: DELETE
2. URL: <http://localhost:3000/api/posts/1>

Erwartung:

- Status 200 OK
- JSON-Objekt des gelöschten Posts

Danach:

- GET /api/posts → der gelöschte Post sollte nicht mehr in der Liste sein
- GET /api/posts/1 → sollte 404 Not Found liefern

Typische Fehlerquellen

- req.params.id ist immer ein String → mit Number(...) oder parseInt(...) umwandeln.
- Bei PUT vergessen viele, den Request-Body als JSON zu senden → in Postman unbedingt raw + JSON wählen.
- Bei PUT und DELETE immer prüfen: existiert das Objekt? → sonst 404 zurückgeben.
- Bei PUT: wenn keine Felder im Body sind → 400 zurückgeben (sonst „Update ohne Update“).

Ausblick

Sie können jetzt CRUD auf API-Ebene komplett:

- Create → POST /api/posts
- Read → GET /api/posts und GET /api/posts/:id
- Update → PUT /api/posts/:id
- Delete → DELETE /api/posts/:id

In der nächsten Unterrichtseinheit ersetzen wir die lokale Liste posts durch eine echte MySQL-Tabelle und führen dieselben Operationen über SQL aus.

From:
<https://wiki.bzz.ch/> - BZZ - Modulwiki



Permanent link:
https://wiki.bzz.ch/modul/m290_guko/learningunits/lu16/theorie/b_update_delete

Last update: **2026/01/02 23:06**