

# Backend Create

## Lernziele

- ☐ CRUD und REST-APIs verstehen
- ☐ Eine Create-REST-API-Route erstellen

## CRUD und REST

### CRUD

Das Akronym CRUD [kɹʌd] deckt die vier grundlegenden Operationen der persistenten Speicherung ab:

- **Create**, *erstelle einen Datensatz*,
- **Read** oder **Retrieve**, *lese einen Datensatz*,
- **Update**, *aktualisiere einen Datensatz*, und
- **Delete** oder **Destroy**, *lösche einen Datensatz*.

Diese Operationen können je nach Kontext oder Umgebung unterschiedlich ausgedrückt werden.

CRUD	MongoDB	SQL	HTTP-Methode	Typische REST-URL (mit HTTP-Methode)
Create	insertOne / insertMany	INSERT	POST	/todos
Read oder Retrieve	findOne / find	SELECT	GET	/todos/[todoId] (einzeln), /todos (alle)
Update	updateOne / updateMany	UPDATE	PUT / PATCH	/todos/[todoId]
Delete oder Destroy	deleteOne / deleteMany	DELETE	DELETE	/todos/[todoId]

💡 Beachte, dass sich die **Create**-Operation auf die HTTP-Methode **POST** bezieht. Du benötigst die entsprechende HTTP-Methode, wann immer du eine der **CRUD**-Operationen durchführen möchtest.

### REST

REST steht für "Representational State Transfer" und bezieht sich auf architektonische Prinzipien und Einschränkungen, wie man seine API strukturiert.

Wir verwenden CRUD-Operationen und HTTP-Methoden mit einer REST-API.

💡 Dies ist eine sehr grundlegende und unvollständige Erklärung. Wenn du mehr darüber erfahren möchtest, was eine API RESTful macht, kannst du [hier](#) darüber lesen.


## Erstellen mit Mongoose

Um einen neuen Eintrag in deiner Datenbank zu erstellen, musst du eine **POST**-API-Route definieren und die **.create**-Methode auf unserem Joke-Modell aufrufen:

```
// pages/api/index.js
if (request.method === "POST") {
  try {
    const jokeData = request.body;
    await Joke.create(jokeData);

    response.status(201).json({ status: "Joke erstellt" });
  } catch (error) {
    console.log(error);
    response.status(400).json({ error: error.message });
  }
}
```

Beachte, dass die **POST**-Route alleine keinen neuen Eintrag in deiner Datenbank erstellt: Du musst dem Submit-Handler deines Formulars mitteilen, diese Route zu verwenden.

 Mehr dazu findest du in den [mongoose-Dokumentationen](#)

---

## POST-Anfragen senden und Daten neu validieren

Da wir unser Witze-Datenarray verändern, müssen wir zwei Aktionen ausführen:

- Daten an unser Backend senden, um sie der Datenbank hinzuzufügen
- Unsere App aktualisieren, damit sie die aktualisierten Daten aus der Datenbank verwendet

Wenn wir unsere Daten nicht neu validieren, spiegelt die App nicht die Änderungen wider, die wir an unserer Datenbank vorgenommen haben. **useSWR** bietet uns eine Methode namens **mutate**, um diese Neuvalidierung für einen bestimmten API-Endpunkt auszulösen, z. B. `/api/jokes`. Wir können sie aus dem Hook-Aufruf wie **data** oder **isLoading** destrukturieren:

```
const { mutate } = useSWR("/api/jokes/");
```

Um eine HTTP-POST-Anfrage mit **fetch** auszuführen, müssen wir ein **Optionsobjekt** an den **fetch**-Aufruf übergeben, das die folgenden Informationen enthält:

- Methode: Ein Verb wie "POST", "PUT" oder "DELETE", das die Art der HTTP-Anfrage definiert
- Den "Content-Type", der in den Anfrage-Headern bereitgestellt wird
- Body: Die Daten, die an den Server gesendet werden

Eine typische "POST"-Anfrage sieht so aus:

```
const response = await fetch("/api/jokes", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify(data),
});
```

💡 Der `body`-Schlüssel stellt den `request.body` in der obigen API-Route dar: Hier werden die tatsächlichen Daten vom Frontend an die API (und dann an das Backend bzw. die Datenbank) übergeben.

Nach einem erfolgreichen "POST"-Fetch, der unseren neuen POST-API-Endpunkt ausgelöst hat, können wir `useSWR` mitteilen, die Daten neu zu validieren, indem wir die `mutate`-Funktion aufrufen. Der gesamte Submit-Prozess sieht folgendermaßen aus:

```
import useSWR from "swr";

export default function JokeForm() {
  const { mutate } = useSWR("/api/jokes");

  async function handleSubmit(event) {
    event.preventDefault();

    const formData = new FormData(event.target);
    const jokeData = Object.fromEntries(formData);

    const response = await fetch("/api/jokes", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(jokeData),
    });

    if (response.ok) {
      mutate();
    }
  }

  return (
    //...
  );
}
```

## Resources

- [What is REST?](#)
- [swr docs](#)

