# Express API Übung

## Einleitung

In dieser Übung lernst du, wie man eine REST API mit Express verwendet. Du wirst Schritt für Schritt die bestehende Anwendung testen und erweitern, um ein besseres Verständnis für die grundlegenden Konzepte zu erhalten.

Ziel ist es, dass du am Ende weißt, wie man einfache Endpunkte erstellt, auf Anfragen antwortet und Query-Parameter verwendet. Die Übung beginnt einfach und wird nach und nach anspruchsvoller.

## Wichtige Ressourcen

- **Projekt-Dokumentation**

  Express Template Documentation

  Verwende diese Dokumentation als Grundlage für die Aufgaben. Sie enthält:
  - Informationen zur Projektstruktur und Einrichtung
  - Anleitungen zum Erstellen von Endpunkten mit Express
  - Hinweise zur Nutzung von Query- und Body-Parametern
  - Beispiele zur Fehlerbehandlung und Statuscodes

# Weitere hilfreiche Dokumentation

1. **Express**
   Offizielle Express-Dokumentation: https://expressjs.com/
   Enthält Details zur Routenverwaltung, Middleware und Anfrageverarbeitung.
2. **Node.js**
   Offizielle Node.js-Dokumentation: https://nodejs.org/
   Eine nützliche Referenz zu Node.js und dessen Grundlagen.
3. **Postman**
   Offizielle Postman-Dokumentation: https://learning.postman.com/
   Hilfreich, um zu verstehen, wie man HTTP-Anfragen testet und simuliert.

# Vorbereitung

Bevor wir starten, stelle sicher, dass du Node.js und npm installiert hast.

1. Erstelle eine Datei `index.js` und kopiere den Beispielcode aus der Unterrichtssitzung hinein oder clone die den entsprechenden code aus Toms repo: https://github.com/tomtechstarter/backend-neu/blob/main/index.js.
2. Öffne das Terminal und navigiere zum Ordner, in dem sich `index.js` befindet.
3. Installiere die benötigten Pakete:

   ```
   npm install express cors
   ```

4. Starte die Anwendung mit:

   ```
   node index.js
   ```

Nun sollte die Meldung `Express App is running on http://localhost:5050` im Terminal erscheinen.

## Aufgabe 1: API testen (ca. 20 Minuten)

In dieser Aufgabe lernst du, die verschiedenen Endpunkte der API zu testen.

1. Öffne Postman oder einen Webbrowser.
2. Sende folgende Anfragen und beobachte die Antworten:
    - **Basis-Endpunkt**:
        - URL: `http://localhost:5050/`
        - Antwort: Sollte die Nachricht `"Hello my name is Tom"` sein.
    - **Alle Todos abrufen**:
        - URL: `http://localhost:5050/todos/all`
        - Antwort: Eine Liste aller Todos im JSON-Format.

- **Ein spezifisches Todo nach ID abrufen**:
  - URL: `http://localhost:5050/todos/byid?todoId=1`
  - Antwort: Das Todo mit der ID 1.
- **Todos eines spezifischen Benutzers abrufen**:
  - URL: `http://localhost:5050/todos/byuserid?userId=1`
  - Antwort: Eine Liste der Todos des Benutzers mit der `userId` 1.

Notiere dir die Antworten und überprüfe, ob alles wie erwartet funktioniert.

## Aufgabe 2: Eigene Filter hinzufügen (ca. 30 Minuten)

Jetzt wirst du einen eigenen Filter-Endpunkt hinzufügen, der alle Todos mit einem bestimmten Namen sucht.

1. Füge in `index.js` einen neuen GET-Endpunkt `/todos/byname` hinzu.
2. Verwende einen Query-Parameter `name`, um nach dem Namen des Todos zu filtern.
3. Falls kein Todo mit dem Namen gefunden wird, soll der Server eine leere Liste zurückgeben.

Beispiel für eine Anfrage:

- URL: `http://localhost:5050/todos/byname?name=Milch holen`
- Antwort: Das Todo-Objekt mit dem Namen `"Milch holen"`, falls vorhanden.

## Aufgabe 3: Fehlerbehandlung verbessern (ca. 30 Minuten)

Baue nun einfache Fehlerbehandlungen ein.

1. **Für `/todos/byid`**: Falls `todoId` fehlt, soll die Antwort `"No Todo Id provided"` sein.
2. **Für `/todos/byuserid`**: Falls `userId` fehlt, soll die Antwort `"No User Id provided"` sein.

Überprüfe, dass die Fehlernachrichten korrekt zurückgegeben werden, wenn die benötigten Query-Parameter fehlen.

## Aufgabe 4: POST-Anfrage hinzufügen (ca. 45 Minuten)

Erstelle eine POST-Route, um neue Todos hinzuzufügen.

1. Implementiere in `index.js` einen neuen Endpunkt `POST /todos`.
2. Der Endpunkt soll neue Todos in das `todos`-Array einfügen.
3. Stelle sicher, dass `name` und `userId` im Body der Anfrage vorhanden sind.
    - Wenn einer dieser Werte fehlt, soll der Server mit `"Name and UserId are required"` und einem Status `400` antworten.

Beispiel für eine Anfrage in Postman:

- Methode: POST
- URL: `http://localhost:5050/todos`
- Body: `{ "name": "Einkaufen", "userId": 3 }`

Testen: Sende mehrere Anfragen mit verschiedenen `name`- und `userId`-Werten und überprüfe, ob die Todos im Array hinzugefügt werden.

## Aufgabe 5: DELETE-Anfrage für mehrere Todos (ca. 45 Minuten)

Erstelle eine DELETE-Route, um mehrere Todos gleichzeitig zu löschen.

1. Implementiere in `index.js` einen neuen Endpunkt `DELETE /todos`.
2. Der Endpunkt soll mehrere `todoId`s als Query-Parameter akzeptieren und alle entsprechenden Todos löschen.
3. Wenn eine ID nicht existiert, soll die Antwort `"Todo not found"` sein.

Beispiel für eine Anfrage:

- URL: `http://localhost:5050/todos?todoId=1&todoId=2`
- Antwort: Gibt die gelöschten Todos zurück.

## Bonusaufgabe: PUT-Anfrage zum Aktualisieren (optional, ca. 30 Minuten)

Diese Aufgabe ist optional und etwas schwieriger.

1. Füge eine PUT-Route hinzu, die den Namen eines bestehenden Todos aktualisiert.
2. Verwende `todoId` als Query-Parameter und `name` im Body.
3. Wenn das Todo nicht existiert, soll die Antwort `"Todo not found"` sein.

Beispiel für eine Anfrage:

- Methode: PUT
- URL: `http://localhost:5050/todos/update?todoId=1`
- Body: `{ "name": "Neue Aufgabe" }`

Testen: Sende die Anfrage und überprüfe, ob der Name des Todos korrekt aktualisiert wurde.