Wiederholung REST APIs und Express

Was ist eine REST API?

- **API**: Schnittstelle, die Kommunikation zwischen verschiedenen Software-Komponenten ermöglicht.
- **REST (Representational State Transfer)**: Architekturprinzipien für APIs, um Daten über das Internet bereitzustellen.
- **Ziel**: Daten und Dienste effizient zwischen Client und Server austauschen.

Warum verwenden wir REST APIs?

- **Standardisierte Kommunikation**: REST bietet einheitliche Methoden zur Kommunikation.
- **Flexibel und erweiterbar**: Anwendbar auf viele verschiedene Systeme und Anwendungsfälle.
- **Trennung von Frontend und Backend**: Ermöglicht, dass verschiedene Frontends (Web, Mobile, etc.) auf dieselbe API zugreifen.

Grundlegende HTTP-Methoden in REST

1. **GET**: Daten vom Server abrufen (Lesen).
2. **POST**: Neue Daten zum Server senden (Erstellen).
3. **PUT**: Vorhandene Daten aktualisieren.
4. **DELETE**: Daten vom Server löschen.

HTTP-Methoden und ihre Funktion

Methode	Funktion	Beispiel
GET	Daten abrufen	`GET /todos`
POST	Neue Daten hinzufügen	`POST /todos`
PUT	Daten aktualisieren	`PUT /todos/1`
DELETE	Daten löschen	`DELETE /todos/1`

Was ist eine Ressource?

- **Ressourcen** sind Objekte, die von der API bereitgestellt werden.
- Beispiel: Ein "Todo" in einer To-do-App.
- Jede Ressource wird durch eine **eindeutige URL** angesprochen.
    - Beispiel: `/todos/1` für ein Todo mit der ID 1.

Beispiel für Ressourcen in einer To-do-App

- **GET /todos**: Alle Todos abrufen.
- **POST /todos**: Ein neues Todo hinzufügen.
- **GET /todos/1**: Ein spezifisches Todo mit der ID 1 abrufen.
- **DELETE /todos/1**: Ein spezifisches Todo mit der ID 1 löschen.

Statuscodes in REST APIs

- **200 OK**: Anfrage war erfolgreich.
- **201 Created**: Ein neues Objekt wurde erfolgreich erstellt.
- **404 Not Found**: Die angeforderte Ressource existiert nicht.
- **500 Internal Server Error**: Serverseitiger Fehler.

Warum sind Statuscodes wichtig?

- **Feedback** für den Client, ob die Anfrage erfolgreich war.
- **Diagnose**: Hinweise auf Fehlerursachen (z.B., fehlende Ressourcen oder falsche Anfrageformate).
- Erleichtert die **Fehlersuche und Wartung** der API.

Was ist Express?

- **Express.js**: Minimalistisches, flexibles Framework für Node.js.
- Hilft beim schnellen Aufbau von **Web-Apps und APIs**.
- **Einfache Routenverwaltung**: Bestimmt, wie Anfragen an Endpunkte behandelt werden.
- **Middleware-Unterstützung**: Zusätzliche Funktionen wie `cors` und `express.json()`.

Vorteile von Express

- **Schnelle Entwicklung** durch minimalistische Struktur.
- **Flexibel** und leicht anpassbar für spezifische Anforderungen.
- **Große Community**: Viele vorgefertigte Lösungen und Module.

Middleware in Express

- **Definition**: Middleware sind Funktionen, die Anfragen bearbeiten, bevor sie die endgültige Route erreichen.
- Beispiele:
  - `cors()`: Erlaubt Anfragen von verschiedenen Quellen (Cross-Origin).
  - `express.json()`: Ermöglicht das Parsen von JSON-Daten im Request-Body.

Beispiel für Middleware in Express

```
const express = require("express");
const cors = require("cors");

const app = express();
app.use(cors());
app.use(express.json());
```

- **cors()**: Erlaubt die API-Nutzung von verschiedenen Quellen.
- **express.json()**: Verarbeitet JSON-Daten in POST-Anfragen.

Was sind Query-Parameter?

- **Query-Parameter**: Zusätzliche Informationen in der URL.
- Beispiel: `/todos?userId=1`
- Verwendet für **Filterung und Auswahl** spezifischer Daten.

Wie funktionieren Query-Parameter?

- Aufbau: Nach einem `?` in der URL.
  - Beispiel: `/todos?userId=1`
- Mehrere Parameter werden durch `&` getrennt:
  - Beispiel: `/todos?userId=1&completed=true`

Unterschied zwischen Query- und Path-Parametern

Typ	Beispiel	Verwendung
Query	`/todos?userId=1`	Filter oder zusätzliche Optionen
Path	`/todos/1`	Zugriff auf spezifische Ressource

Wann verwenden wir Query-Parameter?

- Wenn wir **Daten filtern** oder bestimmte Attribute angeben möchten.
- Beispiel: Alle Todos eines bestimmten Benutzers anzeigen (`userId=1`).

Einführung in Postman

- **Postman**: Tool zum Testen von APIs.
- **Ziel**: Ermöglicht das Senden von HTTP-Anfragen ohne Frontend.
- **Vorteile**:
  - Effizientes Testen und Debuggen von APIs.
  - Einfache Verwaltung und Speicherung von Anfragen.

Warum ist Postman nützlich?

- **Unabhängigkeit vom Frontend**: APIs können getestet werden, ohne eine Benutzeroberfläche zu programmieren.
- **Organisation**: Anfragen und Tests können gespeichert und organisiert werden.
- **Zeitersparnis**: Schnelleres Testen und Debuggen von API-Endpunkten.

Beispiel: GET-Anfrage in Postman

1. Wähle **GET** als Methode.
2. Gib die URL ein, z.B. `http://localhost:5050/todos/all`.
3. Klicke auf **Send**, um die Anfrage zu senden.
4. **Ergebnis**: Die Liste der Todos wird angezeigt.