

# LB03 – Projektarbeit: Backend-API mit Node.js/Express + MySQL

## Einleitung

In den ersten zwei Dritteln des Moduls M290 haben Sie sich intensiv mit relationalen Datenbanken, SQL (DDL, DML, DCL), ERM/ERD, Tabellenbeziehungen, JOINS und Aggregatfunktionen beschäftigt.

In der dritten und letzten Leistungsbeurteilung (LB03) verknüpfen Sie dieses Wissen mit einem Backend-Server: Sie implementieren einen Node.js/Express-Server, der über eine REST-API auf Ihre MySQL-Datenbank zugreift. Ein Frontend wird nicht programmiert – stattdessen simulieren Sie es mit Postman.

**Wichtig:** Das Erklärvideo (Screencast) ist Teil der Leistung. Beide Lernenden müssen darin vorkommen und jeweils einen Teil von Setup/Code verständlich erklären.

## Rahmenbedingungen

- **Sozialform:** Partnerarbeit (2er-Team)
- **Zeit:** ca. 7 Lektionen Projektzeit im Unterricht (Inputs jeweils zu Beginn der Lektion)
- **Abgabe:** Letzter Unterrichtstag des Semesters, **21:00 Uhr**
- **Benotung:** Lineare Notenskala (siehe unten)

## Auftrag

Sie wählen im 2er-Team **einen** der beschriebenen Use Cases aus (pro Klasse: jedes Thema nur einmal). Zu diesem Use Case entwickeln Sie:

- ein sauberes Datenmodell (ERM/ERD),
- eine passende MySQL-Datenbank mit Startdaten und einem AppUser,
- einen Express-Server, der die wichtigsten Funktionen als CRUD-REST-API bereitstellt,
- ein Video-Tutorial (Screencast) von ca. 15 Minuten, in dem Sie Ihre Lösung erklären und demonstrieren.

Zielpublikum des Tutorials sind Ihre Mit-Auszubildenden aus dem 2. Lehrjahr, die die Themen

- Daten & Datenbanken,
- Zugriff auf Daten in einer 3-Schichten-Architektur (Client – Server – Datenbank),
- CRUD-Operationen in SQL und über eine REST-API besser verstehen sollen.

# Anforderungen (MUSS)

## Mindestanforderungen (MUSS)

- Backend läuft (Start über `npm run dev` oder `npm start` oder `node index.js`)
- Mindestens **2 Tabellen** in MySQL (mit PK/FK)
- Mindestens **eine API-Ressource** (z.B. `/api/serien`)
- **CRUD** vollständig über REST nachweisbar: GET, POST, PUT, DELETE
- Mindestens **eine JOIN-Route** (Daten aus 2 Tabellen)
- Mindestens **eine Aggregat-Route** (COUNT/AVG/SUM/MIN/MAX mit GROUP BY)
- Zugriff auf DB **nicht** als root, sondern über einen **AppUser** (Least Privilege)
- Video-Tutorial, wo alle Team-Mitglieder darin vorkommen und einen Teil erklären.

## Inhalt des Video-Tutorials (Screencast, ca. 15 Minuten)

Das Video soll strukturiert und nachvollziehbar sein und mindestens folgende Teile enthalten:

### 1. Einleitung

1. Kurzvorstellung des Use Cases (Problem / Idee)
2. Was Ihre App grob können soll

### 2. Analyse & Datenmodell

1. Erklärung des ERM (Entitäten, Beziehungen, Kardinalitäten)
2. Erklärung des ERD in Crow's-Foot-Notation (Tabellen, PK/FK, Datentypen)

### 3. Datenbank

1. Anlegen der Datenbank & Tabellen per SQL-Skript/Befehlen (DDL)
2. Import der Startdaten per SQL-Skript/Befehlen (DML)
3. Anlegen und Berechtigen eines AppUsers (DCL) + kurzer Hinweis, warum diese Rechte sinnvoll sind

### 4. Backend / Server

1. Aufbau des Node.js/Express-Projekts (wichtigste Dateien kurz erklären: `index.js`, `package.json`)
2. Erklärung der wichtigsten Routen (z.B. GET `/api/...`, POST `/api/...`)

### 5. Tests mit Postman

1. Vorführen der CRUD-Endpunkte (Create, Read, Update, Delete)
2. dabei **zeigen**, dass sich die Daten in der Datenbank wirklich ändern (WebStorm DB-Plugin)
3. mindestens **eine JOIN-Demo**
4. mindestens **eine Aggregat-Demo**
5. sinnvolle HTTP-Statuscodes (z.B. 200, 201, 400, 404, 500)

### 6. Reflexion

1. Jede Person nennt mindestens **2 Learnings** (positiv / herausfordernd)

2. Was würden Sie beim nächsten Mal anders machen?

## 7. Schluss

1. Kurze Zusammenfassung Ihrer Lösung
2. Ausblick / mögliche Erweiterungen

### Wichtig für die Bewertung des Verständnisses

- Beide Teammitglieder müssen im Video **hörbar** sein (eigene Erklärung, nicht nur „ja genau“).
- Beide müssen je einen **eigenen Teil** erklären (z.B. Person A: DB/SQL/DCL, Person B: API-Routen/Postman/Statuscodes).
- Sie müssen während dem Video zeigen, dass Sie Ihren eigenen Code verstehen (Warum dieser Endpoint? Warum dieser JOIN? Was bedeutet `req.params.id?`).

## Abgabe - zu liefernde Lernprodukte (ZIP in Moodle)

Im ZIP-File müssen enthalten sein:

1. **Video-Tutorial** (ca. 15 Min)
  - Format: MP4 (H.264), max. FullHD (1920×1080), min. 1280×720
  - Max. Grösse: 1 GB (Bitrate ca. 3000–5000 kbps)
2. **ERM und ERD** als PDF-Datei
3. **SQL-Skript (DDL)**: Anlegen der Tabellenstruktur (Als PDF/Word oder .sql-File)
4. **SQL-Skript (DML)**: Import / Insert der Startdaten (Als PDF/Word oder .sql-File)
5. **SQL-Skript (DCL)**: AppUser erstellen + Rechte vergeben (Als PDF/Word oder .sql-File)
6. **Datenbank-Dump**: vollständiges SQL-File (Struktur + Daten) zum Wiederherstellen (.sql-File)
7. **Node.js-Projektordner**
  - `package.json`
  - Server-Datei(en) (z.B. `index.js`, `connect.js`)

**Reproduzierbarkeit:** Ihr Projekt muss auf einem anderen Rechner mit Ihren Skripten nachvollziehbar eingerichtet werden können.

## Bewertungsraster (100 Punkte)

Bereich	Max. Punkte	Voll erfüllt	Teilweise erfüllt	Nicht erfüllt
<b>1. Projekt-Setup &amp; Reproduzierbarkeit</b>	<b>8</b>	Start/Setup klar (alle Dateien sind vorhanden und ausführbar), Server startet zuverlässig, Abgabe sauber strukturiert	Läuft grundsätzlich, aber Setup unklar/fehleranfällig	Start nicht möglich / unvollständig

Bereich	Max. Punkte	Voll erfüllt	Teilweise erfüllt	Nicht erfüllt
<b>2. Datenmodell (ERM/ERD)</b>	<b>12</b>	Entitäten/Beziehungen korrekt, PK/FK nachvollziehbar, Kardinalitäten stimmen	Modell vorhanden, aber Lücken/FK/Kardinalitäten unklar	Kein brauchbares Modell
<b>3. MySQL-Implementierung (DDL + Datentypen + Constraints)</b>	<b>10</b>	DDL ausführbar, Datentypen sinnvoll, PK/FK korrekt, Constraints passend	DDL vorhanden, aber Mängel bei Datentypen/Keys/Constraints	Nicht ausführbar / unpassend
<b>4. Startdaten (DML)</b>	<b>6</b>	Mehrere realistische Datensätze, gut testbar	Wenige/teilweise unpassende Daten	Keine Startdaten
<b>5. AppUser &amp; Rechte (DCL, Least Privilege)</b>	<b>8</b>	AppUser funktioniert, Rechte minimal sinnvoll, keine root-Nutzung	AppUser vorhanden, aber Rechte unklar/zu breit	Kein AppUser / root verwendet
<b>6. REST-API Design &amp; Struktur</b>	<b>8</b>	Konsistente Pfade (/api/...), sinnvolle Ressourcen, /:id korrekt	Routen vorhanden, aber inkonsistent/unsauber	Unklar/chaotisch
<b>7. CRUD über Backend → MySQL</b>	<b>20</b>	GET/POST/PUT/DELETE funktionieren, DB ändert sich sichtbar, SQL sauber	CRUD teilweise fehlerhaft/unvollständig	CRUD nicht nachweisbar
<b>8. JOIN-Route</b>	<b>6</b>	Mindestens eine Route mit JOIN, Ergebnis sinnvoll	JOIN vorhanden, aber Ergebnis unklar/falsch	Kein JOIN
<b>9. Aggregat-Route (GROUP BY)</b>	<b>6</b>	Mindestens eine Aggregat-Route korrekt (COUNT/AVG/SUM etc.)	Aggregat vorhanden, aber unklar/falsch	Kein Aggregat
<b>10. Validierung, Fehlerbehandlung, HTTP-Statuscodes</b>	<b>8</b>	400/404/500 sinnvoll, Pflichtfelder geprüft, verständliche Fehlermeldungen	Teilweise vorhanden, teils falsche Codes	Keine Validierung/Fehlerhandling
<b>11. Video-Tutorial (Screencast) &amp; Verständnissnachweis</b>	<b>8</b>	Struktur klar, beide erklären aktiv, Demo mit Postman + DB sichtbar, reflektiert	Video vorhanden, aber unklar/zu kurz/Teamanteil ungleich	Kein Video / Verständnis nicht nachweisbar

**Eigenleistung / Verständnis** Sie müssen Ihre Lösung erklären können (SQL, Datenmodell, API, Tests). Wenn zentrale Teile nicht fachlich korrekt begründet werden können, kann der entsprechende Bereich mit **0 Punkten** bewertet werden.

## Notenberechnung (lineare Skala)

**Note = 1.0 + 5.0 × (Punkte / 100)** (Beispiel: 80 Punkte → 1.0 + 5.0 × 0.80 = 5.0)

From:

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

Permanent link:

[https://wiki.bzz.ch/modul/m290\\_guko/leistungsbeurteilungen/03\\_lb/b\\_projektbeschreibung?rev=1766063969](https://wiki.bzz.ch/modul/m290_guko/leistungsbeurteilungen/03_lb/b_projektbeschreibung?rev=1766063969)

Last update: **2025/12/18 14:19**

