LU15a – Was ist ein Backend-Server?

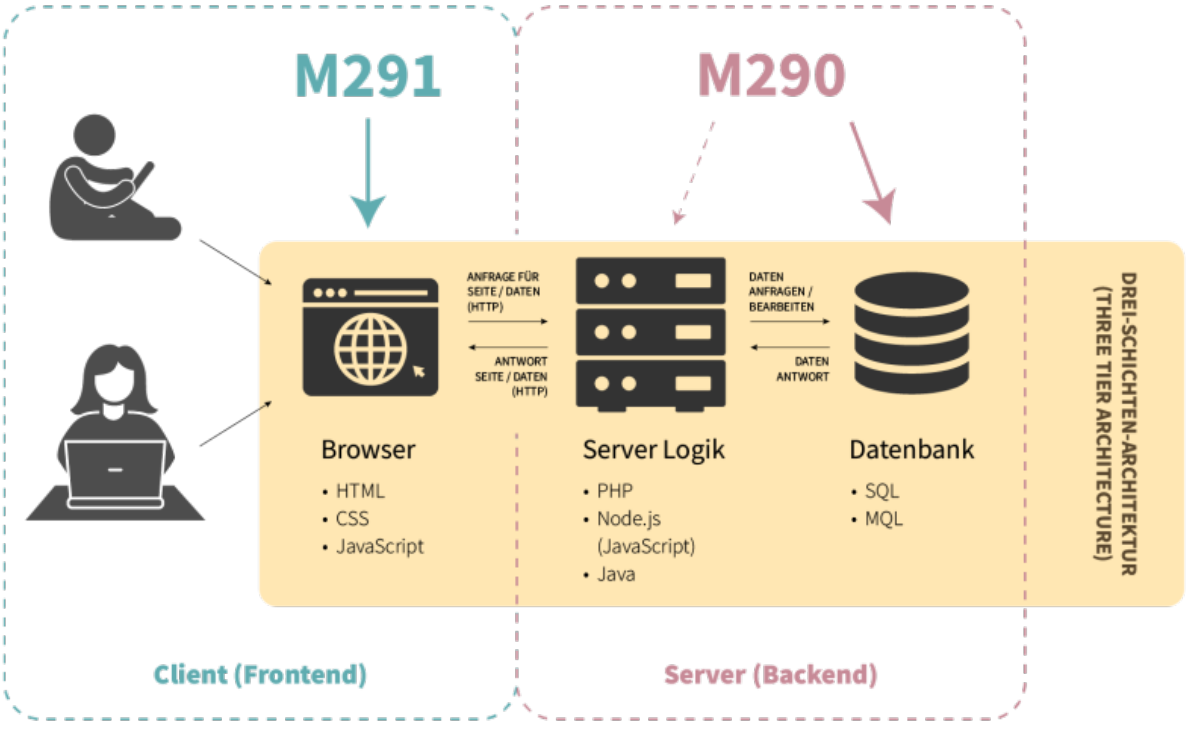
Lernziele

- Sie können erklären, was ein **Backend-Server** ist.
- Sie können den Backend-Server in der **3-Schichten-Architektur** einordnen.
- Sie kennen die Begriffe **Node.js, npm, Express, nodemon, Port, API** und können sie beschreiben.
- Sie verstehen, warum eine Webapplikation besser auf einem **3-Tier-Modell** basiert als auf einem direkten Datenbankzugriff.

3-Tier-Webarchitektur (3-Schichten-Architektur)

Die 3-Tier-Architektur teilt eine Webapplikation in drei Schichten:

1. **Presentation Layer (Frontend)** – Oberfläche im Browser oder in einer App
2. **Application Layer (Backend)** – Logik, Regeln, Validierung
3. **Data Layer (Datenbank)** – Speichern und Verwalten der Daten



Jede Schicht hat eine klar definierte Aufgabe und spricht nur mit der direkt angrenzenden Schicht. Vorteile von einer 3-Tier-Architektur:

- bessere **Sicherheit**
- bessere **Wartbarkeit**
- mehr **Flexibilität**, wenn die Anwendung wächst

Wichtige Begriffe im Node.js-Ökosystem

Begriff	Kurz erklärt	Wofür brauchen wir es im Modul?	Download / Link
Node.js	JavaScript-Laufzeitumgebung, mit der Sie JavaScript **ausserhalb des Browsers** ausführen (z.B. auf dem Server).	Läuft als Basis für Ihren Backend-Server; ohne Node kann Ihr Express-Server nicht starten.	Node.js Download (LTS)
npm	`Node Package Manager`: Standard-Paketmanager für Node.js. Installiert zusätzliche Bibliotheken (Packages).	Sie installieren damit z.B. **Express** und **nodemon** und verwalten Abhängigkeiten im Projekt.	-
Express	Minimalistisches Web-Framework für Node.js, um HTTP-Server und APIs zu bauen.	Sie programmieren damit Ihre **REST-API** (Routen, Request/Response, Status-Codes).	Express-Dokumentation / GitHub
nodemon	Tool, das den Node-Server **automatisch neu startet**, wenn sich Dateien ändern.	Komfort im Unterricht: Kein manuelles `node index.js` nach jeder Code-Änderung nötig.	nodemon auf GitHub
Port	„Türnummer“ auf einem Computer, über die ein Dienst erreichbar ist (z.B. 3000, 8080, 80).	Ihr Server lauscht standardmässig auf **Port 3000** → Browser: `http://localhost:3000/`.	–
API	`Application Programming Interface` – Schnittstelle, über die Software miteinander spricht.	Ihr Express-Server stellt eine **HTTP-API** bereit: z.B. GET `/api/books` liefert Daten zurück.	–

Warum überhaupt ein Backend-Server?

Bis jetzt haben Sie im Modul M290 vor allem mit der Datenbank gearbeitet:

- Sie haben in **WebStorm** mit dem Datenbank-Plugin gearbeitet.
- Sie haben **SQL-Befehle** (inkl. CRUD) direkt an die Datenbank MySQL geschickt.
- Das war ideal, um SQL zu üben und zu verstehen, wie Tabellen, JOINs und Aggregatfunktionen funktionieren.

Für eine **richtige Webapplikation**, die viele Benutzerinnen und Benutzer über das Internet nutzen, ist dieser direkte Zugriff jedoch **nicht sinnvoll** und auch **nicht sicher**.

Was wäre, wenn der Browser direkt auf die Datenbank zugreifen würde?

Stellen Sie sich vor:

- Ihr Browser würde direkt eine Verbindung zur Datenbank MySQL aufbauen.
- Jede Benutzerin / jeder Benutzer hätte direkten Zugriff auf Tabellen und könnte eigene SQL-Befehle schicken.

Das führt zu mehreren Problemen:

- **Sicherheitsrisiko**: Alle, die die Verbindung kennen, könnten eigene SQL-Befehle ausführen – auch schädliche (z.B. Daten löschen).
- **Keine Kontrolle über die Logik**: Es gäbe keine zentrale Stelle, die prüft, ob eine Aktion erlaubt ist (z.B. darf ein:e Lernende:r wirklich Noten ändern?).
- **Schwierige Wartung**: Wenn sich die Struktur der Datenbank ändert, müssten alle Frontends (Webseiten, Apps, ...) angepasst werden.
- **Technische Abhängigkeit**: Jedes Frontend müsste wissen, wie genau die Datenbank aufgebaut ist (Tabellennamen, Spaltennamen, Datentypen, ...).

Die Lösung: eine zusätzliche Logik-Schicht

Statt den Browser direkt mit der Datenbank sprechen zu lassen, schalten wir eine **Zwischenschicht** dazwischen: den **Backend-Server**.

> **3-Schichten-Architektur in unserem Modul**
>
> Client (Browser / Postman) ⇄ **Backend-Server (Node.js/Express)** ⇄ Datenbank (MySQL)

- Der **Client** (z.B. Browser oder Postman) sendet **HTTP-Anfragen** (z.B. `GET /api/books`).
- Der **Backend-Server**:
  1. nimmt die Anfrage entgegen,
  2. prüft die Daten,
  3. entscheidet, was erlaubt ist,
  4. führt die passenden **SQL-Befehle** auf der Datenbank aus,
  5. formatiert das Ergebnis (z.B. als JSON) und sendet es zurück.
- Die **Datenbank** kennt nur den Server und führt dessen SQL-Kommandos aus – aber kein Client greift direkt darauf zu.

Vorteile des Backend-Servers im Detail

1. Sicherheit

- Die Datenbank ist nicht direkt aus dem Internet erreichbar.
- Nur der Backend-Server hat Zugang zur Datenbank (z.B. über einen eigenen `AppUser` mit klar definierten Rechten).
- Der Server entscheidet:
  1. welche SQL-Befehle erlaubt sind,
  2. welche Daten angezeigt oder verändert werden dürfen.

2. Businesslogik & Validierung

- Fachliche Regeln werden im Backend umgesetzt, z.B.:
    1. `Note muss zwischen 1.0 und 6.0 liegen`,
    2. `Geburtsdatum darf nicht in der Zukunft liegen`,
    3. `Eine Bewertung darf nur 1–5 Sterne haben`.
- Der Browser darf nicht über eigene SQL-Befehle mit der Datenbank kommunizieren, sondern ruft nur definierte **API-Endpunkte** auf

(z.B. `POST /api/trips` oder `GET /api/books`).

- So ist klar geregelt, was das System darf und was nicht.

3. Einheitliche Schnittstelle (API)

- Der Backend-Server stellt eine **API** bereit (z.B. `/api/trips`, `/api/songs`).
- Diese API kann von verschiedenen Clients genutzt werden:
    1. einer Webseite,
    2. einer Mobile-App,
    3. Tools wie Postman.
- Die Datenbank-Struktur kann sich im Hintergrund ändern – die API kann stabil bleiben.

4. Bessere Wartbarkeit & Erweiterbarkeit

- Änderungen an der Datenbank (z.B. neue Tabelle, anderes Feld) werden zuerst im Backend angepasst.
- Frontends müssen nur die API kennen, nicht die Details der Datenbank.
- Neue Funktionen (z.B. zusätzlicher Filter, neue Berechnung) können im Backend ergänzt werden,

ohne dass direkt SQL im Frontend geändert werden muss.

In dieser Phase des Moduls gehen wir den nächsten Schritt:

- Wir schreiben **keine SQL-Befehle, wie SELECT/UPDATE/DELETE/CREATE mehr direkt in WebStorm an die Datenbank**,
- sondern wir bauen einen **Backend-Server mit Node.js/Express**, der für uns mit der Datenbank spricht. (Andere SQL-Befehle, wie das Erstellen von Tabellen und Datenbanken werden wir weiterhin direkt mit Webstorm/Datenbank-Plugin machen).
- Im Unterricht verwenden wir dazu zunächst den Browser und **Postman** als Client, später könnten auch echte Frontends (Web oder Mobile) diese API nutzen.