LU15c – Erster Express-Server mit Node.js

Learning Objectives

- Sie können einen einfachen **Express-Server** programmieren.
- Sie können den Server über das Terminal starten.
- Sie verstehen den **Request-Response-Zyklus** (`app.get`, `res.send`).
- Sie können erklären, warum dies **kein klassisches Website-Frontend**, sondern ein **Backend-Server / API** ist.
- Sie kennen `nodemon` und einfache `npm scripts` für einen komfortablen Entwicklungsablauf.

Unser erster Express-Server

Öffnen Sie die Datei `index.js` und fügen Sie folgenden Code ein:

```
import express from 'express';

const app = express();
const port = 3000;

// Route für GET-Anfragen auf "/"
app.get('/', (req, res) => {
    res.send('Hello World');
});

// Server starten und auf Port 3000 auf Anfragen warten
app.listen(port, () => {
    console.log(`Example app listening on port ${port}`);
});
```

Was passiert in diesem Code?

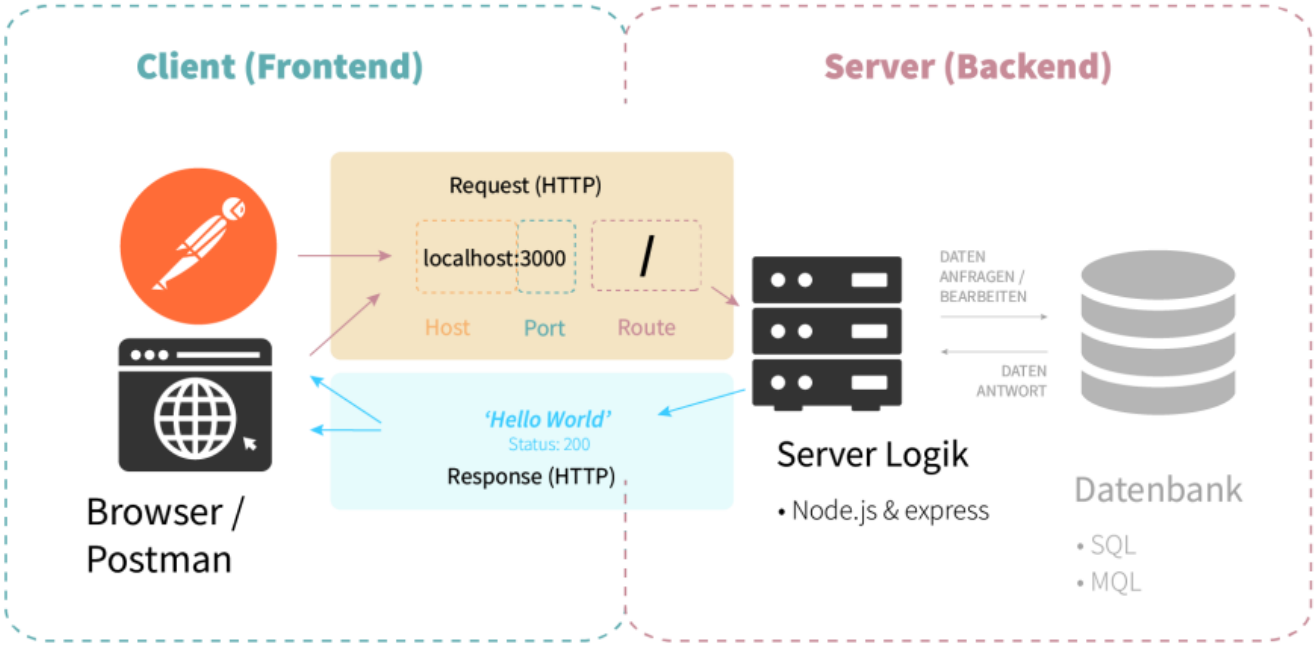
- `import express from 'express';` → Lädt das Express-Framework aus `node_modules`.
- `const app = express();` → Erstellt eine neue Express-Anwendung – das ist Ihr Serverobjekt.
- `const port = 3000;` → Definiert den Port, auf dem der Server lauscht.
- `app.get('/', (req, res) ⇒ { … });` → Definiert eine **Route**:
    - Wenn eine **GET-Anfrage** an `/` kommt,
    - führt Express die Callback-Funktion aus.
    - `req` = Request-Objekt (Infos über Anfrage),

`res` = Response-Objekt (Antwort, die Sie zurückschicken).

- `res.send('Hello World');` → sendet den Text `Hello World` als Antwort zurück an den

Browser oder Postman.

- `app.listen(port, () ⇒ { … });` → Startet den Server und gibt im Terminal eine kurze Meldung aus.



Das Schema zeigt, wie ein Zusammenspiel aus Anfrage (Request) und Antwort (Response) zwischen Client und Server funktioniert.

Server starten und im Browser testen

1. Öffnen Sie in WebStorm das **Terminal** (Projektordner).
2. Starten Sie den Server mit:

```
node index.js
```

1. Im Terminal sollte erscheinen: `Example app listening on port 3000`
2. Öffnen Sie Ihren Browser und geben Sie ein: `http://localhost:3000/`
3. Sie sollten nur das Wort **`Hello World`** im Browser sehen.
4. Server stoppen: Im Terminal `Strg + C` (Windows/Linux) oder `ctrl + C` (macOS) drücken.

Wichtig: Wir haben keinen „klassischen“ Webauftritt

programmiert

Zentrale Idee

Sie haben in dieser Learning Unit **kein HTML**, **kein CSS** und **kein JavaScript für den Browser** geschrieben. Sie haben **JavaScript für den Server** geschrieben.

- Der Browser spielt hier nur die Rolle eines **Clients**, der eine Anfrage an Ihren Server sendet.
- Der Express-Server beantwortet diese Anfrage mit einer Antwort (`Response`).
- Der Browser zeigt einfach an, was im **Response-Body** steht (in unserem Fall der Text `Hello World.`).

Später werden Sie:

- statt einfachem Text **JSON-Daten** zurückgeben,
- mehrere Routen (z.B. `/api/trips`, `/api/books`) definieren,
- und diese mit Ihrer **MySQL-Datenbank** verbinden.

Entwickler-Komfort: npm-Scripts & nodemon

Wenn Sie nach jeder Codeänderung den Server mit `node index.js` neu starten müssen, ist das mühsam. Dafür gibt es zwei Hilfsmittel:

- **`npm scripts`** in `package.json`
- **`nodemon`** für automatischen Neustart

1. nodemon installieren

Installieren Sie `nodemon` als Entwicklungs-Tool:

```
npm install --save-dev nodemon
```

Dadurch ergänzt `npm` Ihre `package.json` um einen Eintrag unter `devDependencies` für `nodemon`.

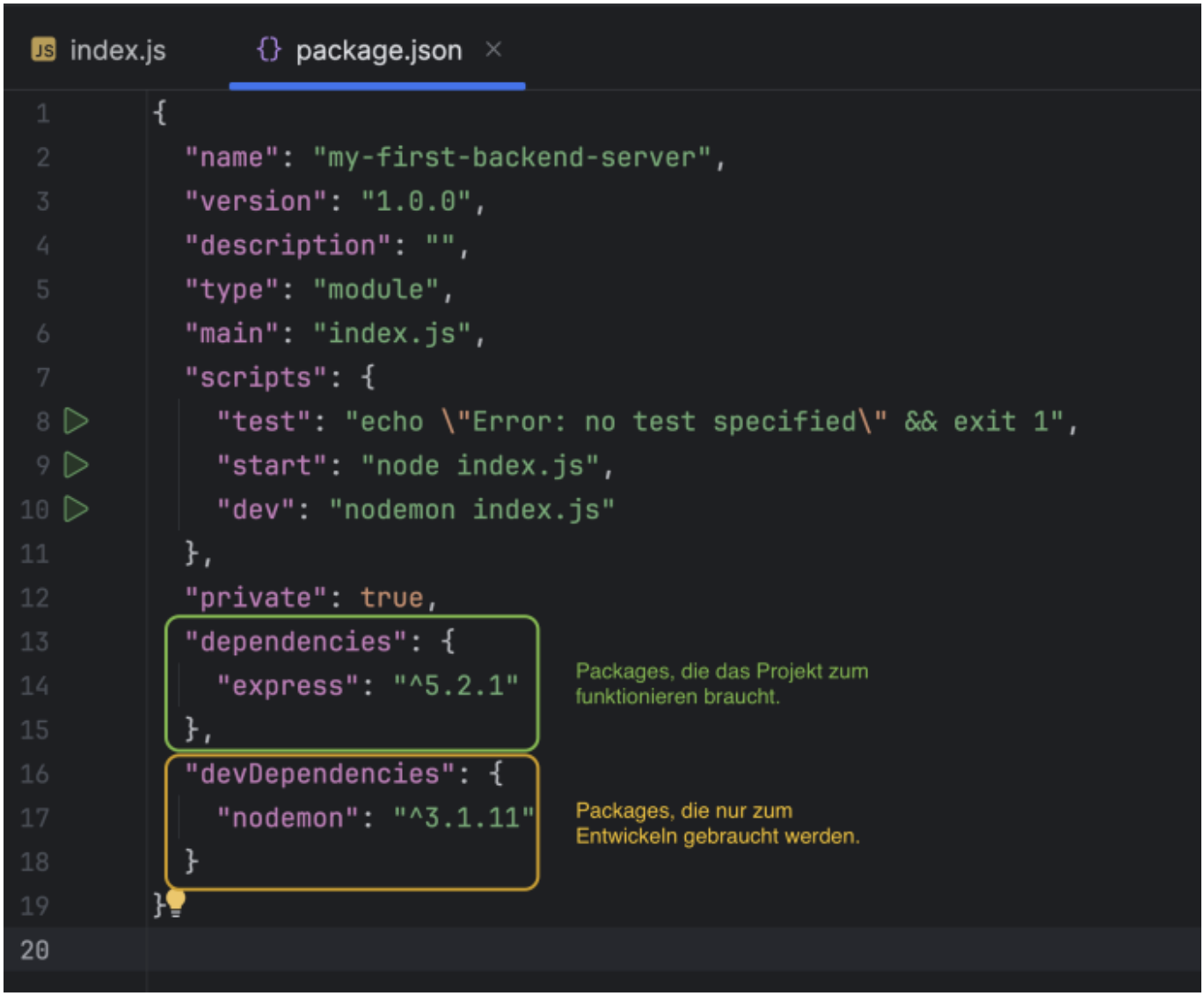
Was bedeutet `devDependencies`?

In `package.json` gibt es zwei wichtige Bereiche für Pakete:

- `dependencies` → Pakete, die Ihre Anwendung **zum Ausführen** braucht (z.B. `express`). Ohne diese Pakete kann der Server in Produktion nicht laufen.

- devDependencies → Pakete, die Sie **nur während der Entwicklung** brauchen (z.B. Test-Frameworks, Build-Tools – und nodemon).

nodemon ist ein typisches **Entwicklungs-Tool**: Es hilft Ihnen beim Programmieren (automatischer Neustart bei Änderungen), wird aber auf einem späteren Produktionsserver **nicht benötigt**. Darum speichern wir es mit `–save-dev` in den devDependencies.



2. Scripts in ''package.json'' ergänzen

Öffnen Sie `package.json` und ergänzen Sie den Abschnitt `scripts` wie folgt (Beispiel, wie im Screenshot oben):

```
"scripts": {
  "start": "node index.js",
  "dev": "nodemon index.js"
}
```

Damit definieren Sie zwei Startvarianten:

- **`npm start`**
  1. Startet den Server mit `node index.js`.
  2. Der Prozess läuft „ganz normal“ ohne automatischen Neustart.
  3. Das entspricht eher einem **Produktivbetrieb**: Der Server läuft stabil, Änderungen am Code erfordern ein manuelles Neustarten.
- **`npm run dev`**
  1. Startet den Server mit `nodemon index.js`.
  2. `nodemon` beobachtet Ihre Dateien und startet den Server **automatisch neu**, wenn Sie etwas ändern und speichern.
  3. Das ist ideal für die **Entwicklung im Unterricht**, weil Sie sofort die Wirkung Ihrer Änderungen sehen.

Wenn Sie nun in `index.js` (oder Ihrer Server-Datei) etwas ändern und speichern, startet `nodemon` den Server bei **`npm run dev`** automatisch neu. Sie müssen `node index.js` nicht jedes Mal von Hand ausführen.

node_modules & IDE-Hinweise

- Der Ordner `node_modules/` kann sehr gross werden. Er wird von `npm` automatisch verwaltet und muss **nicht** manuell bearbeitet werden.
- Für Abgaben (z.B. in Moodle) reicht meist:
  - Quellcode-Dateien (z.B. `index.js`)
  - `package.json` (und optional `package-lock.json`)
- In WebStorm können Sie bei Bedarf die **Inlay-Hints** (kleine Typ-Hinweise) deaktivieren, wenn diese Sie stören (`Settings → Editor → Inlay Hints`).

In den nächsten Learning Units werden Sie:

- weitere Routen in Ihrem Express-Server anlegen,
- Daten nicht nur zurückgeben, sondern auch **annehmen** (POST, PUT, DELETE),
- Ihre **persönlichen Use Cases** (z.B. Reisedatenbank, Lieblingsfilme, Geburtstage) mit einer echten MySQL-Datenbank verbinden und alle CRUD-Operationen **über den Backend-Server** ausführen.