

Electronic Business Development — Lab 3 — APIs, npm & Express

Contacts

Course Instructor

Dr. Nourhan Hamdi

nourhan.hamdi@giu-uni.de
M2.312

Teaching Assistant

Mr. Nour Gaser

nour.aly@giu-uni.de
M2.419

Teaching Assistant

Mr. Omar Alaa

omar.alaaeldin@giu-uni.de
M2.419

Teaching Assistant

Ms. Mariam Khalid

mariam.khalid@giu-uni.de
M2.414

Teaching Assistant

Mr. Younes Zied

younes.zied@giu-uni.de
M2.506

Teaching Assistant

Mr. Mahmoud Hani

TBD
M2.506



Lab agenda

Duration Topic

10 min	Recap: you already used an API
20 min	APIs, REST & HTTP
20 min	Starting a project with npm
15 min	Your first Express server
25 min	Serving listings with GET routes
—	Next lab: the full API



You already used an API

In Lab 2 you wrote this:

```
fetch("https://jsonplaceholder.typicode.com/users/1")  
  .then(res => res.json())  
  .then(user => console.log(user.name));
```

- Your code sent a **request** over the internet to someone else's server.
- That server sent back a **response**, with data in **JSON**.
- You never saw its code. You only knew **which URL to ask**.

Today we build the other side: **our own server**, answering requests of our own.



Meet ebd bn'b

Over the semester we build **ebd bn'b**, a simplified Airbnb: people list rental properties, and guests browse and book them.

It grows one layer at a time:

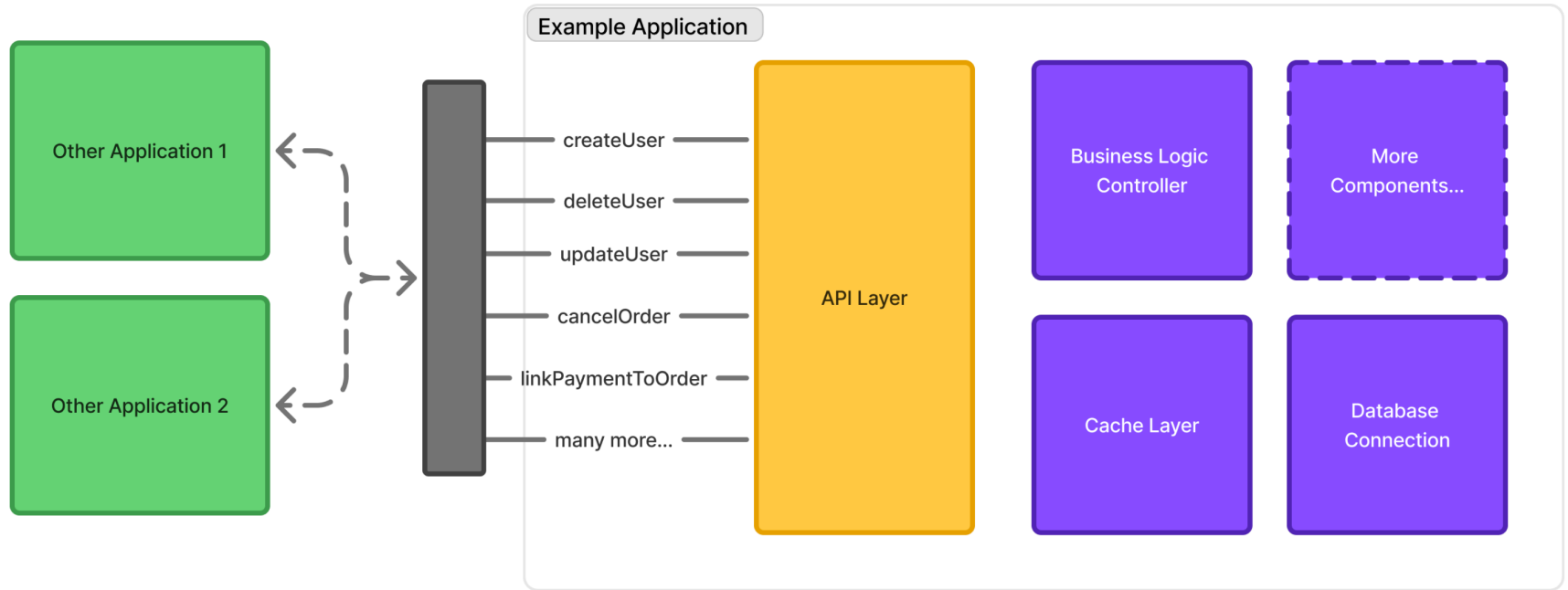
Labs	What we add
Today and next lab	An API with Express that serves listings
Later	A MongoDB database, so listings survive a restart
Later	A React front-end that shows them to guests

Every lab builds on the one before it. Keep your project, commit it, push it.



APIs, REST & HTTP

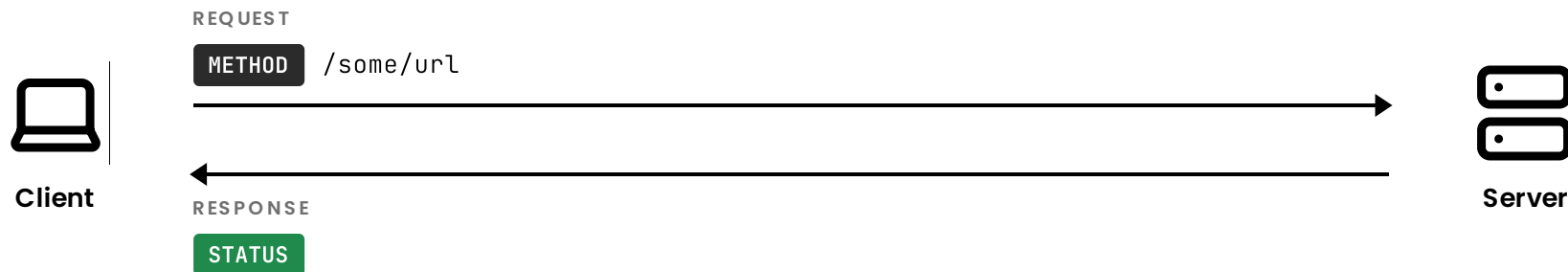
What is an API?





What is a REST API?

REpresentational State Transfer: a set of conventions most web APIs follow.



- Every **resource** has a URL: `/api/listings` is all listings, `/api/listings/2` is one.
- The **HTTP method** says what to do with it: `GET`, `POST`, `PUT`, `DELETE`.
- Data travels as **JSON**, and each request stands on its own.



Anatomy of a request



Client

REQUEST

GET /api/listings/2

RESPONSE

200 OK

```
{ "id": 2, "title": "Beach House", "pricePerNight": 140 }
```



Server

Part What it is

Method What you want done: GET reads, POST creates, PUT updates, DELETE removes

URL Which resource. :id -style parts pick one item

Body Data you send along. Only for POST and PUT ; a GET has none

Status How it went: 200 fine, 404 not found, 500 the server broke



The ebd bn'b API

The same URL with a different method is a different action.

Method	URL	Does
GET	/api/listings	Return all listings
GET	/api/listings/:id	Return one listing, or 404 if it does not exist
POST	/api/listings	Add a new listing
PUT	/api/listings/:id	Update a listing
DELETE	/api/listings/:id	Remove a listing

Today: the two GET routes. **Next lab:** the other three.

Status codes group by their first digit: 2xx it worked, 4xx the client asked for something wrong, 5xx the server broke.



Starting a project with npm

npm

The **Node Package Manager** is installed with  **Node.js**, so you already have it.

It does two jobs:

1. **Describes your project** in a file called `package.json`: its name, how to run it, and what it depends on.
2. **Downloads other people's code** (packages) from [npmjs.com](https://www.npmjs.com), the world's largest code registry.

We use it to install our first package:  **Express**.



npm

npm init

Inside an empty project folder:

```
npm init -y
```

`-y` accepts every default. It creates `package.json`.

Then edit two things by hand:

1. Set `"type"` to `"module"`, so `import` works.
2. Add a `"dev"` script that restarts the server whenever you save.

Run it with `npm run dev`.

```
{
  "name": "ebd-bnb",
  "version": "1.0.0",
  "type": "module",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "dev": "node --watch server.js"
  },
  "dependencies": {
    "express": "^5.1.0"
  }
}
```

The `dependencies` block appears on its own when you install Express. You never write it by hand.

npm install

```
npm install express
```

Three things change. Look at each one.

Anyone who clones your repo gets `node_modules` back with a single `npm install`, so it never goes into git.

File	What it is	Commit?
package.json	Gains "express" under dependencies	Yes
package-lock.json	The exact version of every package installed	Yes
node_modules/	The downloaded code: Express and the ~60 packages it needs	No
.gitignore	You create it, with one line: node_modules/	Yes



Set up ebd bn'b

1. Create a folder called `ebd-bnb` and open it in  **VS Code**.
2. Open a terminal and run `npm init -y`. Open `package.json` and read it.
3. Set `"type": "module"` and add the `"dev"` script from the previous slide.
4. Run `npm install express`. What appeared? What changed in `package.json`?
5. Create `.gitignore` containing `node_modules/`.
6. Make it a  **Git** repository, commit, and push it to a new  **GitHub** repo. Check on GitHub that `node_modules` is **not** there.



Your first Express server

Node.js + Express

 **Node.js** runs JavaScript outside the browser, which is what lets it be a server.

It can already handle web requests on its own, but the code is long and fiddly.

 **Express** is a small framework on top of Node that makes the common parts short:

- **Routes:** "when this method and URL arrive, run this function"
- **Responses:** send text or JSON in one line
- **Middleware:** more on that next lab



Node.js



Express

Hello, world

server.js

```
import express from "express";

const app = express();

app.get("/", (req, res) => {
  res.send("Hello world");
});

app.listen(3000, () => {
  console.log("Server running on http://localhost:3000");
});
```

- `app.get(path, handler)`: when a `GET` request for `path` arrives, run `handler`.
- `req` is the **request** that came in; `res` is the **response** you send back.
- `app.listen(3000)` starts the server on port 3000 of your machine:
`http://localhost:3000`.

Run it

```
npm run dev
```

Then open <http://localhost:3000> in your browser.

The server keeps running until you press **Ctrl+C** in its terminal.

`--watch` restarts it every time you save, so edit, save, and refresh.

✓ IF IT DOES NOT START

You see

Fix

Cannot use import statement outside a module	"type": "module" is missing from package.json
--	---

Cannot find package 'express'	Run <code>npm install express</code> in this folder
-------------------------------	---

EADDRINUSE: address already in use :::3000	Another server is still running; stop it with Ctrl+C
--	---

The browser can't connect

Is the terminal still running? Did it crash? Read the red text



Hello, ebd bn'b

1. Create `server.js` with the code from two slides back.
2. Start it with `npm run dev` and open `http://localhost:3000`.
3. Change the message to `Welcome to ebd bn'b`, save, and refresh. You never restarted anything.
4. Visit `http://localhost:3000/nothing-here`. What does Express send back when no route matches?
5. Commit and push.



Serving listings

A fake database

server.js

```
// Our fake database for now: a plain array that lives in memory
const listings = [
  { id: 1, title: "Nile View Apartment", pricePerNight: 85 },
  { id: 2, title: "Beach House", pricePerNight: 140 },
  { id: 3, title: "Desert Eco-Lodge", pricePerNight: 60 },
];
```

- For now, our listings live in a **plain array**, in the server's memory.
- Each listing is an object with an `id`, so we can find it again.
- A real database replaces this array in a few labs. Everything else barely changes.

Two GET routes

server.js

```
app.get("/api/listings", (req, res) => {
  res.json(listings);
});

app.get("/api/listings/:id", (req, res) => {
  const id = Number(req.params.id); // URL params are always strings
  const listing = listings.find((listing) => listing.id === id);

  if (!listing) {
    return res.status(404).json({ error: "Listing not found" });
  }
  res.json(listing);
});
```

- `:id` is a **route parameter**: for `/api/listings/2`, `req.params.id` is `"2"`, a **string**, so `Number(...)` converts it.
- `res.status(404)` sets the status code; `return` stops the function so we never respond twice.



Serve your listings

1. Add a `listings` array with **three** properties of your own: `id`, `title`, `pricePerNight`.
2. Add `GET /api/listings` and `GET /api/listings/:id`.
3. Test in the browser, and predict each answer before you press Enter:
 - `http://localhost:3000/api/listings`
 - `http://localhost:3000/api/listings/2`
 - `http://localhost:3000/api/listings/99`
 - `http://localhost:3000/api/listings/abc`
4. Press **F12**, open the **Network** tab and reload. Find the method and the status code of each request.
5. Commit and push.



Next lab: the full API

We build all of it together next lab. Try it before then if you like.

The full question

```
// ebd bn'b: the listings API
// We build this in Lab 4. Try it before then!

// 1. Import express and create an app

// ===== MIDDLEWARE =====
// 2. Read JSON request bodies (express.json)

// ===== DATABASE =====
// 3. A "listings" array of rental properties,
//     each with an id, title and pricePerNight

// ===== MODEL =====
// 4. Functions that are the ONLY code
//     allowed to touch the array:
//     getAllListings()
//     getListingById(id)
//     addListing(data)
//     updateListing(id, data)
//     removeListing(id)
```

```
// ===== CONTROLLERS =====
// 5. Read the request, call the model,
//     send the response:
//     handleGetAll -> 200 + all listings
//     handleGetOne -> 200 + listing, or 404
//     handleCreate -> 201 + new listing, or 400
//     handleUpdate -> 200 + listing, or 400/404
//     handleDelete -> 204 (no body), or 404

// ===== ROUTES =====
// 6. Connect each method + URL to a controller
//     GET    "/api/listings"
//     GET    "/api/listings/:id"
//     POST   "/api/listings"
//     PUT    "/api/listings/:id"
//     DELETE "/api/listings/:id"

// ===== START =====
// 7. Start the server on port 3000
```



Before next lab

Install an API client. The address bar can only send `GET`, so for `POST`, `PUT` and `DELETE` we need a tool that can send any method. Pick one:

-  **Postman**: a desktop app. postman.com/downloads
-  **Thunder Client**: an extension inside VS Code. Search "Thunder Client" in the Extensions panel.

Bring your `ebd-bnb` **project**, working and pushed. Next lab starts from it.

Model, controllers, routes: the section names in the question are explained properly next lab. For now, read them as "the part that touches the data", "the part that decides what to answer", and "the part that maps URLs".



Useful references

EXPRESS

[Express: official documentation](#)

[Express: Hello world](#)

[Express: basic routing](#)

[Introduction to Express \(GeeksforGeeks\)](#)

[Introductory video to Express](#)

HTTP & NPM

[MDN: an overview of HTTP](#)

[MDN: HTTP response status codes](#)

[npm docs: package.json](#)

Nothing on this slide is required.

Questions?

Open floor for any questions

