# Node.js

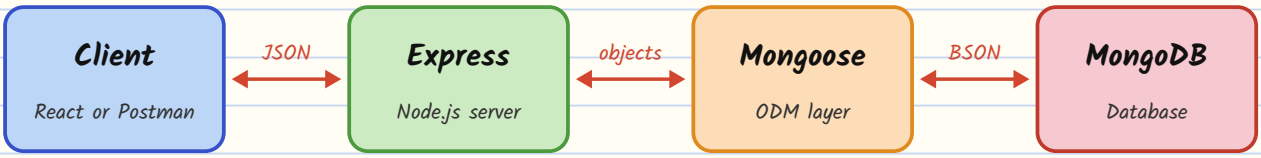
Node.js · Express · REST APIs · MongoDB, explained simply

Node.js Notes · easy to understand · free for students

## 1. Start with the Runtime

- **Node.js** runs JavaScript outside the browser, on your computer or a server. It is built on Chrome's **V8** engine.
- It is **event-driven** and **non-blocking**: while it waits for a file or database, it handles other requests. This is great for APIs, chat apps and streaming.

IDEA One waiter serves many tables. He does not stand in the kitchen waiting for one dish. He takes the next order and comes back when the food is ready. That is Node.




Request goes right, response comes back left

| Browser JavaScript | Node.js |
|---|---|
| Has window, document, DOM | Has process, fs, http, os |
| Talks to web pages | Talks to files, network, database |
| Runs on the user's device | Runs on your server |

- Install the **LTS** version from `nodejs.org`. Check with `node -v` and `npm -v`. Run a file with `node app.js`. Use `node --watch app.js` to restart on every change.
- Weak spot: heavy CPU work (video, big calculations) blocks the single main thread. Use worker threads or another service for that.

## 2. Modules and npm

- Every file is a **module**. Two systems exist. Use **ES modules** in new code by adding `"type": "module"` in `package.json`.

| | CommonJS (old) | ES modules (modern) |
|---|---|---|
| Import | `const x = require('x')` | `import x from 'x'` |
| Export | `module.exports = x` | `export default x` |
| File type | Default in older code | Needs type module or .mjs |

**EXAMPLE** Share a function between files.

```
// math.js
export const add = (a, b) => a + b;

// app.js
import { add } from "./math.js";
console.log(add(2, 3));    // 5
```

| npm command | What it does |
|---|---|
| `npm init -y` | Create package.json |
| `npm install express` | Add a library to dependencies |
| `npm install -D nodemon` | Add a tool only for development |
| `npm run dev` | Run a script from package.json |
| `npm uninstall x` | Remove a library |

**EXAMPLE** Scripts in package.json.

```
"scripts": {
  "start": "node server.js",
  "dev": "node --watch server.js"
}
```

- `node_modules` is huge. Never push it to Git, add it to `.gitignore`. `package-lock.json` locks exact versions, so keep it. In versions, `^4.1.0` allows minor updates.

# 3. Core Modules and Files

- **Core modules** come with Node. No install needed.

| Module | Use |
|---|---|
| fs | Read and write files |
| path | Build file paths that work on any system |
| http | Create a basic web server |
| os | Information about the computer |
| events | EventEmitter for custom events |
| crypto | Hashing and random values |
| process | Env variables (`process.env`) and arguments |

**EXAMPLE** Read and write a file.

```
import fs from "fs/promises";

await fs.writeFile("note.txt", "Hello");
const text = await fs.readFile("note.txt", "utf8");
console.log(text);       // Hello
```

**EXAMPLE** A server using only the http module.

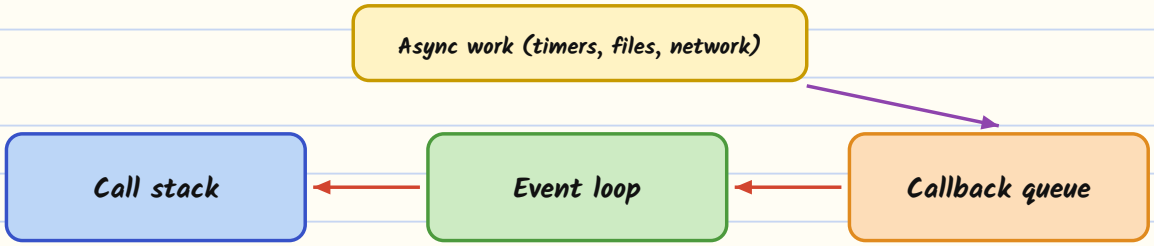
```
import http from "http";

const server = http.createServer((req, res) => {
  res.writeHead(200, { "Content-Type": "text/plain" });
  res.end("Hello");
});
server.listen(3000);
```

- Use **streams** for big files so the whole file is not loaded in memory. Use `path.join('a', 'b')` instead of joining strings with slashes.

# 4. Async Code and Events

- Slow tasks (files, database, network) are **asynchronous**. Three styles: **callbacks** (old), **promises** (`.then`), **async/await** (best, reads like normal code).




*When the call stack is empty, the loop moves the next callback onto it*

**EXAMPLE** async/await with error handling.

```
async function getUser(id) {
  try {
    const res = await fetch(`https://api.example.com/users/${id}`);
    return await res.json();
  } catch (err) {
    console.error(err.message);
  }
}
```

**EXAMPLE** Run tasks together.

```
const [a, b] = await Promise.all([getUser(1), getUser(2)]);
```

EXAMPLE What prints first.

```
console.log("1");
setTimeout(() => console.log("2"), 0);
Promise.resolve().then(() => console.log("3"));
console.log("4");        // prints 1, 4, 3, 2
```

- Order: normal code first, then **promise callbacks** (microtasks), then timers and I/O callbacks.

REMEMBER Never block the event loop. Avoid long sync loops and `readFileSync` inside request handlers. One slow request would freeze everyone.

EXAMPLE EventEmitter.

```
import { EventEmitter } from "events";

const bus = new EventEmitter();
bus.on("order", (id) => console.log("Order", id));
bus.emit("order", 101);       // Order 101
```

# 5. Express Basics

- **Express** is a small, popular web framework for Node. It makes routes, request data and responses easy. Install with `npm install express`.

EXAMPLE Your first Express server.

```
import express from "express";
const app = express();
app.use(express.json());          // read JSON bodies

app.get("/", (req, res) => res.send("Hello Express"));

app.listen(3000, () => console.log("Server on port 3000"));
```

| Object | Useful parts |
|---|---|
| req (request) | req.params URL parts, req.query after ?, req.body JSON, req.headers |
| res (response) | res.send(), res.json(), res.status(404), res.redirect() |

EXAMPLE Params and query.

```
app.get("/users/:id", (req, res) => {
  res.json({ id: req.params.id, page: req.query.page });
});
// GET /users/5?page=2   gives   { id: "5", page: "2" }
```

REMEMBER If `req.body` is undefined, you forgot `app.use(express.json())`. This is the most common beginner bug.

# 6. Routing and Middleware

- **Middleware** is a function `(req, res, next)` that runs between the request and the response. Use it for logging, login checks, parsing and validation. Order matters: they run top to bottom.

> IDEA Airport security. Every passenger passes the checks one by one, and each check either lets you through (`next()`) or stops you.

EXAMPLE A logger middleware.

```
const logger = (req, res, next) => {
  console.log(req.method, req.url);
  next();                    // go to the next step
};
app.use(logger);
```

| Middleware | Job |
|---|---|
| `express.json()` | Turn JSON body into req.body |
| `express.static('public')` | Serve files like images and HTML |
| `cors()` | Allow requests from your React app |
| `helmet()` | Add safe HTTP headers |
| `morgan('dev')` | Log every request |
| `express-rate-limit` | Stop too many requests |

### ➤ Split routes with Router

EXAMPLE routes/users.js and app.js.

```
// routes/users.js
import { Router } from "express";
const router = Router();
router.get("/", listUsers);
router.post("/", createUser);
export default router;

// app.js
app.use("/api/users", userRoutes);
```

# 7. Building REST APIs

- **REST** uses HTTP methods on URLs and sends **JSON**. Use plural nouns: `/api/todos`, `/api/todos/5`.

| Method | Route | Action | Success code |
|---|---|---|---|
| GET | /api/todos | List all | 200 |
| GET | /api/todos/:id | Get one | 200 |
| POST | /api/todos | Create | 201 |
| PUT / PATCH | /api/todos/:id | Update | 200 |
| DELETE | /api/todos/:id | Remove | 204 |

EXAMPLE In-memory CRUD (before using a database).

```
let todos = [];
app.get("/api/todos", (req, res) => res.json(todos));

app.post("/api/todos", (req, res) => {
  const todo = { id: Date.now(), text: req.body.text };
  todos.push(todo);
  res.status(201).json(todo);
});

app.delete("/api/todos/:id", (req, res) => {
  todos = todos.filter(t => t.id !== Number(req.params.id));
  res.sendStatus(204);
});
```

| Code | Meaning |
|---|---|
| 200 / 201 / 204 | OK / Created / No content |
| 400 | Bad request, the client sent wrong data |
| 401 / 403 | Not logged in / not allowed |
| 404 | Not found |
| 409 | Conflict, like a duplicate email |
| 500 | Server error |

- Good design: version it (`/api/v1`), paginate lists (`?page=1&limit=10`), and return errors in one shape like `{ message: 'Not found' }`.
- Test with **Postman**, Thunder Client (VS Code) or curl: `curl localhost:3000/api/todos`.

# 8. MongoDB Basics

- **MongoDB** is a **NoSQL** database. It stores JSON-like **documents** in **collections**. There is no fixed table structure, so documents can differ.

| SQL | MongoDB |
|---|---|
| Table | Collection |
| Row | Document |
| Column | Field |
| Join | Embedding, or $lookup and populate |
| Primary key | _id (ObjectId, made automatically) |

**EXAMPLE** One document.

```
{
  "_id": "665f1c2a9b1e8a0012ab34cd",
  "name": "Asha",
  "age": 20,
  "skills": ["node", "mongodb"],
  "address": { "city": "Delhi" }
}
```

- Get a free cloud database from **MongoDB Atlas**, or install it locally. **Compass** is the visual tool and **mongosh** is the terminal.

**EXAMPLE** CRUD in mongosh.

```
use shop
db.users.insertOne({ name: "Asha", age: 20 })
db.users.find({ age: { $gt: 18 } })
db.users.updateOne({ name: "Asha" }, { $set: { age: 21 } })
db.users.deleteOne({ name: "Asha" })
```

| Operator | Meaning |
|---|---|
| $gt $lt $in | Greater than, less than, in a list |
| $and $or | Combine conditions |
| $set $inc $push | Change a field, add a number, add to an array |

- **Embed** data you always read together (address inside user). **Reference** by id when data is big or shared (user id inside order). Add an **index** on fields you search often: `createIndex({ email: 1 }, { unique: true })`.

**EXAMPLE** Aggregation pipeline, total paid per user.

```
db.orders.aggregate([
  { $match: { status: "paid" } },
  { $group: { _id: "$userId", total: { $sum: "$amount" } } }
])
```

# 9. Mongoose and CRUD

- **Mongoose** adds **schemas** (shape and rules) and **models** (tools to read and write) on top of MongoDB. Install with `npm install mongoose`.

**EXAMPLE** Connect to the database.

```
import mongoose from "mongoose";
await mongoose.connect(process.env.MONGO_URI);
```

**EXAMPLE** Schema and model.

```
const userSchema = new mongoose.Schema({
  name:  { type: String, required: true, trim: true },
  email: { type: String, required: true, unique: true, lowercase: true },
  age:   { type: Number, min: 0 }
}, { timestamps: true });

export default mongoose.model("User", userSchema);
```

| Method | Does |
|---|---|
| `User.create(data)` | Insert a document |
| `User.find(filter)` | Get many. Chain `.sort()` `.limit()` `.skip()` `.select()` |
| `User.findById(id)` | Get one by id |
| `User.findByIdAndUpdate(id, data, { new: true })` | Update and return the new version |
| `User.findByIdAndDelete(id)` | Delete |
| `User.countDocuments()` | Count |

**EXAMPLE** Routes with Mongoose.

```
app.get("/api/users", async (req, res) => {
  const users = await User.find().sort({ createdAt: -1 }).limit(10);
  res.json(users);
});

app.post("/api/users", async (req, res) => {
  const user = await User.create(req.body);
  res.status(201).json(user);
});

app.put("/api/users/:id", async (req, res) => {
  const user = await User.findByIdAndUpdate(req.params.id, req.body, { new: true });
  if (!user) return res.status(404).json({ message: "Not found" });
  res.json(user);
});
```

EXAMPLE Link two collections with ref and populate.

```
// in Post schema
author: { type: mongoose.Schema.Types.ObjectId, ref: "User" }

// when reading
const posts = await Post.find().populate("author", "name");
```

# 10. Authentication and Tokens

- **Authentication** = who are you. **Authorization** = what can you do. Never store plain passwords. Save a **bcrypt** hash. A **JWT** proves the user is logged in.
- Flow: user logs in, server checks the password and sends a signed **token**, the client sends `Authorization: Bearer token` on every request, and middleware verifies it.

EXAMPLE Hash a password and make a token.

```
import bcrypt from "bcrypt";
import jwt from "jsonwebtoken";

const hash = await bcrypt.hash(password, 10);          // on register
const ok = await bcrypt.compare(password, user.password); // on login
const token = jwt.sign(
  { id: user._id },
  process.env.JWT_SECRET,
  { expiresIn: "1h" }
);
```

EXAMPLE Auth middleware.

```
export function auth(req, res, next) {
  const header = req.headers.authorization || "";
  const token = header.replace("Bearer ", "");
  try {
    req.user = jwt.verify(token, process.env.JWT_SECRET);
    next();
  } catch {
    res.status(401).json({ message: "Invalid token" });
  }
}
// use: app.get("/api/me", auth, (req, res) => res.json(req.user));
```

REMEMBER Keep `JWT_SECRET` in `.env`, never in code. Never send the password hash back: use `.select('-password')`. Keep token life short.

# 11. Errors, Security and Config

- Keep secrets in a **.env** file: `PORT`, `MONGO_URI`, `JWT_SECRET`. Load it with `node --env-file=.env server.js` (Node 20.6+) or the `dotenv` package. Add `.env` to `.gitignore`.

EXAMPLE 404 and global error handler (put them last).

```
app.use((req, res) => res.status(404).json({ message: "Route not found" }));

app.use((err, req, res, next) => {
  console.error(err);
  res.status(err.status || 500).json({ message: err.message });
});
```

- In **Express 5**, errors thrown in async handlers go to the error handler automatically. In Express 4, wrap the code in try/catch and call `next(err)`.

| Security step | Why |
|---|---|
| helmet | Safe HTTP headers |
| cors with exact origin | Only your frontend can call the API |
| express-rate-limit | Blocks brute force and spam |
| Validate input (Zod, Joi) | Never trust req.body |
| Avoid raw req.body in queries | Stops NoSQL injection like { $gt: '' } |
| express.json({ limit: '10kb' }) | Blocks huge bodies |
| HTTPS and bcrypt | Protect data on the way and at rest |

EXAMPLE Clean folder layout.

```
server/
  src/
    config/db.js     routes/      controllers/
    models/          middleware/  app.js
  server.js    .env    package.json
```

- Deploy on Render or Railway with MongoDB Atlas. Set the env variables there, read the port from `process.env.PORT`, and use **PM2** or Docker for a long-running process.

# 12. Projects and Practice

| Project | What you practice |
|---|---|
| 1. Notes API | CRUD, Express routes, Mongoose |
| 2. URL shortener | Params, redirects, unique ids, indexes |
| 3. Auth system | Register, login, bcrypt, JWT, roles |
| 4. Blog API | Posts, comments, populate, pagination |
| 5. E-commerce backend | Products, cart, orders, validation, aggregation |
| 6. Chat app | Socket.io, events, real-time updates |

## Common mistakes

| Mistake | Fix |
|---|---|
| req.body is undefined | Add `express.json()` before routes |
| Forgetting await | Await every promise from the database |
| Blocking sync code in handlers | Use async versions |
| Secrets in the code | Use .env and .gitignore |
| Sending password hash back | Use `.select('-password')` |
| CORS error in the browser | Use `cors()` with your frontend origin |

## Practice questions

1. What is Node.js? Why is it called non-blocking?
2. Explain the event loop with an example.
3. CommonJS vs ES modules.
4. What is middleware in Express? Write one.
5. PUT vs PATCH. Which status code for create and delete?
6. SQL vs MongoDB. When do you choose MongoDB?
7. What is Mongoose? Write a schema with validation.
8. How does JWT authentication work?
9. How do you secure an Express API?
10. Build a Notes REST API with Express and MongoDB.

- Build one complete API first with login. Then connect it to a React frontend and deploy it.