Environment + Config: A Different Setting for Every Machine

Your laptop and the real server are different machines: dotenv keeps ports and secrets in one .env file per machine — never inside the code.

Introduction

Your code runs in more than one place: your laptop today, a classmate's machine tomorrow, a real server when the project goes live (the roadmap's 7.6 topic is this same idea from the hosting side). Each machine honestly differs — the free port, the database address, the JWT secret, whether it should behave as a careful live system or a chatty practice one. Beginners hard-code these values, then edit the code for every machine and eventually share a file with a **real password inside it**.

The professional pattern separates **code** from **configuration**. The code never changes; each machine keeps a small text file named `.env` holding its own settings, and the **dotenv** package loads them into `process.env`, where the code reads them like any variable. Secrets stop travelling with the code — and the same project runs everywhere by changing one little file, never the program. *(Roman Urdu: Code sab ke liye aik, setting har machine ki apni — aur raaz hamesha `.env` mein, code mein kabhi nahi.)*

Key points

- **`.env` is a plain settings file:** one `NAME=value` per line — `PORT=3000`, `MONGO_URL=...`, `JWT_SECRET=...`. No quotes needed for simple values, no commas, no JavaScript — it is a list, not code.
- **dotenv loads it once, at the very top:** `require("dotenv").config()` reads `.env` and places every line into `process.env`. It must run before any line that reads a setting, so it stands among the first lines of the server file.
- **Read settings with a fallback:** `const PORT = process.env.PORT || 3000;` — if the machine provides a port (real hosting always does), use it; otherwise the practice default serves. Lesson 7.4's hosting depends on exactly this line.

  (Roman Urdu: Machine jo de woh lo, warna apni pehchan wali setting — `||` yehi kaam karta hai.)
- **Everything in `process.env` is text:** `process.env.PORT` is the string `"3000"`, not a number. Convert with `Number(...)` when a real number is needed — the same text lesson as route parameters in N.3.
- **Development vs production is a setting too:** `NODE_ENV` is the community's agreed name — `development` on your laptop (chatty logs, open error details) and `production` live (calm logs, hidden details — lesson N.9's behaviour switch).
- **`.env` is never shared:** it goes into `.gitignore` on day one and is never uploaded, emailed or committed. What you *do* share is `.env.example` — the same file with the values emptied — so a teammate knows which settings exist without knowing any secret.

- **Missing settings should fail loudly at start:** if `MONGO_URL` is absent, crashing at startup with a clear message beats running a broken server that fails mysteriously per request.

Code examples

Example 1 — The .env file and reading it

```
PORT=3000
MONGO_URL=mongodb://localhost:27017/codescope
JWT_SECRET=meri-gizli-chaabi-786
NODE_ENV=development
```

```
// server.js — the dotenv line goes at the very top
require("dotenv").config();
const express = require("express");

const app = express();
const PORT = process.env.PORT || 3000;

console.log("Mode:", process.env.NODE_ENV);
console.log("Database:", process.env.MONGO_URL);

app.get("/", (req, res) => {
  res.send("Server in " + process.env.NODE_ENV + " mode");
});

app.listen(PORT, () => {
  console.log("Server running on port " + PORT);
});
```

What it does: Four settings live in `.env`; one `config()` line at the top loads them all. From then on the code reads `process.env.NAME` wherever a value is needed — the port for `listen`, the mode for messages, the database string for Mongoose (lesson N.6). Notice the code contains **no secret and no machine-specific number at all**: copy this folder to any computer with its own `.env`, and it runs correctly there without a single edit.

What you will see: the terminal prints the mode and database lines from *your* `.env`, the server starts on port 3000, and `/` answers `Server in development mode`. Change `PORT` to `5000` in `.env`, restart — the same untouched code now listens on 5000.

Example 2 — Fail loudly: settings the app cannot live without

```
require("dotenv").config();

const required = ["MONGO_URL", "JWT_SECRET"];
for (const name of required) {
  if (!process.env[name]) {
    console.log("Missing setting: " + name + " — add it to .env and restart");
    process.exit(1);
  }
}

console.log("All required settings present — starting the server");
```

What it does: Before anything else starts, a small guard walks the list of settings the app truly cannot run without. A missing one is reported by name, with the remedy in the same line, and the process exits deliberately (`process.exit(1)` — the honest full stop). Compare the alternative: booting happily and then failing inside some poor user's request an hour later, with no hint that a setting was the cause.

What you will see: with both settings present, the "all present" line prints and startup continues; delete the `JWT_SECRET` line from `.env` and the server refuses to start, naming exactly what is missing.

Example 3 — Sharing the shape, not the secrets

```
PORT=
MONGO_URL=
JWT_SECRET=
NODE_ENV=
```

```
.env
node_modules
```

What it does: The first block is `.env.example` — committed and shared proudly. It lists every setting's *name* with empty values: a new teammate copies it to `.env` and fills in their own machine's values. The second block is `.gitignore` — the two lines that keep the real `.env` (and the downloaded-packages folder from lesson N.1) out of every commit and upload. Together these two tiny files are the whole etiquette: the shape is public, the secrets are private.

What you will see: a fresh copy of the project made from the shared files contains `.env.example` and **no** `.env` — the project politely refuses to run until the new owner creates their own `.env`, exactly as Example 2 taught it to.

Quick recap

- **Configuration lives outside code**, in a per-machine `.env` file of `NAME=value` lines, loaded by **dotenv** into `process.env`.
- Read with fallbacks (`process.env.PORT || 3000`); remember env values are always **text**.
- `NODE_ENV` separates **development** behaviour from **production** behaviour.
- `.env` is never shared or committed — `.env.example` shares the shape, `.gitignore` enforces the privacy, and missing required settings **fail loudly at startup**.

Common mistakes

1. **Hard-coding a secret "just for testing"** — the test file gets shared, committed or screenshotted, and the test password becomes a real breach. The fix: the first time a value differs per machine or must stay private, it goes into `.env` — no temporary exceptions.
2. **Loading dotenv too late** — calling `config()` after the lines that read the settings means those reads already happened and got `undefined`. The fix: `require("dotenv").config()` is the first meaningful line of the entry file, before routes, models or connections.
3. **Committing `.env` once and thinking deleting it later fixes it** — in git, the old version stays in the history forever, password included. The fix: `.env` enters `.gitignore` **before the first commit**; if a secret ever did land in history, change the secret itself — it is compromised, not merely misplaced.

Practice

1. Move the students API's port and a new setting, `SCHOOL_NAME`, into `.env`. The home route should answer `Welcome to <SCHOOL_NAME>`. Run the server twice with two different `.env` values and confirm the code file never changed between runs.
2. Create `.env.example` and `.gitignore` for the project from Example 3, then deliberately remove `MONGO_URL` from your `.env` and start the server with the Example 2 guard in place. Write down the exact message — this is the message a teammate should meet on their first run.

Practice quiz

1. What is a .env file, and how is it loaded?

- A JavaScript file, imported like any module
- A plain settings file, one NAME=value per line, loaded once at the very top with require("dotenv").config(), which places every line into process.env
- A JSON file, parsed by hand
- A database table

Answer: A plain settings file, one NAME=value per line, loaded once at the very top with require("dotenv").config(), which places every line into process.env

2. What does const PORT = process.env.PORT || 3000; do, and what type is process.env.PORT?

- It always uses 3000, and PORT is a number
- If the machine provides a port (real hosting always does) use it, otherwise the practice default serves; and everything in process.env is text, "3000" not 3000, so convert with Number(...) when a real number is needed
- It sets the port to 3000 forever
- It reads the port from package.json

Answer: If the machine provides a port (real hosting always does) use it, otherwise the practice default serves; and everything in process.env is text, "3000" not 3000, so convert with Number(...) when a real number is needed

3. What do you share with a teammate instead of .env, and why?

- Nothing, teammates guess the settings
- .env.example, the same file with the values emptied, so a teammate knows which settings exist without knowing any secret; .env itself goes into .gitignore on day one and is never uploaded, emailed or committed
- A copy of .env with a warning
- Only the passwords, the rest is obvious

Answer: .env.example, the same file with the values emptied, so a teammate knows which settings exist without knowing any secret; .env itself goes into .gitignore on day one and is never uploaded, emailed or committed

---

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/node-environment-config · More free tools and notes at https://scopetreasure.com