



The Development Environment & Servers Mastery Workbook

Table of Contents

 Prerequisites	1
How to Use This Workbook	2
 Philosophy Behind This Workbook	3
DE & Servers	4
1. The Terminal	4
2. Setting Up a Professional Workspace	5
3.1 Version Control with Git (Windows)	6
3.2 Version Control with Git (macOS)	7
4. Node.js	8
5. Managing Dependencies	10
6. Building & Bundling Apps	11
7. Running Apps Locally	11
8. Understanding Servers	12
9. Serving Dynamic Content	13
10. Persistence (Saving Data)	13
11. Deployment	14
12. Monitoring & Maintenance	15



Prerequisites

Before starting, make sure you can:

- A solid understanding of **HTML, CSS, and JavaScript fundamentals** — including variables, functions, loops, arrays, and basic DOM manipulation.
- Ability to **operate your computer's file system** — creating, renaming, and managing folders.
- Familiarity with **installing software** on your operating system (Windows or macOS).
- Comfort using a **code editor** such as Visual Studio Code — opening projects, saving files, and using the integrated terminal.
- Basic ability to **copy, paste, and execute commands** in a terminal or command-line interface.

How to Use This Workbook

This document is **not a textbook**. It will not hand you the answers. Instead, it **gives you the right questions to ask yourself** — questions every developer must be able to answer to master the topic at a global standard.

Here's how to use it effectively:

1. Ask Yourself First

- Before looking things up, try to explain the answer in your own words.
- If you can't, that's fine — it means you found a gap in your knowledge.
- If a **new question pops up in your own mind** that's not in here, that's your curiosity leading you deeper — write it down and explore it.

2. Leverage All Resources

- Use **Google**, **Stack Overflow**, and **ChatGPT** to research.
- Read documentation, articles, and examples.
- Find a way to practice and produce results.

3. Learn by Doing

- Each section has **project exercises**.
- Completing these exercises forces you to practice and discover the answers naturally.
- Don't skip them — doing is how you'll turn "theory" into mastery.

4. Reflect and Explain

- After finding an answer, try teaching it back:
 - Explain to a friend, or a fellow developer.
 - Write notes.
 - Or even record yourself explaining.
- **If you can explain clearly, you've truly learned it.**

5. Iterate and Improve

- Revisit questions regularly.
- As you grow, your answers will become deeper and more precise.

Philosophy Behind This Workbook

This is a “**find the answer within yourself**” document — **the programming version.**

- The **questions** represent the knowledge every web developer must internalize.
- **Be curious** → always ask “why does this work this way?”
- The **resources** (Google, Stack Overflow, ChatGPT) are your tools — but the true goal is that **the understanding lives inside you**, not just in your search history.
- The **exercises** are opportunities to struggle, explore, and discover.
- **Expect mistakes** → debugging is how you learn.
- **Reflect** → explain new concepts in your own words.

By the time you’ve asked and answered everything here — and built the exercises — you won’t just “know Development Environment & Servers.” You’ll **understand them so deeply that you can build, debug, and explain any project with confidence.**

DE & Servers

1. The Terminal

- What is a terminal (or command line), and why do developers still use it?
- What is the difference between a terminal, a shell, and a command prompt?
- How do I navigate folders (cd, ls / dir)?
- How can I create, rename, and delete files or folders from the terminal?
- How do I run a JavaScript file with node?
- What is the meaning of . (dot) and .. (double dot) in paths?
- How do I check which folder I'm currently in (pwd)?
- How do pipes (|) and redirection (>, >>) work?
- What is an environment variable, and how can I view or set it from the terminal?
- How do I stop a running process (Ctrl+C)?
- What are command history and auto-completion, and how do they speed up work?
- Why is learning the terminal important even when you use a GUI editor?

Exercise – Terminal Explorer

- Open your system terminal (or VSCode's integrated terminal).
- Create a folder called `terminal-playground` on your desktop.
- Inside it:
 - Make two files: `hello.txt` and `notes.txt`.
 - Rename `hello.txt` to `greeting.txt`.
 - Append text to `notes.txt` using `echo` and `>>`.
- Create a subfolder `scripts`, move `greeting.txt` there.
- Write a simple JS file `hello.js`:

2. Setting Up a Professional Workspace

- What is a Development Environment and why is it called that?
- How is it related to project folders, files, and organization?
- How should I organize a project's files for clarity?
- Which VSCode extensions improve speed (Prettier, ESLint, GitLens)?
- What is NodeJS, and what problem does it solve?
- What's the difference between running JS in the browser vs Node?
- When do I use DevTools vs the terminal?
- Why does folder organization matter for collaboration?

Exercise – Configure Your Environment

- Create a folder `my-first-env`.
- Open it in VSCode, add a basic `index.html` and `script.js`
- Run `node -v` and `npm -v` on your terminal to verify Node/npm are installed.

3.1 Version Control with Git (Windows)

- What is Git, and why do developers use it?
- What's the difference between Git (local) and GitHub (remote)?
- What does `git init` do inside a project folder?
- What are commits, and why do they need messages?
- What is a branch? Why not just edit `main` all the time?
- What does it mean to “push” and “pull”?
- How can you view your Git history?
- What happens if you delete your local project — how can you recover it using GitHub?

Exercise – My First Git Project (Windows)

- Download and install git from their official website. Make sure to enable git in command prompt terminal or any preferred terminal
- Open your project directory via terminal
- Initialize Git using `git init`.
- Configure your name using `git config -global user.name "Type name here"` and your email using `git config -global user.email "type@email.here"`
- Create a file `hello.js`, add some code.
- Stage using `git add .` and commit using `git commit -m "Initial commit"`
- Create a new GitHub repo and copy its HTTPS URL
- Connect using `git remote add origin <your-repo-url>` then `git branch -M main` and push using `git push -u origin main`
- Check GitHub and verify if your code appeared
- Make a small change in your codes -> commit -> push again

3.2 Version Control with Git (macOS)

💡 Note: Read **3.1 Version Control on Windows** first — it contains general knowledge about Git and essential commands that apply to both Windows and macOS users.

- Why is the `gh` CLI useful compared to raw Git commands?
- What's the difference between `git push` and `gh repo create`?
- How does authentication work using GitHub CLI (`gh auth login`)?
- How do you create and clone repositories from the terminal?
- How can you check if your local folder is already linked to a GitHub repo?
- How do you make a pull request from the terminal?

Exercise – Using the GitHub CLI (macOS)

- Install the GitHub CLI using brew install gh on any terminal on your MacBook
- Authenticate with GitHub
 - run `gh auth login`
 - Choose GitHub.com
 - Choose HTTPS
 - Confirm using your browser.
- Inside your project folder:
 - `git init`
 - `git add .`
 - `git commit -m "Initial Commit"`
- Create and link a repo directly from the terminal:
 - `gh repo create my-first-repo --public --source=. --remove=origin`
 - `git push -u origin main`
- Edit your code and commit again:
 - `git add .`
 - `git commit -m "Update code"`
 - `git push`
- Open a pull request directly from the terminal using `gh pr create --fill`

4. Node.js

- What is Node.js, and why was it created?
- How is Node.js different from the JavaScript we use in browsers?
- What is the **V8 engine**, and what does it do?
- What is a **runtime environment**, and what role does it play?
- How can Node.js run both command-line programs and web servers?
- How do **modules** work in Node.js (require, import)?

- What are **CommonJS** and **ES Modules**, and how do they differ?
- What are the global objects available in Node (e.g., `__dirname`, `process`, `console`)?
- What happens when you run `node app.js` in your terminal?
- What is the **event loop**, and how does it make Node handle many connections efficiently?
- How does Node handle file operations and networking without freezing your app?

Exercise – “My First Node Script”

- Using a terminal, make a new folder
- Create a file ‘hello.js’

```
console.log("Hello from Node.js!");
```

- Run the file using `node hello.js`
- Observe what happens in your terminal
- Create another file ‘math.js’:

```
function add(a, b) {  
  return a + b;  
}
```

```
module.exports = { add };
```

- In app.js, import and use it:

```
const { add } = require('./math');  
console.log("5 + 3 =", add(5, 3));
```

- Run the app using `node app.js`
- Observe what happens

5. Managing Dependencies

- What is npm, and what problem does it solve?
- What is the difference between **Node.js** and **npm**?
- How does npm work?
- What is package.json and why is it important?
- How do you properly manage the package.json of your project?
- What’s the difference between dependencies and devDependencies?

- Why shouldn't you edit `node_modules` directly?

Exercise – Add a Library to your project.

- Run `npm init -y` on your terminal
- Install `axios` and check how it appears in `package.json`.

6. Building & Bundling Apps

- Why do we need bundlers or compilers?
- What's the difference between development and production builds?
- What are source maps and tree-shaking?

Exercise – Try a Bundler

- On a new folder, install `Vite` (`npm create vite@latest`) on your project. Run it outside your existing project as it will create a new project for you.
- Explore the generated structure.

7. Running Apps Locally

- Why isn't opening an HTML file with `file://` enough for modern apps?
- What does a local development server actually do?
- How do I start and stop a local server safely?
- What is the difference between static serving and dynamic serving?
- What is CORS, and when does it appear during local testing?

- How do browser tools (Network tab, Console) help while the app runs locally?
- What are localhost and 127.0.0.1?

Exercise – Local Playground

- On your vite project, serve the app locally using: `npx serve` or `vite preview` on the terminal
- Open the given localhost URL.
- Add a simple fetch call to <https://jsonplaceholder.typicode.com/users>.
- Display results on the page
- Observe changes and reloading behavior.

8. Understanding Servers

- What happens step-by-step when you type a URL in the browser?
- What is a request, and what is a response?
- What does it mean for a server to “listen” on a port?
- What are HTTP methods (GET, POST, PUT, DELETE), and when are they used?
- What are status codes, and why are they important?
- What’s the difference between a web server and an app server?

Exercise – Your First Server

- Make a folder “simple-server”
- Initialize node using `npm init -y` and `npm i express` on your project terminal
- Create `server.js` which serves your project into `localhost` port `3000` using `express`
- Run the file in your terminal using `node server.js`
- Visit `localhost:3000` in your browser

9. Serving Dynamic Content

- How can a server send different data depending on the route?
- What are routes and route parameters?
- When should a server respond with HTML vs JSON?
- How do you send query parameters (`/search?q=term`)?
- What’s the difference between query params and body data?
- What is a template engine, and why might you use one?

Exercise – Express Routing

- Extend your `express` app with

```
app.get('/user/:name', (req, res) => {  
  res.send(`Hello ${req.params.name}!`);  
});
```

- Add a `/data` route returning JSON.
- Test routes in the browser and observe URL patterns

10. Persistence (Saving Data)

- Why can’t we store everything in memory?

- What are databases, and what problems do they solve?
- What's the difference between SQL and NoSQL?
- What is CRUD (Create, Read, Update, Delete)?
- How can environment variables protect DB credentials?

Exercise – Local JSON Database

- Create db.json manually with a few users
- Use the fs module to:
 - Read existing data
 - Add a new entry
 - Rewrite the file
- Create routes to GET and POST users
- Verify that data persists after restarting

11. Deployment

- What does “deploying” mean in practical terms?
- How is deployment different from local testing?
- What are build commands (`npm run build`) and why are they needed?
- What services can host Node apps for free (Render, Railway, Vercel)?
- What are environment variables in production?
- Why is HTTPS important?

Exercise – Deploy your App

- Push your project to GitHub.
- Create a free account on Render or Railway.
- Link your repo and deploy your Express app.
- Set environment variables if needed.
- Test your live URL and share it.

12. Monitoring & Maintenance

- How do you know your deployed app is working correctly?
- What are logs, and how do you read them?
- How can you catch and handle errors gracefully?
- What happens if your app crashes — how can you restart it?
- How do you safely update dependencies in production?

Exercise – Keep It Alive

- Add error handling middleware in your Express app:

```
app.use((err, req, res, next) => {  
  console.error(err.stack);  
  res.status(500).send('Something broke!');  
});
```

- Deploy the updated version.
- Trigger a test error to confirm logs appear on your host dashboard.
- Update one dependency and redeploy.