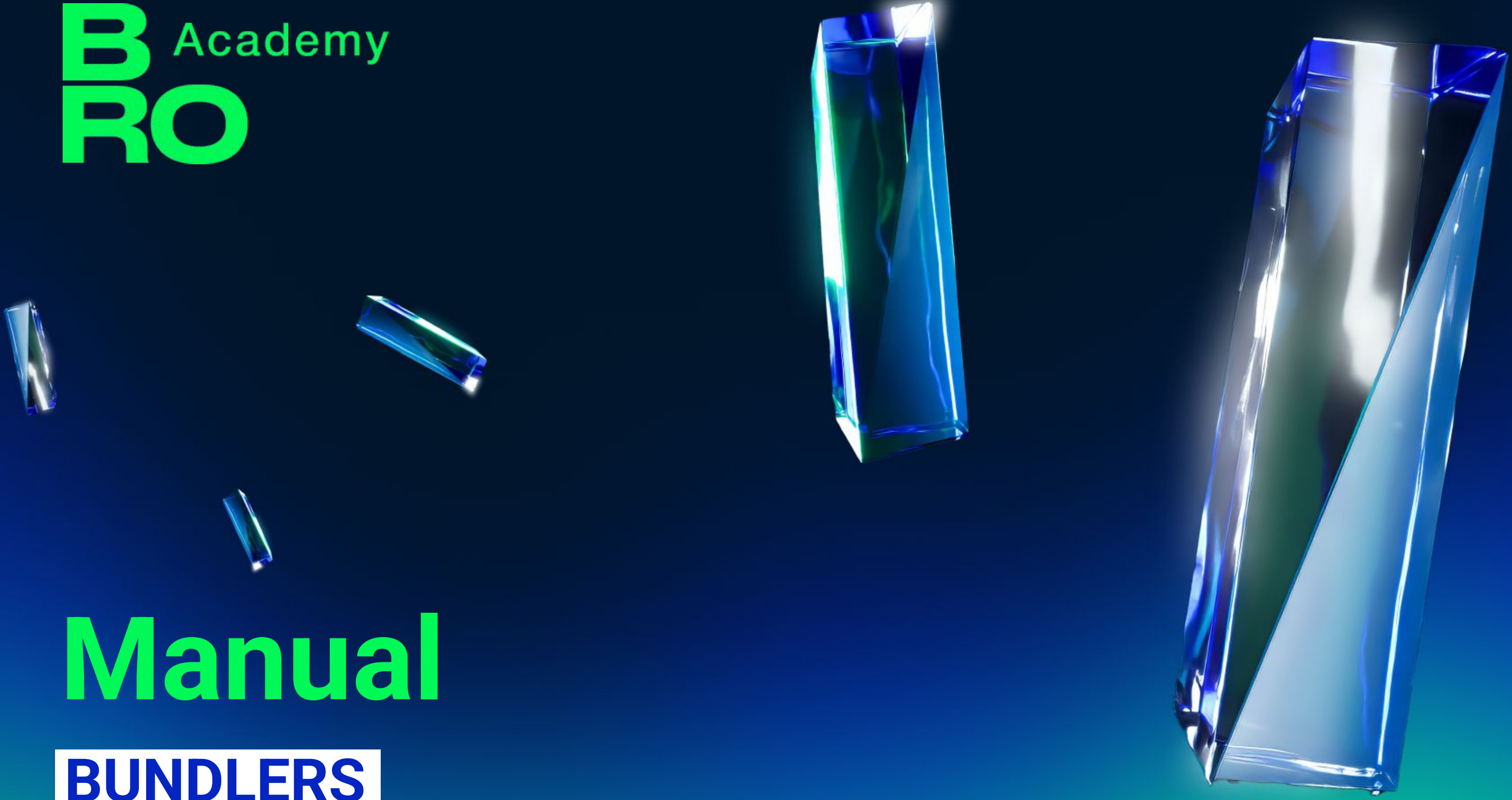
BRO Academy



# Manual

BUNDLERS

# Automation

**Web development** involves many routine tasks that can be significantly simplified through **automation**. In this text, we will explore how automation helps developers work **faster, more efficiently, and with fewer errors**.

The **main goal of automation** is to eliminate repetitive actions for developers.

Automation is the process of transferring routine development tasks, which can be performed manually, to a computer. It converts files written in a developer-friendly format into compressed code that is optimized for browsers and generates the final application package.

Automation **saves developers' time**, helps **identify and fix errors**, and **optimizes code** for browsers, which improves **website loading speed**.

A **bundler** is a tool that helps assemble (or put together) all parts of your project so that it works as a whole.

Imagine you are putting together a **puzzle**. You have many different pieces, and you need to fit them together to create a complete picture.

Similarly, a **web project** consists of different parts, such as **HTML, CSS, JavaScript, images, and other files**. A bundler helps take all these pieces and combine them into a **final, optimized version** of the project.

So, a **bundler** is a tool that helps gather all the parts of a project and make it **more efficient and well-structured**.

🔹 **A file packager** 📦

🔹 A tool that collects files into one, optimizes them, and speeds up the website. 🚀

💡 Many brilliant things are born from laziness. Build systems make developers' lives easier! 😎

## Types of bundlers

- **Webpack** – The most popular and powerful, but complex
- **Parcel** – Simple, but slow
- **Vite** – Fast and convenient

💡 **We will use Gulp**

## What does a bundler do?

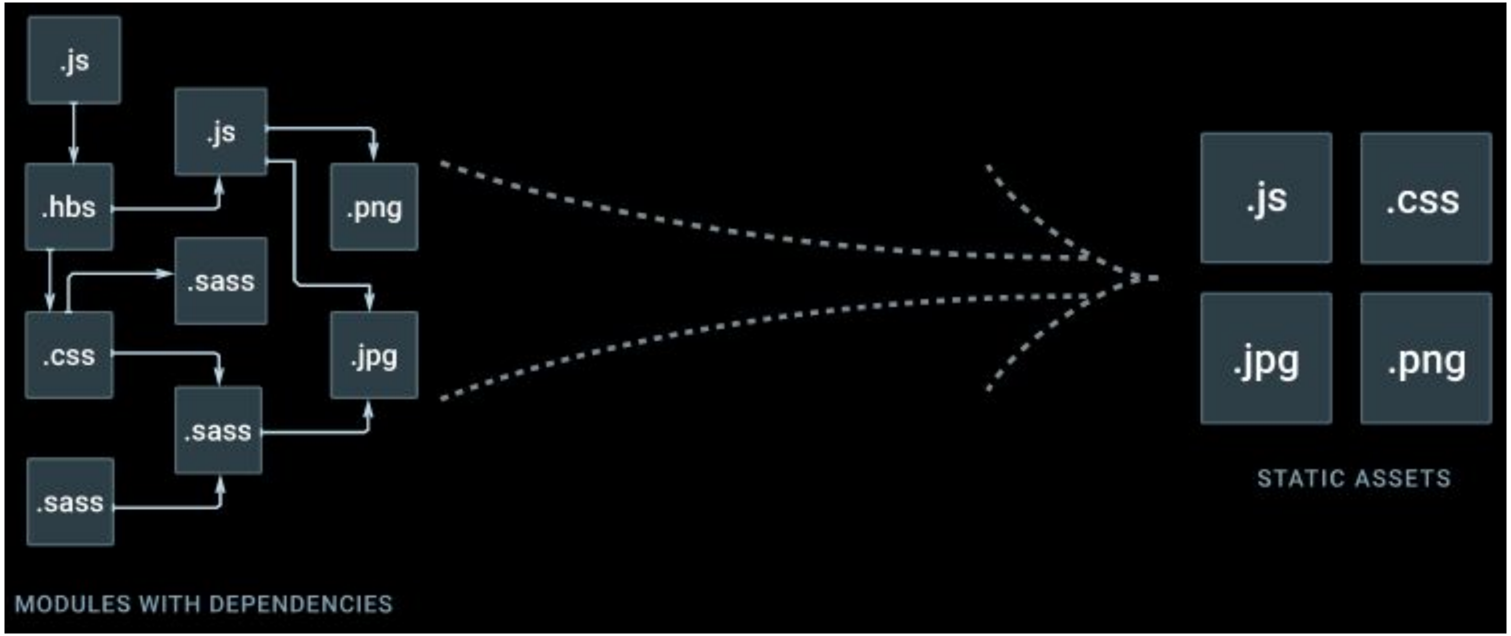
✅ Combines files

File transformation:
✅ Removes duplication
✅ Minifies code
✅ Supports SCSS, PostCSS, images

Without a bundler:

❌ Many separate CSS and JS files
❌ Code duplication makes maintenance difficult
❌ Slow loading (no minification)
❌ No convenient tools (SCSS, PostCSS, images)

Automation also helps speed up a website by optimizing files. For example, without **image compression**, **CSS and JavaScript minification**, a page may take **up to 5 seconds** to load. However, automatic processing of these resources can reduce the loading time to **just 1 second**, significantly improving the **user experience**.


.js
.js
.hbs
.png
.sass
.css
.jpg
.sass
.sass
MODULES WITH DEPENDENCIES
.js
.css
.jpg
.png
STATIC ASSETS

## 1️⃣ Development Server & Live Reload

📌 **Runs a local development server with auto-reloading:**

- `browser-sync` → **Serves files and refreshes browser automatically** when changes occur.

💡 **Why?**

✅ **Instant feedback** while coding (no need to manually refresh).

✅ Allows **multiple devices to preview changes live**.

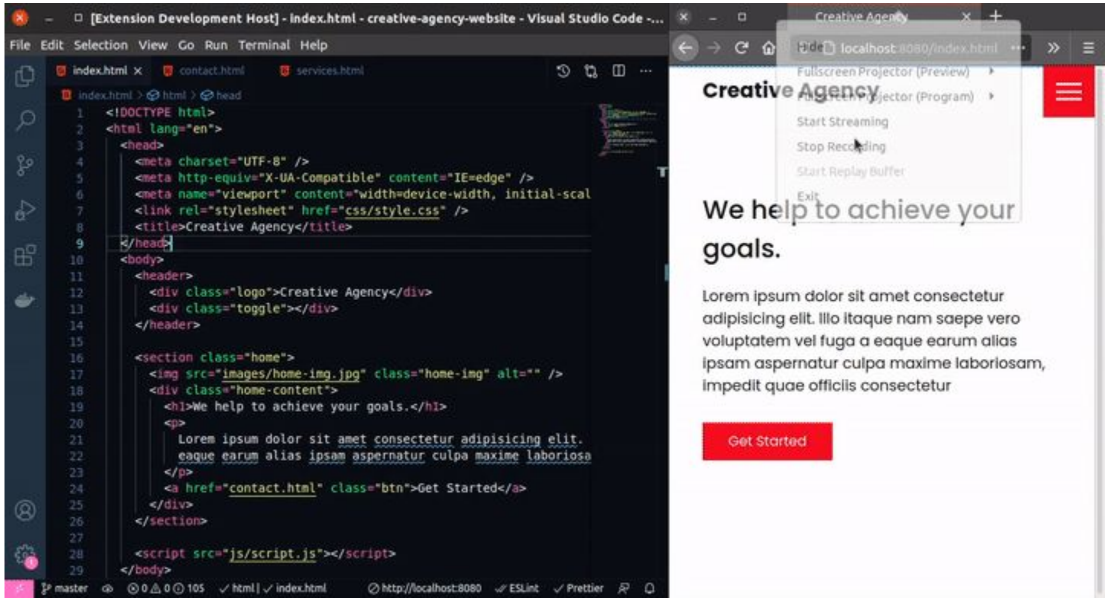
## Live Server

**Automatic Page Refresh**

**Edit →**

**Save →**

**Browser Updates!**


[Extension Development Host] - index.html - creative-agency-website - Visual Studio Code -...
File Edit Selection View Go Run Terminal Help
index.html
contact.html
services.html
index.html > html > head
master
http://localhost:8080
ESLint
Prettier
Creative Agency
localhost:8080/index.html
Fullscreen Projector (Preview)
Fullscreen Projector (Program)
Start Streaming
Stop Recording
Start Replay Buffer
Exit
Get Started

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scal
    <link rel="stylesheet" href="css/style.css" />
    <title>Creative Agency</title>
  </head>
  <body>
    <header>
      <div class="logo">Creative Agency</div>
      <div class="toggle"></div>
    </header>

    <section class="home">
      <img src="images/home-img.jpg" class="home-img" alt="" />
      <div class="home-content">
        <h1>We help to achieve your goals.</h1>
        <p>
          Lorem ipsum dolor sit amet consectetur adipisicing elit.
          eaque earum alias ipsam aspernatur culpa maxime laboriosa
        </p>
        <a href="contact.html" class="btn">Get Started</a>
      </div>
    </section>

    <script src="js/script.js"></script>
  </body>
```

We help to achieve your goals.

Lorem ipsum dolor sit amet consectetur adipisicing elit. Illo itaque nam saepe vero voluptatem vel fuga a eaque earum alias ipsam aspernatur culpa maxime laboriosam, impedit quae officiis consectetur

## 2️⃣ CSS Processing

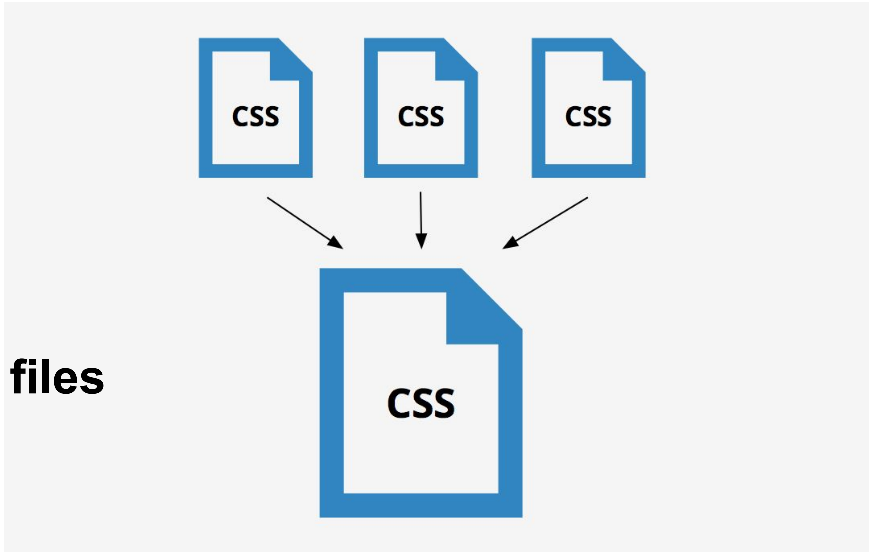
📌 **Tools for working with SCSS & PostCSS:**

- `sass` → Compiles SCSS to CSS
- `postcss` → Processes CSS with plugins
- `autoprefixer` → Adds vendor prefixes automatically
- `csso` → Minifies CSS for better performance
- `sourcemaps` → Generates **source maps** for debugging

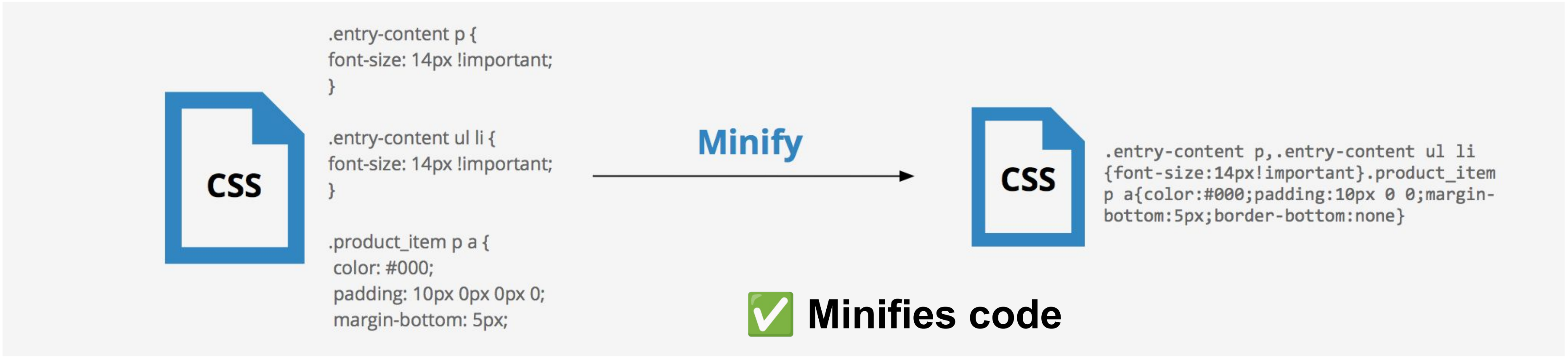
💡 **Why?**

✅ **SCSS support** with clean syntax and mixins
✅ **Autoprefixer ensures cross-browser compatibility**
✅ **Minified CSS loads faster** for better performance
✅ **Source maps make debugging easier** in DevTools

## Work with styles files



✅ **Combines files**



```
.entry-content p {
font-size: 14px !important;
}

.entry-content ul li {
font-size: 14px !important;
}

.product_item p a {
 color: #000;
 padding: 10px 0px 0px 0;
 margin-bottom: 5px;
```

```
.entry-content p,.entry-content ul li
{font-size:14px!important}.product_item
p a{color:#000;padding:10px 0 0;margin-
bottom:5px;border-bottom:none}
```

✅ **Minifies code**

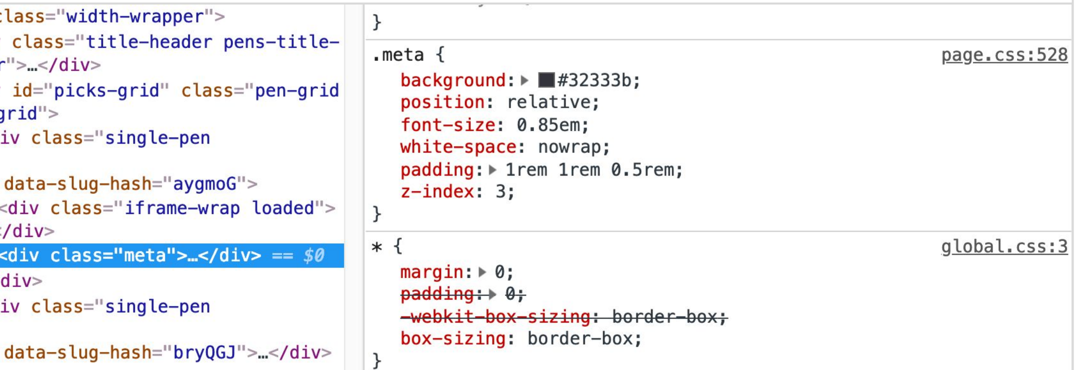
## Source map

A **source map** is a file that maps **compiled (minified, transpiled) code** back to its **original source code**. It allows developers to debug **minified or compiled** JavaScript, CSS, or SCSS **as if they were working with the original files**.

✅ **Think of it as a "translator" between compiled and original code.**

The browser **links the compiled CSS back to the original SCSS file**.

In DevTools, you can **see and edit the original SCSS instead of the compiled CSS**.

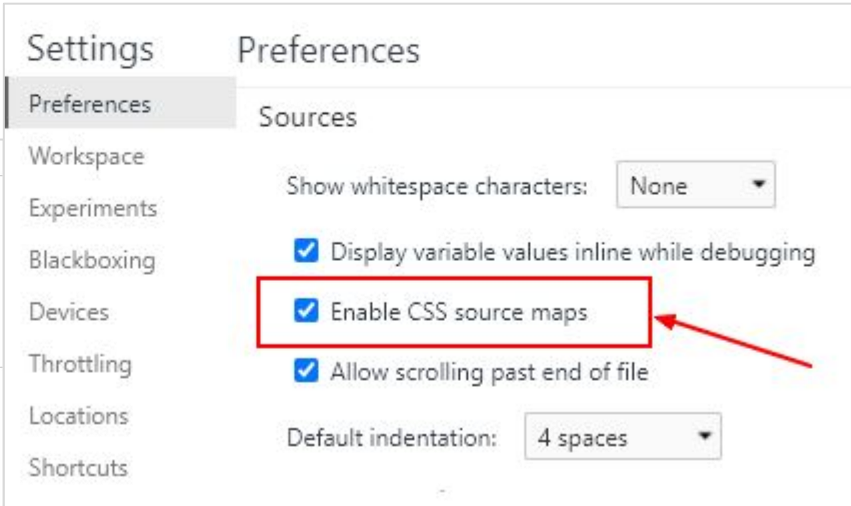


```
lass="width-wrapper">
 class="title-header pens-title-
">…</div>
 id="picks-grid" class="pen-grid
grid">
iv class="single-pen

 data-slug-hash="aygmoG">
<div class="iframe-wrap loaded">
/div>
<div class="meta">…</div> == $0
div>
iv class="single-pen

 data-slug-hash="bryQGJ">…</div>
```

```
}
.meta {                                          page.css:528
   background: ▶ ■#32333b;
   position: relative;
   font-size: 0.85em;
   white-space: nowrap;
   padding: ▶ 1rem 1rem 0.5rem;
   z-index: 3;
}
* {                                              global.css:3
   margin: ▶ 0;
   ~~padding: ▶ 0;~~
   ~~-webkit-box-sizing: border-box;~~
   box-sizing: border-box;
}
```



Settings — Preferences

Sources

Show whitespace characters: None

✓ Display variable values inline while debugging

✓ Enable CSS source maps

✓ Allow scrolling past end of file

Default indentation: 4 spaces

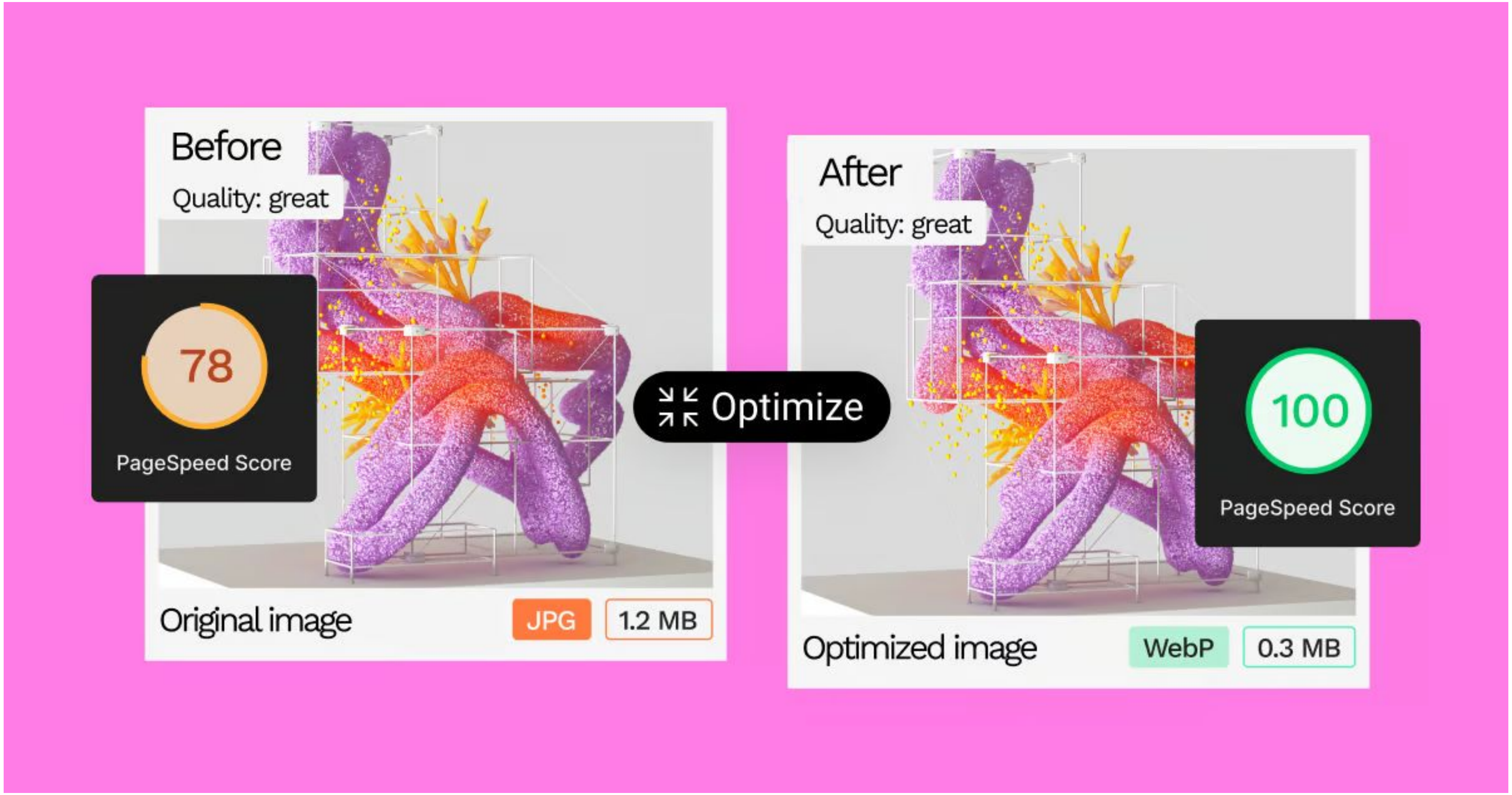
## Automatic Image Processing

One important aspect of **automation** is **image processing**. For example, to support **high-pixel-density screens (Retina)**, images need to be created in **two resolutions**: **standard (1x)** and **high-resolution (2x)**.

If done manually, this process would require:

1. Exporting the image
2. Optimizing it in one tool
3. Resizing it in another
4. Optimizing it again
5. Finally adding it to the project

This process can be **automated**: simply **upload the original image once**, and the **build system** will automatically generate the required versions with **optimized sizes and file weights**.


Before
Quality: great
78
PageSpeed Score
Original image
JPG
1.2 MB
Optimize
After
Quality: great
100
PageSpeed Score
Optimized image
WebP
0.3 MB

## Linting (Code Checking)

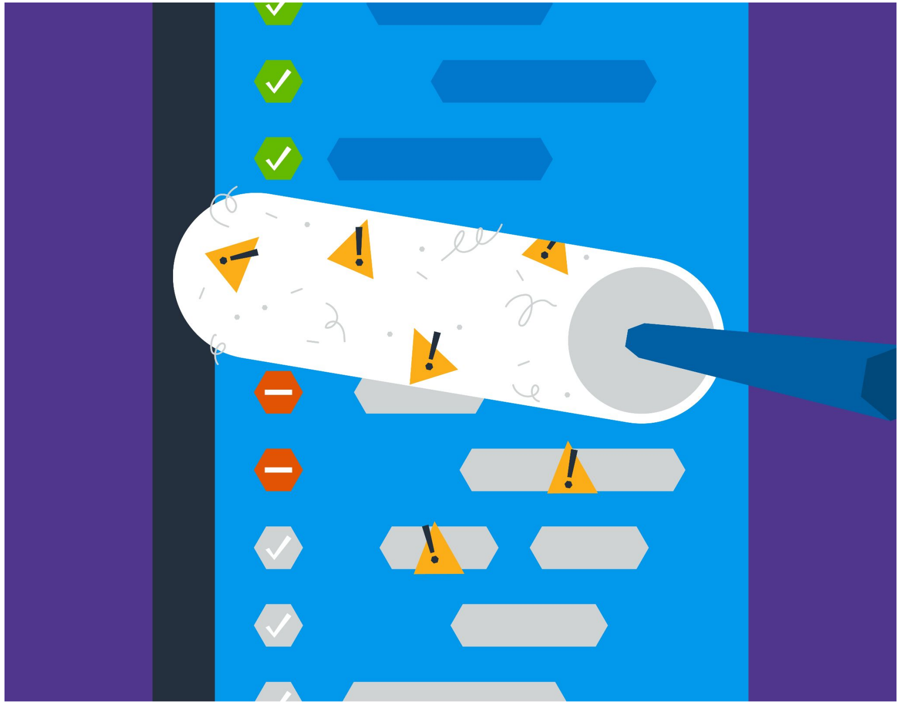
1️⃣**.editorconfig** – Unified indentation and spaces
2️⃣**stylelint** – Ensures consistent CSS coding style
3️⃣**prettier** – automatically formats code to follow a consistent style

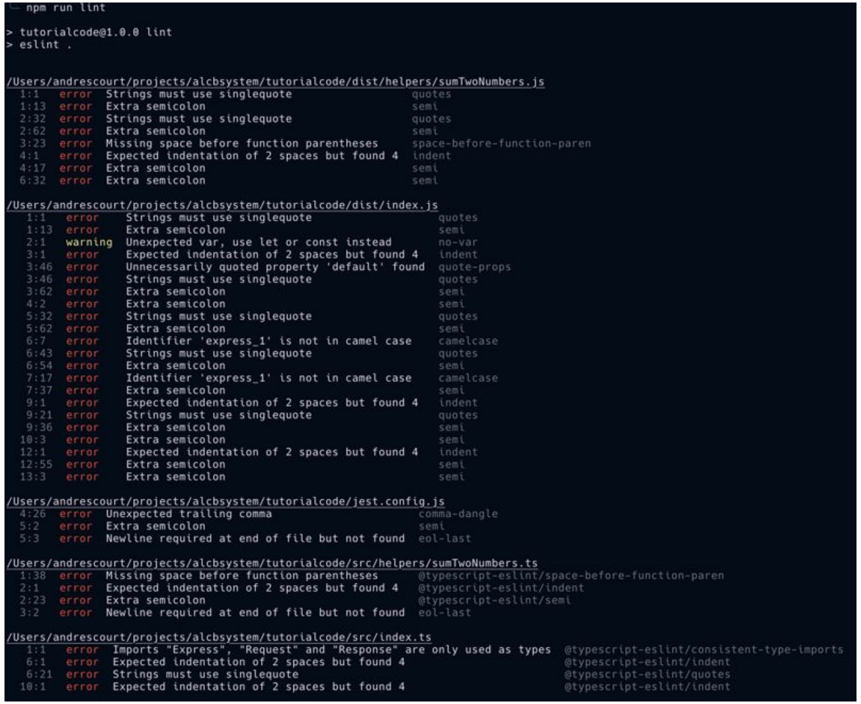
Every developer writes code in their own way, which can create **chaos in a team**. **Automation** helps standardize the process by enforcing **unified code formatting rules**. For example, a **build system** can automatically adjust **line breaks, indentation**, and prevent the use of **prohibited elements**. As a result, the **code remains clean and understandable** for all team members.

Another important aspect that a **build system** manages is **project structure**. Developers often have **personal coding habits**—one person writes code in one way, another in a completely different way, and a mentor may have a different approach as well.

To **avoid debates** over formatting choices—such as whether to place a **line break at the end of a file** or whether to use **English or Cyrillic variable names**—we rely on **build rules** that define coding standards. **There's no arguing with the build system.**

If you **hand over your project** to another **team member**, they will follow the exact same rules as you. **This eliminates disagreements**, ensuring **consistency** across the entire project.





```
└─ npm run lint

> tutorialcode@1.0.0 lint
> eslint .


/Users/andrescourt/projects/alcbsystem/tutorialcode/dist/helpers/sumTwoNumbers.js
  1:1   error  Strings must use singlequote                   quotes
  1:13  error  Extra semicolon                                semi
  2:32  error  Strings must use singlequote                   quotes
  2:62  error  Extra semicolon                                semi
  3:23  error  Missing space before function parentheses      space-before-function-paren
  4:1   error  Expected indentation of 2 spaces but found 4  indent
  4:17  error  Extra semicolon                                semi
  6:32  error  Extra semicolon                                semi

/Users/andrescourt/projects/alcbsystem/tutorialcode/dist/index.js
   1:1   error    Strings must use singlequote                   quotes
   1:13  error    Extra semicolon                                semi
   2:1   warning  Unexpected var, use let or const instead       no-var
   3:1   error    Expected indentation of 2 spaces but found 4  indent
   3:46  error    Unnecessarily quoted property 'default' found  quote-props
   3:46  error    Strings must use singlequote                   quotes
   3:62  error    Extra semicolon                                semi
   4:2   error    Extra semicolon                                semi
   5:32  error    Strings must use singlequote                   quotes
   5:62  error    Extra semicolon                                semi
   6:7   error    Identifier 'express_1' is not in camel case    camelcase
   6:43  error    Strings must use singlequote                   quotes
   6:54  error    Extra semicolon                                semi
   7:17  error    Identifier 'express_1' is not in camel case    camelcase
   7:37  error    Extra semicolon                                semi
   9:1   error    Expected indentation of 2 spaces but found 4  indent
   9:21  error    Strings must use singlequote                   quotes
   9:36  error    Extra semicolon                                semi
  10:3   error    Extra semicolon                                semi
  12:1   error    Expected indentation of 2 spaces but found 4  indent
  12:55  error    Extra semicolon                                semi
  13:3   error    Extra semicolon                                semi

/Users/andrescourt/projects/alcbsystem/tutorialcode/jest.config.js
  4:26  error  Unexpected trailing comma                       comma-dangle
  5:2   error  Extra semicolon                                 semi
  5:3   error  Newline required at end of file but not found  eol-last

/Users/andrescourt/projects/alcbsystem/tutorialcode/src/helpers/sumTwoNumbers.ts
  1:38  error  Missing space before function parentheses      @typescript-eslint/space-before-function-paren
  2:1   error  Expected indentation of 2 spaces but found 4   @typescript-eslint/indent
  2:23  error  Extra semicolon                                @typescript-eslint/semi
  3:2   error  Newline required at end of file but not found  eol-last

/Users/andrescourt/projects/alcbsystem/tutorialcode/src/index.ts
   1:1   error  Imports "Express", "Request" and "Response" are only used as types  @typescript-eslint/consistent-type-imports
   6:1   error  Expected indentation of 2 spaces but found 4                        @typescript-eslint/indent
   6:21  error  Strings must use singlequote                                        @typescript-eslint/quotes
  10:1   error  Expected indentation of 2 spaces but found 4                        @typescript-eslint/indent
```

## .editorconfig

📌 **A configuration file that defines basic code formatting rules**

✅ **Supported by most editors**
✅ **Focuses on core formatting settings:**

- Spaces/tabs
- Indentation
- Encoding
- Line ending style

❌ **Does not check syntax or code structure**, only formatting! 🎯

```
root = true

[*]
indent_style = space
indent_size = 2
end_of_line = lf
insert_final_newline = false
trim_trailing_whitespace = true
charset = utf-8
```

Ensures that all files in the project **use the same type of line breaks**, regardless of the developer's OS.

If different developers use different operating systems, Git might **show unnecessary changes** due to line-ending differences.

Prevents adding a **blank line at the end of files.**

Some editors **automatically add an empty newline at the end of a file;** this setting disables that behavior.

**Automatically removes spaces or tabs at the end of each line when saving the file.**

Trailing whitespace is unnecessary and can cause **Git diffs** to show unwanted changes.

## stylelint

📌 **Ensures consistent CSS coding style** 🎨

✅ **Checks CSS/SCSS** code for errors and enforces coding style rules.

✅ **Prevents common mistakes** (e.g., unknown properties, missing quotes).

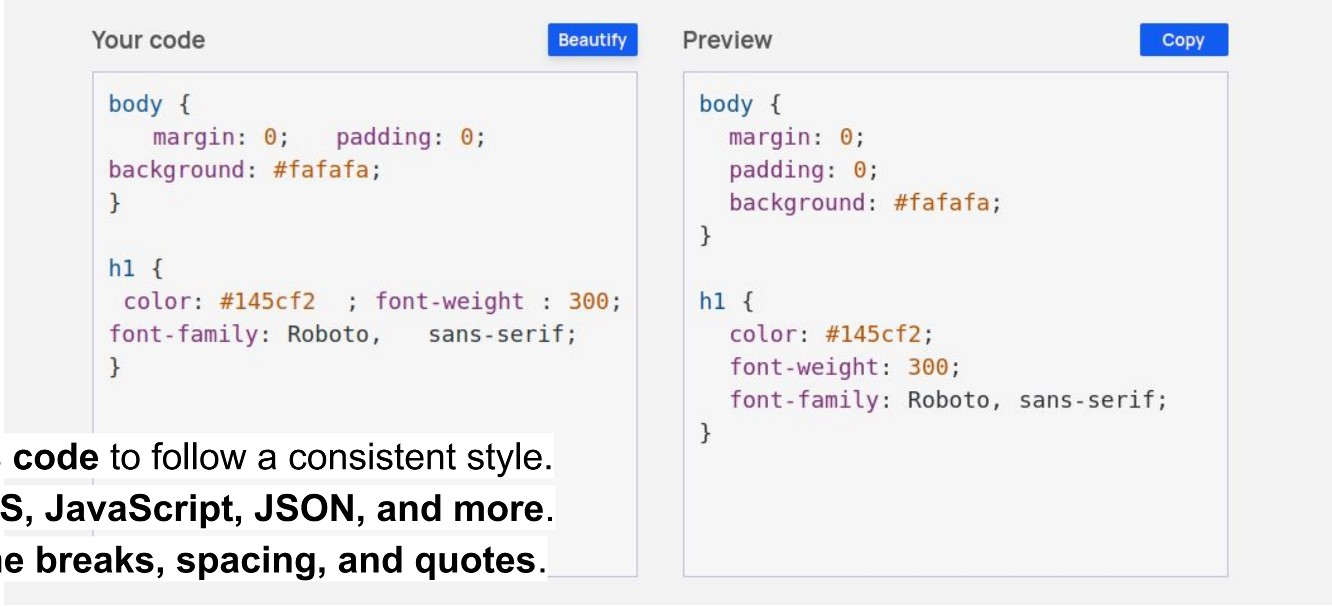
✅ **Ensures the use of BEM** naming convention for class names.

🔹 **Keeps your styles clean and maintainable!** 🚀

Example `stylelint.config.js`:

```
module.exports = {
  extends: "stylelint-config-standard",
  rules: {
    "indentation": 2,
    "color-hex-case": "lower",
    "string-quotes": "double",
    "no-empty-source": true
  }
};
```

## prettier



```
body {
   margin: 0;    padding: 0;
background: #fafafa;
}

h1 {
 color: #145cf2   ; font-weight : 300;
font-family: Roboto,    sans-serif;
}
```

```
body {
  margin: 0;
  padding: 0;
  background: #fafafa;
}

h1 {
  color: #145cf2;
  font-weight: 300;
  font-family: Roboto, sans-serif;
}
```

**What does it do?**

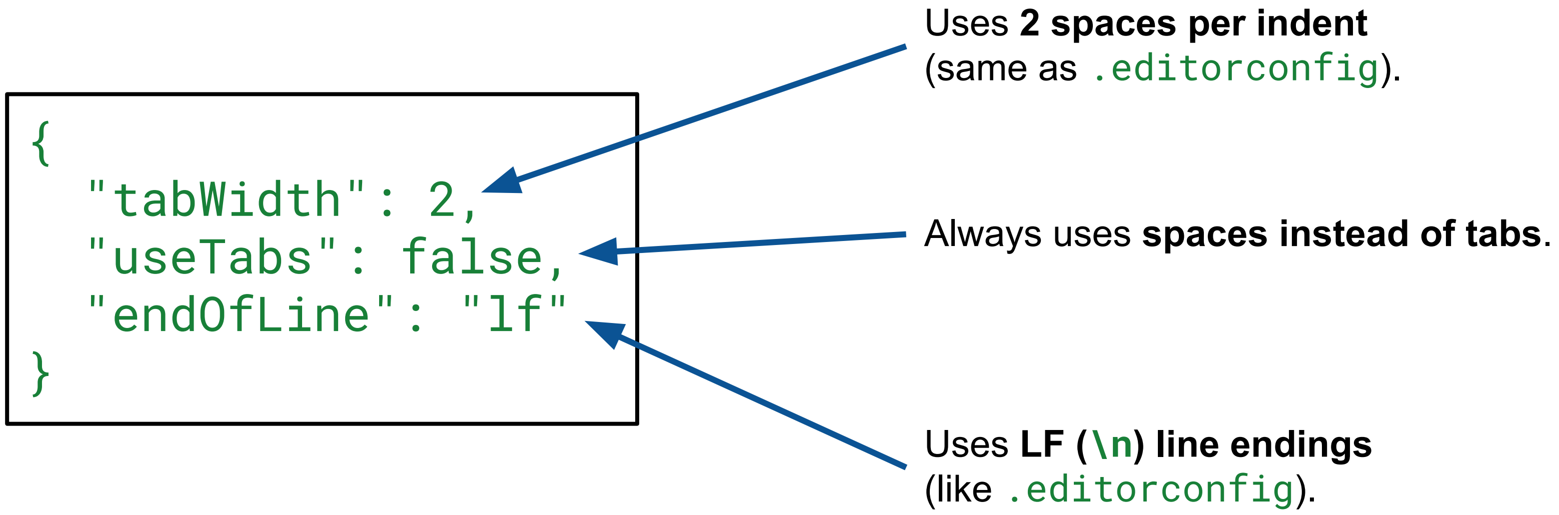
- Automatically **formats code** to follow a consistent style.
- Works with **CSS, SCSS, JavaScript, JSON, and more**.
- **Fixes indentation, line breaks, spacing, and quotes**.

💡 **Why use `prettier`?**

✅ **Formats code automatically** so you don’t have to.

✅ **Works with multiple languages** (JavaScript, CSS, JSON, etc.).

✅ **Prevents unnecessary code diffs** in Git (e.g., inconsistent spacing).



```
{
  "tabWidth": 2,
  "useTabs": false,
  "endOfLine": "lf"
}
```

Uses **2 spaces per indent**
(same as `.editorconfig`).

Always uses **spaces instead of tabs**.

Uses **LF (`\n`) line endings**
(like `.editorconfig`).

## 📌 The Key Differences Between These Tools

| Feature | `.editorconfig` | `stylelint` | `prettier` |
|---|---|---|---|
| **Controls formatting in the editor** | ✅ Yes | ❌ No | ❌ No |
| **Linting & error checking** | ❌ No | ✅ Yes | ❌ No |
| **Automatic code formatting** | ❌ No | ❌ No | ✅ Yes |
| **Works with** | **All file types** | **CSS & SCSS** | **JS, CSS, JSON, etc.** |
| **Example rules** | Indentation, line endings | Class naming, color rules | Line breaks, spacing |

## Node.js – JavaScript Runtime Outside the Browser

And this is a crucial part where **any build process begins**. In web development, **every build process** by default runs using **JavaScript**. And to execute this **JavaScript**, **Node.js** is required.

**Node.js** is a **platform** that allows **JavaScript to run outside the browser**. It is essential for running **automated tools** and managing **build processes**.

### 📌 Why do we need it?

✅ **Allows running tools**, such as bundlers
✅ **Manages dependencies** in a project
✅ **Works with packages**

## NPM

**Node Package Manager** is a **package manager for JavaScript**.

**What Does npm Do for Automation?**

- **Installs** software packages from external libraries directly into our project.
- **Runs tasks** we define for the task manager **Gulp** using console commands.
- **Transfers/publishes** packages from our computer to a shared library if needed.

But how does **npm** know which programs we want to use and how to download them?

We **install all required tools at once**. In this case, **npm** retrieves this information from the **package.json** file.

📌 **Main functions of npm:**

✅ **Installs and manages libraries and tools**

✅ **Works through the command line (terminal)**

✅ **Uses `package.json` to store project dependencies**

💡 **npm comes bundled with Node.js** – if you've installed Node.js, you already have npm!

## package.json

### 📌 The `package.json` file stores project commands and dependencies.

The **package.json** file can be thought of as a **README file on steroids**. It allows you to:

- Define **project dependencies**
- Write **scripts** that run during project **build and testing**
- Store **project version information**
- Provide a **project description**

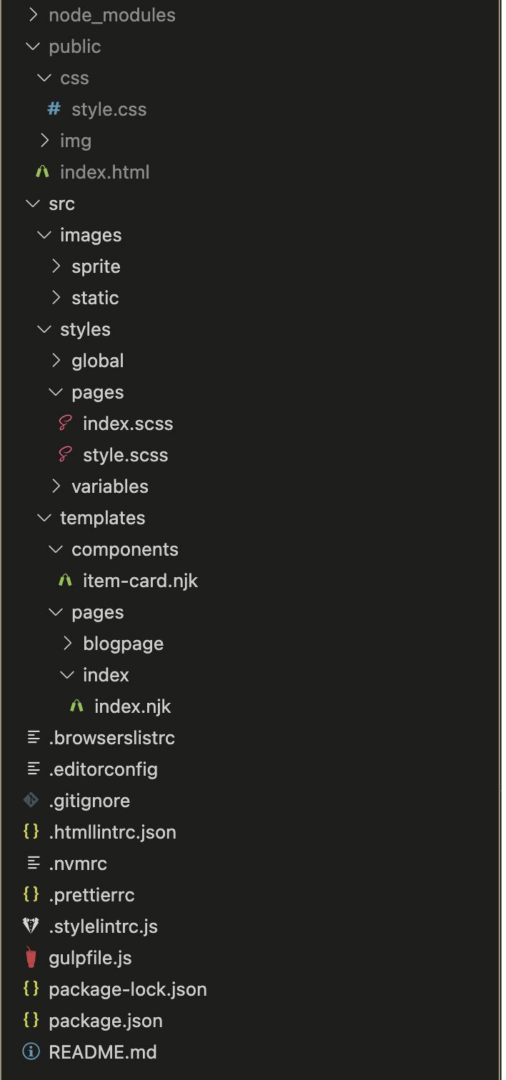
One of the most important features of **package.json** is its ability to **manage dependencies**.

However, the way **dependencies** are handled in this file can seem a bit concerning. Imagine a **package that depends on another package**, which in turn depends on yet another package. This **dependency chain** can grow **infinitely long**, making it difficult to track all the required modules.

✅ **Example of `package.json`:**

```
{
  "name": "binabox-build-gulp",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",

  "scripts": {
    "start": "NODE_ENV=production
gulp build",
    "dev": "gulp serve --lint"
  },
```



```
> node_modules
v public
  v css
    # style.css
  > img
  index.html
v src
  v images
    > sprite
    > static
  v styles
    > global
    v pages
      index.scss
      style.scss
    > variables
  v templates
    v components
      item-card.njk
    v pages
      > blogpage
      v index
        index.njk
.browserslistrc
.editorconfig
.gitignore
.htmllintrc.json
.nvmrc
.prettierrc
.stylelintrc.js
gulpfile.js
package-lock.json
package.json
README.md
```

- **`public/`** — final build of the project
  - `css/style.css` — styles after SCSS compilation
  - `img/` — project images
  - `index.html` — compiled HTML file
- **`src/`** — source files of the project
  - `images/` — image storage
    - `sprite/` — possibly SVG sprites
    - `static/` — regular images
  - `styles/` — source SCSS files
    - `global/` — global styles
    - `pages/` — styles for individual pages
    - `variables/` — SCSS variables
  - `templates/` — Nunjucks template files
    - `components/` — reusable components, e.g., `item-card.njk`
    - `pages/` — pages (e.g., `index.njk`, `blogpage/`)

## Configuration Files

- **`.editorconfig`**, **`.prettierrc`**, **`.stylelintrc.js`**, **`.htmllintrc.json`** — configuration for formatting and linting code.
- **`.gitignore`** — files excluded from the repository.
- **`gulpfile.js`** — Gulp configuration file, responsible for automating the build process.
- **`package.json`** — list of dependencies and scripts for working with the project.

## lets do it

## Open Terminal and install Node.js

## Unix Installation – Linux & MacOS

```
curl https://get.volta.sh | bash
```

## Windows Installation

```
winget install Volta.Volta
```

## Restart Terminal

## Select a default Node version

```
volta install node
```

## download files from repository

https://github.com/bro-academy/binabox-build-gulp

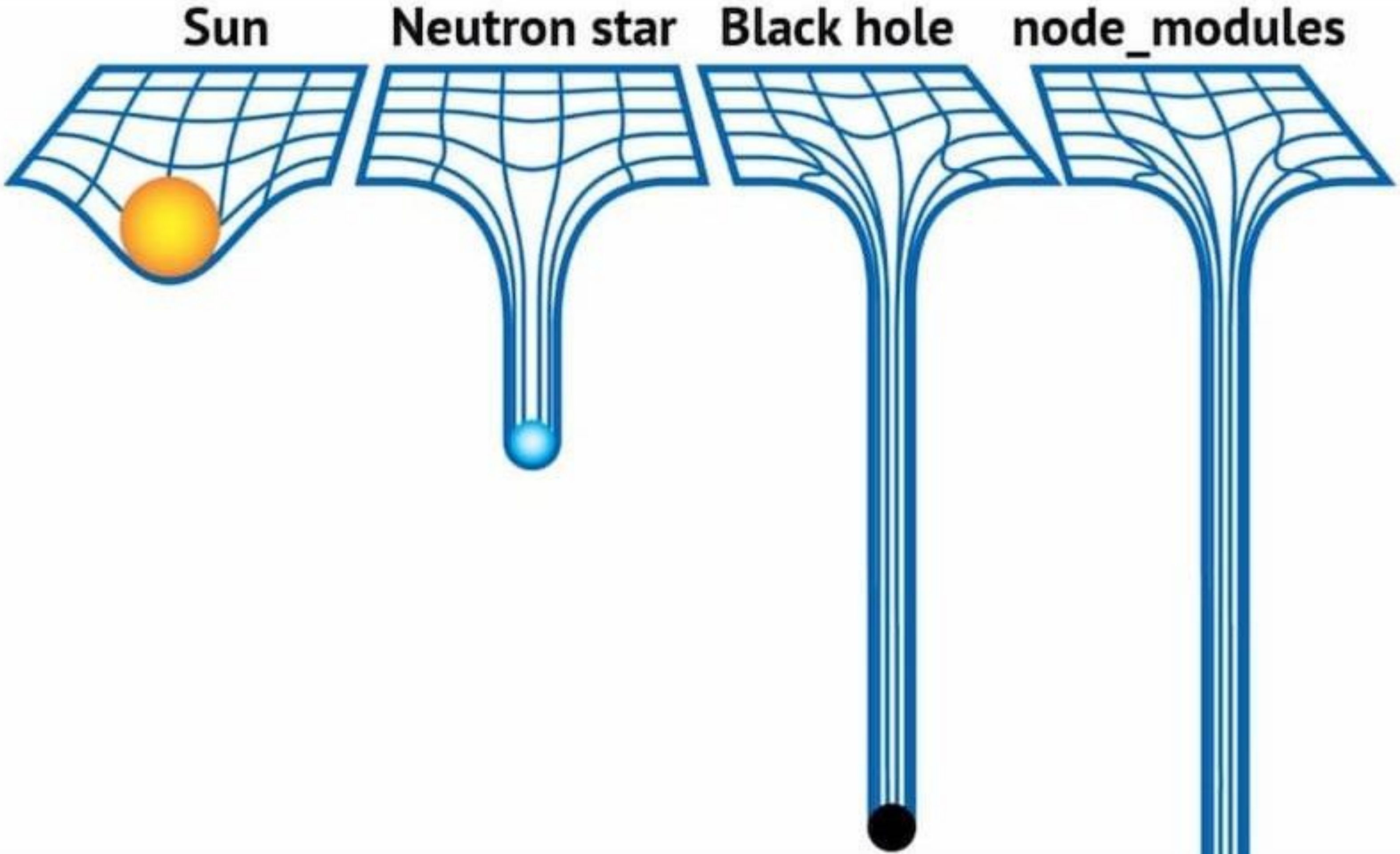
```
ctrl + c -> ctrl + v
```

## Open your project and start

`npm ci` – install dependencies

↓

`node_modules` – creates


Sun
Neutron star
Black hole
node_modules

## .gitignore

```
.DS_Store
Thumbs.db
*.log
node_modules/   – too heavy
public/   – version for prod
```

## Build commands for start

`npm start` – will generate site (static build)

`npm run dev` – will generate site, show **lint errors**, watch files and **re-run build on change**(**live server**)

`ctrl + c` – stop watch