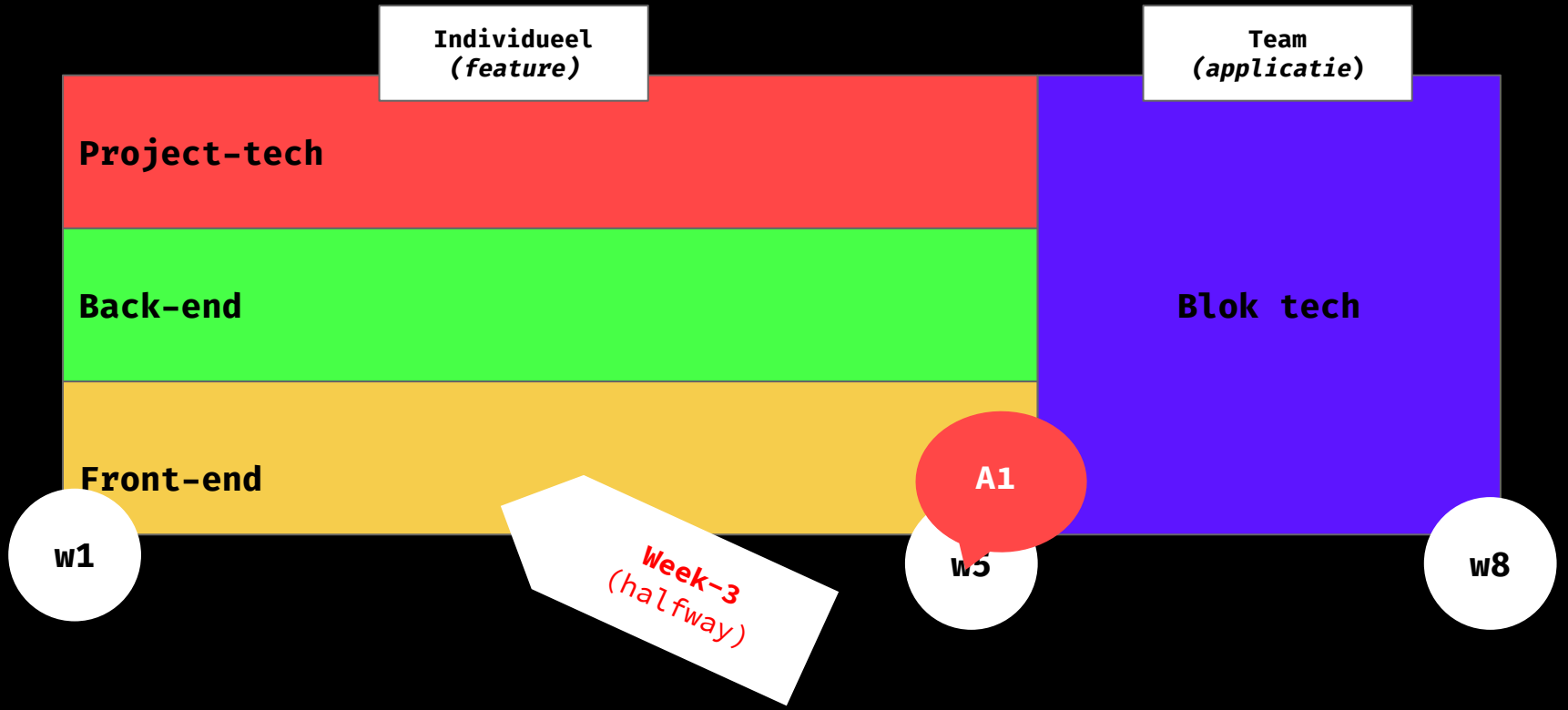
project-tech

Env && Linting && Build

lab 3/8

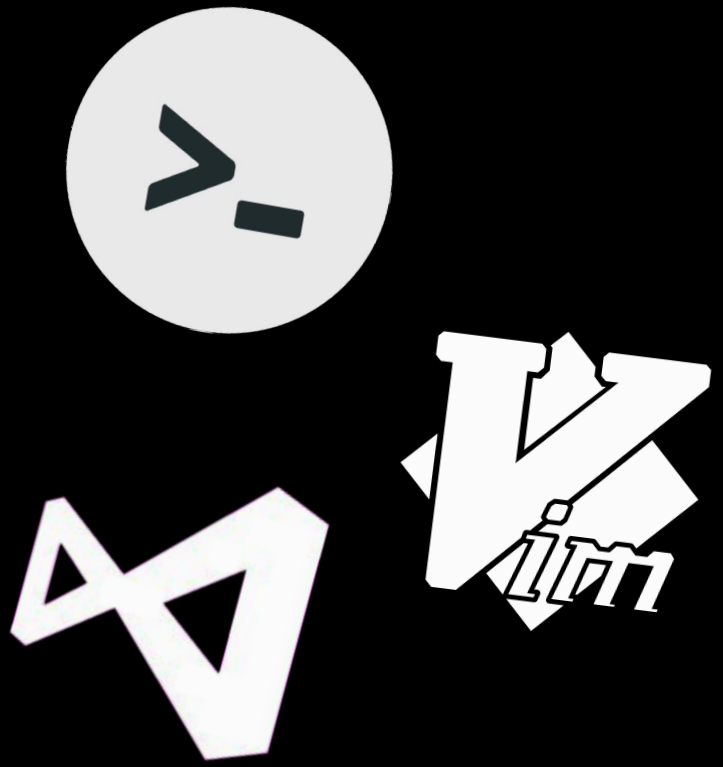
Stand-up!

Show what you did

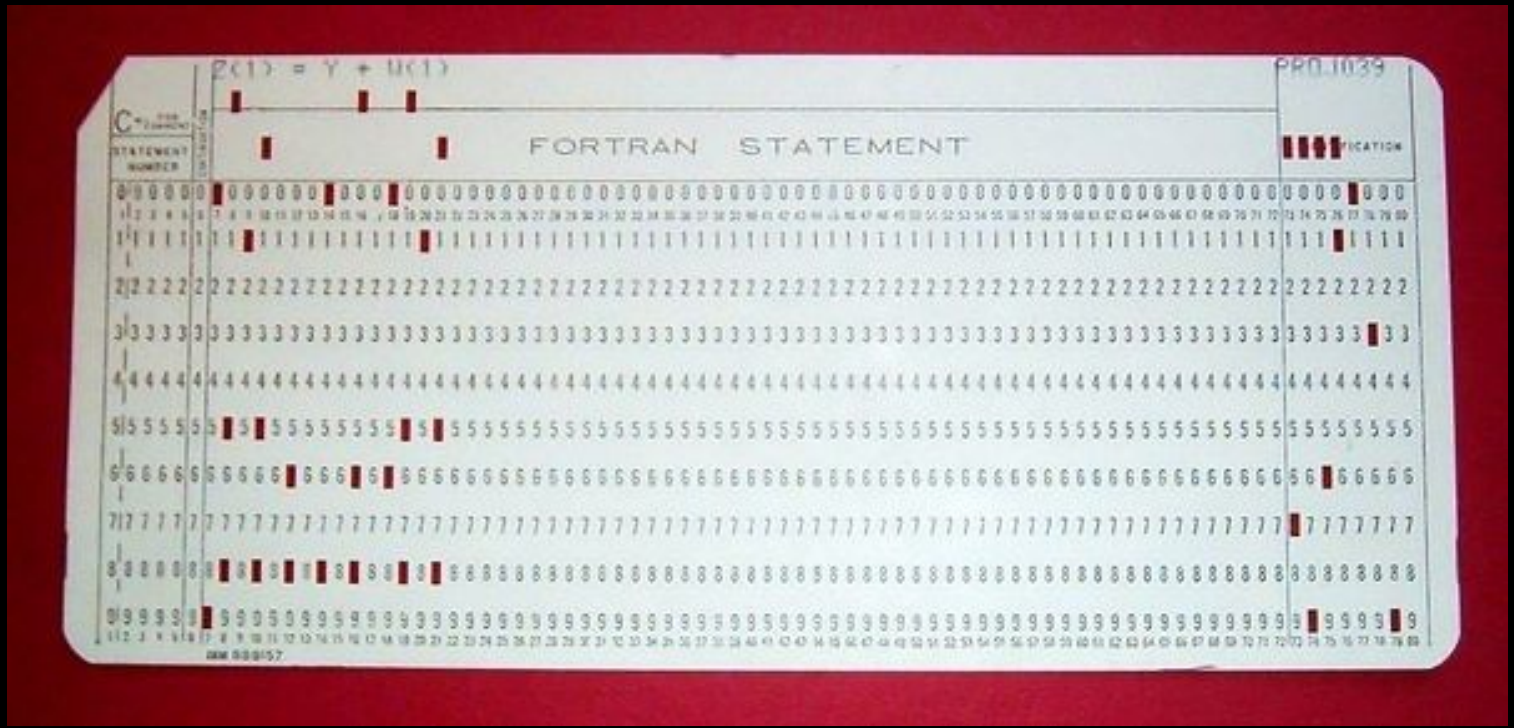

Individueel
(feature)
Team
(applicatie)
Project-tech
Back-end
Front-end
Blok tech
A1
w1
Week-3
(halfway)
w5
w8

today

I. ~~Standup~~
II. Local Dev environment
III. Linters & Formatters
IV. Build Tools

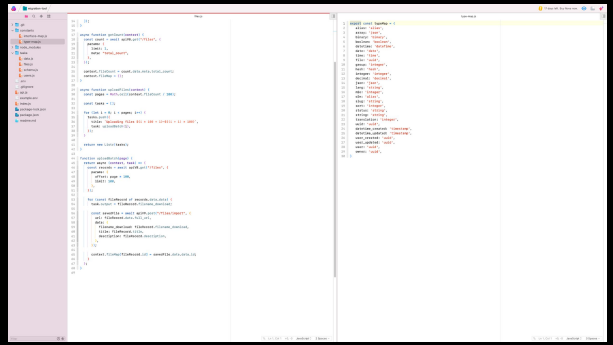


Dev Environment

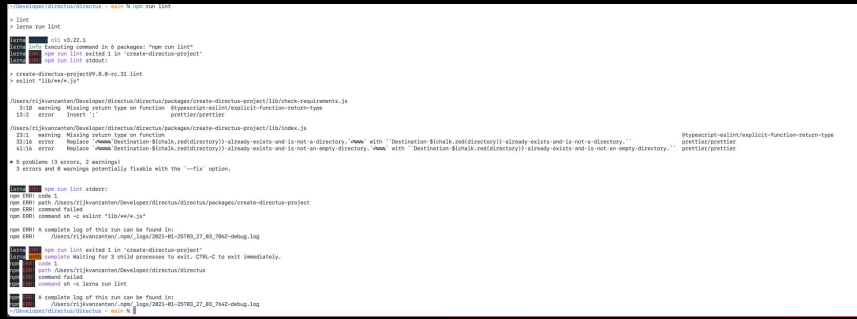

Z(1) = Y + W(1)
PRO 1039
C
STATEMENT NUMBER
FORTRAN STATEMENT
ICATION


IBM

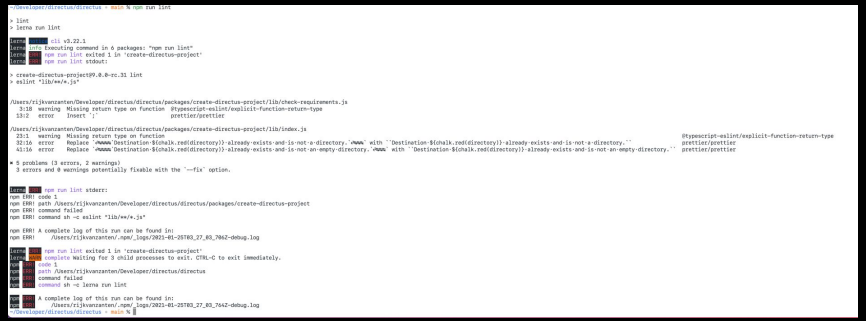

MacBook Pro



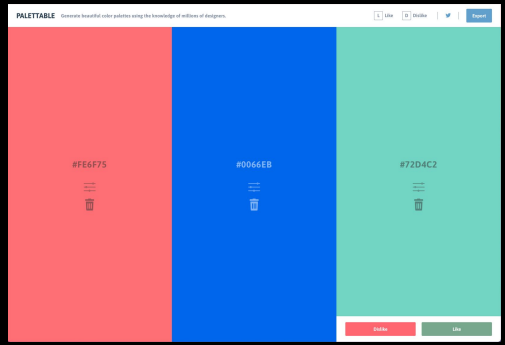
1. Code Editors



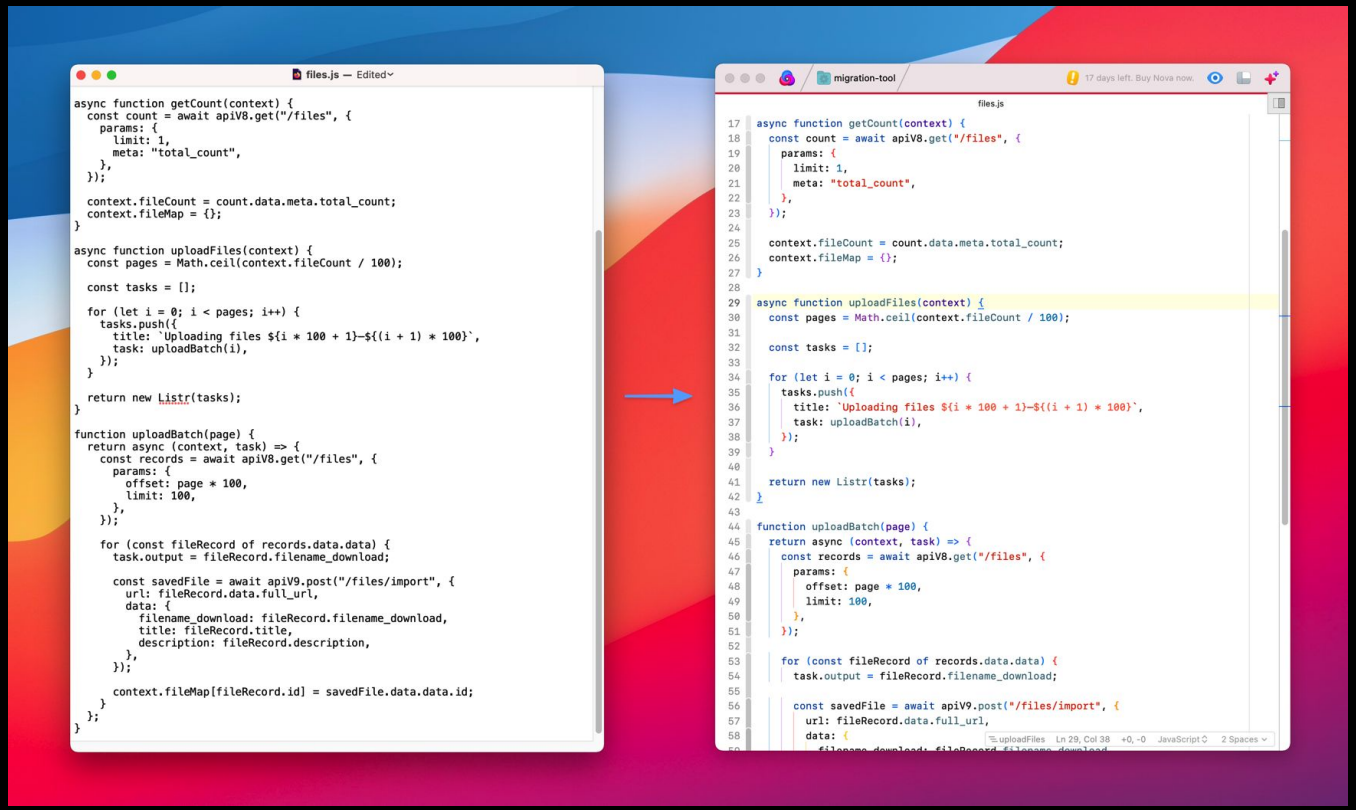
3. Build Tools



2. Linters/Formatters



4. Non-code tools


files.js — Edited
async function getCount(context) {
const count = await apiV8.get("/files", {
params: {
limit: 1,
meta: "total_count",
},
});
context.fileCount = count.data.meta.total_count;
context.fileMap = {};
}
async function uploadFiles(context) {
const pages = Math.ceil(context.fileCount / 100);
const tasks = [];
for (let i = 0; i < pages; i++) {
tasks.push({
title: `Uploading files ${i * 100 + 1}–${(i + 1) * 100}`,
task: uploadBatch(i),
});
}
return new Listr(tasks);
}
function uploadBatch(page) {
return async (context, task) => {
const records = await apiV8.get("/files", {
params: {
offset: page * 100,
limit: 100,
},
});
for (const fileRecord of records.data.data) {
task.output = fileRecord.filename_download;
const savedFile = await apiV9.post("/files/import", {
url: fileRecord.data.full_url,
data: {
filename_download: fileRecord.filename_download,
title: fileRecord.title,
description: fileRecord.description,
},
});
context.fileMap[fileRecord.id] = savedFile.data.data.id;
}
};
}
migration-tool
17 days left. Buy Nova now.
files.js
async function getCount(context) {
const count = await apiV8.get("/files", {
params: {
limit: 1,
meta: "total_count",
},
});
context.fileCount = count.data.meta.total_count;
context.fileMap = {};
}
async function uploadFiles(context) {
const pages = Math.ceil(context.fileCount / 100);
const tasks = [];
for (let i = 0; i < pages; i++) {
tasks.push({
title: `Uploading files ${i * 100 + 1}–${(i + 1) * 100}`,
task: uploadBatch(i),
});
}
return new Listr(tasks);
}
function uploadBatch(page) {
return async (context, task) => {
const records = await apiV8.get("/files", {
params: {
offset: page * 100,
limit: 100,
},
});
for (const fileRecord of records.data.data) {
task.output = fileRecord.filename_download;
const savedFile = await apiV9.post("/files/import", {
url: fileRecord.data.full_url,
data: {
uploadFiles Ln 29, Col 38 +0, -0 JavaScript 2 Spaces

code

- Code is “just” text
- Computers don’t care what your code
- Make writing (reading) source code easier
- Don’t work hard, work smart #tailopez
- Personal preference!


Visual Studio Code


Brackets

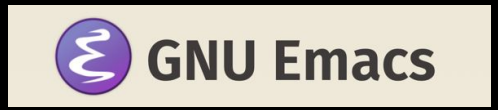

WS
WebStorm


Sublime Text

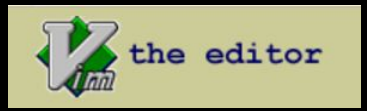

nova

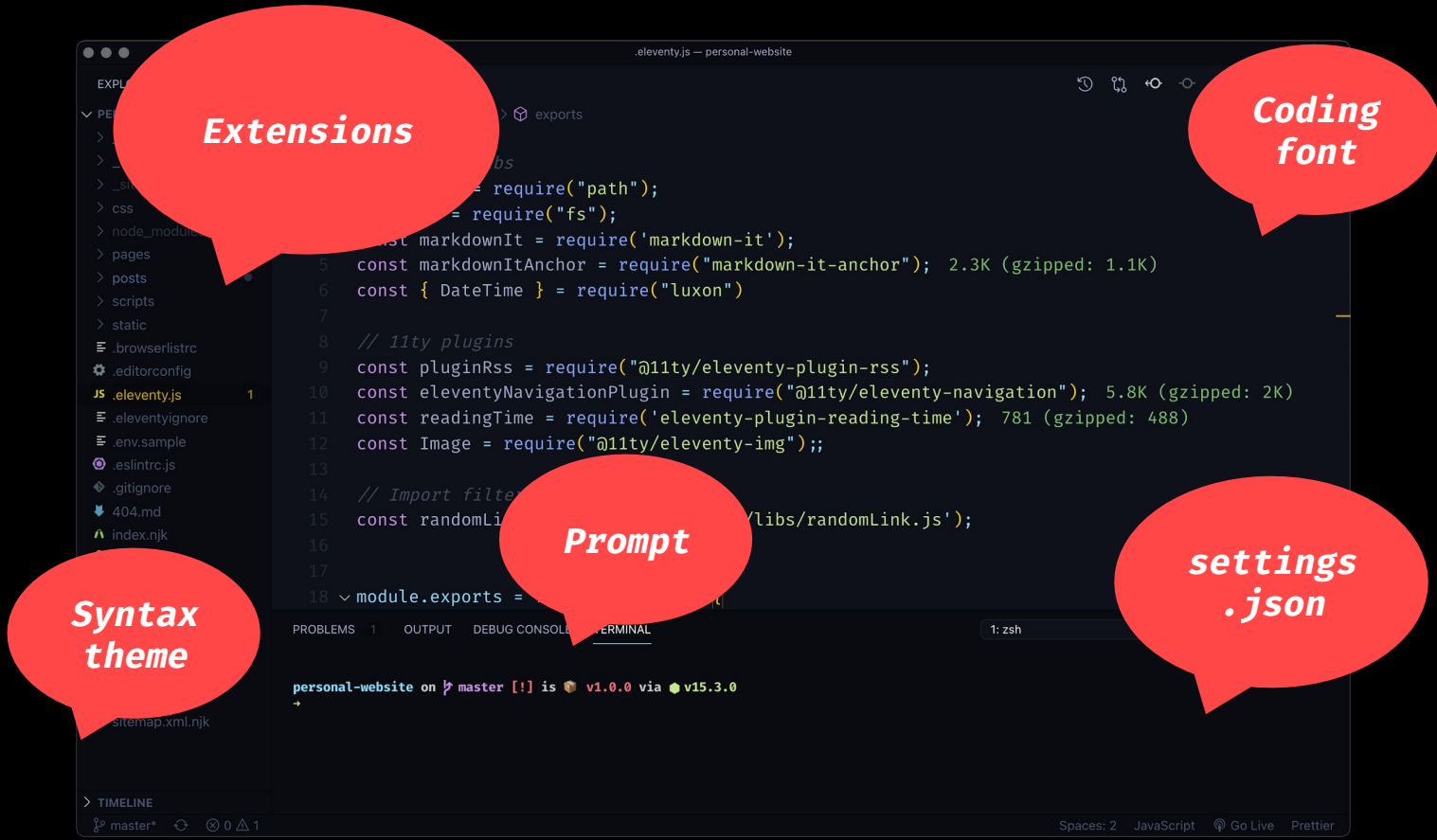

ATOM

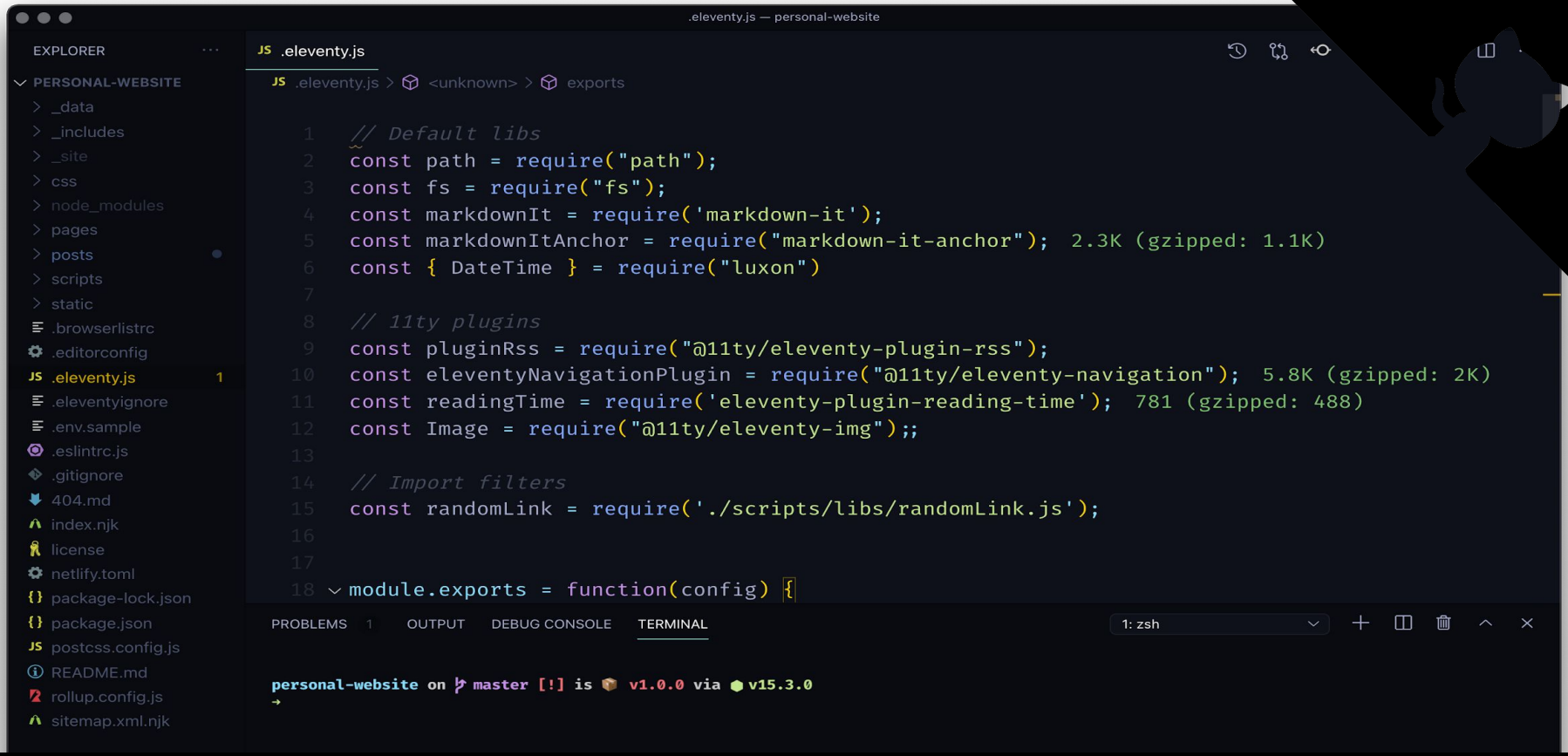

Visual Studio


GNU Emacs


Xcode 12


the editor


Extensions
Coding font
Syntax theme
Prompt
settings .json
.eleventy.js — personal-website
exports
const markdownIt = require('markdown-it');
const markdownItAnchor = require("markdown-it-anchor");
const { DateTime } = require("luxon")
// 11ty plugins
const pluginRss = require("@11ty/eleventy-plugin-rss");
const eleventyNavigationPlugin = require("@11ty/eleventy-navigation");
const readingTime = require('eleventy-plugin-reading-time');
const Image = require("@11ty/eleventy-img");;
const randomLi
/libs/randomLink.js');
module.exports =
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: zsh
personal-website on master [!] is v1.0.0 via v15.3.0
TIMELINE
master*
Spaces: 2
JavaScript
Go Live
Prettier



live demo **dev environment**

Linters & Formatters

linters

?

A **linter** catches inconsistencies throughout your project and reduces your *chances of making logical errors.*

Syntax: Linters check the source code for programmatic mistakes (bugs, errors)

formatter ?

A **formatter** can help maintain a *consistent coding style*. It’s basically a ‘style guide’ for how you write code.

Stylistic: beautifies (and formats) your code to make it more readable.

Linters & formatters

why

Ensuring a good coding style has a couple upsides:

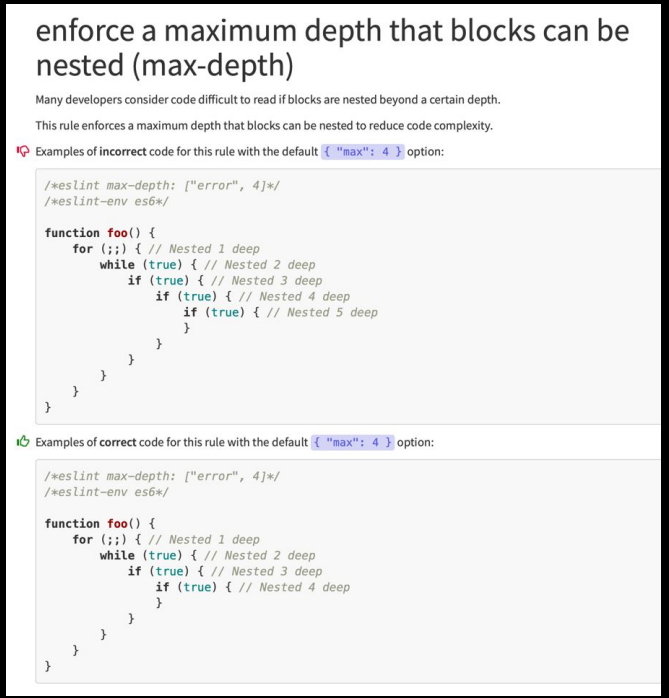
- It helps you avoid bugs
- Help other developers read through your code
- Save time and avoid stress
- Well organized, cleanly written code can be its own documentation!
- Improves team collaboration

linter example

ESlint

- Checks for a wide range of syntax- and stylistic errors
- Can help you write better code by enforcing best practices
- Makes sure everybody codes in the same “dialect”

https://eslint.org



enforce a maximum depth that blocks can be nested (max-depth)

Many developers consider code difficult to read if blocks are nested beyond a certain depth.

This rule enforces a maximum depth that blocks can be nested to reduce code complexity.

Examples of **incorrect** code for this rule with the default `{ "max": 4 }` option:

```
/*eslint max-depth: ["error", 4]*/
/*eslint-env es6*/

function foo() {
    for (;;) { // Nested 1 deep
        while (true) { // Nested 2 deep
            if (true) { // Nested 3 deep
                if (true) { // Nested 4 deep
                    if (true) { // Nested 5 deep
                    }
                }
            }
        }
    }
}
```

Examples of **correct** code for this rule with the default `{ "max": 4 }` option:

```
/*eslint max-depth: ["error", 4]*/
/*eslint-env es6*/

function foo() {
    for (;;) { // Nested 1 deep
        while (true) { // Nested 2 deep
            if (true) { // Nested 3 deep
                if (true) { // Nested 4 deep
                }
            }
        }
    }
}
```

formatter example

- Makes sure everybody's **editor settings are the same**
- Ensures you never end up **with mixed tabs/spaces**

https://editorconfig.org

editorconfig

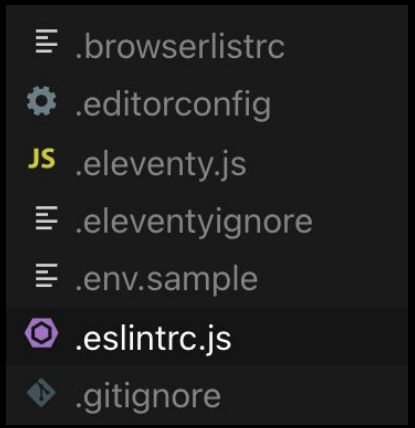
```
# EditorConfig is awesome:
https://EditorConfig.org

# top-most EditorConfig file
root = true

# Unix-style newlines with a newline ending
every file
[*]
end_of_line = lf
insert_final_newline = true

# Matches multiple files with brace expansion
notation
# Set default charset
[*.{js,py}]
charset = utf-8

# 4 space indentation
[*.py]
indent_style = space
indent_size = 4

# Tab indentation (no size specified)
[Makefile]
indent_style = tab

# Indentation override for all JS under lib
directory
[lib/**.js]
indent_style = space
indent_size = 2
```

linters

configuration

Linters & formatters **need configuration files to tell them what rules to enforce.**
(.rc = runtime configuration)

- .editorconfig
- .eslintrc
- .stylelint



linters

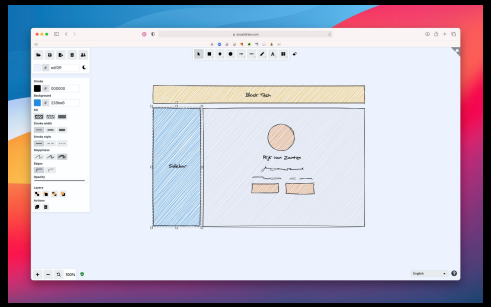
how to use

- In **an extension** in your text editor
- **Using commands** in the CLI
- **Run scripts** in package.json

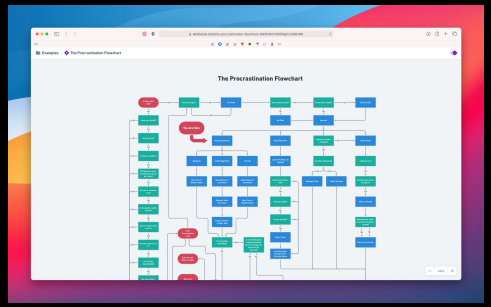
live demo **linters & formatters**

non-code tools

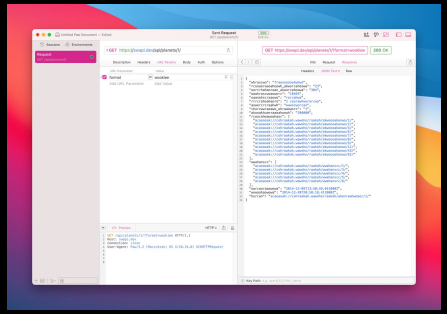
applications



Excalidraw



Whimsical



Postman

And many many more ...

non-code tools

healthy

- #stayhydrated
- Get enough sleep
- Reduce your brightness
- Make your workspace comfortable
- Take enough breaks

Build Tools

build tools ?

Build tools are additional tooling you can use in your project and are mostly used for ***automating* tasks or *transforming* code.**

preprocessors, bundlers, postprocessor, compiling #buzzword #buzzword, ai, machine learning

build tools ?

Build tools are additional tooling you can use in your project and are mostly used for ***automating* tasks or *transforming* code.**

preprocessors, bundlers, postprocessor,

Note: If you don't know what you are doing and build tools feel too much. **Then don't ;-)**
But *zwarte piste* go ahead.

example

sass

.scss *gets processed (compiled) to* ***.css***

SCSS

```scss
section {
    height: 100px;
    width: 100px;

    .class-one {
        height: 50px;
        width: 50px;

        .button {
            color: #074e68;
        }
    }
}
```

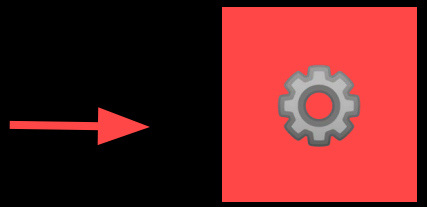
CSS

```css
section {
    height: 100px;
    width: 100px;
}

section .class-one {
    height: 50px;
    width: 50px;
}

section .class-one .button {
    color: #074e68;
}
```

example

build

```
// Input

css/
├─ typography.scss
├─ layout.scss
└─ colors.scss
```



```
// Build
```


```
// Output

css/
└─ style.css
```

Building

building

> Start building out the interface (html & css) of your matching-app feature.

Synopsis

- **Time**: 6:00h
- **Due**: before week 3

Assignment

Based on the concept, job story, requirement list and wireframe from the previous week start building out the front-end of the feature you are going to make for the matching-application. You can build the interface with the **templating engine** as you learned in back-end as opposed to 'plain' HTML & CSS.

- Turn your wireframe **into HTML pages**. Do a HTML breakdown of your wireframe and use semantic HTML as learned in previous courses.

work on **local, linting, build**

exit;

see you in lab-4!