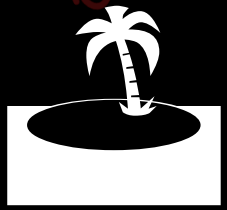
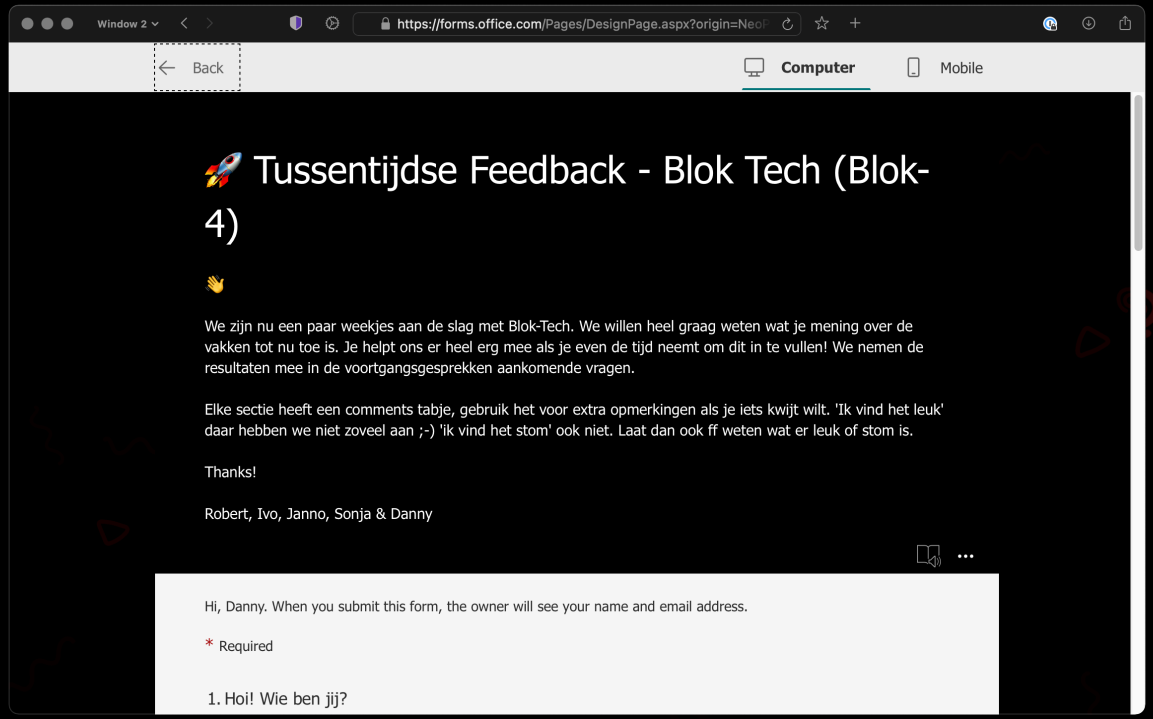
project-tech

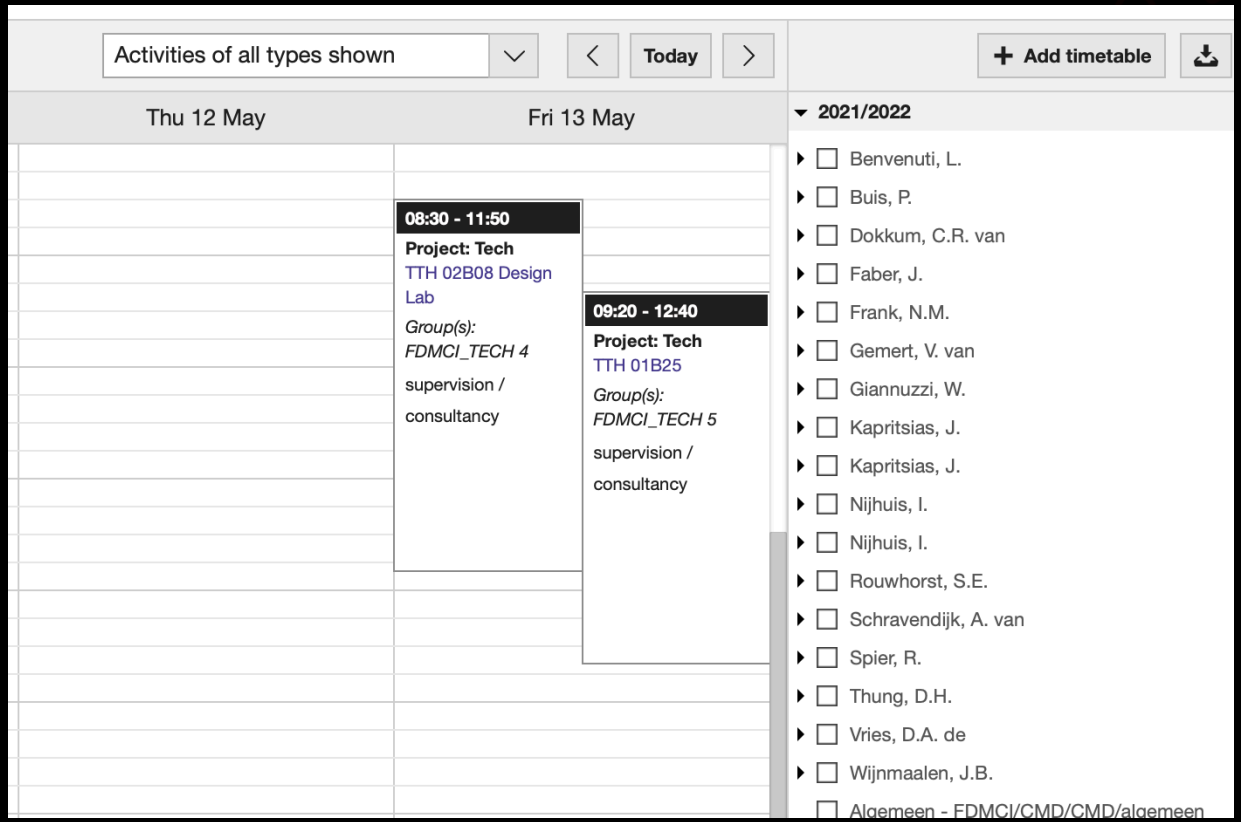
Env && Linting && Build

lab 3/8

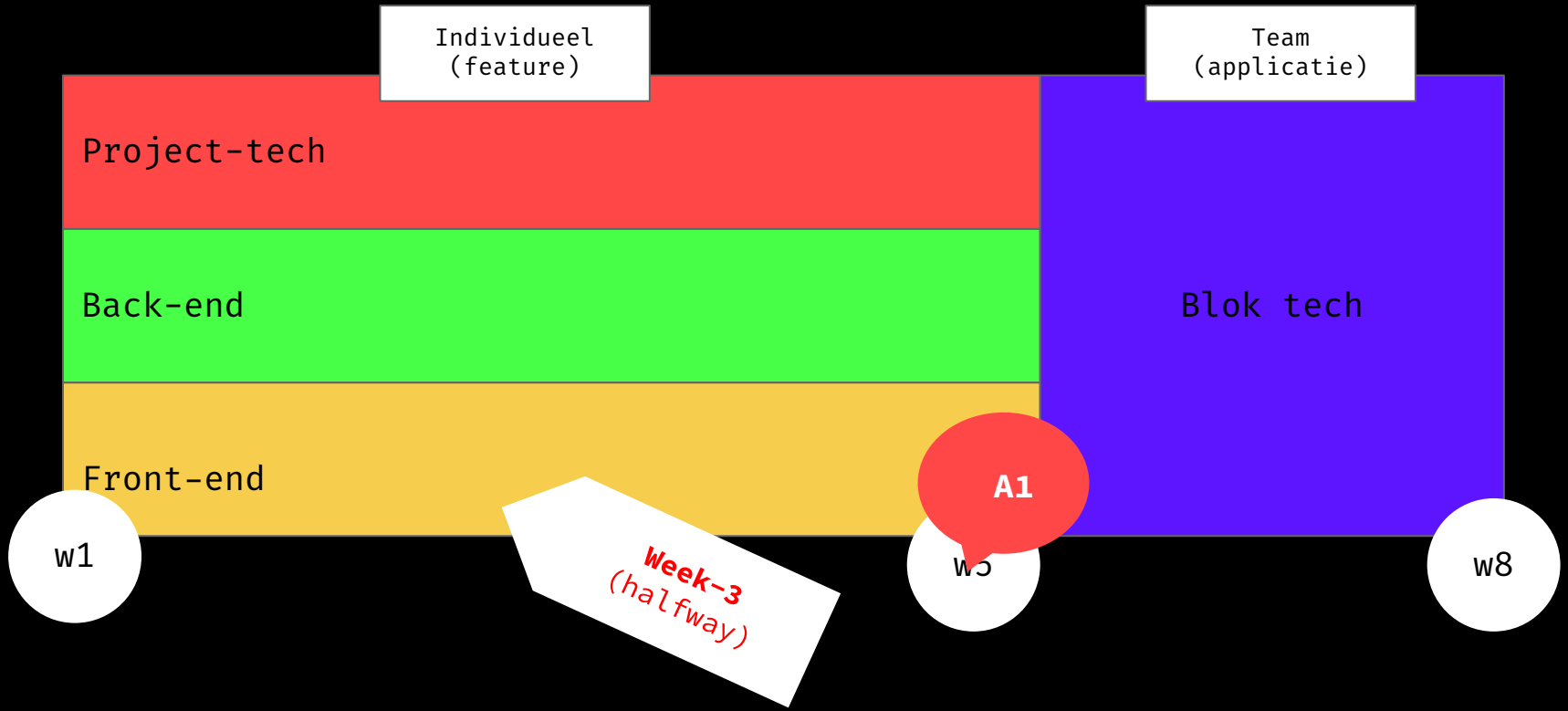




Geef je feedback!



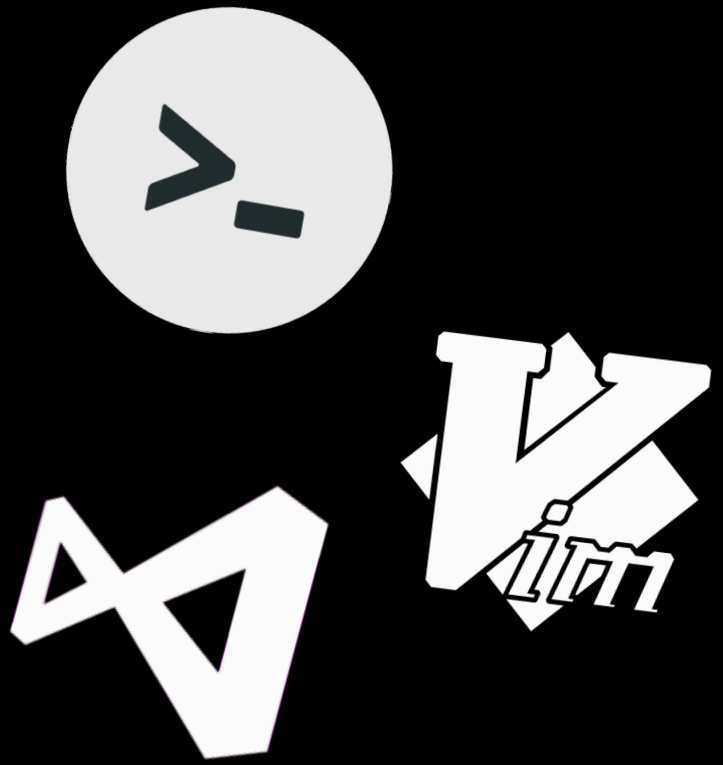
Voortgangsgesprek


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

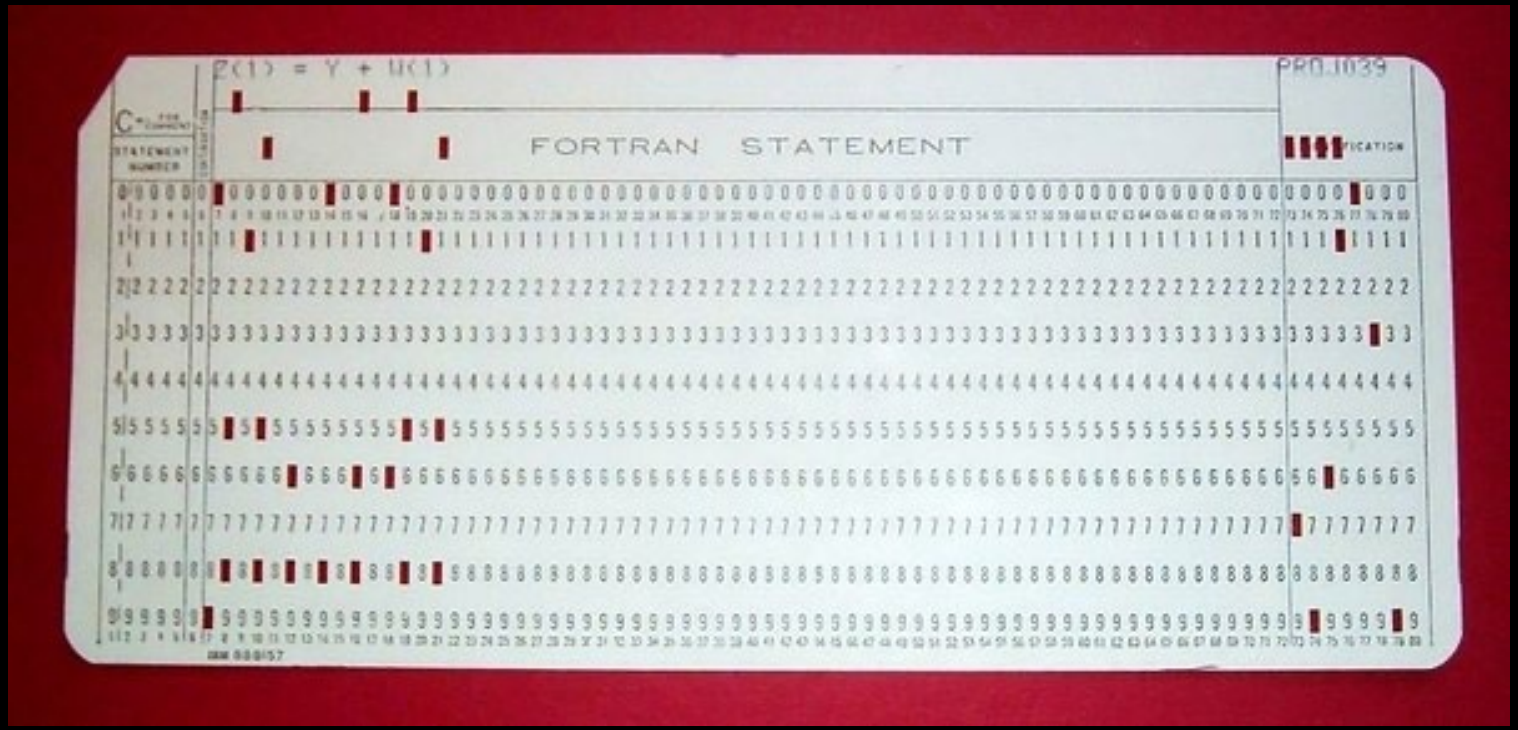
Stand-up!

today

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

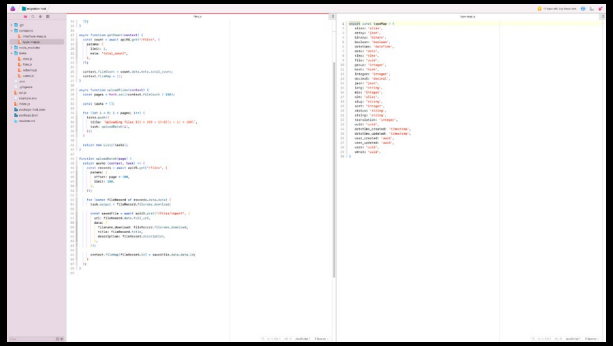


Dev Tooling

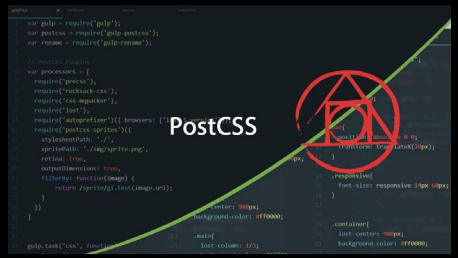


Steve Baker
https://qph.fs.quoracdn.net/main-qimg-4da0567cc18f277fd5bf46fd9667c963-c

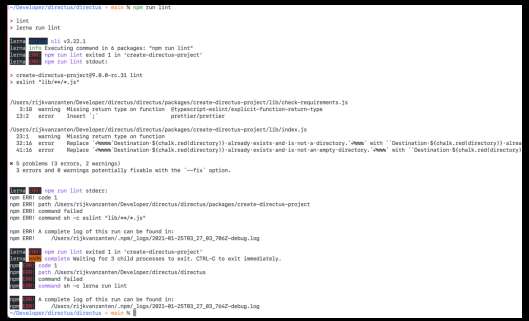

MacBook Pro



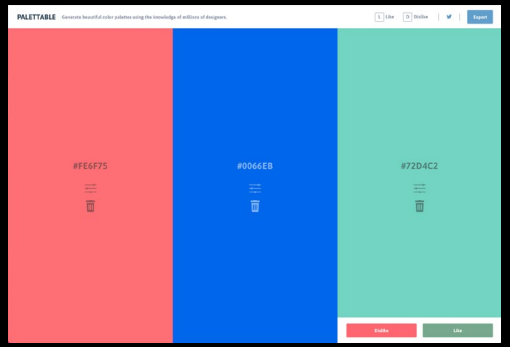
Code Editors (ide’s)



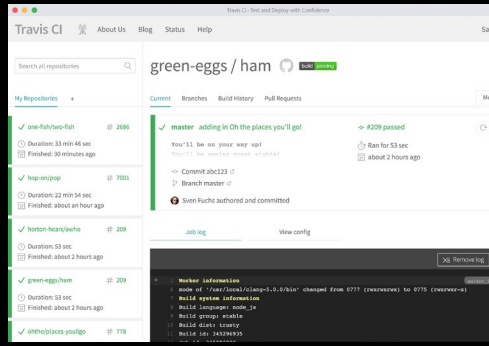
Build tools



Linters/Formatters



“Other” non-code tools



Testing (CI Intergration)

Code-Editors

Code Editors

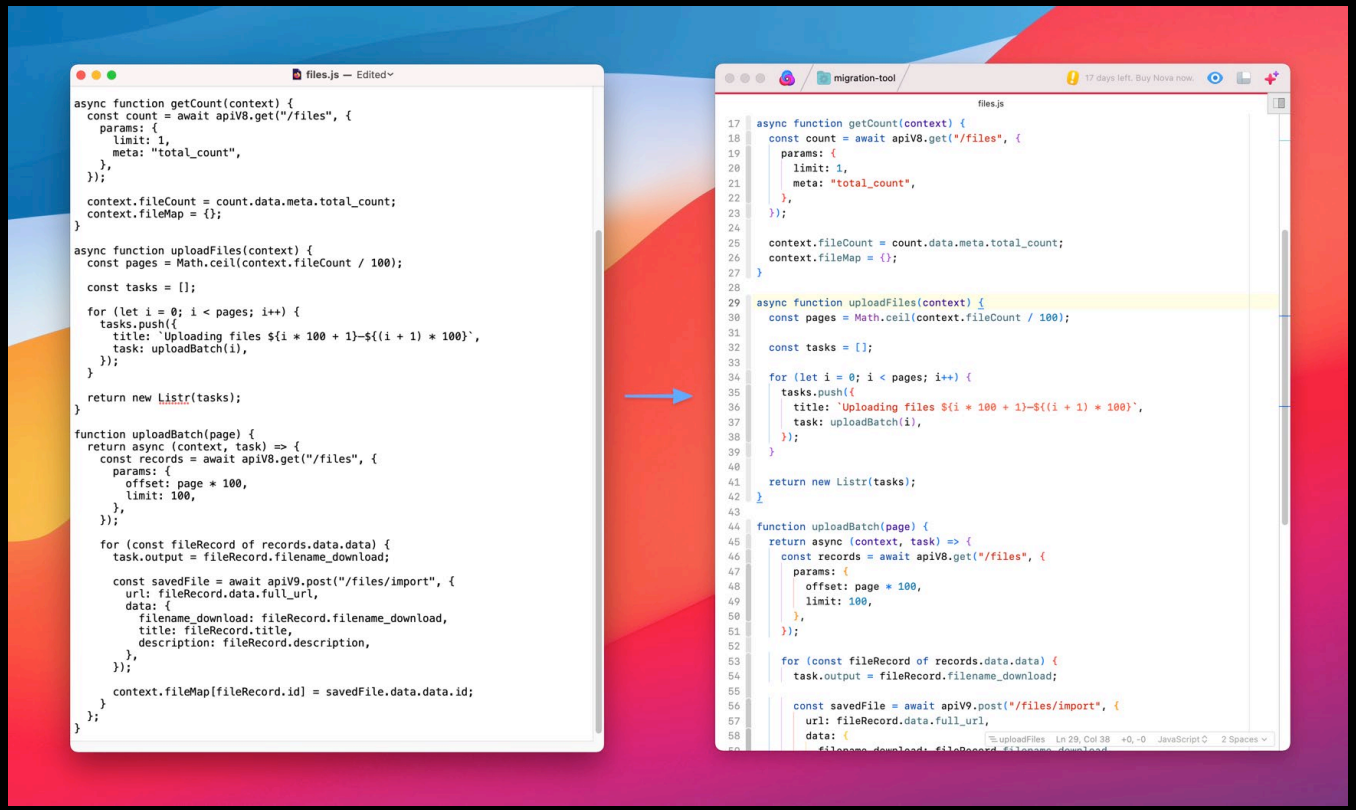
- Code is “just” text
- Computers don’t care what your code looks like
- Your code-editor is the most important tool in your arsenal

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: {
          filename_download: fileRecord.filename_download,
          title: fileRecord.title,
          description: fileRecord.description,
        },
      });

      context.fileMap[fileRecord.id] = savedFile.data.data.id;
    }
  };
}
```

type-map.js

```
export const typeMap = {
    alias: 'alias',
    array: 'json',
    binary: 'binary',
    boolean: 'boolean',
    datetime: 'dateTime',
    date: 'date',
    time: 'time',
    file: 'uuid',
    group: 'integer',
    hash: 'hash',
    integer: 'integer',
    decimal: 'decimal',
    json: 'json',
    lang: 'string',
    m2o: 'integer',
    o2m: 'alias',
    slug: 'string',
    sort: 'integer',
    status: 'string',
    string: 'string',
    translation: 'integer',
    uuid: 'uuid',
    datetime_created: 'timestamp',
    datetime_updated: 'timestamp',
    user_created: 'uuid',
    user_updated: 'uuid',
    user: 'uuid',
    owner: 'uuid',
}
```


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 Editors

- Specialized for writing software
- Make writing and reading source code easier
- Syntax Highlighting, Find and Replace, Extensions
- Personal Preference!


Visual Studio Code


Brackets

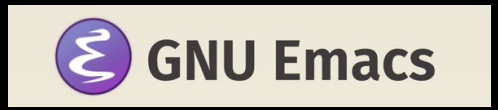

WS
WebStorm


Sublime Text

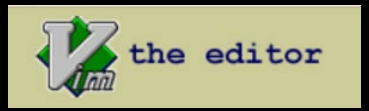

nova

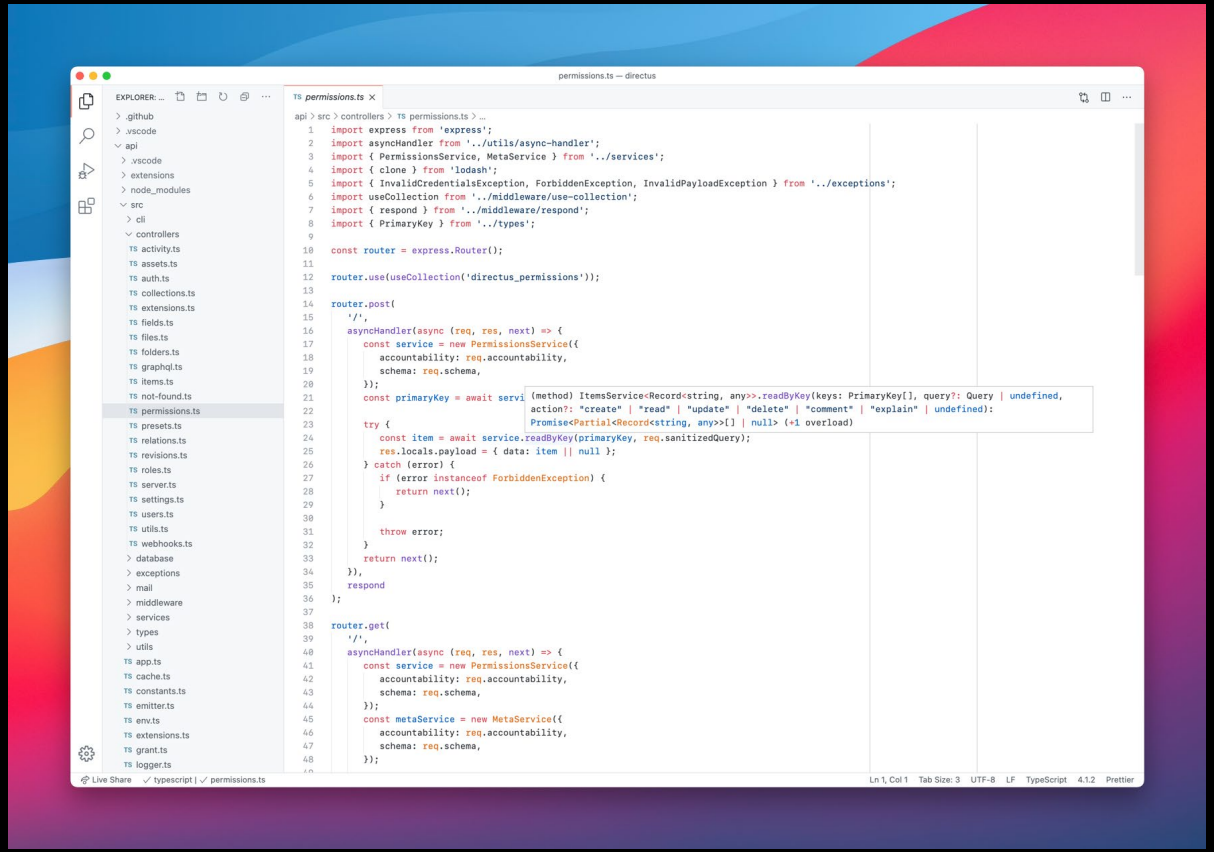

ATOM


Visual Studio

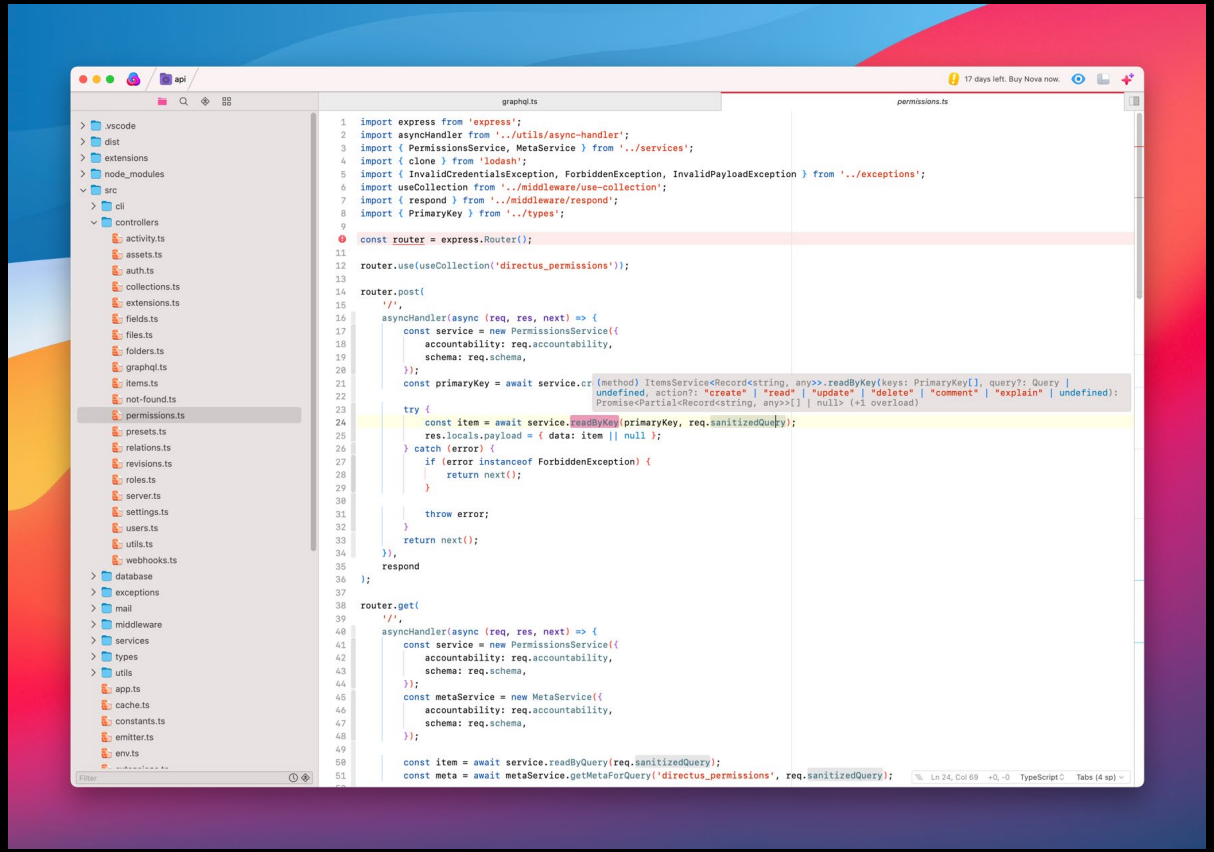

GNU Emacs


Xcode 12


the editor

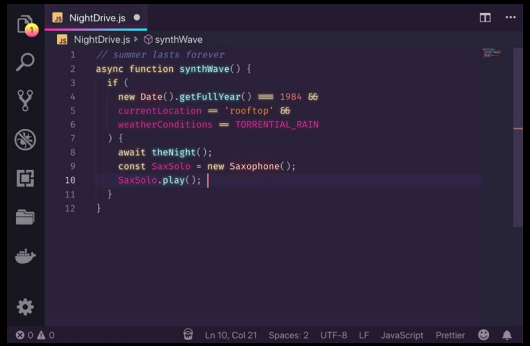


Visual Studio Code

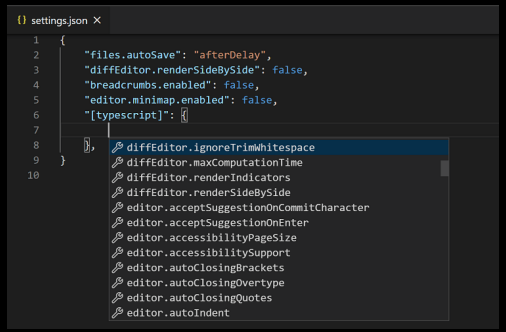

api
17 days left. Buy Nova now.
graphql.ts
permissions.ts
.vscode
dist
extensions
node_modules
src
cli
controllers
activity.ts
assets.ts
auth.ts
collections.ts
extensions.ts
fields.ts
files.ts
folders.ts
graphql.ts
items.ts
not-found.ts
permissions.ts
presets.ts
relations.ts
revisions.ts
roles.ts
server.ts
settings.ts
users.ts
utils.ts
webhooks.ts
database
exceptions
mail
middleware
services
types
utils
app.ts
cache.ts
constants.ts
emitter.ts
env.ts
import express from 'express';
import asyncHandler from '../utils/async-handler';
import { PermissionsService, MetaService } from '../services';
import { clone } from 'lodash';
import { InvalidCredentialsException, ForbiddenException, InvalidPayloadException } from '../exceptions';
import useCollection from '../middleware/use-collection';
import { respond } from '../middleware/respond';
import { PrimaryKey } from '../types';
const router = express.Router();
router.use(useCollection('directus_permissions'));
router.post(
'/',
asyncHandler(async (req, res, next) => {
const service = new PermissionsService({
accountability: req.accountability,
schema: req.schema,
});
const primaryKey = await service.cr
(method) ItemsService<Record<string, any>>.readByKey(keys: PrimaryKey[], query?: Query | undefined, action?: "create" | "read" | "update" | "delete" | "comment" | "explain" | undefined): Promise<Partial<Record<string, any>>[] | null> (+1 overload)
try {
const item = await service.readByKey(primaryKey, req.sanitizedQuery);
res.locals.payload = { data: item || null };
} catch (error) {
if (error instanceof ForbiddenException) {
return next();
}
throw error;
}
return next();
}),
respond
);
router.get(
'/',
asyncHandler(async (req, res, next) => {
const service = new PermissionsService({
accountability: req.accountability,
schema: req.schema,
});
const metaService = new MetaService({
accountability: req.accountability,
schema: req.schema,
});
const item = await service.readByQuery(req.sanitizedQuery);
const meta = await metaService.getMetaForQuery('directus_permissions', req.sanitizedQuery);
Ln 24, Col 69
+0, -0
TypeScript
Tabs (4 sp)

Nova

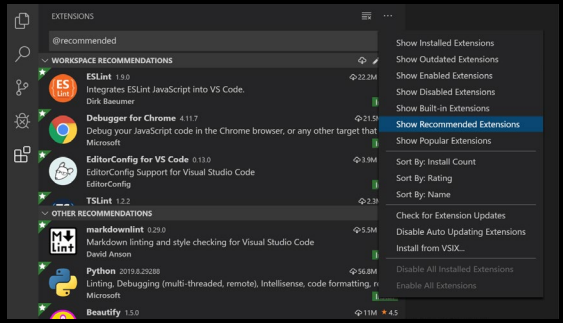
Code Editor Customisation



1) Look and Feel

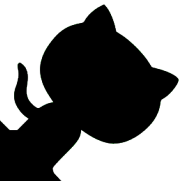


2) Settings



3) Extensions

Assignment (±30m)



Research the functionalities of your IDE development and document the research in your wiki. Look into the following subjects:

- Themes (f.e. color themes, fonts)
- Settings (f.e. settings.json)
- Extensions (f.e. extensions marketplace)

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.

```
vagrant@precise32:~$ eslint uploader.js

uploader.js
   1:13  error  'angular' is not defined              no-undef
   1:28  error  Strings must use doublequote          quotes
   7:15  error  Strings must use doublequote          quotes
   7:28  error  Strings must use doublequote          quotes
   7:34  error  Missing "use strict" statement        strict
   9:20  error  Expected '===' and instead saw '=='   eqeqeq
  10:17  error  Expected '===' and instead saw '=='   eqeqeq
  14:20  error  Strings must use doublequote          quotes
  27:16  error  'FormData' is not defined            no-undef
  34:17  error  'XMLHttpRequest' is not defined      no-undef
  35:1   error  Trailing spaces not allowed          no-trailing-spaces
  36:12  error  Strings must use doublequote          quotes
  45:1   error  Unexpected blank line at end of file  eol-last

â 13 problems

vagrant@precise32:~$
```

Syntax
(eslint)

```
function HelloWorld({
  greeting = "hello",
  greeted = '"World"',
  silent = false,
  onMouseOver,
}) {
  if (!greeting) {
    return null;
  }

  // TODO: Don't use random in render
  let num = Math.floor(Math.random() * 1e7)
    .toString()
    .replace(/\.\d+/gi, "");

  return (
    <div
      className="HelloWorld"
      title={`You are visitor number ${num}`}
      onMouseOver={onMouseOver}
    >
      <strong>
        {greeting.slice(0, 1).toUpperCase() + greeting.slice(1).toLowerCase()}
      </strong>
      {greeting.endsWith(",") ? (
        " "
      ) : (
        <span style={{ color: "grey" }}>", "</span>
      )}
      <em>{greeted}</em>
      {silent ? "." : "!"}
    </div>
  );
}
```

Formatter
(prettier)

Configuration File Formats

ESLint supports configuration files in several formats:

- **JavaScript** - use `.eslintrc.js` and export an object containing your configuration.
- **JavaScript (ESM)** - use `.eslintrc.cjs` when running ESLint in JavaScript packages that specify `"type":"module"` in their `package.json`. Note that ESLint does not support ESM configuration at this time.
- **YAML** - use `.eslintrc.yaml` or `.eslintrc.yml` to define the configuration structure.
- **JSON** - use `.eslintrc.json` to define the configuration structure. ESLint's JSON files also allow JavaScript-style comments.
- **package.json** - create an `eslintConfig` property in your `package.json` file and define your configuration there.

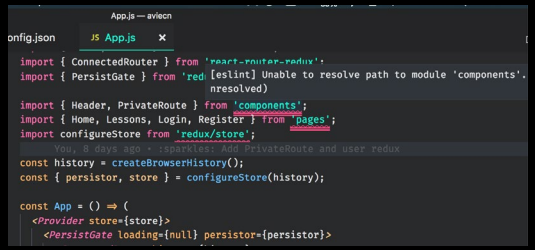
If there are multiple configuration files in the same directory, ESLint will only use one. The priority order is as follows:

1. `.eslintrc.js`
2. `.eslintrc.cjs`
3. `.eslintrc.yaml`
4. `.eslintrc.yml`
5. `.eslintrc.json`
6. `package.json`

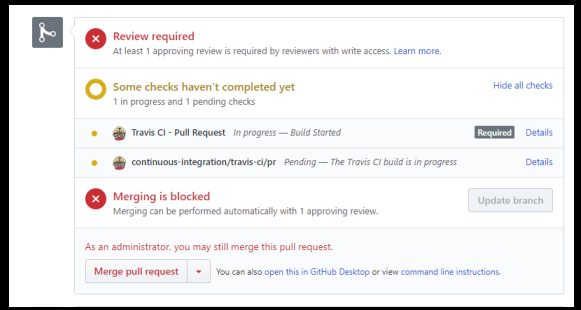
Installing an extension or package in your project is usually not enough.
More often then not these linters and formatters *only work if you actually configure them.*

```
vagrant@precise32:~$ eslint uploader.js

uploader.js
   1:13  error  'angular' is not defined              no-undef
   1:28  error  Strings must use doublequote          quotes
   7:15  error  Strings must use doublequote          quotes
   7:28  error  Strings must use doublequote          quotes
   7:34  error  Missing "use strict" statement        strict
   9:20  error  Expected '===' and instead saw '=='   eqeqeq
  10:17  error  Expected '===' and instead saw '=='   eqeqeq
  14:20  error  Strings must use doublequote          quotes
  27:16  error  'FormData' is not defined            no-undef
  34:17  error  'XMLHttpRequest' is not defined      no-undef
  35:1   error  Trailing spaces not allowed          no-trailing-spaces
  36:12  error  Strings must use doublequote          quotes
  45:1   error  Unexpected blank line at end of file eol-last

✖ 13 problems

vagrant@precise32:~$
```

CLI Output (run scripts)



IDE (extensions)



GitHub (testing)

Linters & formatters

why

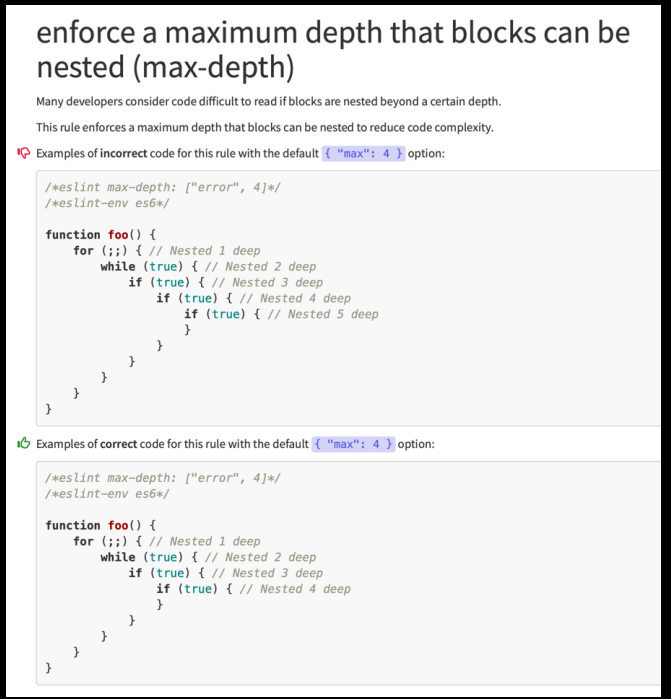
Ensuring a good coding style has a couple upsides:

- 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

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
                }
            }
        }
    }
}
```

Prettier

- Opinionated code formatter
- Rewrites your code to be consistent and adhering to the rules
- Very useful, but can cause friction

https://prettier.io/

```
function HelloWorld({greeting = "hello", greeted = '"World"', silent = false, onMouseOver,}) {

  if(!greeting){return null};

    // TODO: Don't use random in render
  let num = Math.floor (Math.random() * 1E+7).toString().replace(/\.\d+/ig, "")

  return <div className='HelloWorld' title={`You are visitor number ${ num }`} onMouseOver={onMouseOver}>

    <strong>{ greeting.slice( 0, 1 ).toUpperCase() + greeting.slice(1).toLowerCase() }</strong>
    {greeting.endsWith(",") ? " " : <span style={{color: '\grey'}}>", "</span> }
    <em>
    { greeted }
    </em>
    { (silent)
      ? "."
      : "!"}

    </div>;

}
```

Before

```
function HelloWorld({
  greeting = "hello",
  greeted = '"World"',
  silent = false,
  onMouseOver,
}) {
  if (!greeting) {
    return null;
  }

  // TODO: Don't use random in render
  let num = Math.floor(Math.random() * 1e7)
    .toString()
    .replace(/\.\d+/gi, "");

  return (
    <div
      className="HelloWorld"
      title={`You are visitor number ${num}`}
      onMouseOver={onMouseOver}
    >
      <strong>
        {greeting.slice(0, 1).toUpperCase() + greeting.slice(1).toLowerCase()}
      </strong>
      {greeting.endsWith(",") ? (
        " "
      ) : (
        <span style={{ color: "grey" }}>", "</span>
      )}
      <em>{greeted}</em>
      {silent ? "." : "!"}
    </div>
  );
}

```

After

.editorconfig

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

https://editorconfig.org

```
# 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

# Matches the exact files either package.json
or .travis.yml
[{package.json,.travis.yml}]
indent_style = space
indent_size = 2
```

Assignment (±30m)

Research one linter and **one formatter** and see what the functionalities are.
Additionally try to get one up and running in your project;

- Linters; help you catch **syntax errors** (f.e. ESLint, Stylelint, Markdownlint)
- Formatters; helps you **format** you code (f.e. Prettier, Editorconfig)

live demo linters & formatters

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

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

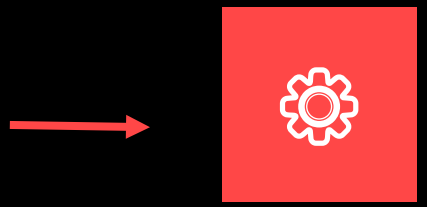
```css
section {
    height: 100px;
    width: 100px;
}

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

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

example

build

// Input

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



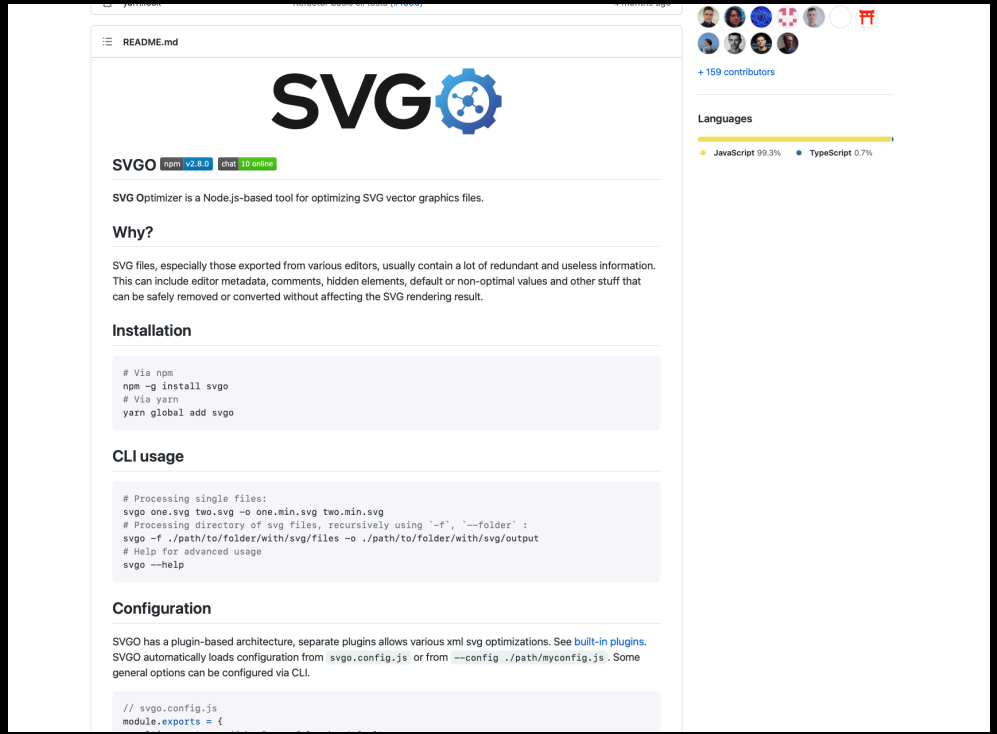
// Build

// Output

```
css/
└─ style.css
```

example

images



README.md

SVGO

SVGO npm v2.8.0 chat 10 online

SVG Optimizer is a Node.js-based tool for optimizing SVG vector graphics files.

Why?

SVG files, especially those exported from various editors, usually contain a lot of redundant and useless information. This can include editor metadata, comments, hidden elements, default or non-optimal values and other stuff that can be safely removed or converted without affecting the SVG rendering result.

Installation

```
# Via npm
npm -g install svgo
# Via yarn
yarn global add svgo
```

CLI usage

```
# Processing single files:
svgo one.svg two.svg -o one.min.svg two.min.svg
# Processing directory of svg files, recursively using `-f`, `--folder` :
svgo -f ./path/to/folder/with/svg/files -o ./path/to/folder/with/svg/output
# Help for advanced usage
svgo --help
```

Configuration

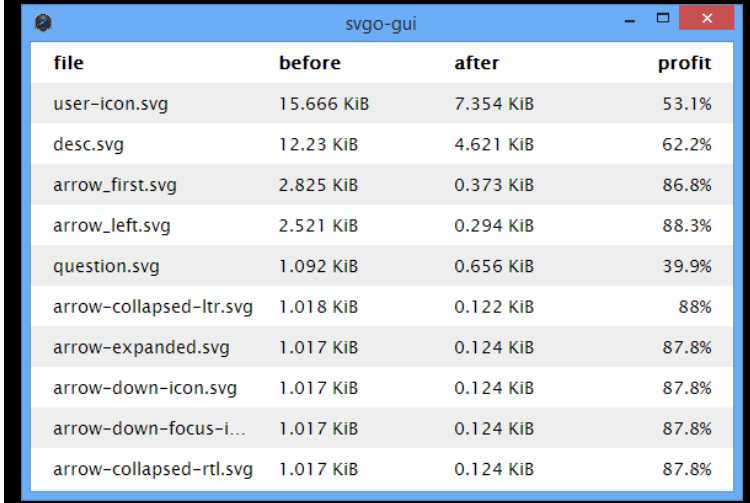
SVGO has a plugin-based architecture, separate plugins allows various xml svg optimizations. See built-in plugins. SVGO automatically loads configuration from `svgo.config.js` or from `--config ./path/myconfig.js`. Some general options can be configured via CLI.

```
// svgo.config.js
module.exports = {
```

+ 159 contributors

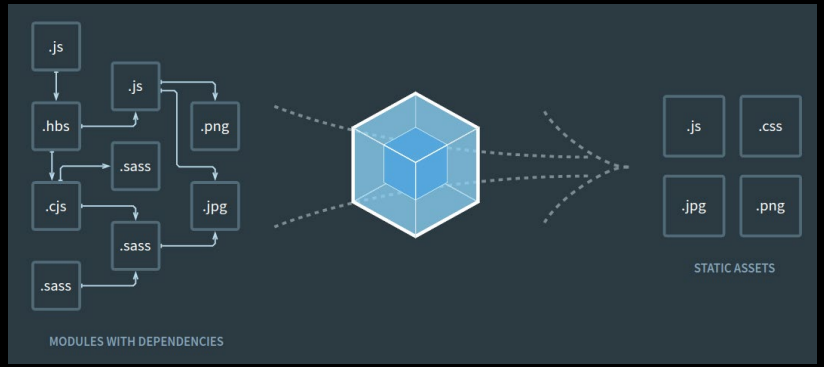
Languages

JavaScript 99.3% TypeScript 0.7%



file	before	after	profit
user-icon.svg	15.666 KiB	7.354 KiB	53.1%
desc.svg	12.23 KiB	4.621 KiB	62.2%
arrow_first.svg	2.825 KiB	0.373 KiB	86.8%
arrow_left.svg	2.521 KiB	0.294 KiB	88.3%
question.svg	1.092 KiB	0.656 KiB	39.9%
arrow-collapsed-ltr.svg	1.018 KiB	0.122 KiB	88%
arrow-expanded.svg	1.017 KiB	0.124 KiB	87.8%
arrow-down-icon.svg	1.017 KiB	0.124 KiB	87.8%
arrow-down-focus-i...	1.017 KiB	0.124 KiB	87.8%
arrow-collapsed-rtl.svg	1.017 KiB	0.124 KiB	87.8%

Types



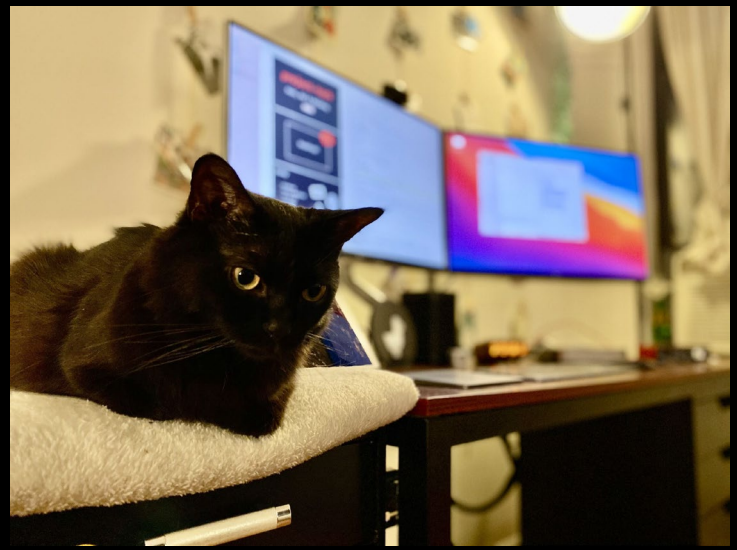
Bundlers (automation)

Processors (transpiling)

live demo build tools

Non-Virtual “Tooling”

- Stay hydrated
- Get enough sleep
- Reduce your brightness
- Make your workspace comfortable
- Take enough breaks



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!