Formation proposée par Ronin et ValT

Kesako ?

- Un framework vachement sympa pour faire du développement web
- Regroupe du HTML, JS et CSS en un seul fichier .astro

astro

Les avantages

Framework modulaire via:

- Layouts,
- Components

--> template réutilisable

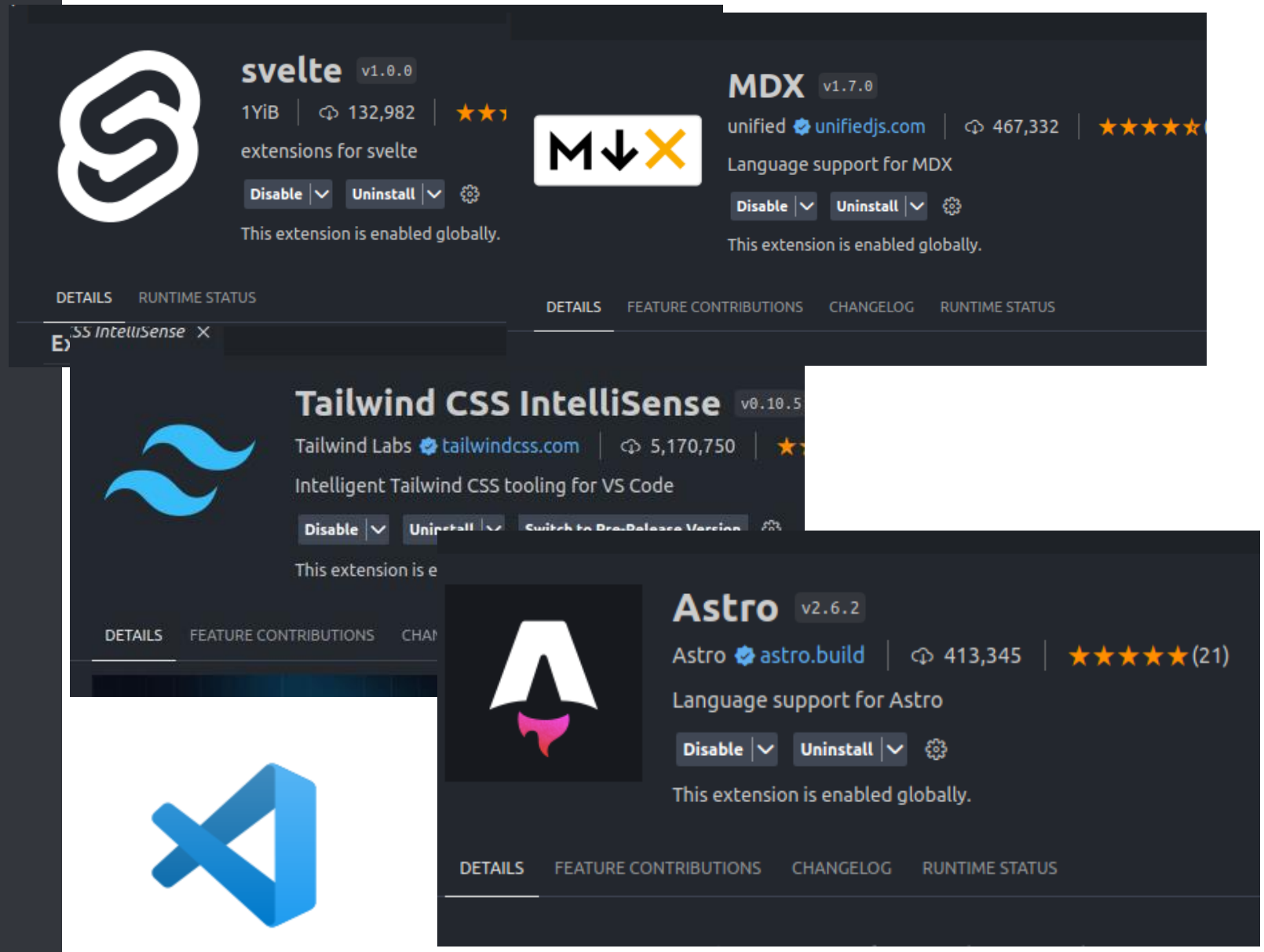
- Giga optimisé -> charge les components quand il a besoin

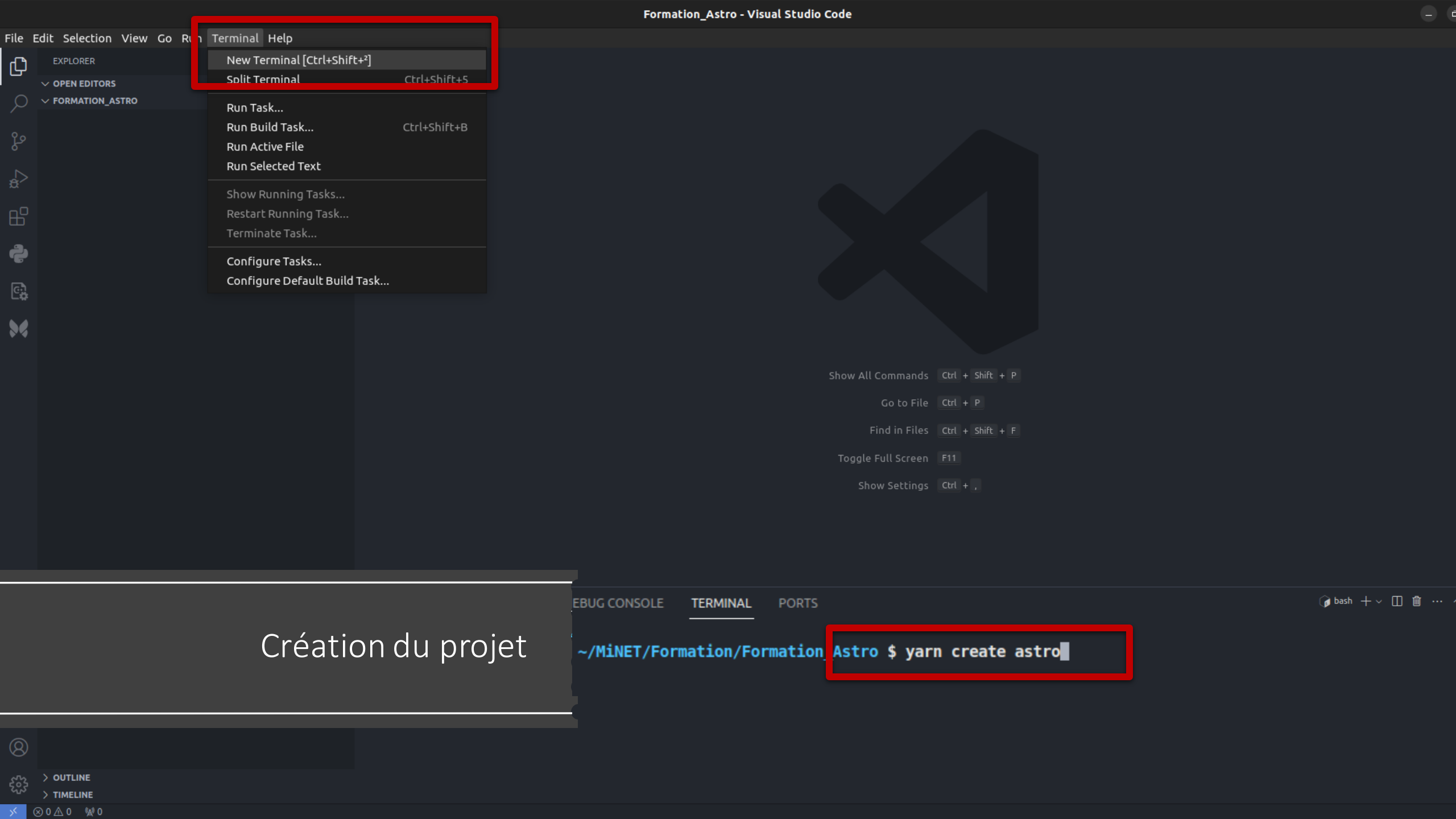
Simplifie les changements

Ce que nous devons apprendre à faire, nous l'apprendrons en le faisant - Aristote

Comment qu'on fait ?

1. Dans VSCode :
   - Installer les extensions:
     - Astro
     - Svelte
     - Tailwind CSS Intellisense
     - MDX





Création du projet

Houston vous parle

- Donnez lui votre dossier : .
- Choisissez Empty Project (sauf si vous voulez essayer les templates)
- Dites oui à Typescript et prenez l'option recommended
- Dites oui au repo git sauf si vous aimez développé dans votre coin
- Et...
- C'est TIPARRRRRRRRRRRRRRRRRRRR

```
yaienex@yaienex{} : ~/MiNET/Formation/Formation_Astro $ yarn dev
yarn run v1.22.21
warning package.json: No license field
$ astro dev

 astro  v4.2.6 ready in 275 ms

  Local    http://localhost:4321/
  Network  use --host to expose

14:03:04 watching for file changes...
```

Lancer le projet

Taper Yarn dev dans le terminal

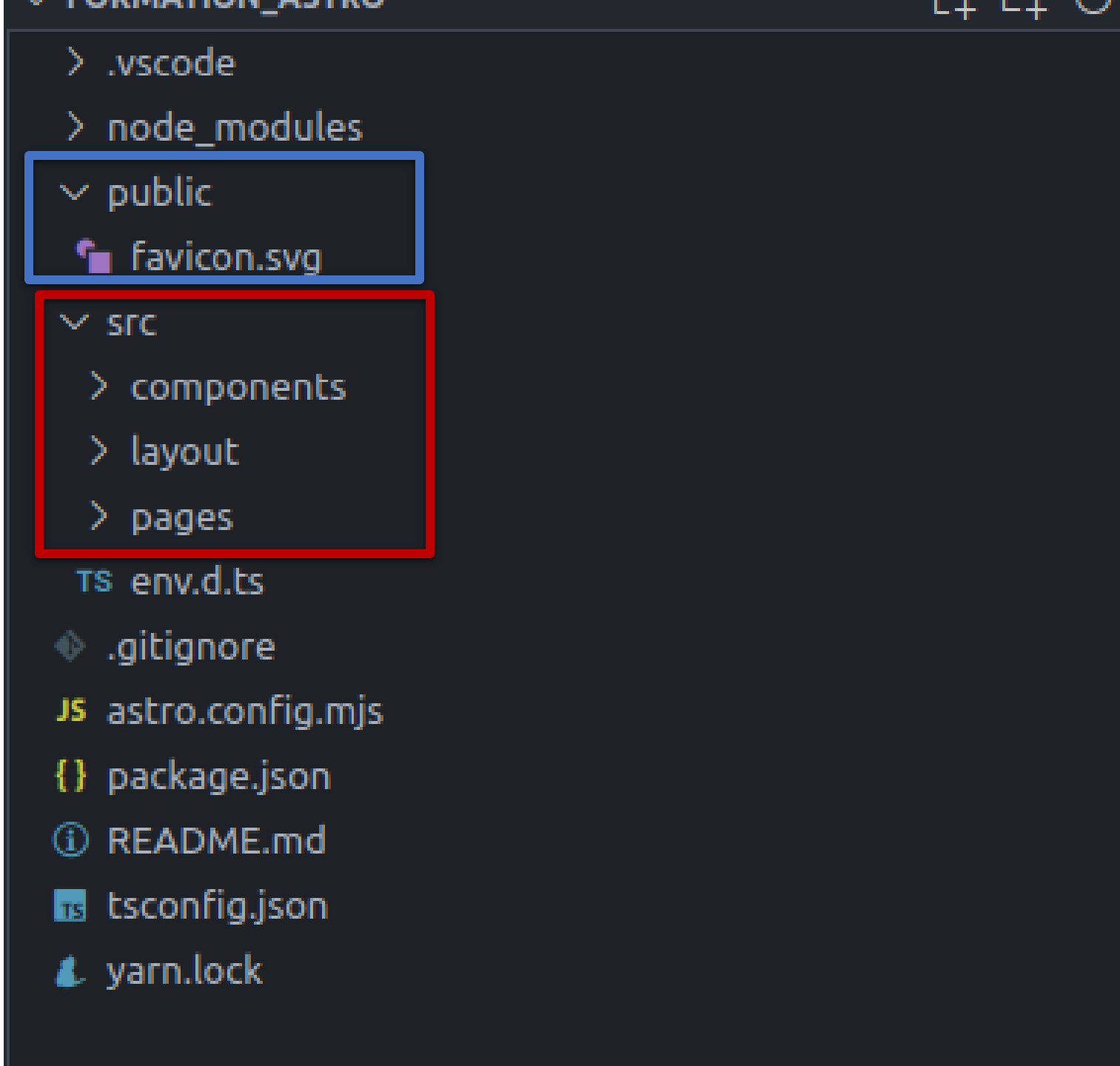
-> tous les changements seront pris en compte en live (sauf si ça amène à une erreur)

Keskon a là ?

Dossier public -> tout vos imports (Images, Vidéos, Fichiers)

Dossier src :

- Components -> contient des .astro qui sont des composants réutilisables et personnalisables
- Layout -> Template en .astro dont il faut abuser !
- Pages -> contient les pages de votre site en .astro toujours. Notamment index.astro



Keski change ?

Html : rien

Rootage des pages: rien

CSS :

- si tu connais Tailwind: rien
- Si tu connais pas : tout

Utiliser Tailwind CSS

Tailwind CSS Intellisense fait du remplissage aka pas besoin de tout connaître

https://tailwindcss.com/docs/installation

```
<style>

    .B{
        background-color: aqua;
        display: flex;
        font-size: large;
    }


</style>

<p class="B"> B  </p>
```

```
<p class="bg-cyan-400 flex text-lg"> B</p>
```

Ajouter Tailwind avec la commande yarn astro add tailwind et accepter tout

```
yaienex@yaienex{} : ~/MiNET/Formation/Formation_Astro $ yarn astro add tailwind
yarn run v1.22.21
warning package.json: No license field
$ astro add tailwind
✔ Resolving packages...
17:12:24
  Astro will run the following command:
  If you skip this step, you can always run it yourself later
```

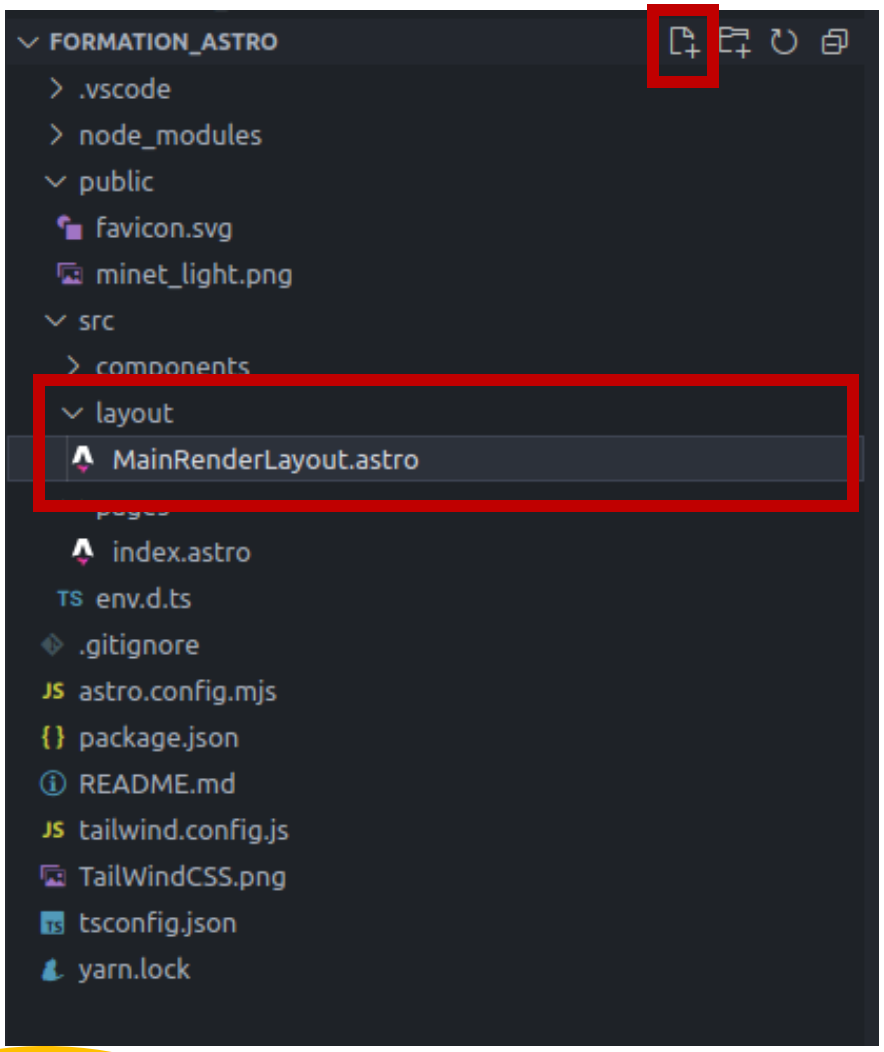
Attention

- Certaines valeurs de couleurs ne sont pas accessibles
- Tous les arguments n'existent pas forcément
- Faire gaffe aux fautes d'orthographe sinon ça marche po
- Écart de valeur prédéterminé: bg-cyan-400 marche mais pas bg-cyan-341

Jusqu'à maintenant

Triste un peu

Astro



Notre premier Layout

Bien le finir en .astro et toujours commencer par une majuscule pour les components et layouts

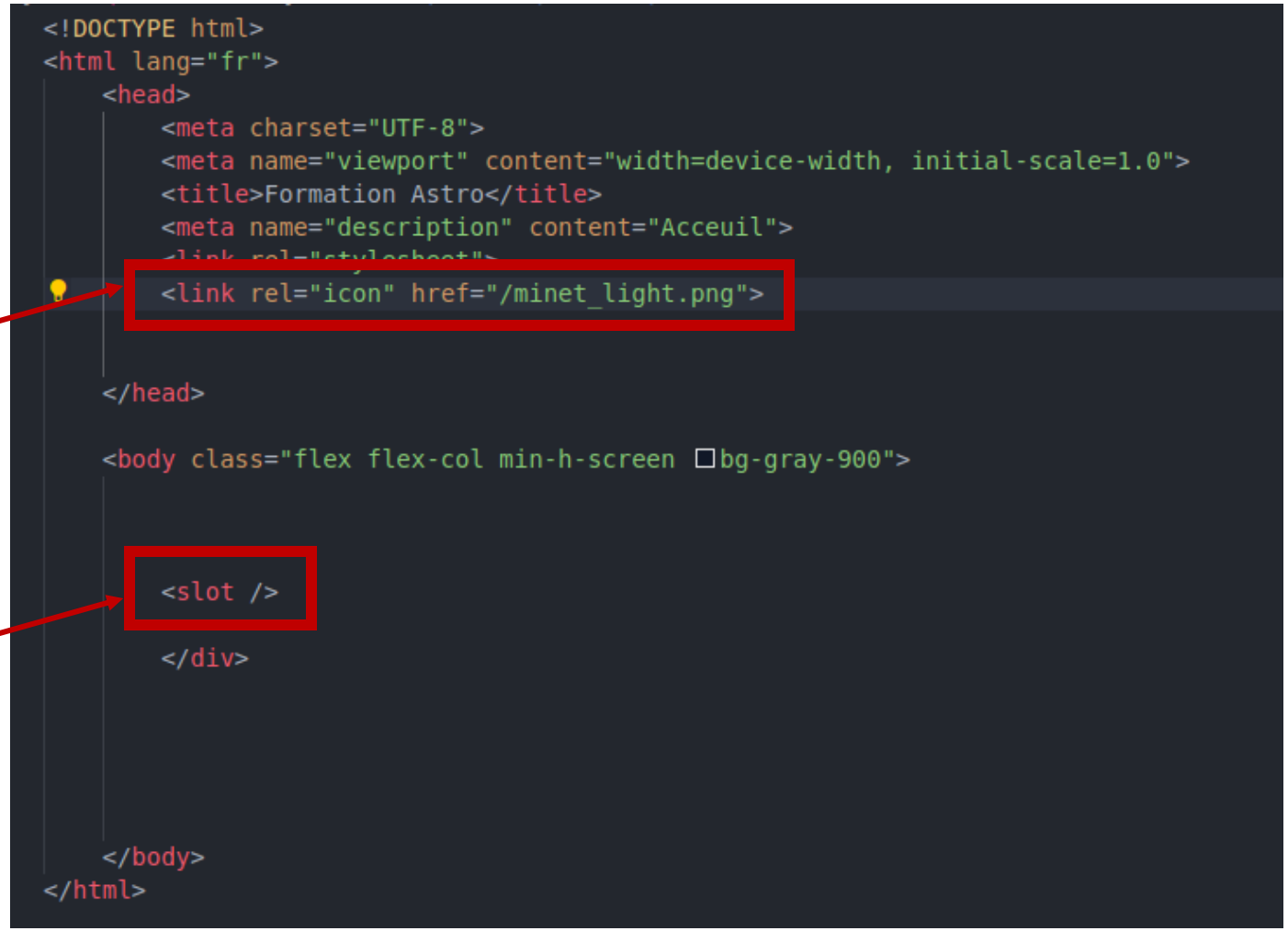
D'ailleurs bonne pratique: finir le nom du fichier par Layout

Créer un template à utiliser sur la plupart de vos pages

On accède au élément du dossier public via /elt et c'est tout

Cette balise sert à indiquer que l'on insèrera tout le contenu à la place de slot

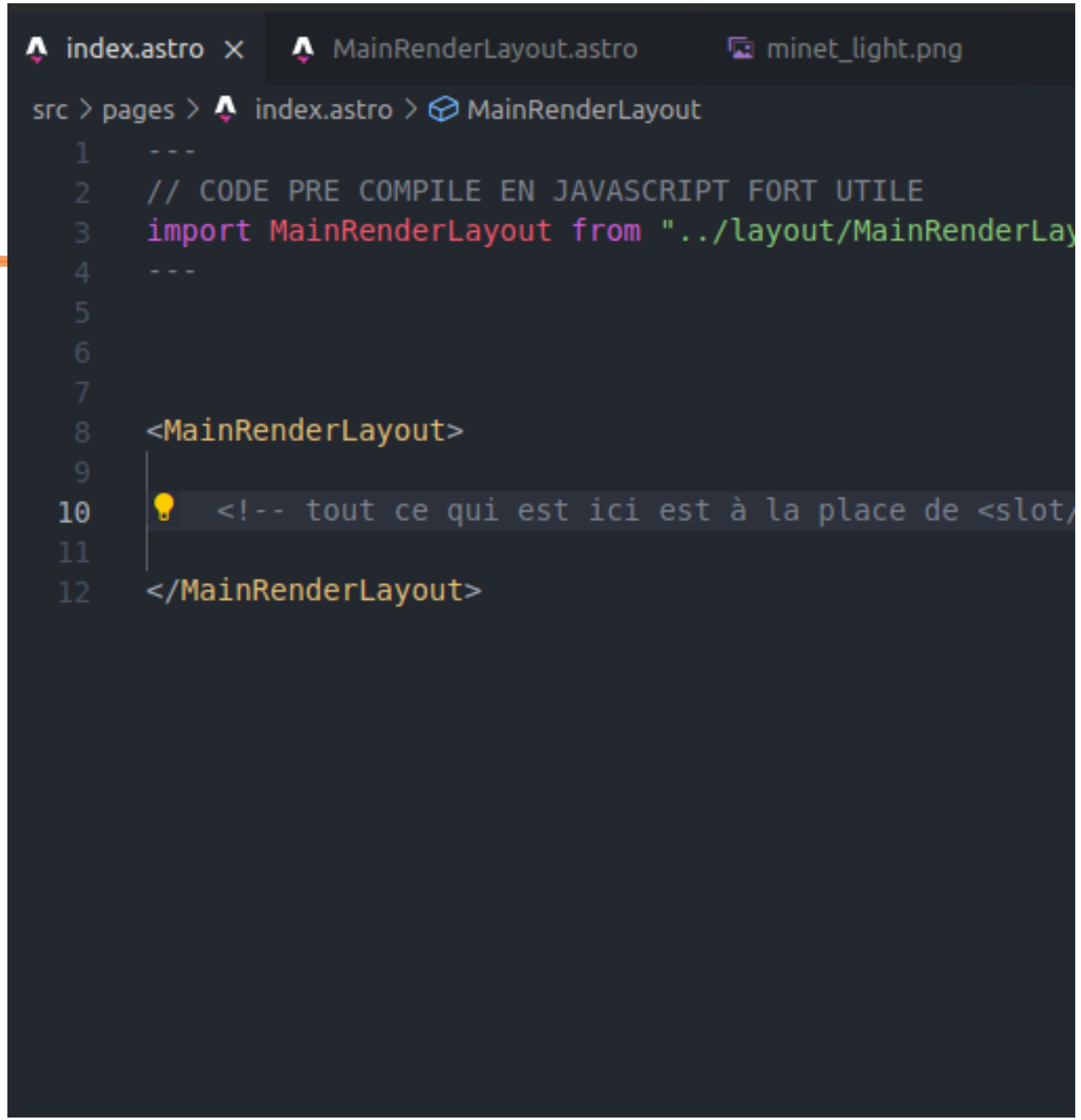
MainRenderLayout.astro



Dans index.astro

On import notre Layout et pouf le code est réduit à que ça

Mais Kesako que les --- ??



Les --- ---

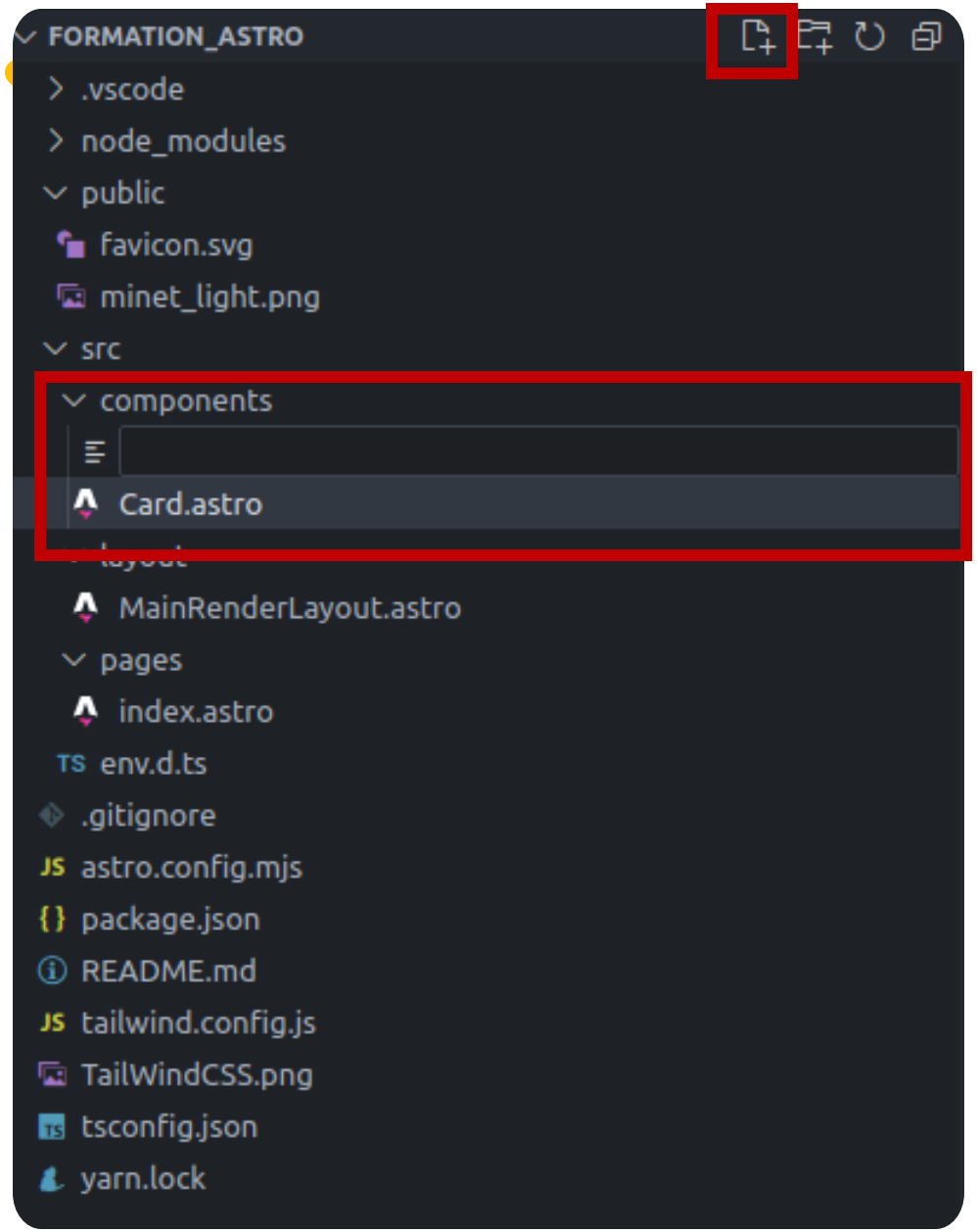
Du javascript pour le plus souvent importer toutes sortes de choses comme des Components ou des données

```
//import des components
import AccueilLayout from '../layouts/AccueilLayout.astro'

import Render from '../components/Render.astro'


//code
var DATA:any;
const search =  Astro.url.searchParams.get("q")!.toString();
const filter = Astro.url.searchParams.get("f")!.toString();

async function fetchSearchResults(recherche:string,filtre:string){

    const res = await import(`./search.json`); //comme on est coté serveur on a juste à importer et pas fetch
    const data = await ( await res.get(recherche,filtre)).json(); //ne pas croire le message d'erreur

    return data;

}
//on récupère le tableau
DATA =  await  fetchSearchResults(search,filter);
---
```

Exemple plus complet de ce que l'on peut faire

Notre premier Component

Même chose majuscule + .astro



Exemple de Card.astro

```
---
const {name, image,link } = Astro.props
---

<div class="m-2 rounded-xl h-80 w-auto bg-gradient-to-r from-white to-slate-900 hover:from-pink-500 hover:to-yellow-500 hover:shadow-xl ">
    <p class="text-xs mb-2 text-purple-950 dark:text-purple-200"></p>

    <img class="text-purple-950  " width="400" height="400" src={image}></p>


    <a class="m-2 px-4 hover:shadow-lg py-2 font-semibold text-sm bg-purple-700 text-white rounded-lg shadow-sm w-fit" href="http://google.com">
        Read more &rarr;
    </a>
</div>
```

Tous les codes copiables sont en fin de TP

Kesako ça ?

Astro nous permet de donner des éléments à nos components ! Ils ne sont pas tous pareil après tout

On va donc pouvoir passer trois arguments à notre components name, image et link

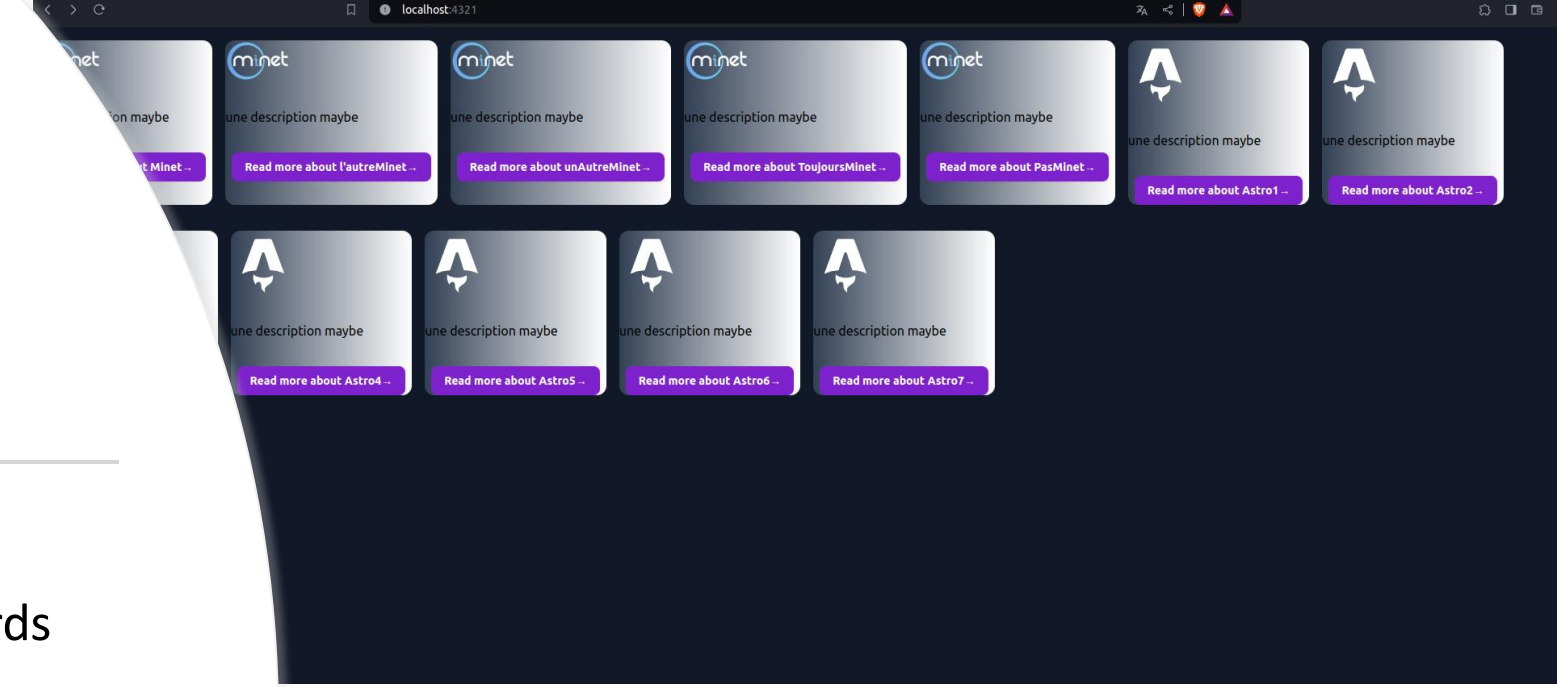
```
---
const {name, image,link } = Astro.props
---
```

On ya accède dans le code html par {image} pour récuperer l'adresse de l'image

```
<img class="text-purple-950  " width="400" height="400" src={image}></p>
```

Notre premier rendu

On importe bien une image que l'on met dans ./public et en ajoutant quelques Cards à index.astro on obtient:




```
from "../components/Card.astro"

erLayout>

tout ce qui est ici est à la place de <slot/> -->
rd name="Yes" image="/minet_light.png"/>
ard name="Oui" image="/minet_light.png"/>
Card name="Oui" image="/minet_light.png"/>
<Card name="Oui" image="/minet_light.png"/>
<Card name="Oui" image="/minet_light.png"/>
```

On récapitule

Components -> réutilisable / modulable via des variables utilisables via {var} dans la partie html

Layouts -> réutilisable / <slot/> détermine l'espace où le "code" html sera contenu

TailWindCSS -> fort pratique et faut prendre en main l'écriture

--- --- -> Javascript utilisable dans le fichier html

Pour accéder au contenu du dossier public on peut directement faire /element

Mais on en veut plus !

Admettons j'ai une liste avec des noms et une autre avec les liens correspondants. On vas pas s'amuser à tout taper (surtout s'il y a 1000 liens)

Déjà, déclarons nos variables dans index.astro

```
const allCards = [
    {name:"Minet",image:"/minet_light.png"},
    {name:"l'autreMinet",image:"/minet_light.png"},
    {name:"unAutreMinet",image:"/minet_light.png"},
    {name:"ToujoursMinet",image:"/minet_light.png"},
    {name:"PasMinet",image:"/minet_light.png"}]// un tableau de dictionnaire
```

Autre moyen de le faire

Si vous réutilisez plein de fois un tableau de variable ou une base de données par exemple, il vaut mieux créer un fichier const.ts dans lequel vous mettez toutes vos variables que vous importez par la suite

```
import {ALLCARDS} from "../const"
```

Utilisable de la même manière que le tableau qu'on a créé juste avant

Il faut bien exporter la constante que vous voulez utilise sinon VSCode va vous dire qu'il ne connaît pas

Place à la magie

Comme vous l'avez, on peut invoquer des variables avec {} (si on est dans le code html. Si vous êtes dans les --- vous pouvez direct le faire) mais on peut aussi faire d'autre trucs comme une boucle for !!

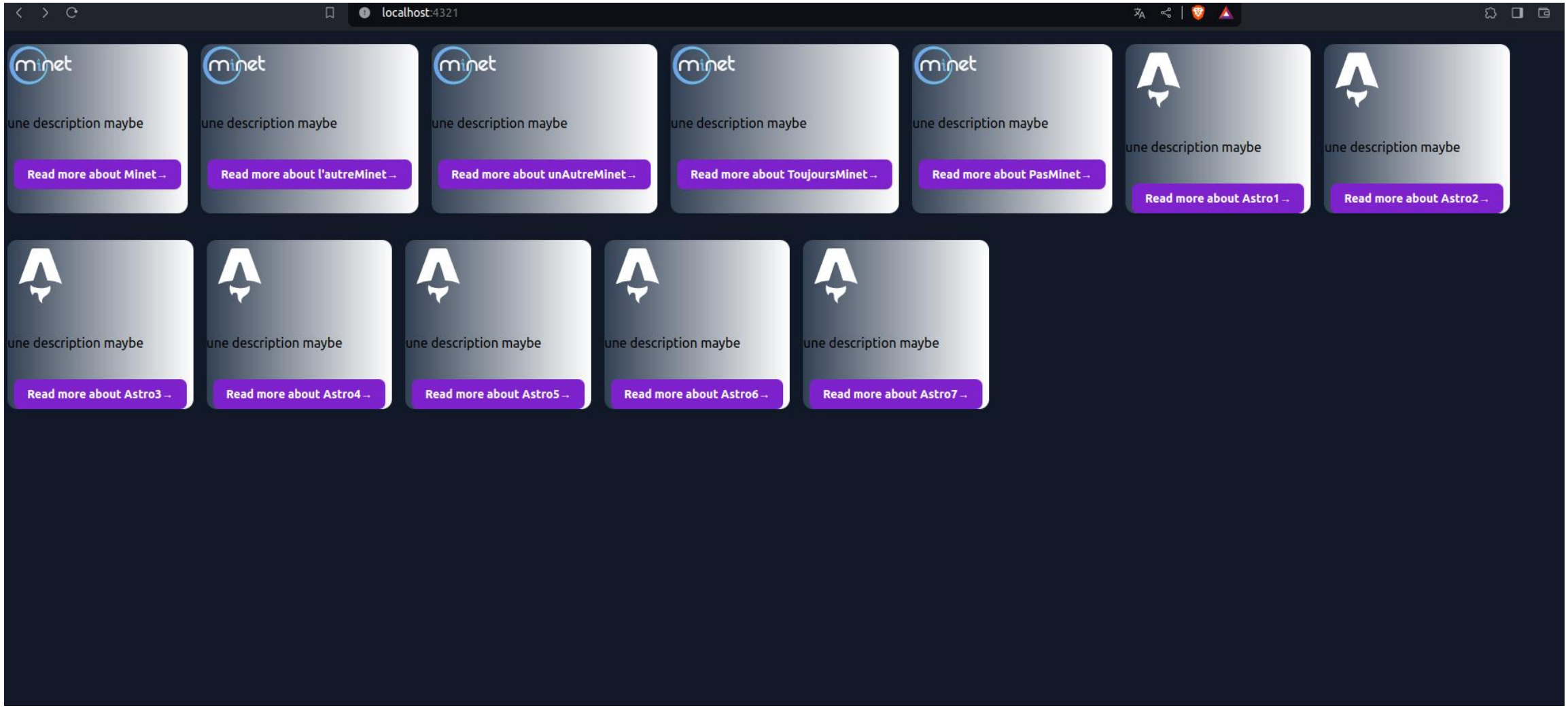
On doit l'écrire ainsi :

```
{
tableau.map((element) =>
{element}
)}
```

```
{allCards.map((card) =>
    <Card name={card.name} image={card.image}/>
)}
```

Comme card est un dictionnaire on accède au sous élément grâce à card.nomAttribut

On a le même résultat que tout à l'heure mais en mieux codé



Autre manière de le faire

Si vous pensez que passer tous les arguments d'une card à notre component c'est pas très smart, on peut directement lui envoyer la card ! Ce qui nous donnerai

Dans index.astro

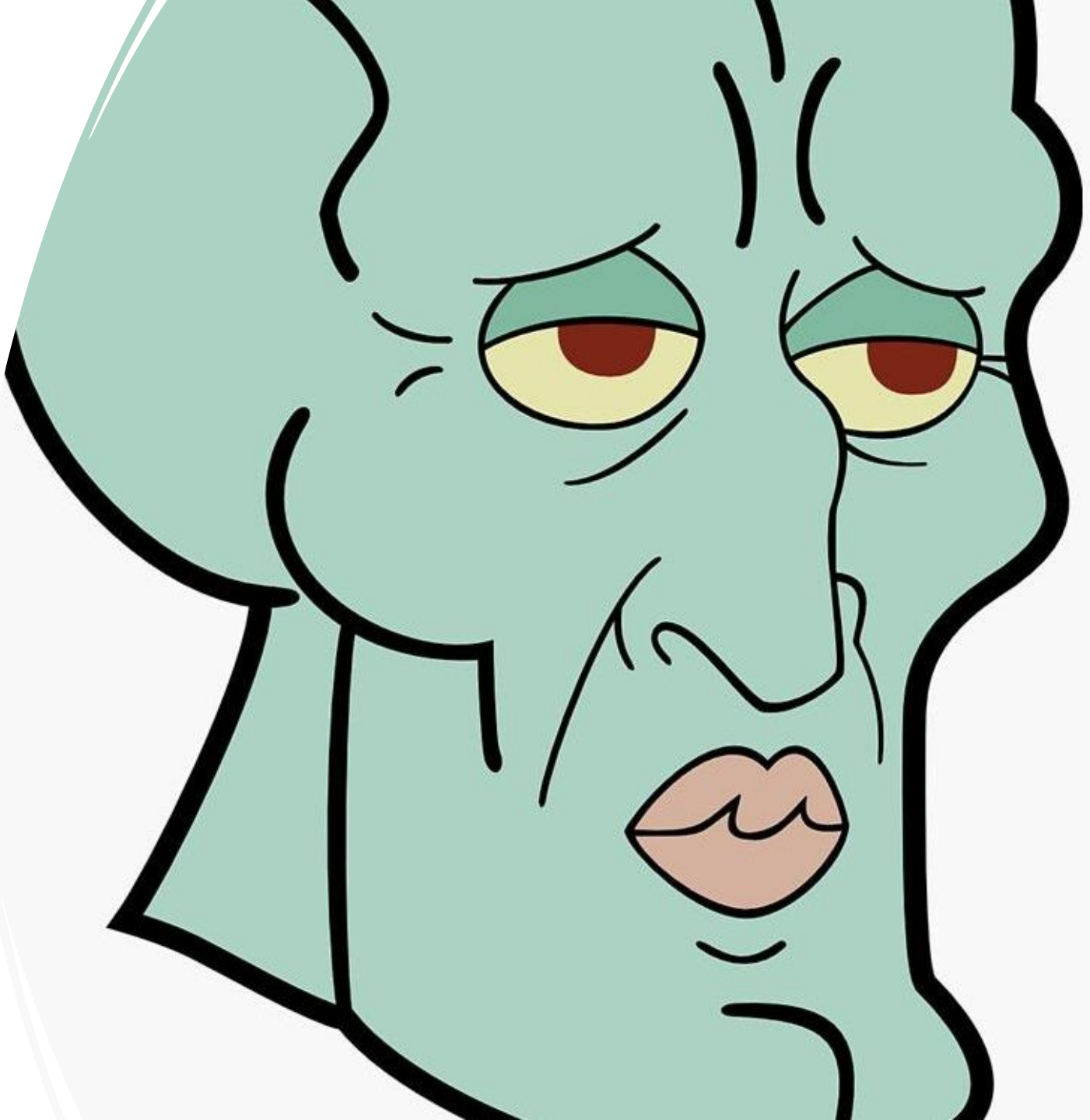
```
{   ALLCARDS.map((card) =>
    <CardDirect card={card}/>
)
}
```

Dans le component

```
---
const {card} = Astro.props
---

<div class="m-2 rounded-xl h-80 w-auto bg-gradient-to-r from-white to-slate-900 hover:from-pink-500 hover:to-yellow-500 hover:shadow-xl ">
    <p class="text-xs mb-2 text-purple-950 dark:text-purple-200"> </p>

    <img class="text-purple-950  " width="400" height="400" alt="Flag_Oui" src={card.image}></p>


    <a class="m-2 px-4 hover:shadow-lg py-2 font-semibold text-sm bg-purple-700 text-white rounded-lg shadow-sm w-fit" href="http://google.com">
        Read more about {card.name}&rarr;
    </a>
</div>
```

Et on a toujours le même rendu

C'est magnifique



On peut aussi ...

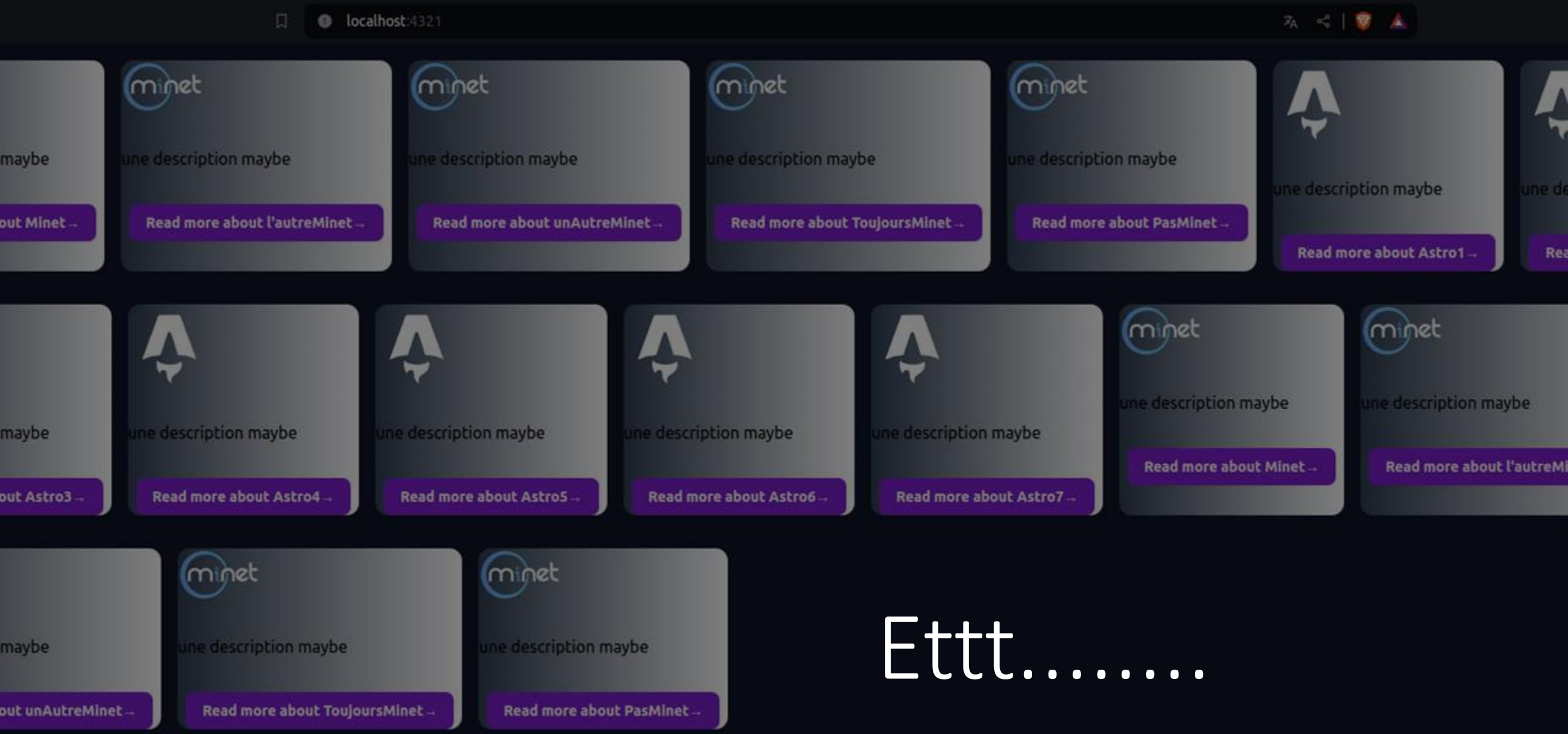
Faire des structures conditionnelles !
Admettons qu'on veuille filtrer nos card avec seulement les cards de Minet (faut en ajouter d'autres au préalable)

On en rajoute de la même manière qu'on a déjà fait puis on ajoute la structure

```
{
tableau.map((element) =>(

(test(element) ? oui : non )

) )}
```

```
{   ALLCARDS.map((card) => (
    (card.name.indexOf("Minet") !== -1 ?  <CardDirect card={card}/> : <p></p> )


))}
```

On check si notre nom contient la chaîne de caractère "Minet" si oui alors on affiche sinon on ne fait rien
Si vous ne faites rien avec la donnée, mettez une balise quelconque vide, ça évitera d'avoir des [Object] qui spawn



Ettt.....

En affichant une fois sans, puis avec la condition

index.astro | Card.astro | RickRoll.md ● | CardDirect.astro

rc > pages > chansons > RickRoll.md

```
---
name: "RickRoll"
layout: "../../layout/MainRenderLayout.astro"
---

<iframe width="1000" height="500" src="https://www.youtube.com/embed/dQw4w9WgXcQ?si=OmxR9K5tP3S6uQbx" title="YouTube video pla
allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share" allowfullscreen></
```

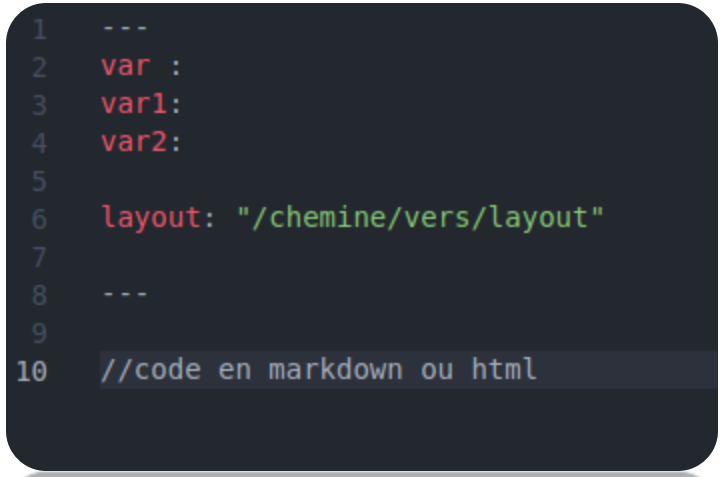
Deux dernières features mais pas des moindres

On a installé une extension du doux nom de MarkDown au tout débuuut et bah on va l'utiliser

Je vous propose de créer un (ou plusieurs) fichiers finissant en .md, mais... il faut mettre quoi dedans ?

Vous avez un petit exemple au dessus qui met une balise pour une vidéo youtube

La syntaxe en .md

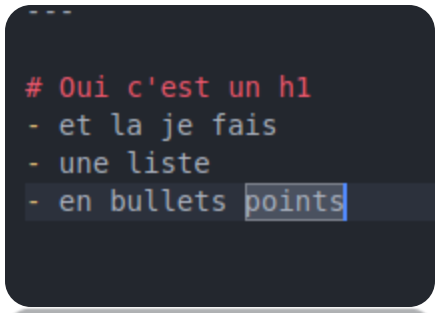


```
---
var :
var1:
var2:

layout: "/chemine/vers/layout"

---

//code en markdown ou html
```

On a encore des --- --- dans lesquels on va mettre autant de variables concernant notre fichier (l'année, l'auteur, une image etc) + il faut mettre un layout qui va être utiliser comme base pour cette page (et oui un .md en fait c'est une page html)

https://www.markdownguide.org/cheat-sheet/

Ensuite on peut écrire comme dans un .html ou .astro, mais l'idéal c'est d'utiliser la syntaxe .md



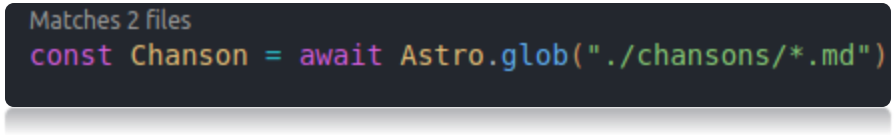
```
---

# Oui c'est un h1
- et la je fais
- une liste
- en bullets points
```

Il se peut que Tailwind interfère avec la syntaxe et que vous ne voyez pas vraiment un h1 si vous allez voir la page

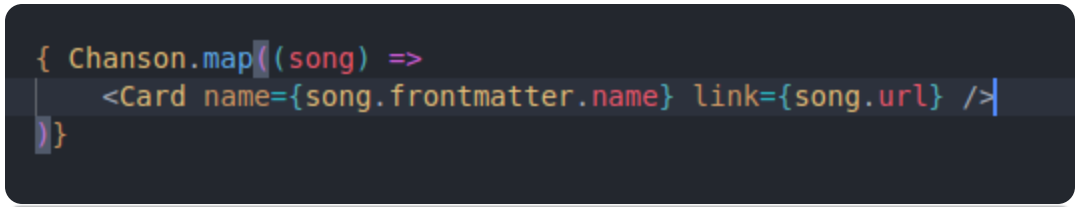
Et les arguments ?

Bah oui on met des arguments mais comment on s'en sert.

Déjà faut importer tous les fichiers .md dans la page où vous allez les utiliser grâce à la fonction glob



On reconnaît que le *.md veut dire qu'on importe tous les fichiers qui ont l'extension .md



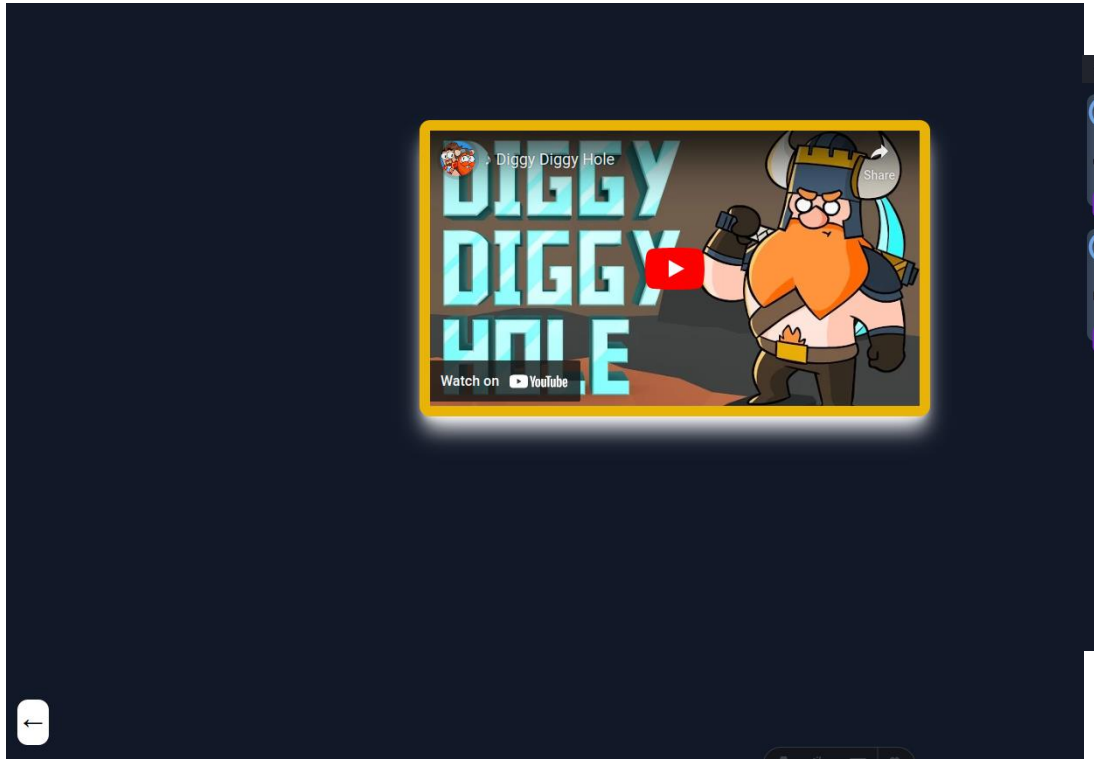
Un élément de notre collection Chanson possède 4 paramètres utiles:

- url : le chemin vers le fichier
- frontmatter: tableau des arguments personnels
- name : nom du fichier

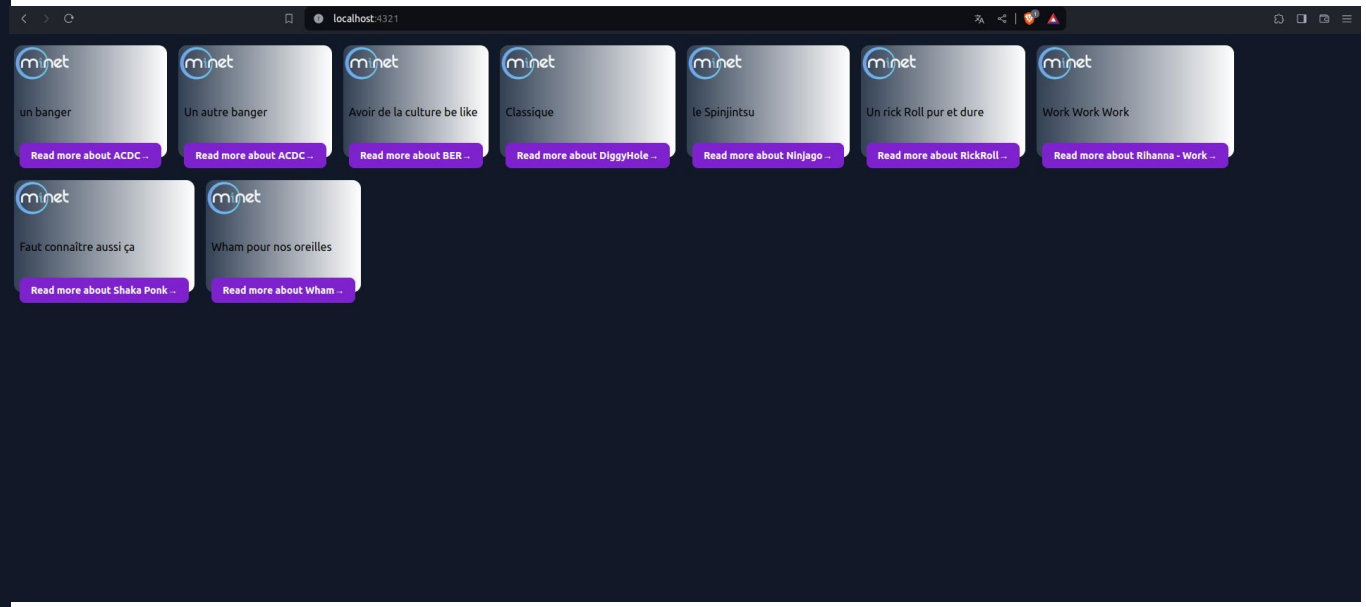
Et donc vos arguments sont tous bien au chaud dans frontmatter et vous y accédez comme d'hab, frontmatter.nomVariable.

On met tout ça en place et.....

J'ai choisi de faire une librairie avec tout pleins de vidéos



Quand on clique sur un des liens pouf une vidéo



La page principale (j'en ai mis quelques uns à titre d'exemple mais vous pouvez en avoir autant que vous voulez ofc)

Magnifique

Pour aller vraiment plus loin

Vous avez tous les éléments nécessaires pour faire à peu près ce que vous désirez. Mais comme vous êtes des grands malades, je vais vous parler des bases de données avec Astro grâce à Xata

Xata, c'est kwa ?

Serverless database platform powered by PostgreSQL - source leur site

Elle vous permet d'avoir plein de Tables différentes au même endroit et c'est surtout gratuit !

Comme je suis grand prince je vais vous donner l'accès à une base de données Xata que j'héberge sur mon compte

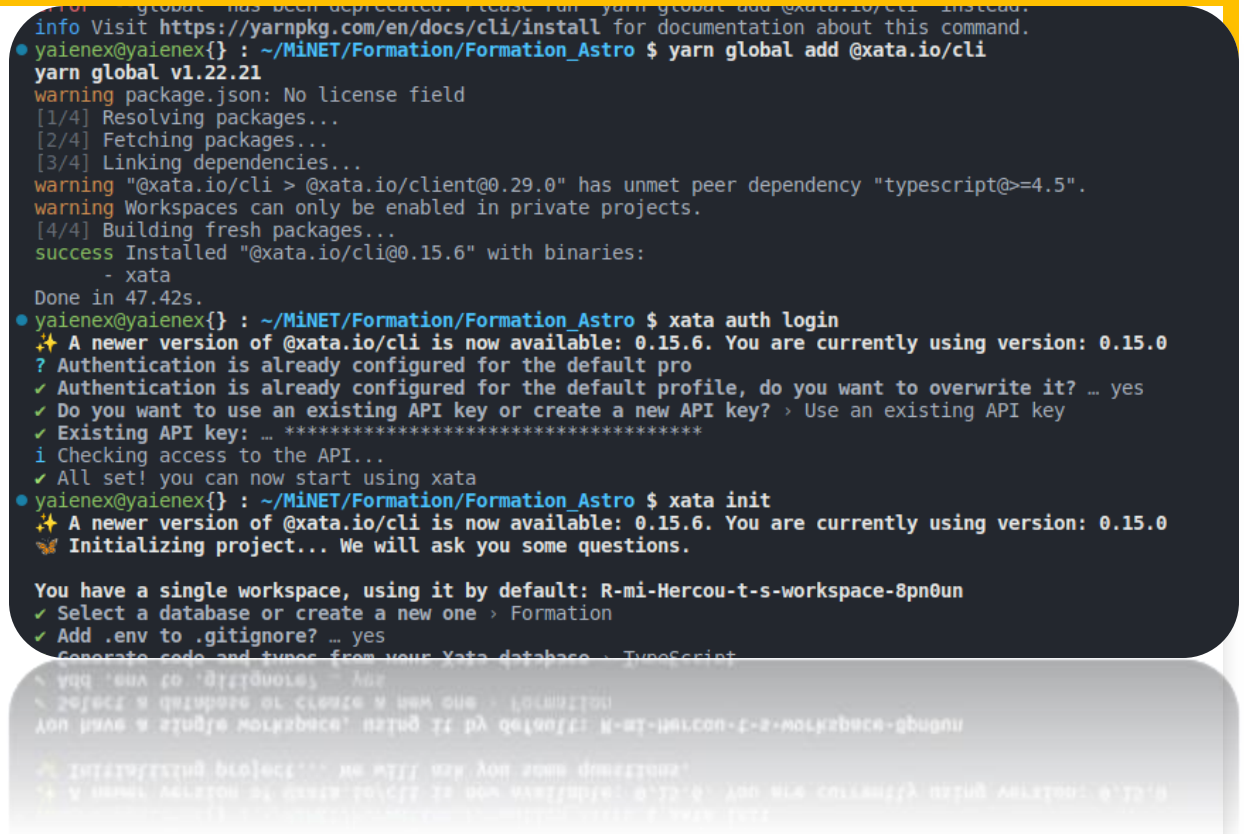
Mais comment faire ?

Entrer la commande yarn global add @xata.io/cli

Puis xata auth login et donnez lui cette API key (n'en créez pas de nouvelles):

xau_7WSiyjuOWsB9PvG9fTqyQku4Pko6Y75c5

Enfin lancez la base de données avec xata init, selectionnez la db Formation et on est bon !



```
import { getXataClient } from "../xata"


const data = getXataClient();
```

Et mtn ?

On va pas trop modifier notre code, la db possède juste des utilisateurs et on va les mettre dans nos cards !

Tout d'abord, dans index il faut importer la base de données puis on va extraire tout les utilisateurs dans un tableau avec une commande que l'on détaille à la slide d'après

```
const data = getXataClient();
const Users = await data.db.Users.getAll();
```

On va chercher dans les bases de données (db = database)

On récupère la table Users

On récupère tous les utilisateurs sous la forme d'un tableau

Si on vous sort une erreur parlant d'ApiKey, il faut aller dans le fichier xata.ts, retrouver la fonction getXataClient() et ajouter ceci

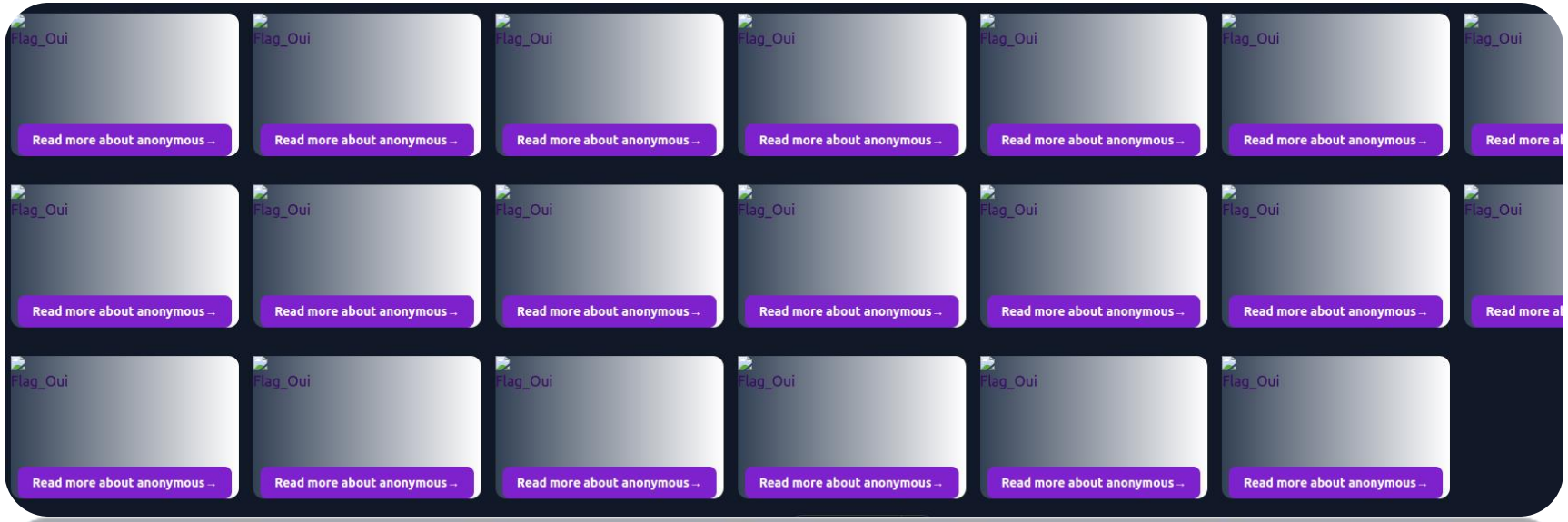
```
export const getXataClient = () => {
  if (instance) return instance;

  instance = new XataClient({ apiKey : "xau_7WSiyjuOWsB9PvG9fTqyQku4Pko6Y75c5"});
  return instance;
};
```

Et....

On l'emploie de la même manière avec map et on accède au nom des utilisateurs avec le nom de l'attribut ici name

```
{Users.map((user:any) =>

    <Card name={user.name} />

)}
```

On est bon !! Tout le monde s'appelle anonymous c normal



Pour aller plus loin

Utiliser ce projet git pour vous familiariser avec les interactions avec la DB. C'est un peu complexe donc si vous avez des questions, n'hésitez pas si ça vous intéresse !

Et vous verrez qu'il s'agissait de la V1 de mon projet dev ahah

https://gitlab.com/Yaienex/Inventaire.git

CODES

Card.astro

```
---
const {name, image,link } = Astro.props
---



<div class="mx-2 my-4 rounded-xl flex flex-col space-y-8 h-40 w-auto bg-
    gradient-to-r from-slate-700 to-white hover:from-pink-500 hover:to-
    yellow-500 hover:shadow-xl ">
<img class="text-purple-950 " width="80" height="200" alt="Oui"
    src={image}></p>
<div class="whitespace-normal">une description maybe </div>
<a class="m-2 px-4 hover:shadow-lg py-2 font-semibold text-sm bg-
    purple-700 text-white rounded-lg shadow-sm w-fit"
    href="http://google.com">
Read more about {name}&rarr;
</a>
</div>
```

Index.astro

```
---
// CODE PRE COMPILE EN JAVASCRIPT FORT UTILE
import MainRenderLayout from "../layout/MainRenderLayout.astro"
import Card from "../components/Card.astro"
import CardDirect from "../components/CardDirect.astro"

const allCards = [
{name:"Minet",image:"/minet_light.png"},
{name:"l'autreMinet",image:"/minet_light.png"},
{name:"unAutreMinet",image:"/minet_light.png"},
{name:"ToujoursMinet",image:"/minet_light.png"},
{name:"PasMinet",image:"/minet_light.png"}]// un tableau de dictionnaire

import {ALLCARDS} from "../const"

const Chanson = await Astro.glob("./chansons/*.md")
import { XataClient } from "../xata"
const data = new XataClient({apiKey: "xau_7WSiyjuOWsB9PvG9fTqyQku4Pko6Y75c5"});
const Users = await data.db.Users.getAll();

---

<MainRenderLayout>

{ Chanson.map((song) =>
<Card name={song.frontmatter.name} link={song.url} description={song.frontmatter.description} image={song.frontmatter.image} />
)}

{Users.map((user:any) =>

<Card name={user.name} />
)}

</MainRenderLayout>
```

Layout.astro

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Formation Astro</title>
<meta name="description" content="Acceuil">
<link rel="stylesheet">
<link rel="icon" href="/minet_light.png">
</head>



<body class="flex flex-wrap bg-gray-900">




<slot />




</body>
</html>
```