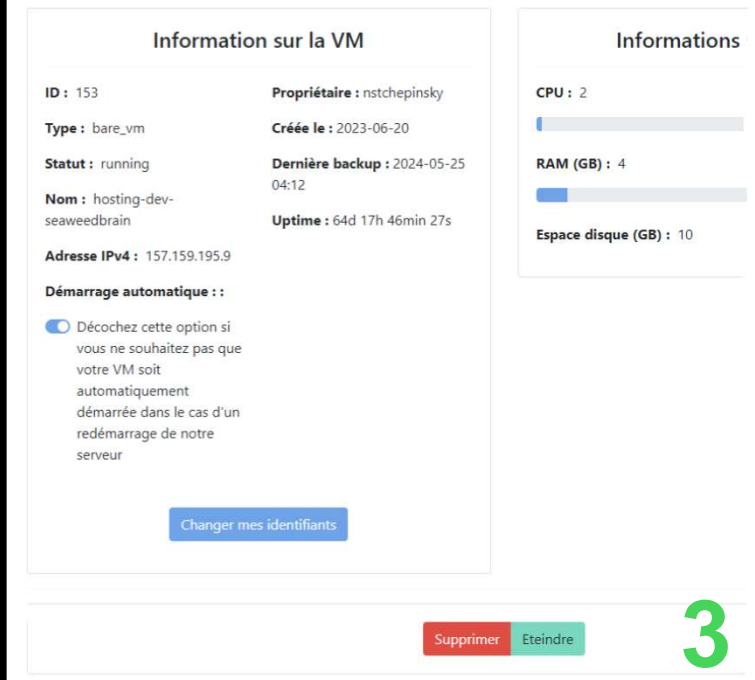
Formation

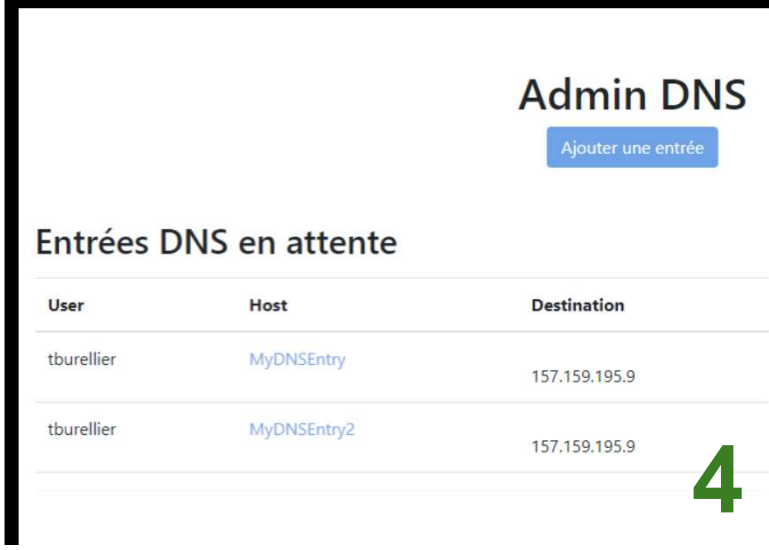
Par ValT et Orchidium

HOSTING

Que propose hosting ?

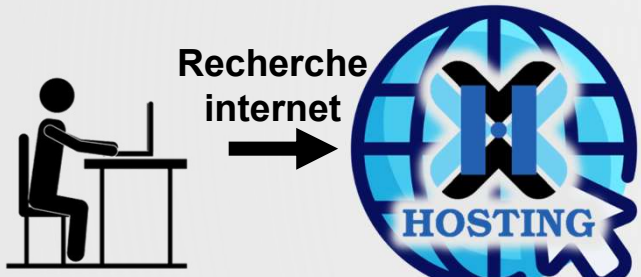
1. Authentification
2. Création de VMs
3. Manager la Vm
4. Créer une entrée DNS



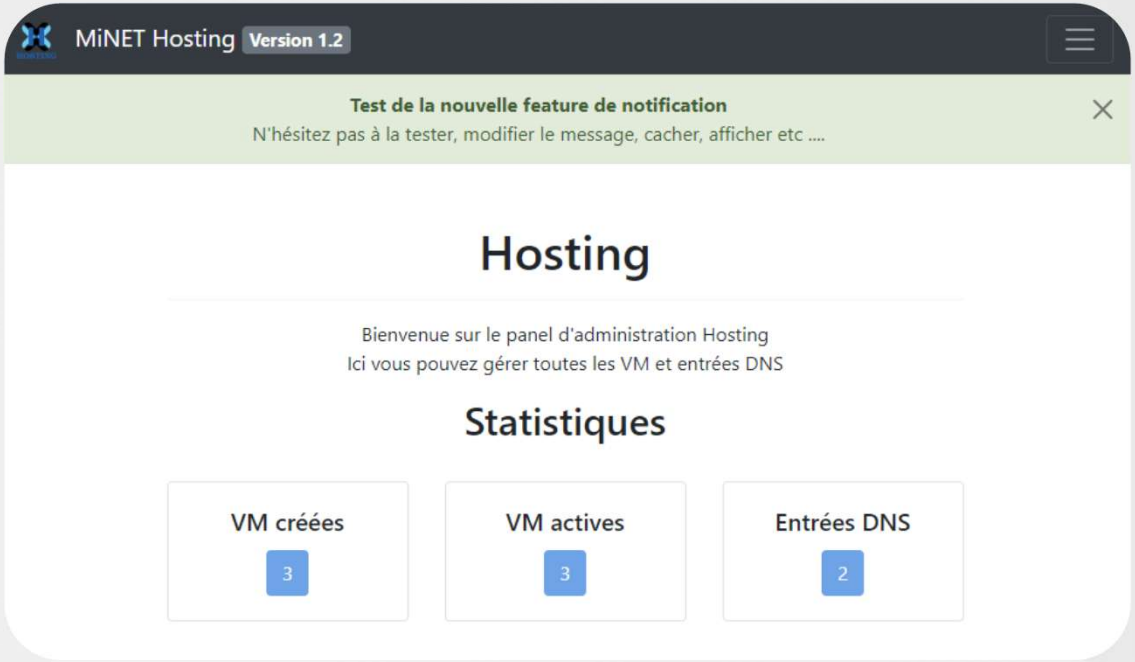


C'est quoi le fonctionnement ?

Site
Hosting



ANGULAR

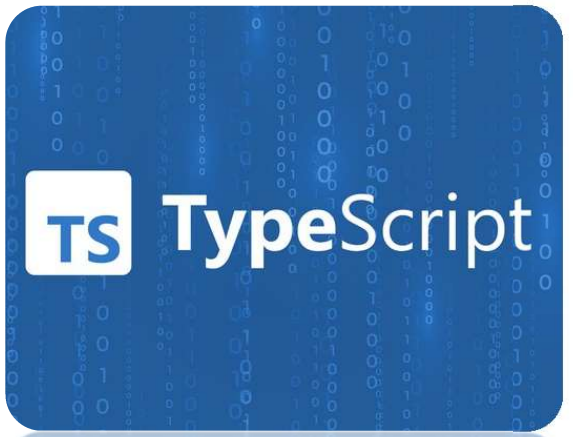


On commence en se connectant au site hosting. C'est le frontend, la partie avec lequel interagit le client, réalisé en Angular

Angular c'est un framework web

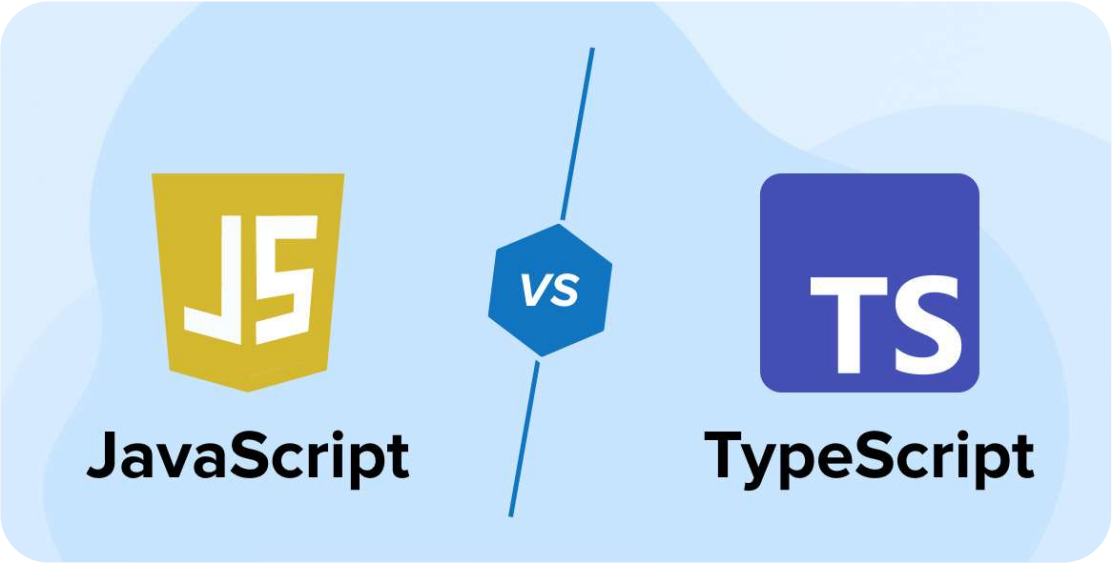
Un framework possède pleins d'outils préconçus pour faire rapidement des gros sites web

Pas besoin de réinventer la roue



ANGULAR

Ça se code en HTML, CSS et en Typescript



Aucun typage : ne te diras pas si tu fais n'importe quoi

Aucune structure : difficile de s'y retrouver

Aucune optimisation : ton code sera tel que tu l'as écrit dans ton site

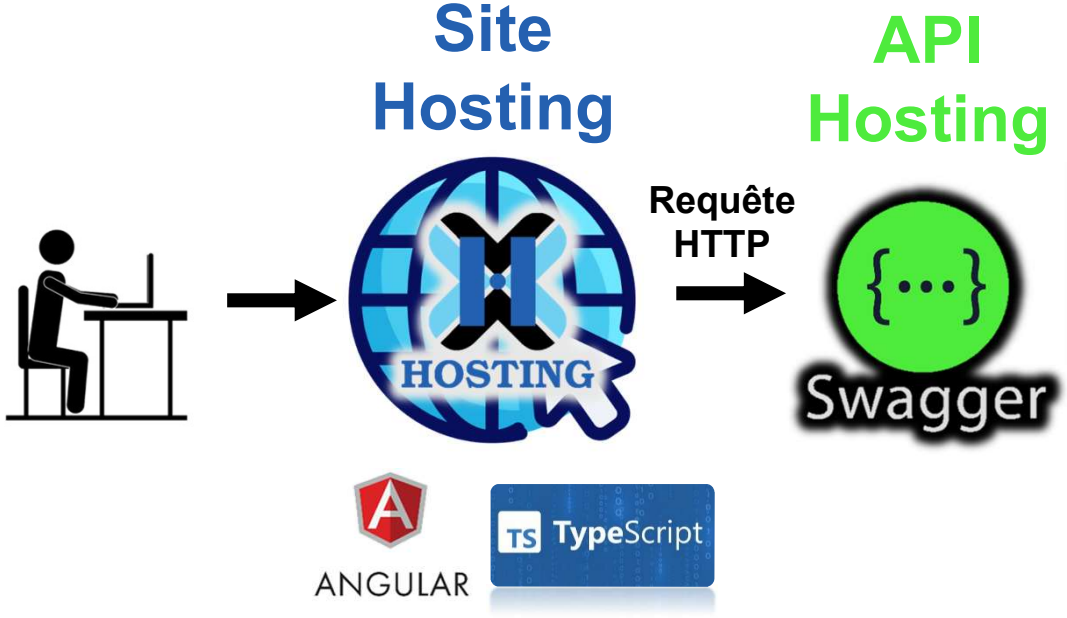
Typage strict : t'indique si y'a des erreurs

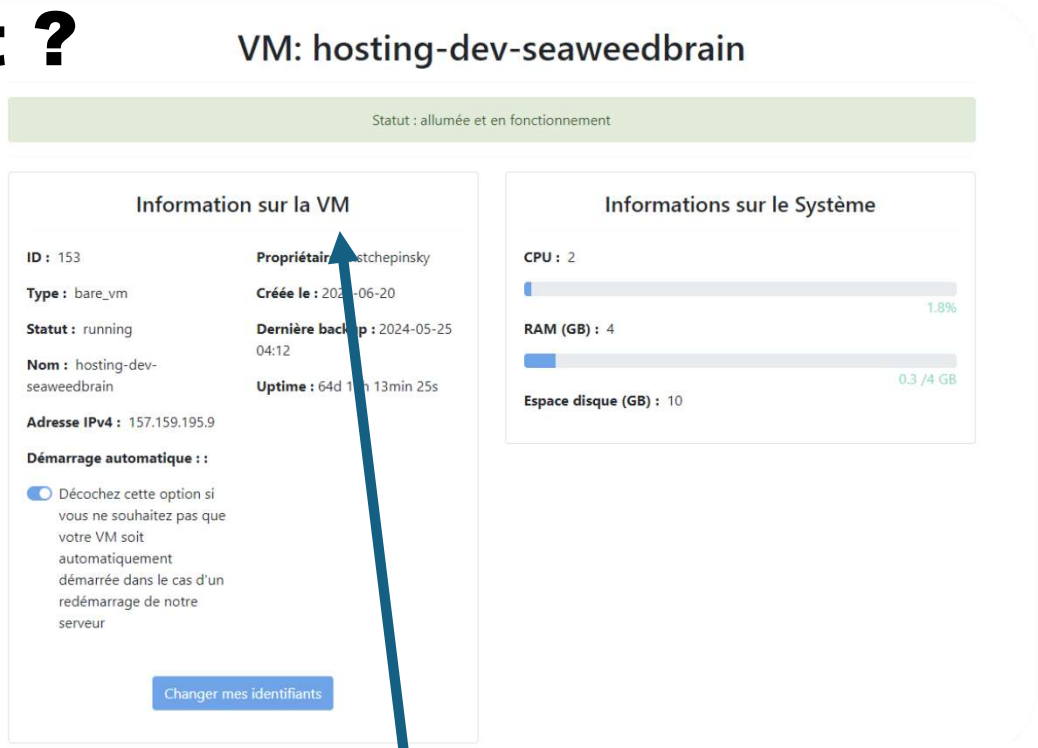
Gère la programmation orientée objet : permet de créer des classes et interfaces

Se compile de manière optimisée en javascript

Privilégié par tous les gros frameworks

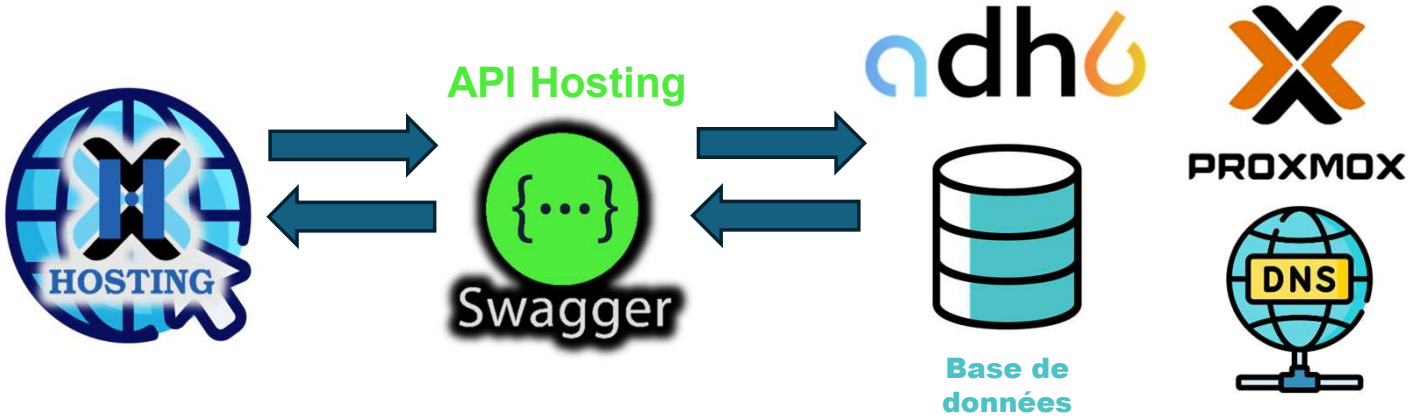
C'est quoi le fonctionnement ?





Quand j'affiche du contenu, genre les infos de cette vm, ou que je clique sur un bouton, ça exécute des fonctions Typescript qui exécute des requêtes HTTP vers l'API d'Hosting

Une API permet à n'importe quelle application A de communiquer avec une application B via des lignes de commandes



Dans notre cas, c'est le frontend Hosting qui veut accéder aux VMs proxmox, au DNS MiNET, ou aux adhérents adh6

En gros, une API, c'est le dictionnaire pour parler à une application

User

Allez sur api-hosting.minet.net/2.0/ui

- GET `/account_state/{username}` get the freeze state of a user account
- GET `/expired` get the list of users with expired cotisation
- GET `/ips` get the list of a user ip addresses
- GET `/max_account_ressources` max ressources a single account can have
- GET `/notification` Try to fetch a notification to display for the user
- PATCH `/notification` Modify the notification (for admin only)

DNS

- GET `/dns` get all user's dns entries. For admin, it returns all dns entries
- POST `/dns` create a dns entry
- POST `/dns/validation` validate a dns entry by id
- DELETE `/dns/{dnsid}` delete dns entry by id in database, also delete the dns entry in the proxmox server if it exists (validated)
- GET `/dns/{dnsid}` get a dns entry by id

VM

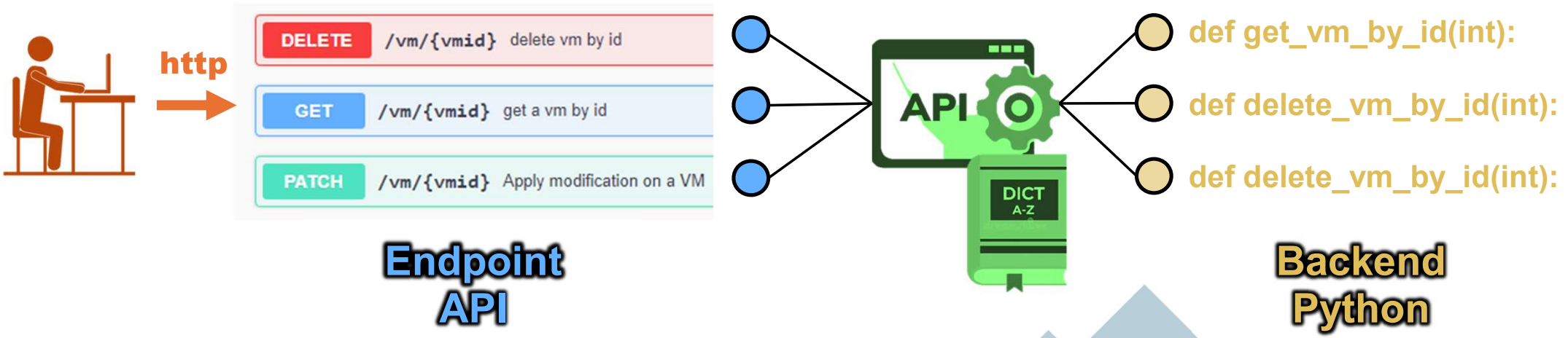
Voici l'ensemble des endpoints HTTP qui ont été codés pour hosting

- GET `/history/{vmid}` get the ip history of a vm

Un EndPoint, c'est une action que l'API nous permet de réaliser en faisant une requête http

(C'est une définition dans le dictionnaire pour dialoguer avec l'application)

Et derrière chaque EndPoint, il y a une fonction python dans le backend

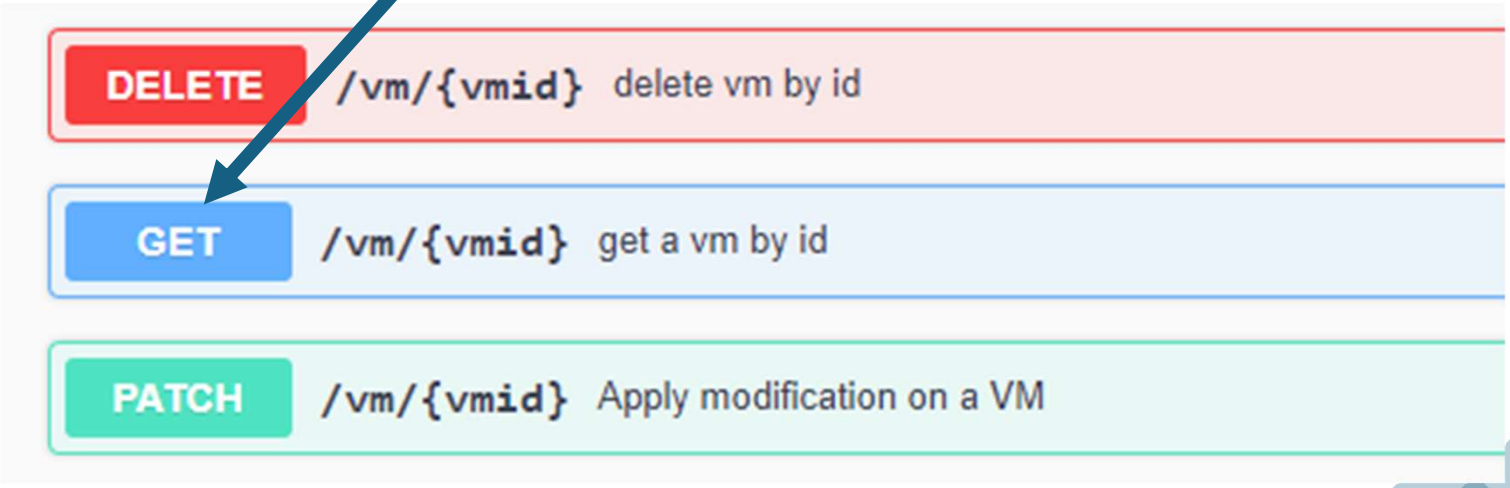


Par exemple, une requête http GET en typescript ça ressemble à ça :

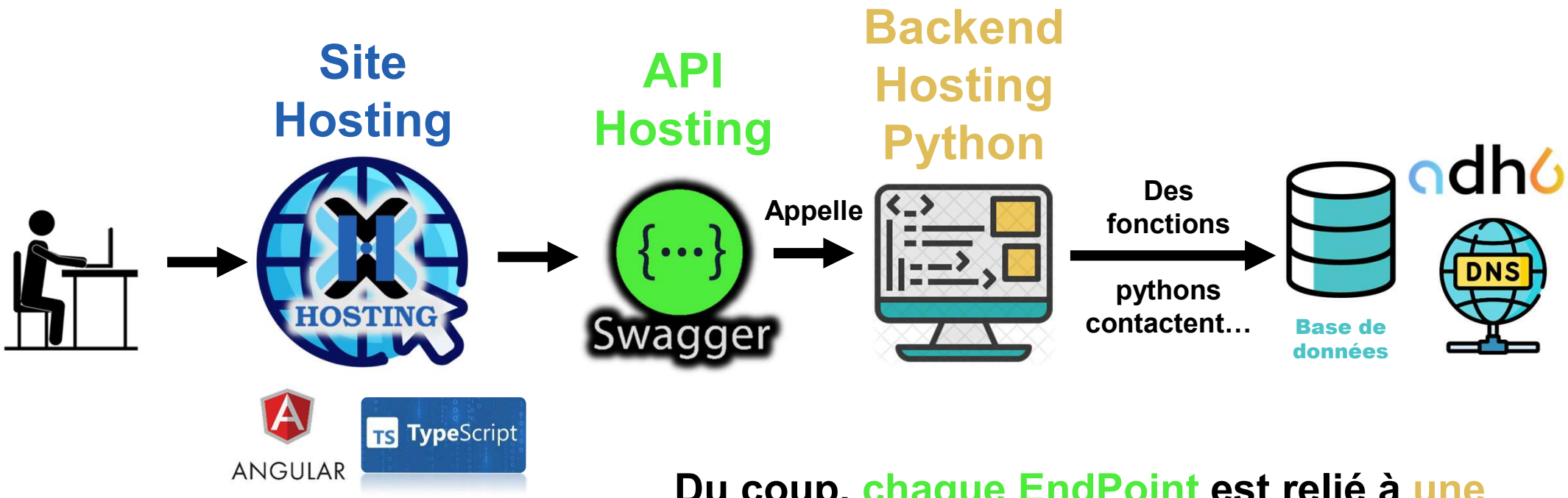
```
this.http.get(« https://api-hosting.minet.net/vm/146 »);
```

(récupère les infos de la vm numéro 146 sur proxmox)

Note : On pourrait faire ça dans n'importe quel langage, depuis n'importe quelle machine
Une API, c'est un dictionnaire qui fonctionne dans toutes les langues

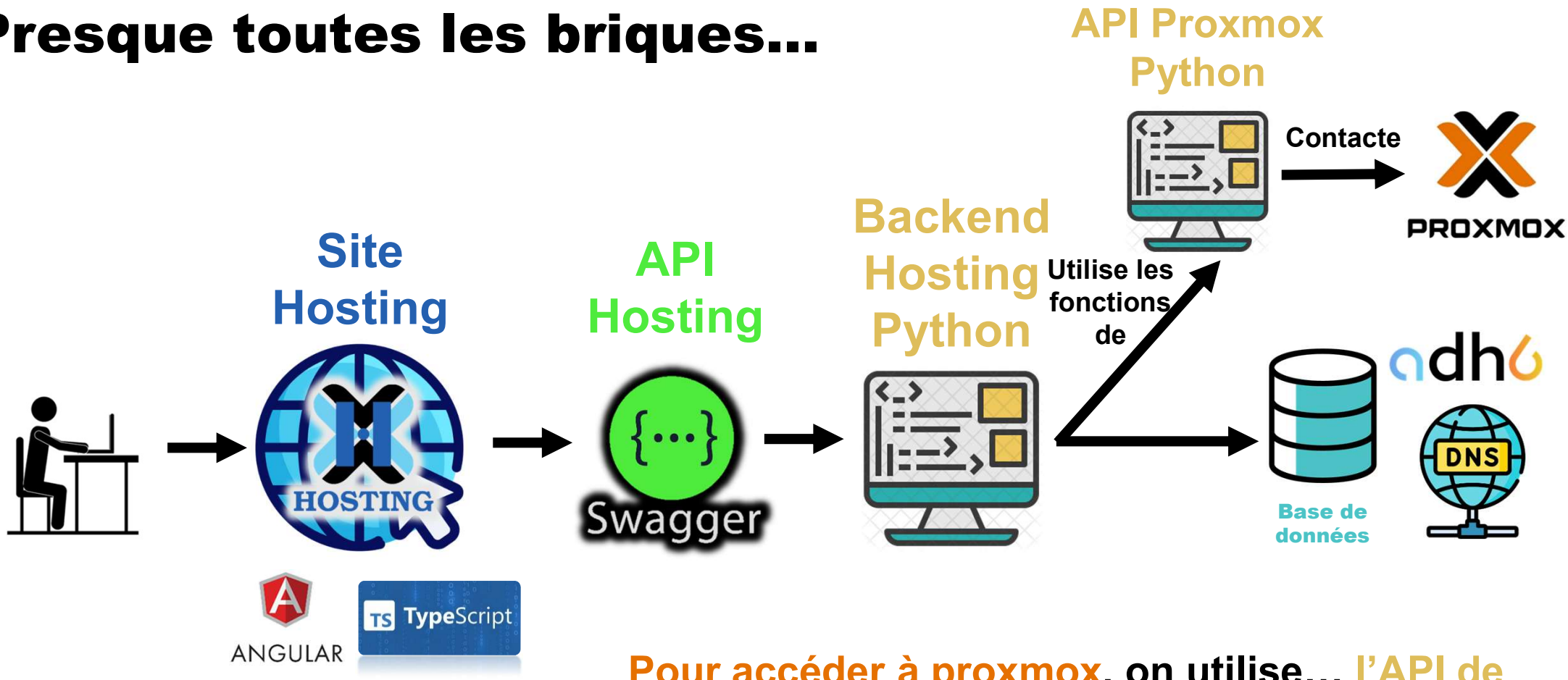


Maintenant on a toutes les briques !

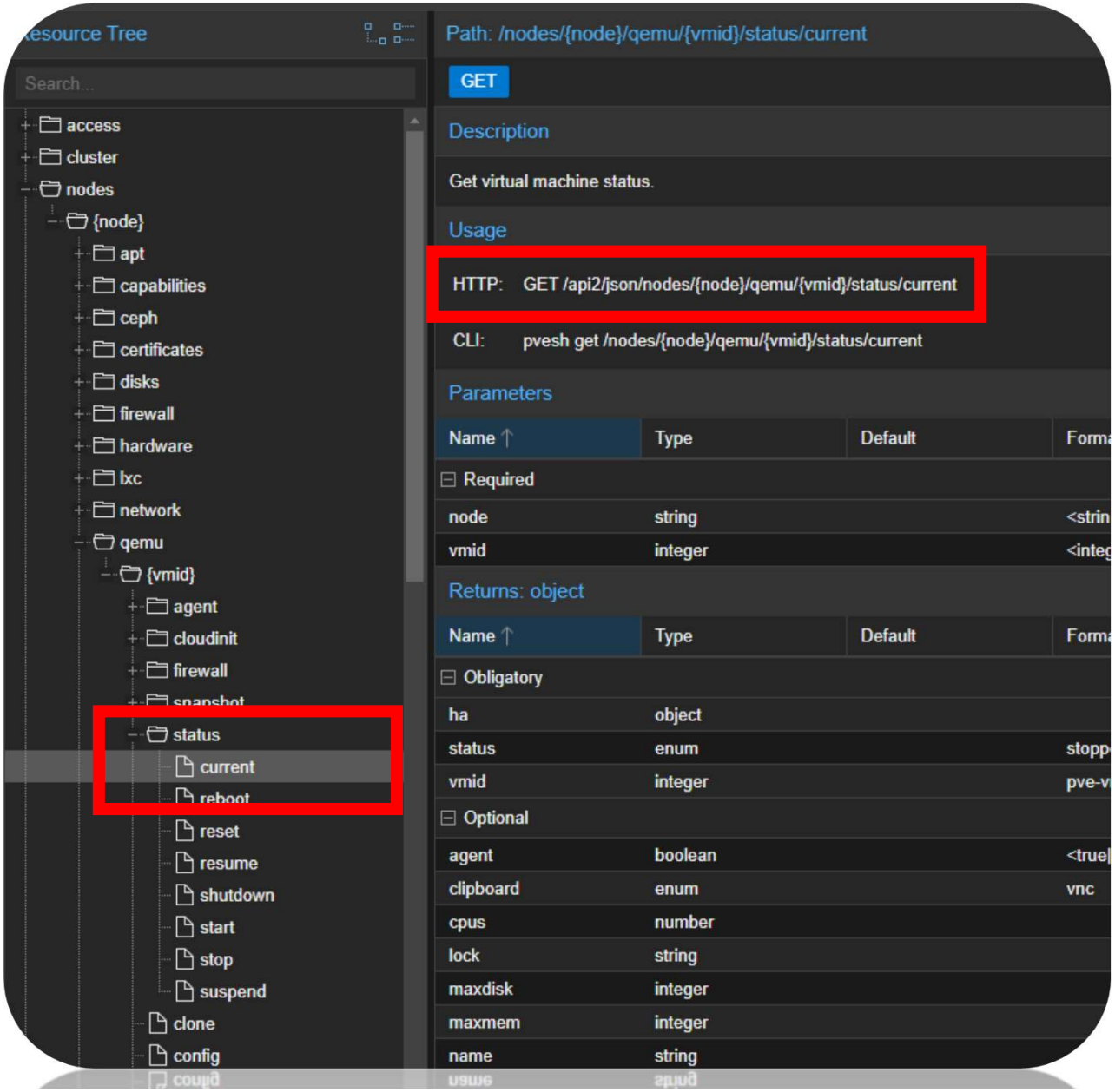


Du coup, chaque EndPoint est relié à une fonction Python dans le backend d'hosting. Et ensuite on utilise tout un tas de librairies Python pour contacter le DNS, la BDD ou adh6

Presque toutes les briques…



Pour accéder à proxmox, on utilise… l'API de proxmox. C'est aussi une librairie python, avec des fonctions déjà toutes faites pour obtenir toutes les infos dispos sur Proxmox



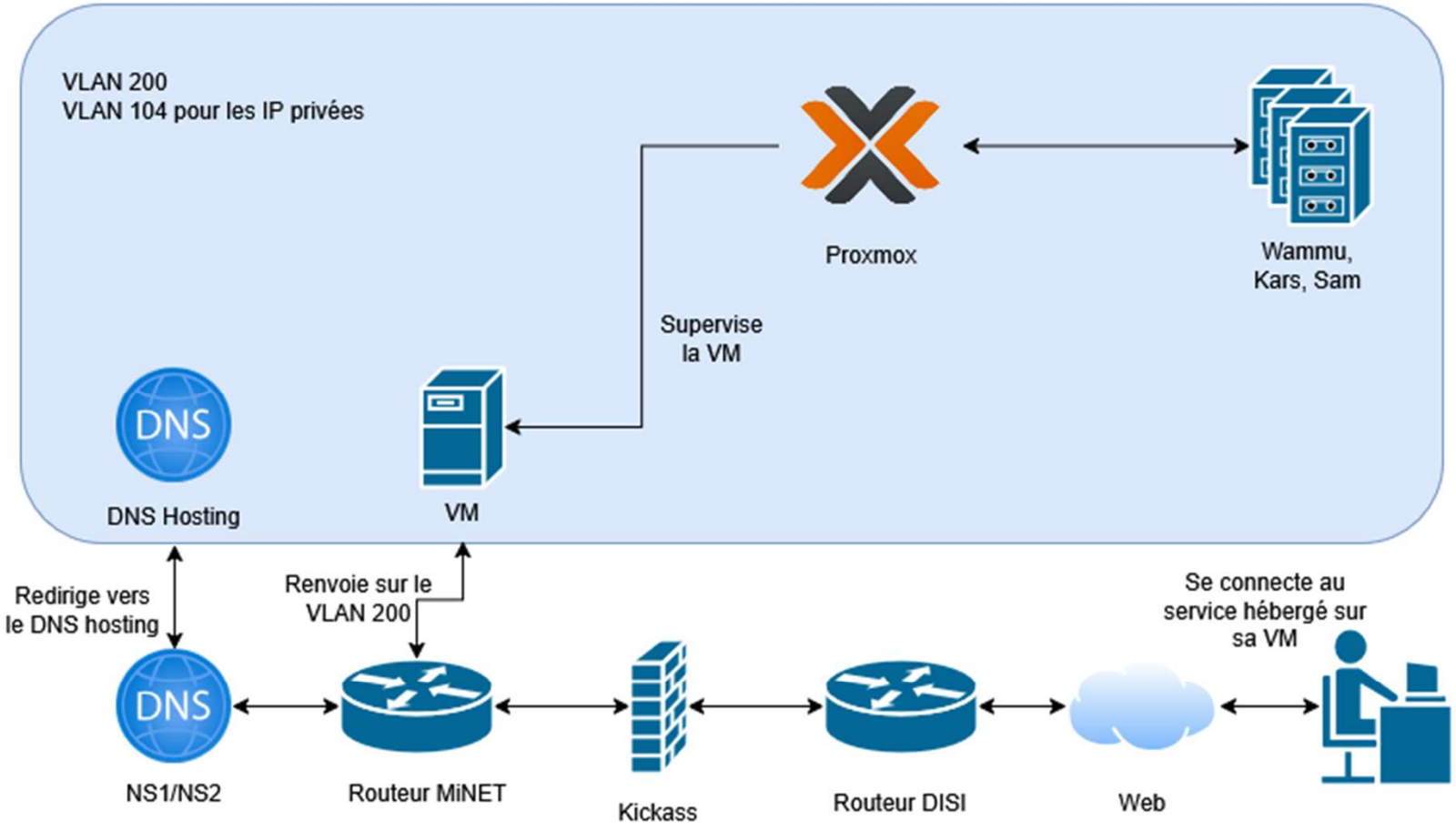
Allez sur ce site :

pve.proxmox.com/pve-docs/api-viewer

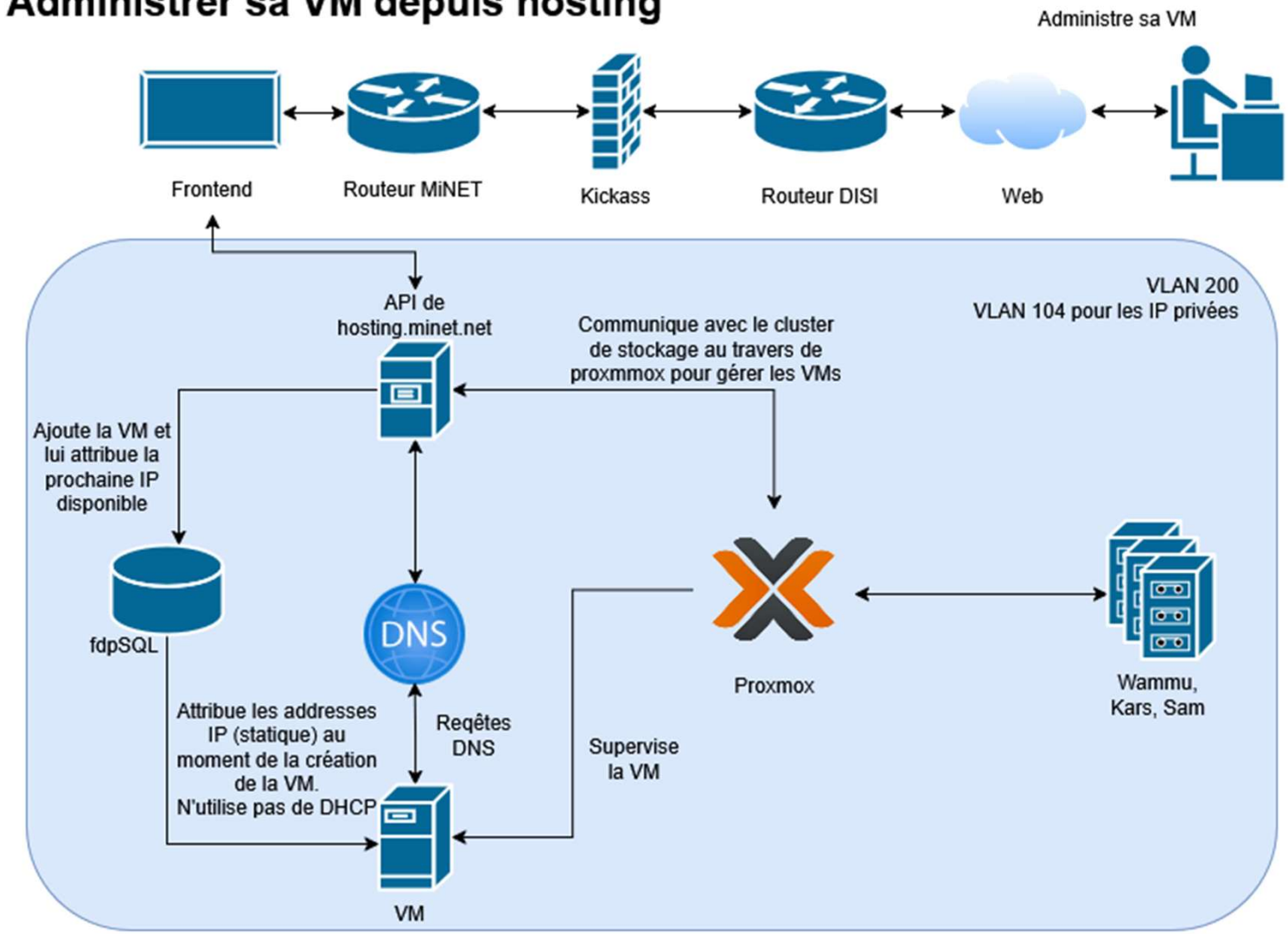
Contient tous les EndPoints de l'API proxmox

Par exemple celui-là c'est pour obtenir les infos de la vm

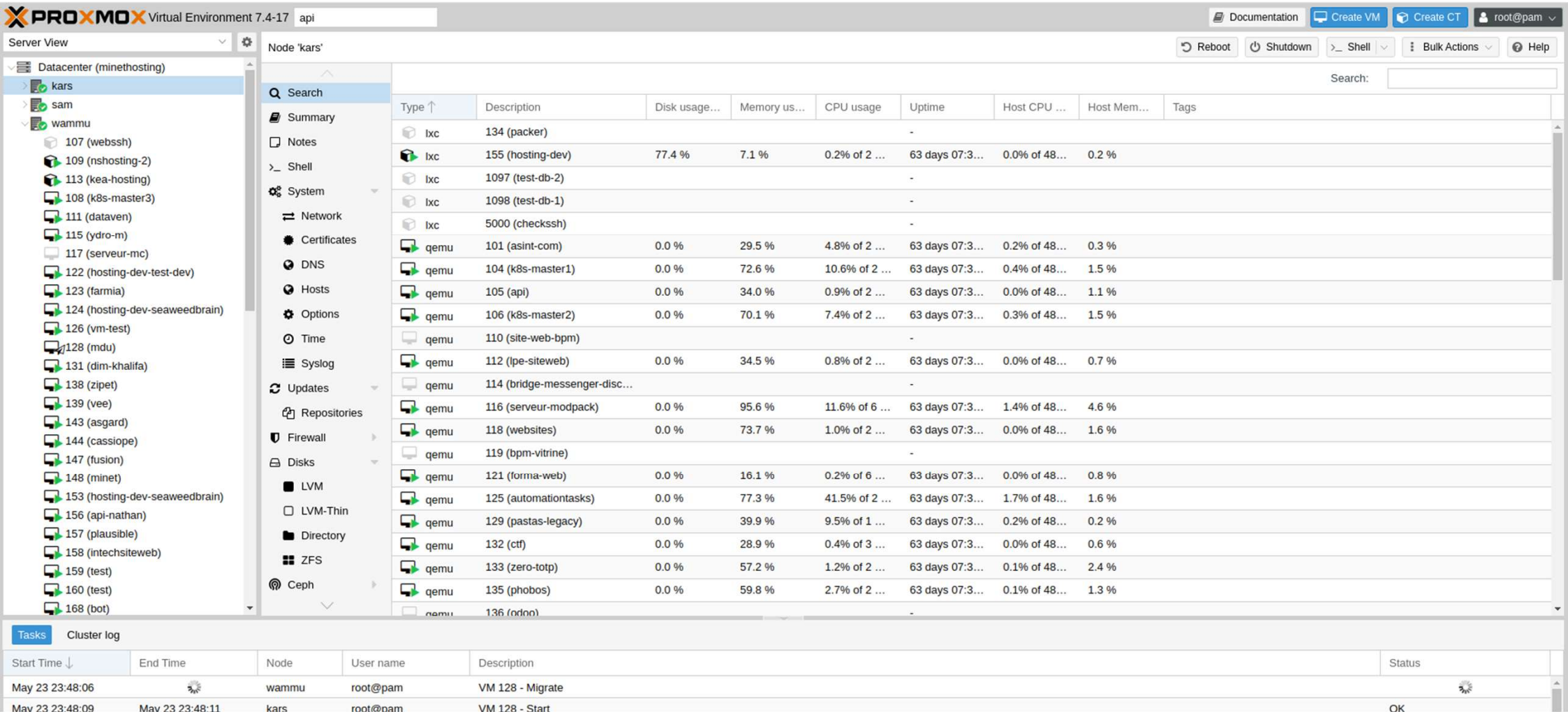
Accéder au service hébergé sur sa VM



Administrer sa VM depuis hosting



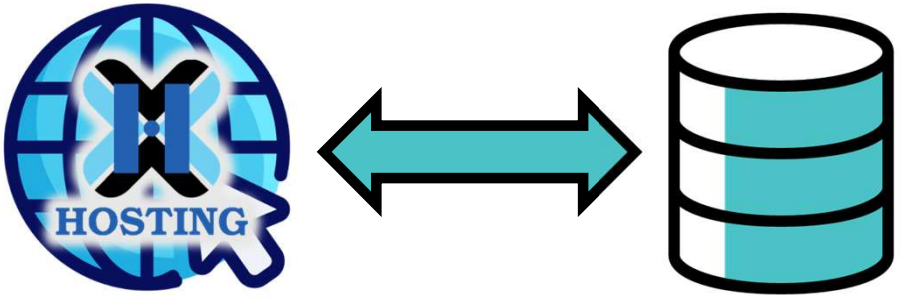
Et du coup oui ! Derrière hosting, il y a juste un proxmox avec toutes les VMs des adhérents. Comme sur la Dev ou la Prod



Il faut savoir qu'il y a TROIS Hosting

Hosting.minet.net : production

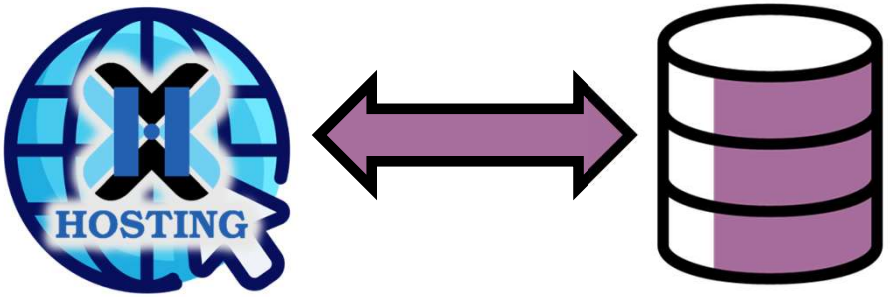
C'est le site qu'on connait



Base de données de production : toutes les VMs des adhérents

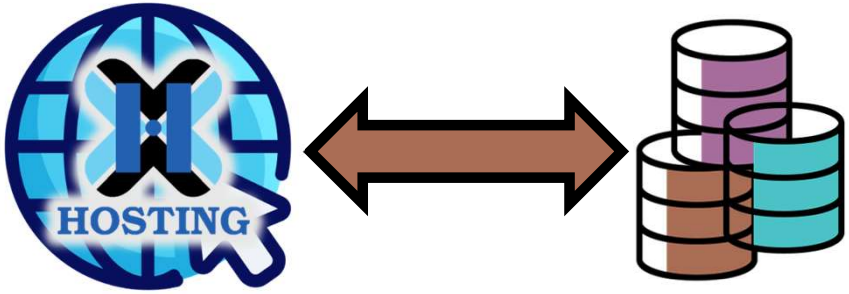
Hosting-dev.minet.net : développement

C'est un site qui est aussi accessible en permanence, mais qui contient du code pas encore push sur main sur gitlab



Base de données de dev : c'est sur le même cluster, mais pas la même bdd

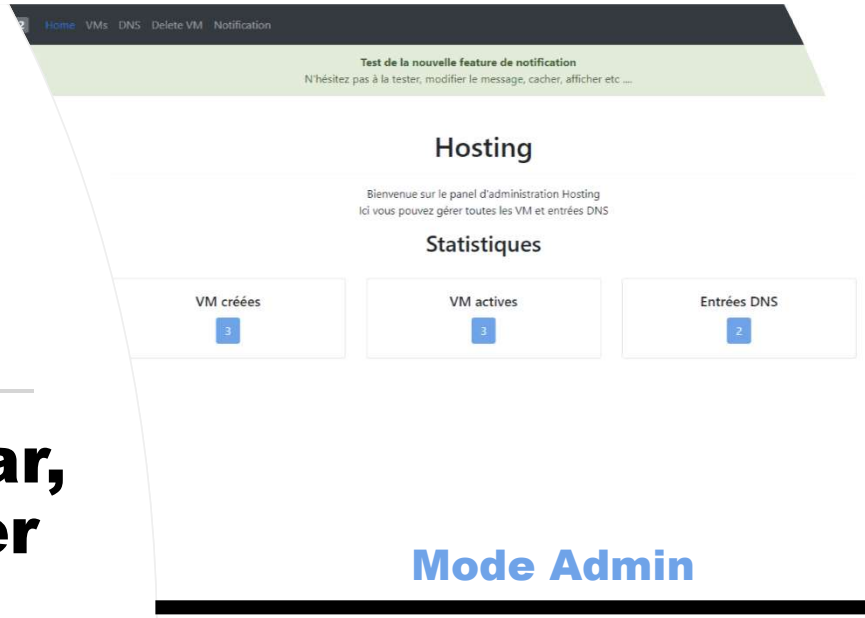
Hosting-local.minet.net : vous quand vous lancez le projet sur votre pc



On peut choisir quelle base utiliser en local. Et y'a une base dédiée au test de code

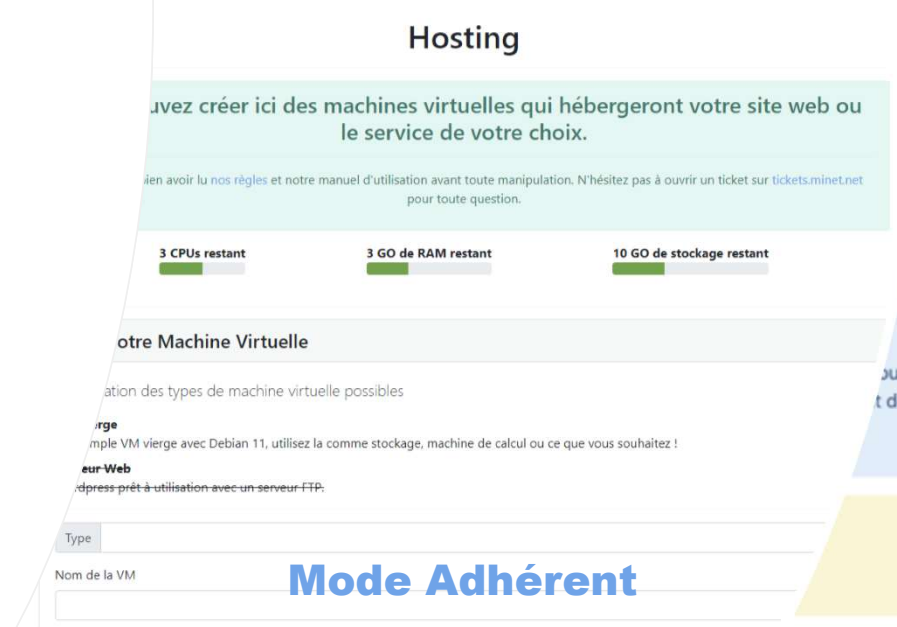
Grâce à Angular, on peut afficher des trucs totalement différents en fonction de qui est connecté

Y’a en gros 4 modes :



Mode Admin

Mode Pas Connecté



Mode Adhérent

Mode Cotisation expiré

Freeze State

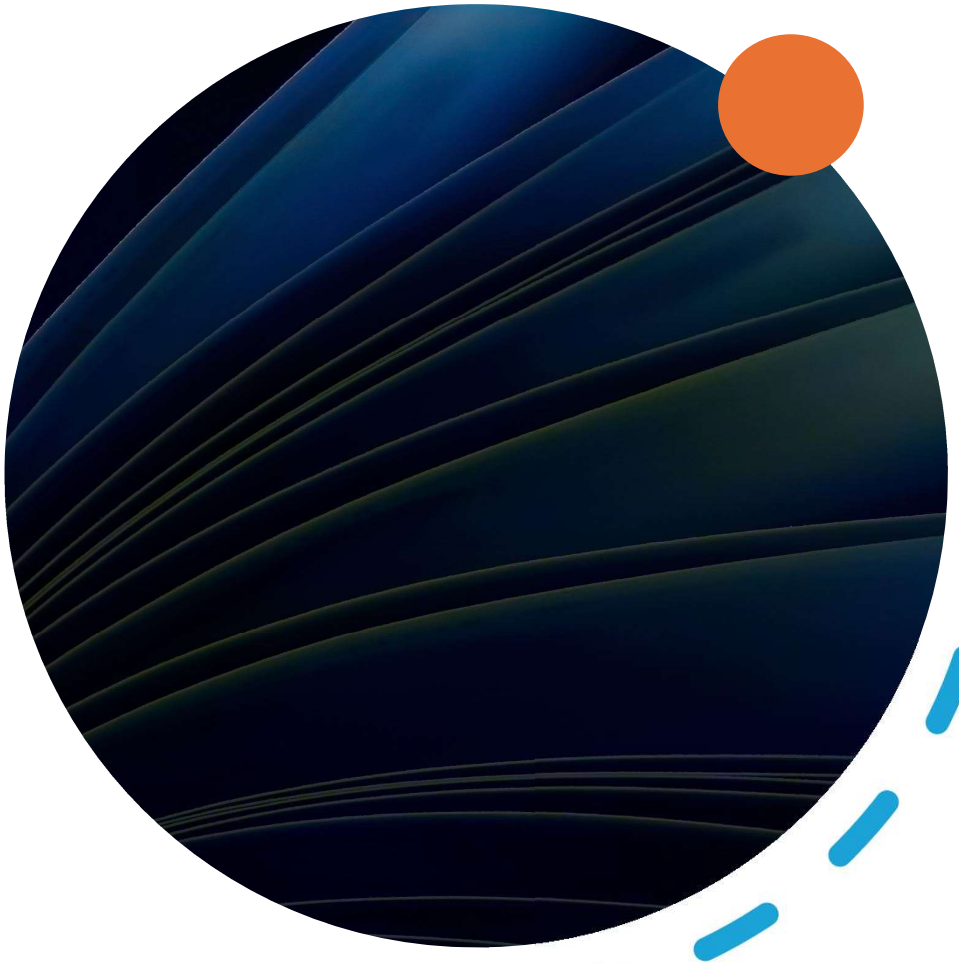
Un adhérent a un freeze state de 1.0 quand il cotise

Dès que sa cotisation expire, il passe à un freeze state de 2.1

Le freeze state d'un adhérent expiré va changer toutes les semaines et il recevra un mail : 2.1 → 2.2 → 2.3 → 2.4 → 3.1 → 3.2 → 3.3 → 3.4 → 4.1 → …

A partir d'un freeze state de 4, on peut supprimer les VMs de l'adhérent

Les freezes states sont gérés sur le jenkins de minet (faut trouver l'IP sur le wiki)



Passons à du concret, on va voir comment bien débuter sur hosting

Cloner le repo sur Github :

Le projet est mirroré sur Github pour être accessible à tous :

https://github.com/minet/hostingproxmox

Ensuite faut installer tout un tas de trucs :

$ sudo apt install npm

$ npm install -g @angular/cli

https://doc.ubuntu-fr.org/nodejs
(depuis un PPA)

Setup le projet

Hosting est divisé en 2 dossiers : le frontend avec Angular, le backend avec Flask

Installer les dépendances angular utilisée dans le projet :

```
$ cd frontend && npm install
```

Créer un environnement virtuel python avec les dépendances utilisées par le projet :

```
$ cd backend &&  python3 -m venv venv && source venv/bin/activate
&&  pip3 install -r requirements.txt
```

Setup le projet

Pour finir il faut créer un fichier nommé « .env » dans la racine du projet. Il sert à stocker les variables d'environnement pour se connecter aux différents serveurs d'Hosting

Pour des raisons de sécurité, je ne peux pas les donner ici, et c'est pas grave puisqu'on peut faire cette formation sans. Mais il faudra les demander si vous voulez vraiment travailler sur hosting

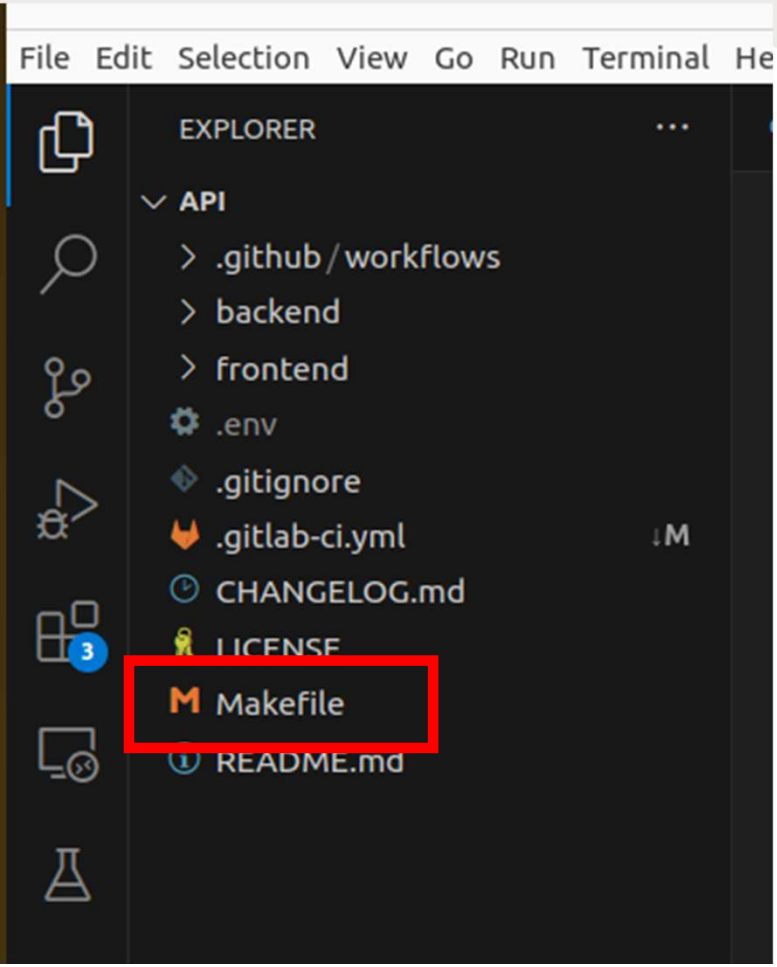
Copier-coller ça dans le .env :

```
export KEYRING_DNS_SECRET=<KEYRING_DNS_SECRET>
export PROXMOX_API_KEY_NAME=<PROXMOX_API_KEY_NAME>
export PROXMOX_API_KEY=<PROXMOX_API_KEY>
export PROXMOX_BACK_DB=<PROXMOX_BACK_DB>
export ADH6_API_KEY=<ADH6_API_KEY>
export PROXMOX_BACK_DB_DEV=<PROXMOX_BACK_DB_DEV>
export ENVIRONMENT= « DEV »
```

Le makefile

C’est un fichier qui permet d’exécuter pleins de commandes rapidement

Notamment ça sert pour faire toutes les commandes pour lancer le projet d’un coup



Makefile

```
.DEFAULT_GOAL := run

install_server:
    rm -rf backend/venv
    python3 -m venv backend/venv
    backend/venv/bin/pip install -r backend/requirements.txt

install_frontend:
    cd frontend && npm install

install_all: install_server install_frontend

run_server:
    . ./.env && . backend/venv/bin/activate && cd backend && python3 -m proxmox_api

run_frontend:
    cd frontend && ng serve --host=127.0.0.1 --disable-host-check

run:
    echo "Starting server ..."
    make run_server & \
    echo "Starting frontend on http://hosting-local.minet.net:4200/ ..." && \
    make run_frontend

clean:
    rm -rf pycache
```

On va PAS pouvoir utiliser le backend

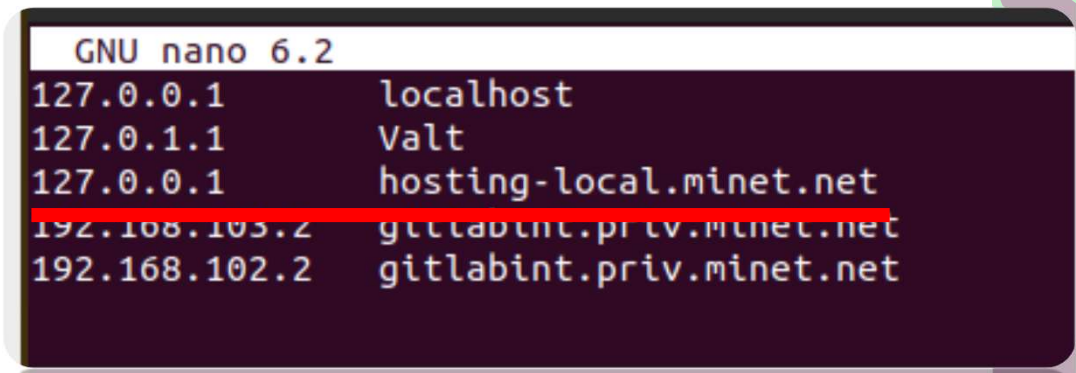
Du coup on va faire $ make run_frontend

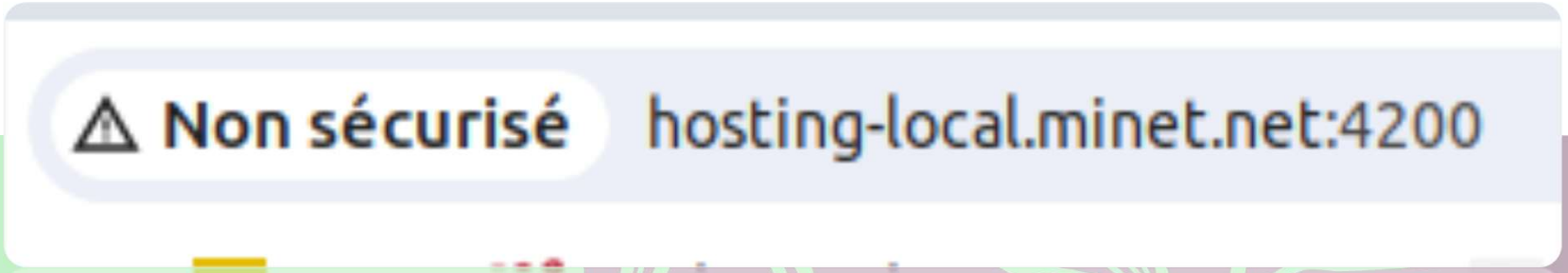
Normalement on pourrait utiliser make pour lancer les 2

Faut modifier le /etc/hosts pour ajouter :

127.0.0.1 hosting-local.minet.net

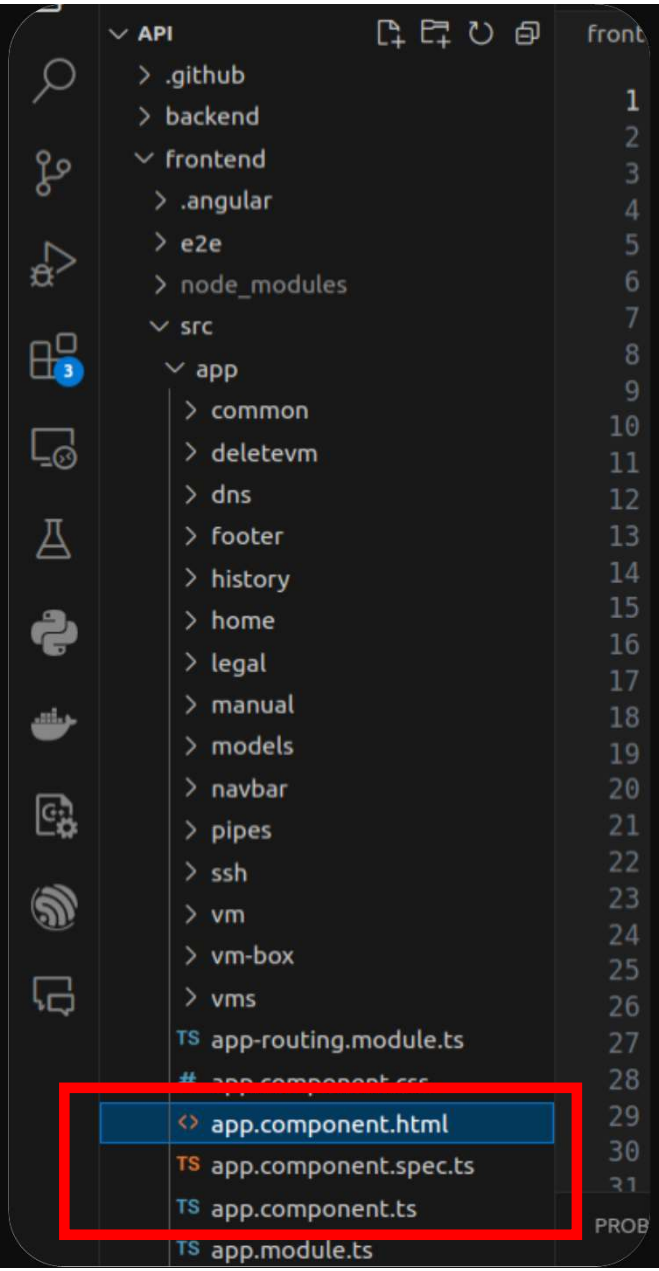
Puis faut aller sur hosting-local.minet.net:4200
(127.0.0.1:4200 fonctionne pas)





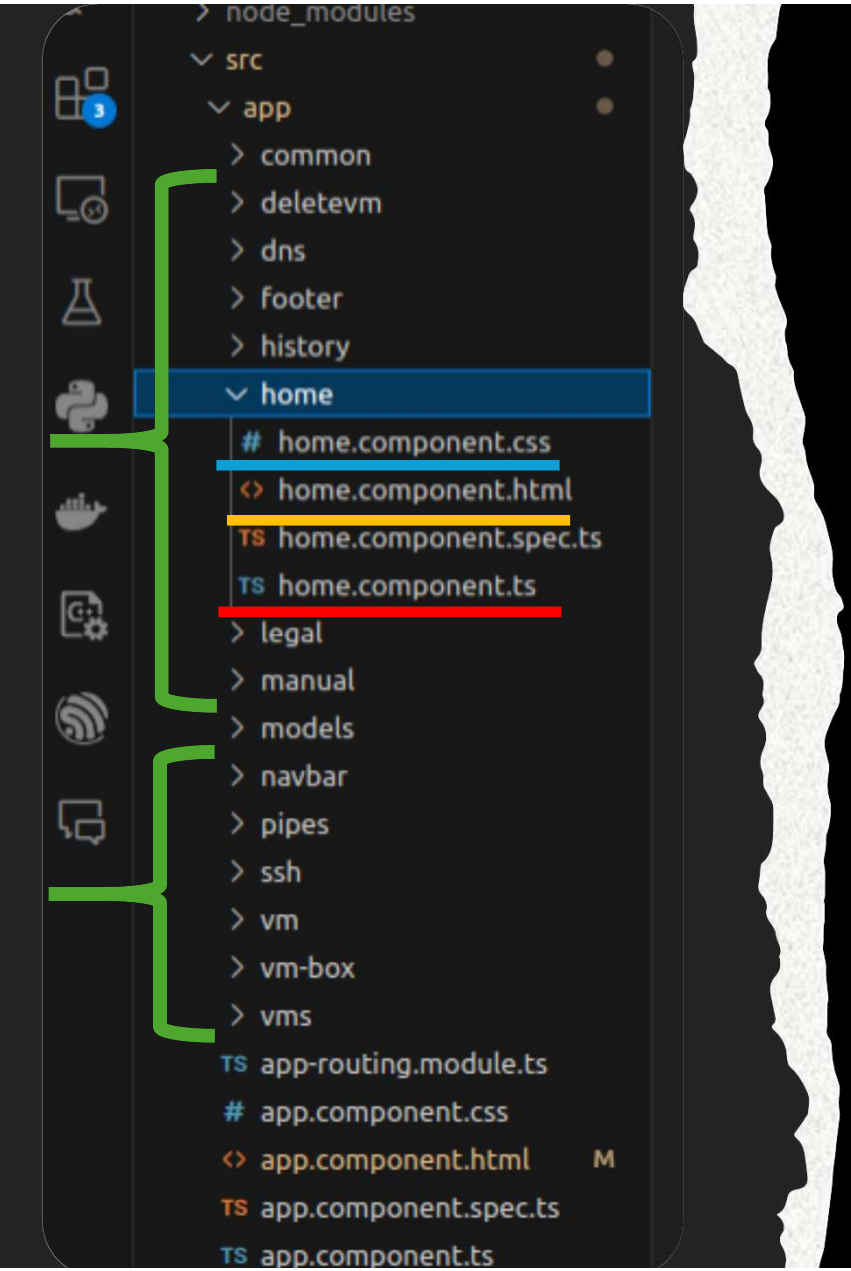
Là vous avez accès à la page d'acceuil et vous pouvez vous connecter au cas de minet avec vos creds d'adhérent

(pas ldap, vous êtes pas admin hosting)



Pour comprendre comment tout ça fonctionne, commençons par revenir à la racine de l'app : app.component.html et app.component.ts

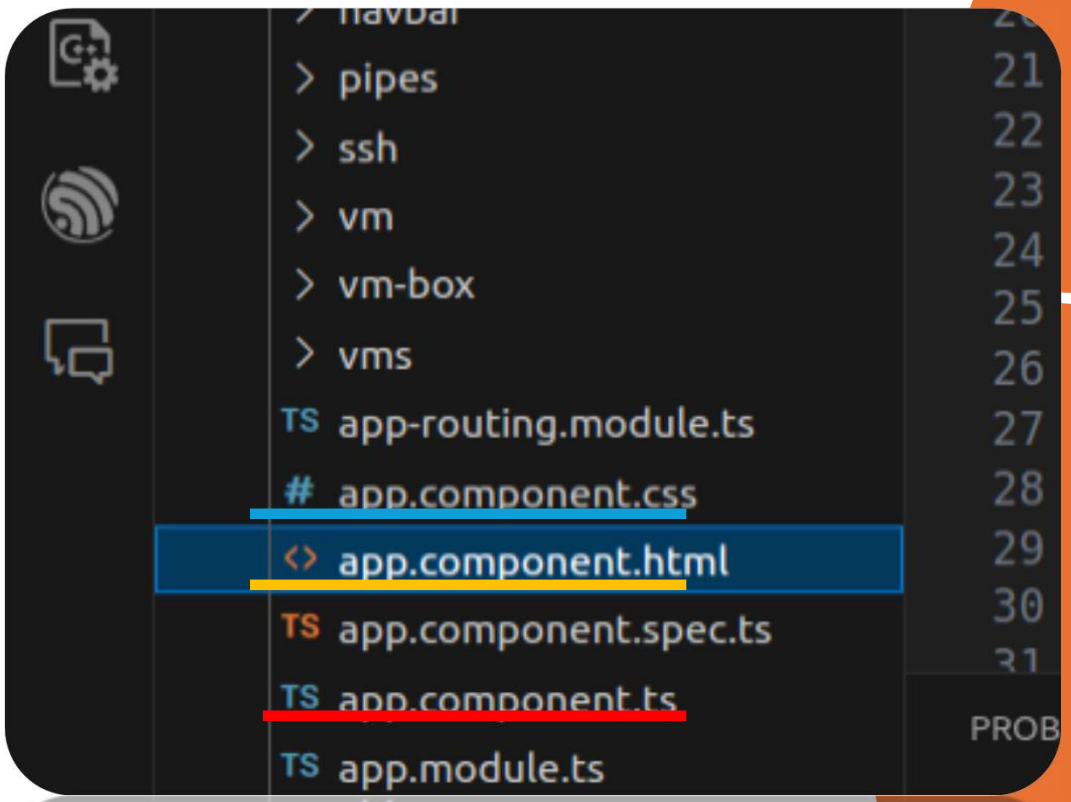
C'est là que démarre notre app



Faut savoir que Angular fonctionne avec un système de component. Un component c'est un fichier html, couplé à un fichier typescript et un fichier css. Les 3 fonctionnent ensembles et sont connectés

Et un component peut ensuite être inséré et réutilisé n'importe où

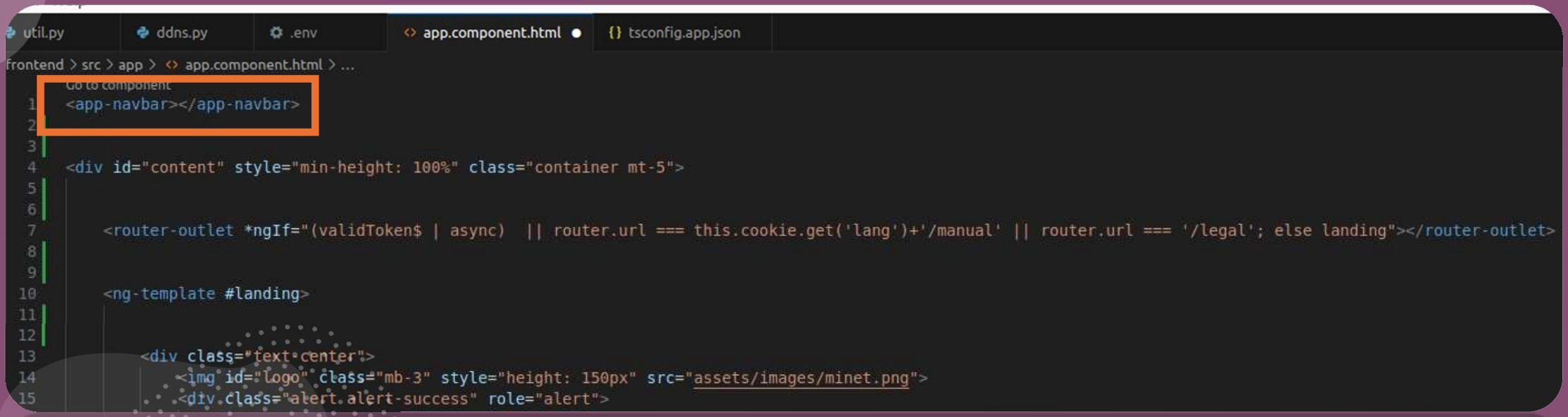
Par exemple, chacun des dossiers que vous voyez là sont des components



Et du coup, y'a un component au-dessus des autres, c'est le app.component. C'est un peu le component qui s'affiche par défaut et sur lequel va venir se greffer tout le reste

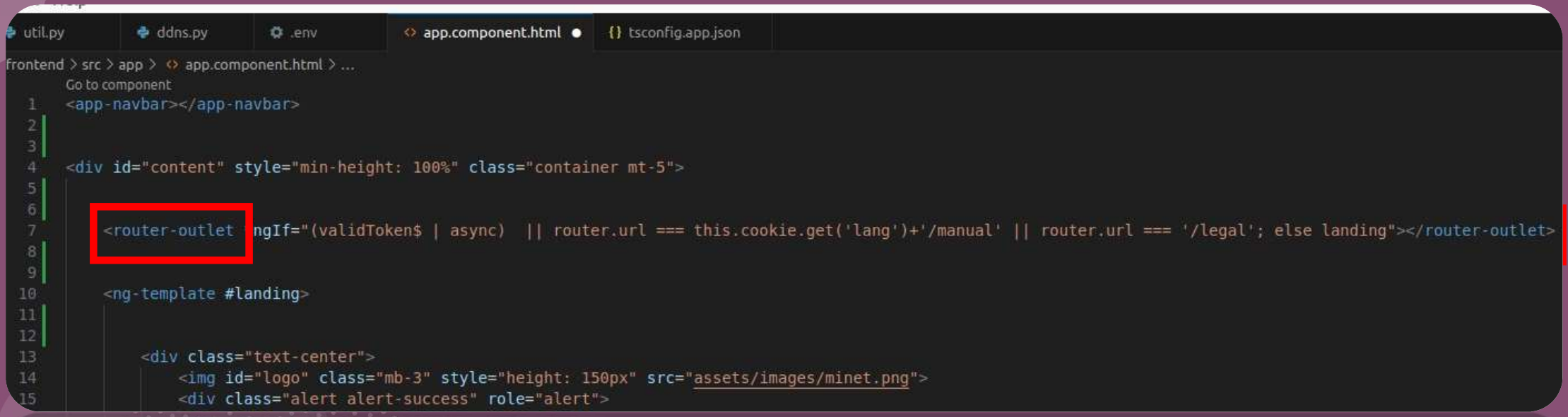
Tout en haut, on voit qu'on insère un app-navbar component

De même tout en bas, on a le footer-component. Vu que c'est le component de base, ça veut dire que peu importe où on sera dans l'application, on aura un navbar, et un footer. Ils sont définis dans les dossiers « navbar » et « footer »



Ensuite on a le "router-outlet". Ça permet d'afficher un component différent en fonction de l'url de votre site

Par exemple, si mon url c'est "/vms", je vais afficher le component vms. Et si mon url c'est "/dns", je vais afficher le component dns



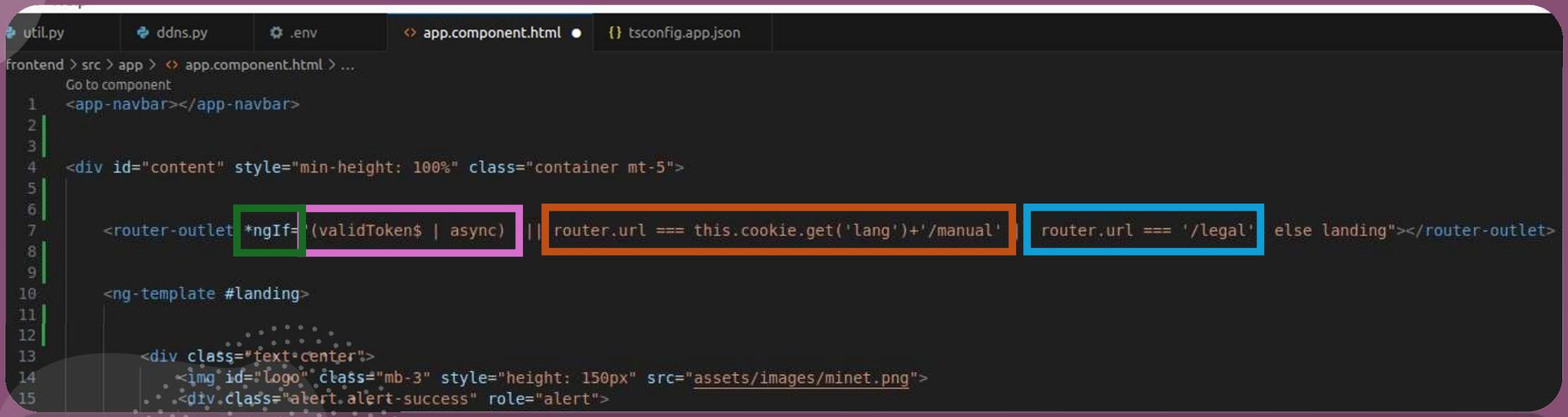
Ce qui va être affiché à la place du router-outlet, c'est défini dans le

app-routing.module

En gros, quand on est sur telle URL, on affiche tel component

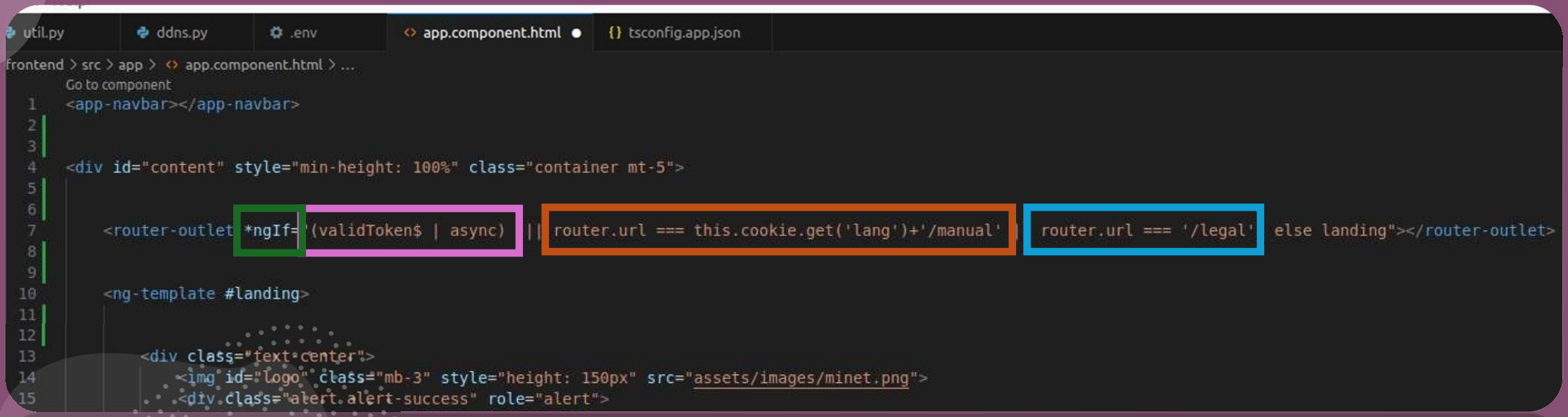
Pour finir, vous pouvez remarquer le *ngIf. Angular possède une feature trop bien : vous pouvez afficher du HTML à certaines conditions !

C'est très pratique, par exemple, pour afficher des boutons visibles que pour les admins. Ou que pour les gens authentifiés



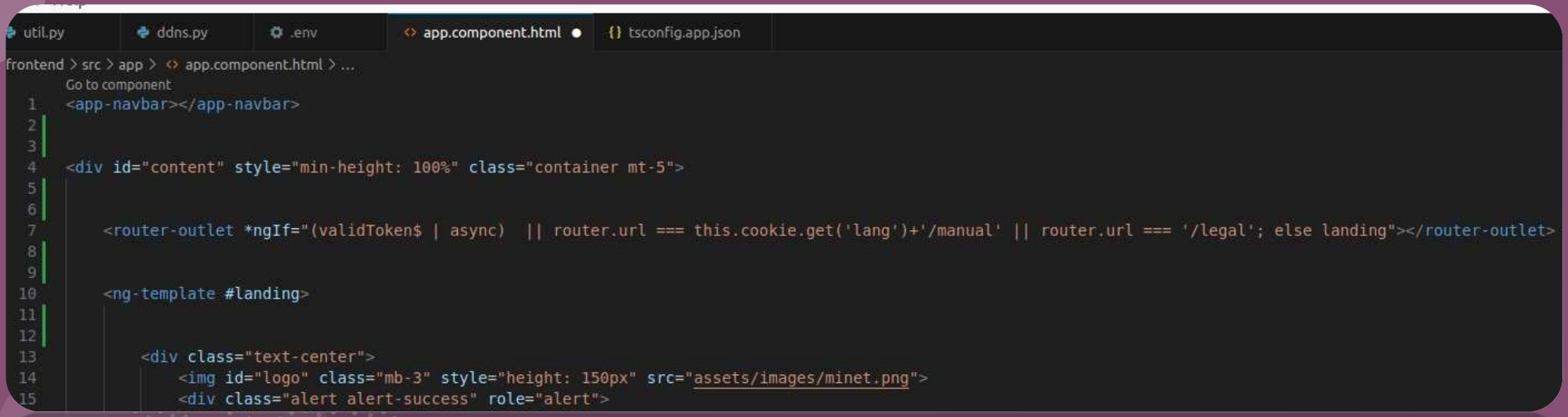
Par exemple ici : le router-outlet affiche des trucs QUE SI : vous êtes authentifié, OU vous êtes sur les routes "/legal" ou "/manual".

Parce que y'a pas besoin de s'authentifier pour voir ces routes.



Et si ces conditions ne sont pas validées, alors on affiche le ng-template "landing"

Ce truc est défini juste en dessous. Globalement ça correspond à la page d'accueil. Et du coup elle sera affichée que si le router-outlet ne s'affiche pas



Ok on va vraiment hostinger !

Votre mission si vous l'acceptez (ça parait dur… enfait c’est 5 lignes de code) :

Créer un nouveau component et le rendre accessible via le router (par flemme, on rentrera l'url à la main pour y accéder)

Créer une variable int, puis une fonction qui augmentent la valeur de cette variable dans le typescript. Faut qu'à chaque fois qu'on entre sur la page, la valeur de la variable soit reset à 3

Créer un bouton qui appelle la fonction. Puis afficher la variable dans le html

Aide

Créer un nouveau component

```
$ ng generate component nomDuComponent
```

Afficher une variable du typescript dans le HTML :

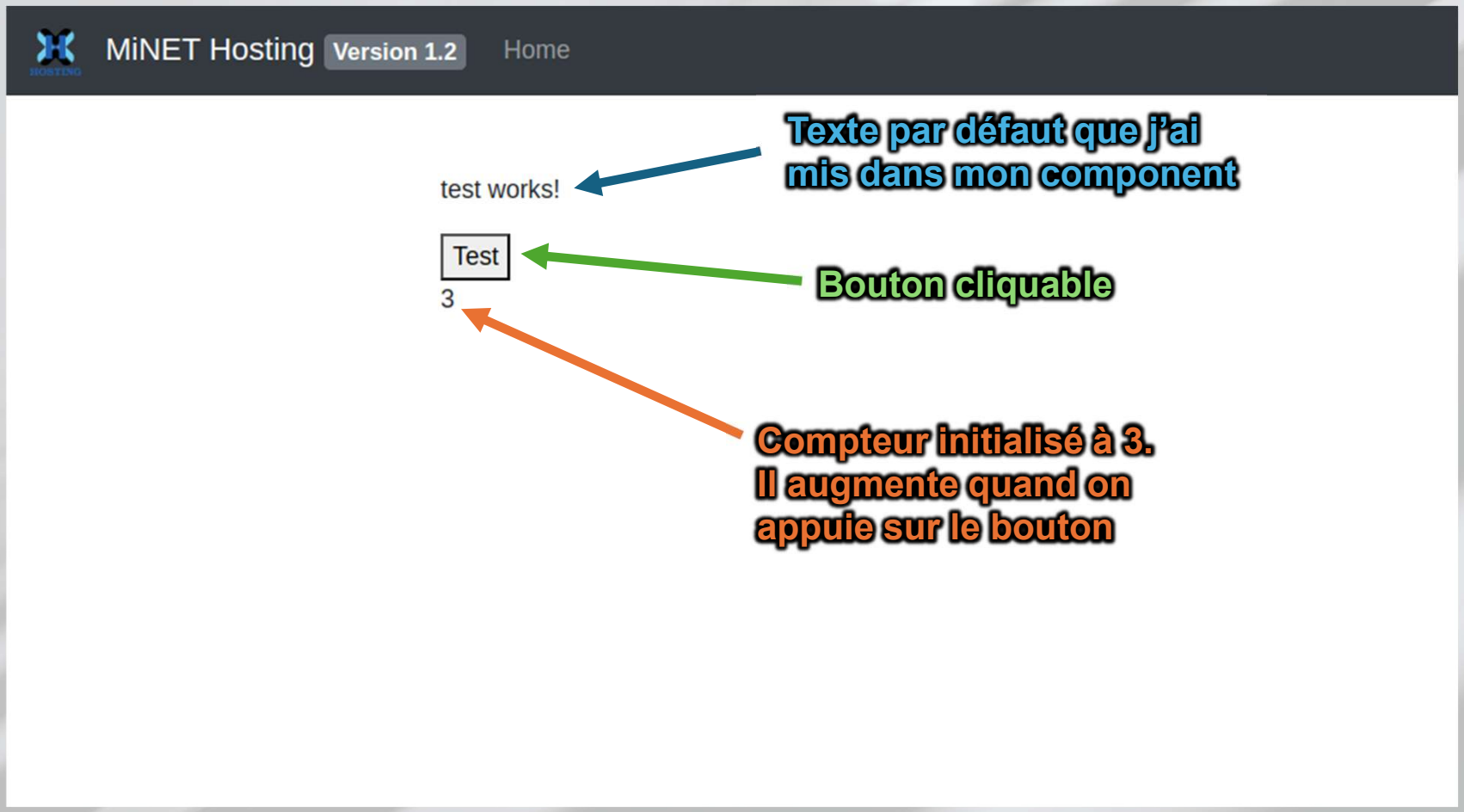
```
{{ variable }}
```

Associer une méthode à un bouton :

```
<button (onClick)="maMethode()">
```

ngOnInit : une méthode qui s'exécute automatiquement à chaque fois qu'on charge le component

WoW c'est beaaau



Pour ceux qui sont plus motivés

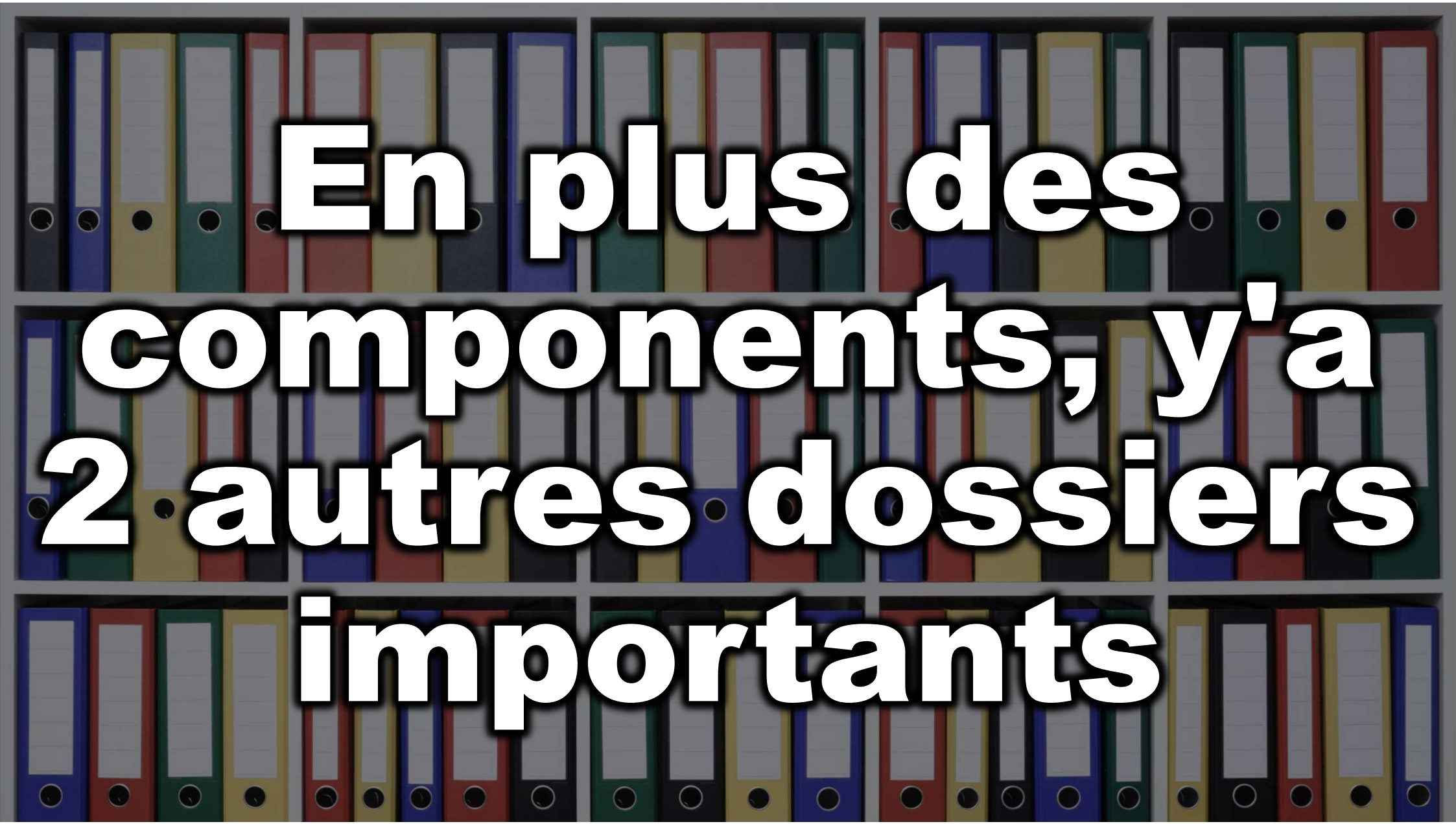
Faites 2 boutons : un pour like et un pour dislike

Du coup il faut aussi faire 2 compteurs pour like et dislike

On ne doit pas pouvoir like et dislike en même temps : si on en fait un ça annule l'autre

Lorsqu'on a like, le compteur s'affiche en vert plutôt que noir. Pareil mais en rouge pour le dislike.

On peut utiliser ngClass avec du CSS : <div [ngClass]="{ nomPropCSS: condition}">


En plus des components, y'a 2 autres dossiers importants

Les modèles :

En gros ça permet de définir des classes avec des propriétés

Genre là je définis l'objet « Vm »

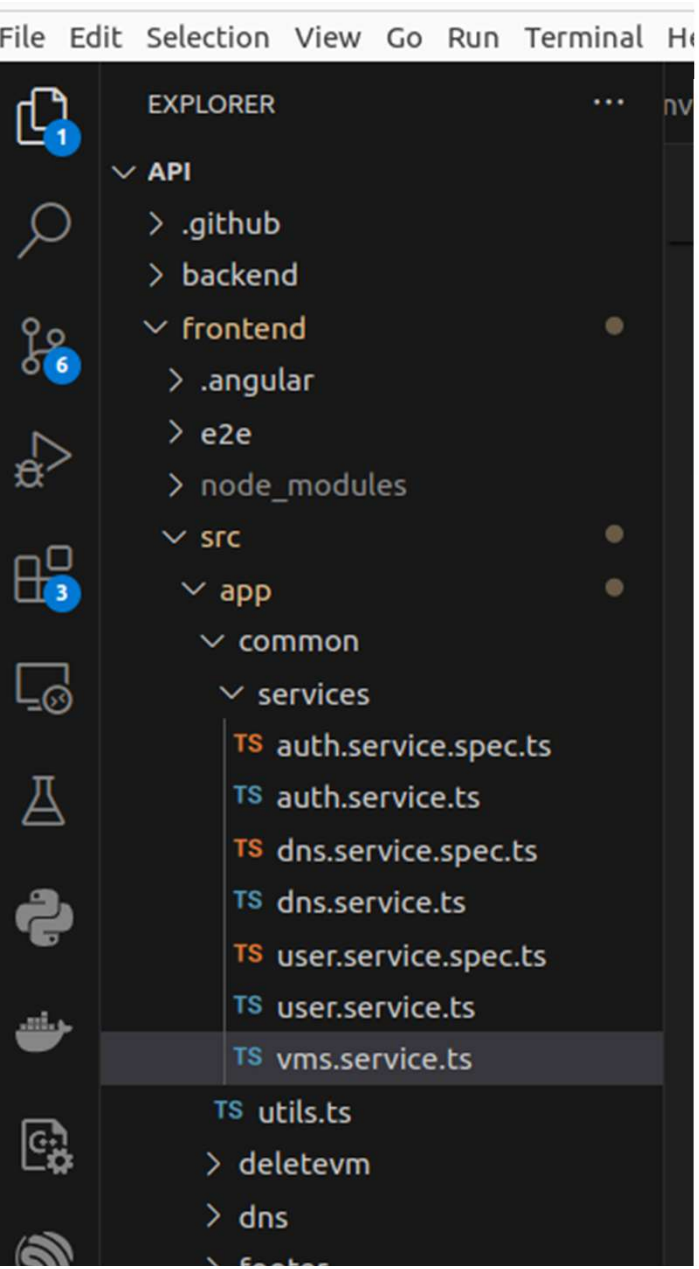
```
const expiredVms: Vm[] = [];
```

Puis, je crée un tableau de VM

```
vm.name = response.body['name'].trim();
vm.status = response.body['status'];
vm.ram = String(Math.floor(response.body['ram']/1000));
vm.disk = response.body['disk'];
vm.cpu = response.body['cpu'];
vm.user = response.body['user'];
vm.autoreboot = response.body['autoreboot'];
vm.ramUsage = response.body['ram_usage'];
vm.cpuUsage = response.body['cpu_usage'];
vm.uptime = response.body['uptime'];
vm.lastBackupDate = response.body['last_backup_date'];
```

Et là je remplie les propriétés de ma VM

(cet exemple c’est dans le service « vms »)



Ensuite y'a les services

Si je quitte puis revient sur un component, toutes ses variables seront réinitialisées à leur valeur de base

Les Services, ça contient des variables qui ne seront jamais réinitialisés tant qu'on ne quitte pas l'application.

Ça permet d'avoir des méthodes accessibles depuis n'importe quelle page, et de stocker des valeurs qu'on peut réutiliser sur tous les components.

Par exemple les infos sur les vms, on ne veut pas avoir à les récupérer à chaque fois qu'on change de page

Pour utiliser les services dans un autre component, il faut l'ajouter dans le constructeur du component

```
18    */
19   export class AppComponent implements OnInit{
20     title = 'Hosting';
21
22     constructor(public userService: UserService,
23                 public vmsService: VmsService,
24                 public dnsService: DnsService,
25                 public user: User,
26                 public router: Router,
27                 private titleService: Title,
28                 public cookie: CookieService)
29                 {
30       const url = window.location.hostname;
```

Ok on va s'intéresser au backend maintenant

DU COUP IL VA FALLOIR OUVRIR LE DOSSIER BACKEND AUSSI

SAUF QUE SANS LES VARIABLES D'ENVIRONNEMENT DU .ENV, ON NE POURRA RIEN FAIRE DE CONCRET

PAR CONTRE ON PEUT QUAND MÊME VOIR RAPIDEMENT C'EST QUOI LA CHAINE D'INTERACTION POUR PASSER DU FRONTEND ANGULAR JUSQU'AUX VMS SUR PROXMOX

Voici un exemple typique pour récupérer les infos d'une VM
Essayez de retracer chaque fonction appelée :

A l'initialisation du component "vm", on appelle la fonction typescript "updateVm" dans le service "vms"

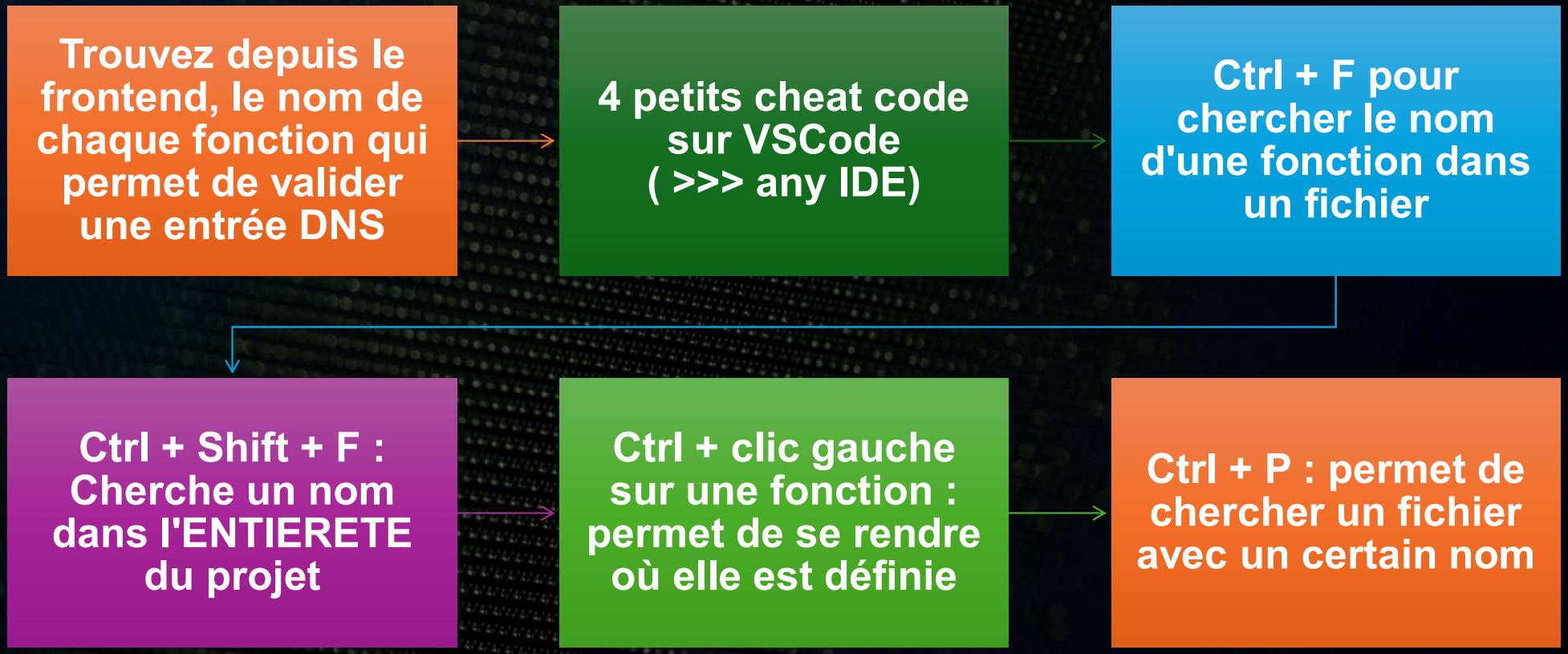
UpdateVm réalise la requête http GET "vm/vmid" à l'API d'hosting (regardez sur api-hosting-dev.minet.net)

Cet endpoint est défini dans "proxmox_api > swagger"

Le swagger appelle la fonction "get_vm_id" du fichier "proxmox_api > controllers > default_controller"

Elle fait appel à pleins de fonction de "proxmox_api > proxmox"

OK ce serait le dernier défi



C’est fini

Allez regarder la formation de l’année dernière et le wiki pour plus de détails