1. Introduction

Ce document est le support de la formation "Créer une appli web en NSI", à destination des enseignants de NSI, et éventuellement pour les élèves motivés.

Il est disponible en ligne à l'adresse :
2025-formation-web-app-python-26f3cf.forge.apps.education.fr.

Une version PDF est disponible à l'adresse :
2025-formation-web-app-python-26f3cf.forge.apps.education.fr/appli-web-nsi.pdf.
attention: les liens internes dans ce PDF ne sont pas cliquables.

La présentation associée à cette formation est disponible à l'adresse :
2025-formation-web-app-python-26f3cf.forge.apps.education.fr/slides.prez.pdf.

Le code source ce document est disponible à l'adresse :
forge.apps.education.fr/nsinormandie/2025-formation-web-app-python

Auteur : Jean-Matthieu BARBIER

2. Table des matières

    - Cookies
    - Sessions
    - Authentification
    - Web tokens
- Bases de données
    - SQLite
        - Utilisation serveur
        - Injection SQL
    - SQL/ORM
        - PostgreSQL
        - MySQL
    - NoSQL
    - Session et authentification
- Déploiement
    - Créer une clef SSH
    - Fournisseur cloud : scaleway
    - Installation, premier serveur
    - Nom de domaine
    - Certificat SSL
    - Mise en production
    - Bricolage chez soi

3. Rappels HTML/CSS

Notre internet est basé sur plusieurs "piliers" :

- **HTML** : HyperText Markup Language
- **CSS** : Cascading Style Sheets
- **JavaScript** : langage de programmation
- **HTTP(s)** : HyperText Transfer Protocol
- **URL** : Uniform Resource Locator

Objectif : rappeler les bases de ces "piliers" pour la création d'une application web.

> **Programme**
>
> - SNT : thème Internet
> - SNT : thème WEB
> - NSI 1ère : Modalités de 'intéraction entre l'homme et la machine / évènements

4. World Wide Web

WWW = "World Wide Web" = "Toile mondiale"

4.1. Qu'est-ce que le WWW ?

Le Web est un **système hypertexte public** fonctionnant sur *internet*. Le Web permet de consulter, avec un navigateur, des pages accessibles via des adresses **URL**, et pouvant contenir des liens vers d'autres pages.

4.2. Histoire du WWW

Le WWW a été inventé par *Tim Berners-Lee* en 1989 au CERN (Organisation européenne pour la recherche nucléaire) à Genève, en Suisse. Il a été conçu pour faciliter le partage d'informations entre les chercheurs et les scientifiques du monde entier.

Le premier site web a été mis en ligne le 6 août 1991. Il s'agissait d'une page d'information sur le projet WWW lui-même, hébergée sur un serveur NeXT à l'époque. Le site contenait des informations sur le projet, des instructions sur la façon de créer des pages web et des liens vers d'autres ressources.

Il est encore possible de consulter cette page à l'adresse suivante :

- avec un navigateur moderne : info.cern.ch/hypertext/WWW/TheProject.html
- avec une vue proche de l'originale : line-mode.cern.ch/www/hypertext/WWW/TheProject.html

Le WWW a rapidement gagné en popularité et est devenu un outil essentiel pour la communication, l'éducation, le commerce et le divertissement.

Au fil des ans, de nombreuses technologies et normes ont été développées pour améliorer l'expérience utilisateur sur le Web; le WWW a également évolué pour inclure des fonctionnalités telles que les réseaux sociaux, le commerce électronique et le streaming.

> **Programme**
>
> - NSI 1ère : Évènements clés de l'histoire de l'informatique

5. HTML

5.1. Problématique

Comment créer un texte pouvant **en mode texte seulement** contenir des informations sur son **contenu**, sa **structure** et sa **présentation**, à la fois lisible par un humain et par une machine ?

C'est dès les annéés 60 que le besoin se fait sentir :

- pour les documents techniques
- pour les documents juridiques
- pour les documents scientifiques

Il faut trouver un moyen de séparer le fond de la forme.

5.2. Historiquement

- dans les années 1969 : GML[1] (Generalized Markup Language) est un langage de balisage développé par IBM pour la publication de documentation technique.
- en 1986 : SGML[2] (Standard Generalized Markup Language) est une norme ISO qui définit un langage de balisage générique.
- en 1991 : HTML[3] (HyperText Markup Language) est un langage de balisage dérivé de SGML, conçu pour créer des pages web. Il a été développé par Tim Berners-Lee au CERN (Organisation européenne pour la recherche nucléaire) à Genève, en Suisse.

[1] fr.wikipedia.org/wiki/Generalized_Markup_Language

[2] fr.wikipedia.org/wiki/Standard_Generalized_Markup_Language

[3] fr.wikipedia.org/wiki/Hypertext_Markup_Language

5.3. Concept

HyperText Markup Language :

- HyperText : concept d'interconnexion de documents par des liens hypertextes, parcours non linéaire
- Markup Language : langage de balisage, qui permet de structurer le contenu d'un document en utilisant des balises

5.3.1. Markup language

> **Définition**
>
> **Markup language** = langage de balisage : langage informatique permettant de structurer un document en utilisant des balises

Une balise est un morceau de texte qui "ouvre" un contexte, et ce même contexte est "fermé" par la même balise. Les contextes peuvent être imbriqués, et chaque balise peut contenir des attributs.

Syntaxe : `<balise>...</balise>`

Exemple :

```
<article id="example">
  <h1>Mon titre</h1>
  <p>Mon texte</p>
  <ul>
    <li>Mon premier élément</li>
    <li>Mon deuxième élément</li>
    <li>
      <ol>
        <li>Mon premier sous-élément</li>
        <li>Mon deuxième sous-élément</li>
      </ol>
    </li>
  </ul>
</article>
```

En arrivant à la ligne "Mon premier sous-élément", on a ouvert successivement les contextes `article > ul > li > ol > li`. En passant, on a ouvert et fermé le contexte `h1` et `p`.

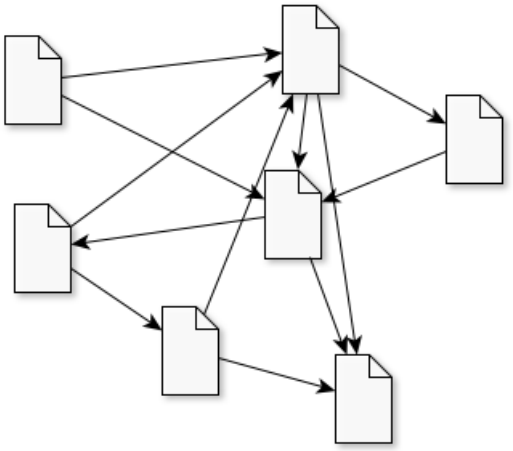
Règles :

- pas de chevauchement de balises, mode poupées russes
- dans l'idéal, balisage **sémantique** : le balisage doit refléter la structure du document, et non sa présentation; cela permet de séparer le fond de la forme, de faciliter la lecture et la compréhension du document par une machine, et de permettre une présentation différente selon le contexte (impression, écran, etc.)

5.3.2. HyperText

> **Définition**
>
> **hypertexte** : Fonction permettant d'établir des liaisons directes entre éléments (texte, image...) de documents différents.

Inclus dans le balisage, le concept d'hypertexte permet de créer des liens entre les documents, et de naviguer entre eux. Cela permet de créer des pages interconnectées, et de créer une toile d'informations. On utilise pour cela la balise `<a>` (anchor) qui permet de créer un lien vers une autre page en utilisant l'attribut `href` (**H**ypertext **REF**erence) pour spécifier l'URL de la page cible.



5.4. Suite de l'histoire

- 1995-1999 : HTML 3.2, HTML 4.0, guerre des navigateurs, balbutiements du CSS
- 2000-2005 : XHTML, CSS2.1, Internet Explorer, époque sombre
- 2007-2012 : HTML5, CSS2, un nouveau souffle, SPA, PWA, WebApps
- 2012-... : le bonheur à nouveau, des standards enfin à peu près respectés & uniformes (et IE disparait progressivement)

Programme

- SNT : thème Web
- NSI 1ère : Évènements clés de l'histoire de l'informatique

6. Syntaxe HTML, URLs

6.1. Ouverture et fermeture

Deux syntaxes possibles pour ouvrir et fermer une balise :

- Ouverture : `<balise>`
- Fermeture : `</balise>`

Ou bien les deux à la fois :

- Ouvrefermeture : `<balise/>`

6.2. Attributs

Les attributs sont des informations supplémentaires sur une balise. Ils sont écrits dans la balise d'ouverture, après le nom de la balise. Ils sont généralement sous la forme `nom_attribut="valeur"` ou juste `nom_attribut` si la seule présence de l'attribut est suffisante pour indiquer son sens.

Example

```
<a href="http://example.com">Un lien</a>
<img src="./monimage.png" alt="Un pingouin" />
<article id="article-1">...</article>
<span class="nompropre">STALLMAN</span>
```

6.3. URL

6.3.1. Décomposition d'une URL

Une **URL** (Uniform Resource Locator) est une adresse qui permet de localiser une ressource sur le Web. Elle est composée de plusieurs parties :

- le protocole (`http` , `https` , `ftp` , `file` , etc.)
- le nom de domaine (`exemple.com`) avec éventuellement un sous-domaine (`www`)
- le chemin d'accès à la ressource (`/site/pages/`)"

- le nom de la ressource (`index.html` , `image.png` , etc.)
- les paramètres de la requête (optionnels) `?param1=valeur1&param2=valeur2`
- le fragment (optionnel) `#fragment` (navigation dans la page)

Example

```
http://www.site.com:8080/path/to/file.html?
param1=valeur1&param2=valeur2#fragment
```

- protocole : `http`
- nom de domaine : `site.com`
- sous-domaine : `www`
- port : `8080`
- chemin d'accès : `/path/to/file.html`
- nom de la ressource : `index.html`
- paramètres de la requête : `param1` (`valeur1`), `param2` (`valeur2`)
- fragment : `fragment`

6.3.2. URLs relatives et absolues

Dans une page située à l'URL `https://exemple.com/site/index.html` , on peut utiliser les types d'URL suivants :

- **URLs complètes** (vers le même site ou vers un autre site)
  - `https://exemple.com/site/page.html`
  - `https://exemple.com/images/pingouin.png`
  - `https://wikipedia.org`
- **URLs absolues** (même protocole, domaine et sous-domaine que la page) :
  - `/chemin/vers/page.html`
  - `/images/pingouin.png`
- **URLs relatives** :
  - `./page.html` ou `page.html`
  - `../images/pingouin.png`

6.4. Balises

Les balises apportent une signification sémantique au document. Le HTML5 a supprimé une grande partie des balises qui n'avaient pas de signification sémantique, la présentation étant déléguée au CSS.

- `article` , `detail` , `figure` , `section` , `nav` , `menu` , `summary` , `aside` ... : organisation
- `h1` , `h2` , `h3` , `h4` , `h5` , `h6` , `p` : niveaux de titre, paragraphe
- `ul` , `ol` , `li` , `dl` , `dt` , `dd` : listes
- `a` , `img` , `audio` , `video` : hyperliens, medias
- `table` , `thead` , `tbody` , `tfoot` , `tr` , `th` , `td` : tableaux
- `form` , `input` , `select` , `button` : formulaires
- `code` , `cite` , `q` , `del` , `kbd` : types de contenu
- `b` , `i` , `em` , `strong` , `col` : présentation
- `span` , `div` : fourre-tout

Liste rapide des balises HTML5 :

HTML 5 Cheat Sheet [TAGS]

New [tags added in HTML5]

<article>	self-contained composition that is independently distributable
<aside>	section of page that consists of content tangentially related to content around it
<audio>	sound content
<bdi>	span of text to be isolated from surroundings for bidirectional formatting purposes
<canvas>	area that can be used to draw graphics via JavaScript
<command>	user invokable command
<datalist>	dropdown list
<datatemplate>	data template
<details>	details of an element
<embed>	embedded content
<figcaption>	caption of figure element
<figure>	group of media content
<footer>	footer for section or page
<header>	header for section or page
<hgroup>	group of headings for section
<keygen>	generated key in a form
<mark>	marked text
<meter>	measurement in defined range
<nav>	navigation links
<output>	represents results of calculation
<progress>	progress of any kind of task
<rp>	parenthesized ruby text
<rt>	ruby text
<ruby>	ruby annotations
<section>	section in a document
<source>	media resources
<summary>	header of a detail element
<time>	date/time
<video>	video
<wbr>	possible line break

Old [unsupported tags]

<acronym>	acronym
<applet>	applet
<basefont>	base font
<bgsound>	background sound
<big>	big text
	centered text
<fn>	footnotes
<font>	text font, size, and color
<frame>	sub window
<frameset>	set of frames
<isindex>	provides searchable index related to current document
<dir>	directory list
<noembed>	no embed section
<noframes>	no frame section
<s>	strikethrough text
<strike>	strikethrough text
<tt>	teletype text
<u>	underlined text
<xmp>	preformatted text

Existing [tags in HTML4 & 5]

<!--...-->	comment
<!doctype>	document type
<a>	hyperlink
<abbr>	abbreviation
<address>	address element
<area>	image map area
<b>	bold text
<base>	base URL for all links in page relative to document root
<bdo>	text direction
<blockquote>	long quotation
<body>	body element
 	single line break
<button>	push button
<caption>	table caption
<cite>	citation
<code>	code text
<col>	attributes for columns
<colgroup>	groups of columns
<dd>	definition description
<del>	deleted text
	generic block-level element
<dfn>	defining instance of a term
<dl>	definition list
<dt>	definition term
	emphasized text
<fieldset>	logically group items in a form
<form>	defines a form
<h1> to <h6>	header 1 to header 6
<head>	document information
<hr>	horizontal rule
<html>	html document
<i>	italic text
<iframe>	inline sub window
<img>	image
<input>	input field
<ins>	inserted text
<kbd>	keyboard text
<label>	label for a form control
<legend>	title in a fieldset
	list item
<link>	resource reference
<map>	image map
<menu>	menu list
<meta>	meta information
<noscript>	no script section
<object>	embedded object
	ordered list
<optgroup>	option group
<option>	option in a drop-down list
<p>	paragraph
<param>	parameter for an object
<pre>	preformatted object
<q>	short quotation
<samp>	sample computer code
<script>	script
<select>	selectable list
<small>	small text
	inline generic container
	strong text
<style>	style definition
<sub>	subscripted text
<sup>	superscripted text
<table>	table
<tbody>	table body
<td>	table cell
<textarea>	text area
<tfoot>	table footer
<th>	table header
<thead>	wraps row containing table headers
<title>	document title
<tr>	table row
	unordered list
<var>	variable

Brought to you by: inmotion hosting

6.5. Ressources

- MDN : Mozilla Developer Network, site de référence
- CanIUse : compatibilité des navigateurs
- Antisèche courte : image résumée des balises HTML5
- Antisèche longue : document PDF très complet sur toutes les balises HTML5
- HTML5 : spécification HTML5 et plus précisément la partie Éléments

---

Programme

- SNT : thème Internet
- NSI 1ère : Modalités de l'interaction entre l'homme et la machine / évènements

7. DOM

7.1. Document Object Model

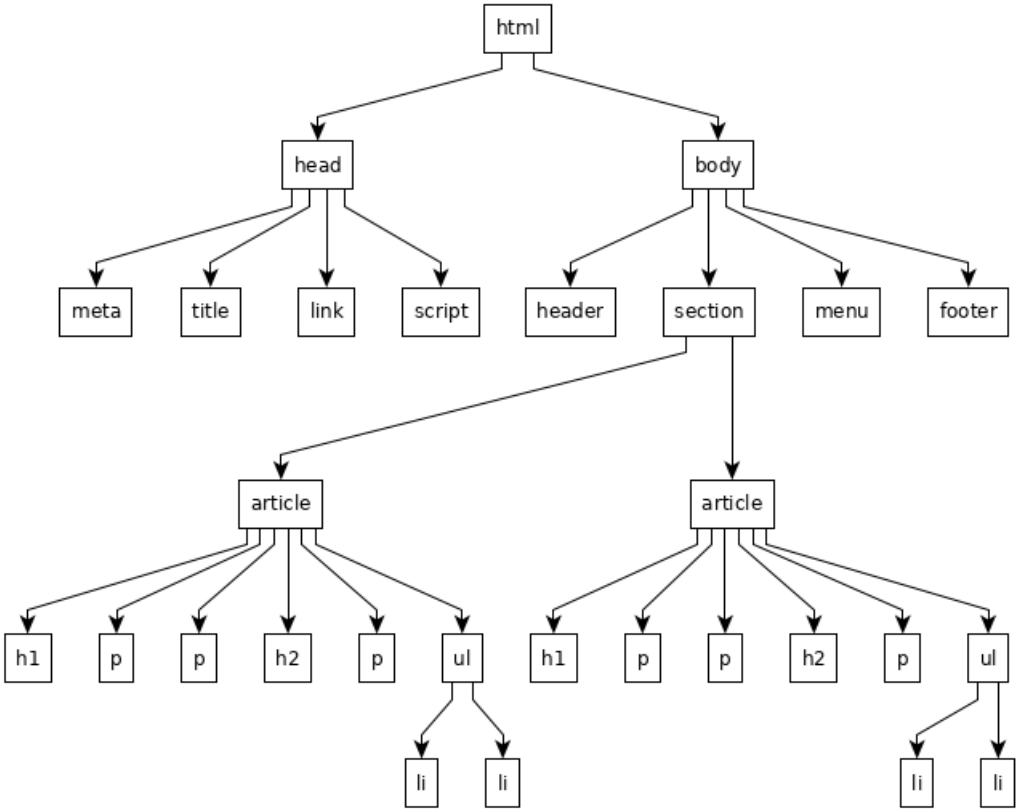
Le **Document Object Model** ou DOM (pour modèle objet de document) est une représentation et une interface de programmation pour les documents HTML. Il correspond à une représentation structurée du document sous forme d'un **arbre** et définit la façon dont la structure peut être manipulée par les programmes, en termes de style et de contenu.

Chaque noeud possède des propriétés et des méthodes, et peut être associé à des *événements*.

La structure étant arborescente, on dispose de toutes les méthodes associées aux arbres, pour naviguer dans l'arbre, pour accéder à un noeud parent, enfant ou frère, pour accéder aux enfants, aux descendants, etc...

Si on se contente ici de la partie "représentation / accès" :

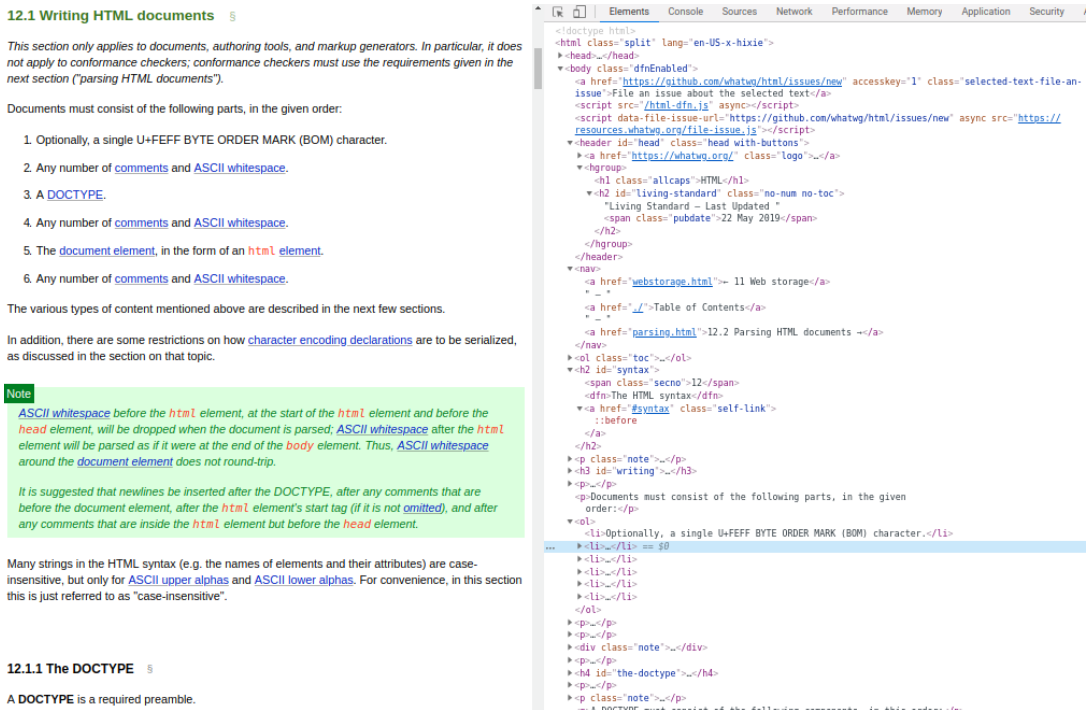
```
<!DOCTYPE html>
<html>
  <head>
     ...
  </head>
  <body>
     ...
  </body>
</html>
```



7.2. Exploration

7.2.1. Dans l'inspecteur

L'exploration du DOM peut se faire à l'aide de l'inspecteur de votre navigateur (F12 ou clic droit > Inspecter).



12.1 Writing HTML documents §

This section only applies to documents, authoring tools, and markup generators. In particular, it does not apply to conformance checkers; conformance checkers must use the requirements given in the next section ("parsing HTML documents").

Documents must consist of the following parts, in the given order:

1. Optionally, a single U+FEFF BYTE ORDER MARK (BOM) character.
2. Any number of comments and ASCII whitespace.
3. A DOCTYPE.
4. Any number of comments and ASCII whitespace.
5. The document element, in the form of an `html` element.
6. Any number of comments and ASCII whitespace.

The various types of content mentioned above are described in the next few sections.

In addition, there are some restrictions on how character encoding declarations are to be serialized, as discussed in the section on that topic.

Note

ASCII whitespace before the `html` element, at the start of the `html` element and before the `head` element, will be dropped when the document is parsed; ASCII whitespace after the `html` element will be parsed as if it were at the end of the `body` element. Thus, ASCII whitespace around the document element does not round-trip.

It is suggested that newlines be inserted after the DOCTYPE, after any comments that are before the document element, after the `html` element's start tag (if it is not omitted), and after any comments that are inside the `html` element but before the `head` element.

Many strings in the HTML syntax (e.g. the names of elements and their attributes) are case-insensitive, but only for ASCII upper alphas and ASCII lower alphas. For convenience, in this section this is just referred to as "case-insensitive".

12.1.1 The DOCTYPE §

A **DOCTYPE** is a required preamble.

Elements Console Sources Network Performance Memory Application Security

```
<!doctype html>
<html class="split" lang="en-US-x-hixie">
 ▶<head>…</head>
 ▼<body class="dfnEnabled">
    <a href="https://github.com/whatwg/html/issues/new" accesskey="1" class="selected-text-file-an-
    issue">File an issue about the selected text</a>
    <script src="/html-dfn.js" async></script>
    <script data-file-issue-url="https://github.com/whatwg/html/issues/new" async src="https://
    resources.whatwg.org/file-issue.js"></script>
   ▼<header id="head" class="head with-buttons">
     ▶<a href="https://whatwg.org/" class="logo">…</a>
     ▼<hgroup>
        <h1 class="allcaps">HTML</h1>
       ▼<h2 id="living-standard" class="no-num no-toc">
          "Living Standard — Last Updated "
          <span class="pubdate">22 May 2019</span>
        </h2>
      </hgroup>
    </header>
   ▼<nav>
      <a href="webstorage.html">← 11 Web storage</a>
      " — "
      <a href="./">Table of Contents</a>
      " — "
      <a href="parsing.html">12.2 Parsing HTML documents →</a>
    </nav>
   ▶<ol class="toc">…</ol>
   ▼<h2 id="syntax">
      <span class="secno">12</span>
      <dfn>The HTML syntax</dfn>
     ▼<a href="#syntax" class="self-link">
        ::before
      </a>
    </h2>
   ▶<p class="note">…</p>
   ▶<h3 id="writing">…</h3>
   ▶<p>…</p>
    <p>Documents must consist of the following parts, in the given
      order:</p>
   ▼<ol>
      <li>Optionally, a single U+FEFF BYTE ORDER MARK (BOM) character.</li>
…   ▶<li>…</li> == $0
     ▶<li>…</li>
     ▶<li>…</li>
     ▶<li>…</li>
     ▶<li>…</li>
    </ol>
   ▶<p>…</p>
   ▶<p>…</p>
   ▶<div class="note">…</div>
   ▶<p>…</p>
   ▶<h4 id="the-doctype">…</h4>
   ▶<p>…</p>
   ▶<p class="note">…</p>
```

7.2.2. Dans la console javascript

Dans la "Console javascript", on peut accéder à la structure du DOM, et explorer les noeuds en javascript.

```
// Accéder à un noeud
let node = document.children[0].children[1]...
node.parentNode // parent
node.children
node.innerHTML // contenu
node.textContent // texte
```

7.2.3. En python

En python, la librairie `beautifulsoup` permet de parcourir et modifier le DOM d'une page HTML.

```
pip install beautifulsoup4 requests
# ou bien
# uv add beautifulsoup4 requests
```

```
from bs4 import BeautifulSoup, Tag, NavigableString
import requests

def affiche_dom(node, indent=0):
    prefix = '  ' * indent
    if isinstance(node, Tag):
        print(f"{prefix}{node.name}")
        for child in node.children:
            affiche_dom(child, indent + 2)

def main():
    url = "https://www.gnu.org/philosophy/philosophy.html"
    response = requests.get(url)
    soup = BeautifulSoup(response.content, "html.parser")
    # parcourir l'arbre de manière récursive
    affiche_dom(soup.html)


if __name__ == "__main__":
    main()
```

Référence : https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Programme

- SNT: thème Web
- NSI 1ère : Modalités d'interaction entre l'homme et la machine
- NSI terminale : Structures de données - Arbres : structures hiéarchiques
- NSI terminale : Algorithmes sur les arbres [binaires] : parcourir un arbre de différentes façons (ou algos sur les graphes...)

8. CSS

> **Définition**
>
> **CSS** = **Cascading Style Sheets** = Feuilles de style en cascade : séquence d'instructions de rendu visuel, auditif ou tactile d'une ressource HTML ou XML

8.1. Principe

Le CSS est un langage informatique qui permet de décrire la présentation d'un document écrit en HTML ou XML (y compris les dialectes XML comme SVG, MathML ou XHTML).

Enjeux :

- **séparation entre la structure & la présentation**
- présentation selon le media (print / screen / tv...)
- cascade des styles : combinaison de différentes source de style

8.2. Exemple : CSS Zen Garden

Le projet CSS Zen Garden a été lancé en 2003 par Dave Shea. Il s'agit d'un projet qui met en avant la puissance du CSS en permettant aux designers de créer des mises en page uniques et créatives en utilisant le même contenu HTML. Le site présente une série de designs réalisés par des designers du monde entier, **tous basés sur le même fichier HTML.**

Lien : CSS Zen Garden

La page HTML sans aucune feuille de style :

CSS Zen Garden

The Beauty of CSS Design

A demonstration of what can be accomplished through CSS-based design. Select any style sheet from the list to load it into this page.

Download the example html file and css file

The Road to Enlightenment

Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible DOMs, broken CSS support, and abandoned browsers.

We must clear the mind of the past. Web enlightenment has been achieved thanks to the tireless efforts of folk like the W3C, WaSP, and the major browser creators.

The CSS Zen Garden invites you to relax and meditate on the important lessons of the masters. Begin to see with clarity. Learn to use the time-honored techniques in new and invigorating fashion. Become one with the web.

So What is This About?

There is a continuing need to show the power of CSS. The Zen Garden aims to excite, inspire, and encourage participation. To begin, view some of the existing designs in the list. Clicking on any one will load the style sheet into this very page. The HTML remains the same, the only thing that has changed is the external CSS file. Yes, really.

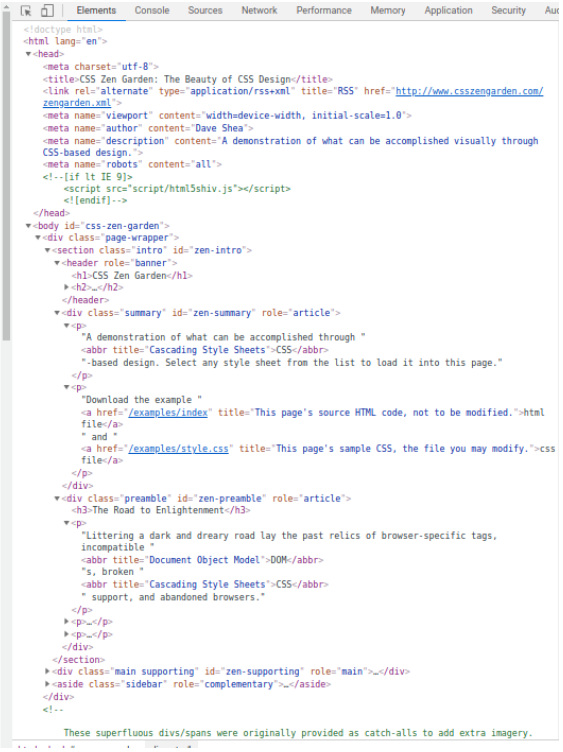
CSS allows complete and total control over the style of a hypertext document. The only way this can be illustrated in a way that gets people excited is by demonstrating what it can truly be, once the reins are placed in the hands of those able to create beauty from structure. Designers and coders alike have contributed to the beauty of the web; we can always push it further.

Participation

Strong visual design has always been our focus. You are modifying this page, so strong CSS skills are necessary too, but the example files are commented well enough that even CSS novices can use them as starting points. Please see the CSS Resource Guide for advanced tutorials and tips on working with CSS.

You may modify the style sheet in any way you wish, but not the HTML. This may seem daunting at first if you've never worked this way before, but follow the listed links to learn more, and use the sample files as a guide.

Download the sample HTML and CSS to work on a copy locally. Once you have completed your masterpiece (and please, don't submit half-finished work) upload your CSS file to a web server



Différentes mises en page réalisées par des designers :

Mid Century Modern
ANDREW LOHMAN, United States

Garments
DAN MALL, United States

Steel
STEFFEN KNOELLER, Germany

Apothecary
TRENT WALTON, United States

Handcrafted
ELLIOT JAY STOCKS, United Kingdom

Fountain Kiss
JEREMY CARLSON, United States

Programme

- SNT : thème Web

9. Syntaxe et sélecteurs

Références : MDN Sélecteurs : MDC CSS Selectors

9.1. Syntaxe

Le CSS associée des **règles de représentation** à des éléments sélectionnés dans le document HTML. Une règle de représentation est composée d'un ou plusieurs **sélecteurs** et d'une **déclaration** qui elle-même est composée d'une ou plusieurs **propriétés** et de leur **valeur**.

Les propriétés sont tout ce qui peut être appliqué à un élément : couleur, taille, marge, police, etc.

Exemple de structure :

```
selecteur1 {
  propriete: valeur;
}

selecteur2,
selecteur3 {
  propriete: valeur;
}
```

9.2. Sélecteurs

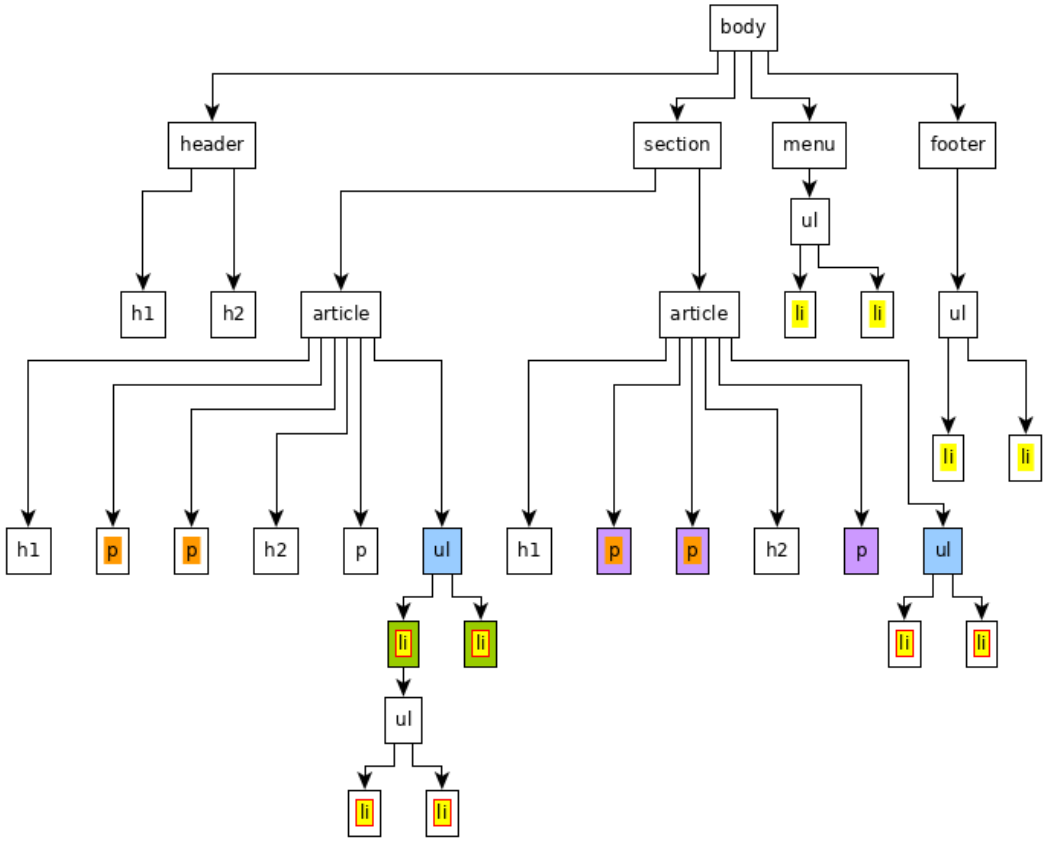
9.2.1. Sélecteurs simples, hiérarchie

1. `li` == sélectionne toutes les balises li du document
2. `article li` == sélectionne toutes les balises li qui sont des descendants (pas forcément directs) d'un article

3. `article > ul` == sélectionne tous les ul qui sont enfant direct d'un article
4. `article > ul > li` == sélectionne tous les enfants direct d'un ul descendant direct d'un article
5. `h1 + p` == sélectionne toutes les balises p qui suivent immédiatement un h1
6. `h1 ~ p` == sélectionne toutes les balises p qui suivent (immédiatement ou non) un h1 et qui ont le même parent

Exemple :

```
li {
  background-color: yellow;
}
article li {
  border: 1px solid red;
}
article > ul {
  background-color: blue;
}
article > ul > li {
  border: 3px solid green;
}
h1 + p {
  background-color: orange;
}
h1 ~ p {
  border: 3px solid purple;
}
```



9.2.2. Sélecteur de classe, d'id et d'attribut

- `.rouge` == sélectionne toutes les balises ayant la classe "rouge"
- `h1.rouge` == sélectionne toutes les balises h1 ayant la classe rouge
- `#article-1` == sélectionne toutes (=LA) balise ayant l'id article-1

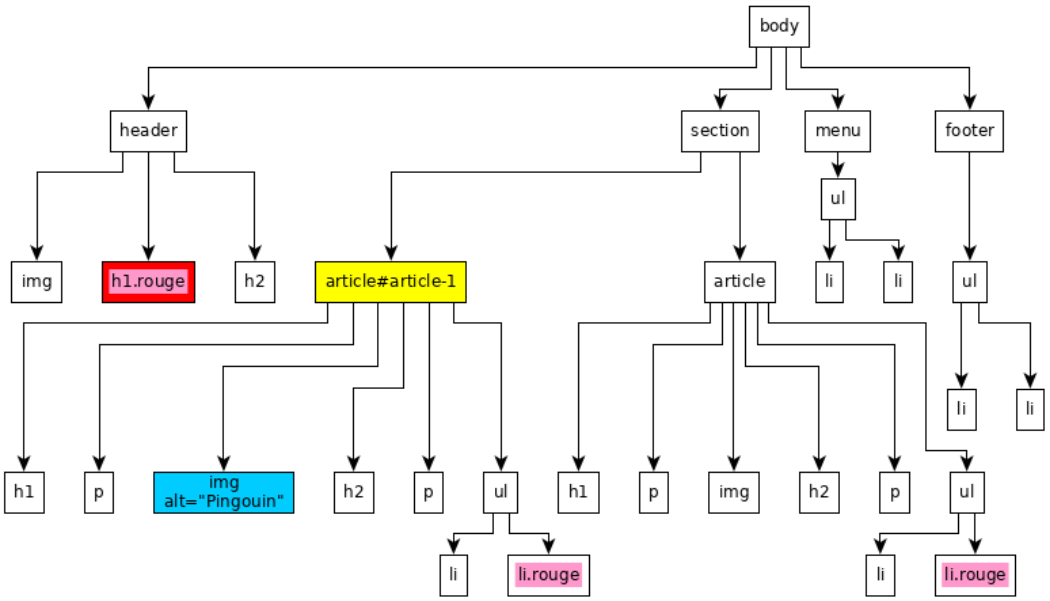
- `img[alt='Pingouin']` == sélectionne toutes les balises img ayant un attribut alt égal à Pingouin

Plusieurs variantes pour le sélecteur d'attribut : MDN CSS Attribute selectors :

- `attr$=valeur` : sélectionne les éléments dont l'attribut se termine par valeur
- `attr^=valeur` : sélectionne les éléments dont l'attribut commence par valeur
- `attr*=valeur` : sélectionne les éléments dont l'attribut contient valeur
- `attr~=valeur` : sélectionne les éléments dont l'attribut contient valeur
- ...

Exemple :

```
.rouge {
  background-color: red;
}
h1.rouge {
  border: 1px solid red;
}
#article-1 {
  background-color: yellow;
}
img[alt="Pingouin"] {
  background-color: blue;
}
```



9.2.3. Sélecteurs de pseudo-classes et pseudo-éléments

Références : MDN CSS Pseudo-classes et MDN CSS Pseudo-elements

- `:hover` : lorsque la souris survole l'élément
- `:active` : lorsque l'élément est cliqué
- `:focus` : lorsque l'élément a le focus
- `:first-child` : lorsque l'élément est le premier enfant de son parent
- `:last-child` : lorsque l'élément est le dernier enfant de son parent
- `:nth-child(n)` : lorsque l'élément est le n-ième enfant de son parent
- `:nth-of-type(n)` : lorsque l'élément est le n-ième enfant de son parent du même type
- `:before` : avant le contenu de l'élément

- `:after` : après le contenu de l'élément
- `::first-letter` : première lettre de l'élément
- `::first-line` : première ligne de l'élément
- `::selection` : sélection de l'élément
- ...

9.3. Propriétés

9.3.1. Toutes les propriétés

Les propriétés régissent tous les aspect de la présentation d'un élément : arrière plan, bordures, polices, affichage de texte, positionnement, ...

Liste "antisèche" de 5 pages, très complète, de 2009 (il manque quelques valeurs) et Référence MDN CSS (à jour)

Quelques exemples "classiques":

- `font-size` : en mm, px, pt, em, rem
- `font-weight` : normal, bold, 200, 300..
- `color` : #RRGGBB ou #RGB ou rgb(255,255,255) ou rgba(255,255,255,1)
- `background-image` : url('http://adresse/image')
- ...

9.3.2. Valeurs

Les valeurs dépendent de la propriété; certaines propriétés acceptent des "mots" préféfinis (ex: `text-align: left|right|start|end|center|justify;`), d'autres acceptent des valeurs dimensionnelles (ex: `font-size: 12px;`), d'autres acceptent des valeurs de type couleur (ex: `color: #RRGGBB;`).

Valeurs dimensionnelles

- `px` : pixels = pixels "réels" (ex: 12px) sur l'écran
- `pt` : points = 1/72 de pouce (ex: 12pt)
- `mm` : millimètres (ex: 12mm) et `cm` : centimètres (ex: 12cm)
- `em` : taille de la police du **parent**
- `rem` : taille de la police de l'**élément racine**
- `%` : pourcentage de la taille de l'élément parent
- `vh` : 1% de la hauteur de la fenêtre
- `vw` : 1% de la largeur de la fenêtre
- `vmin` : 1% de la plus petite dimension de la fenêtre
- `vmax` : 1% de la plus grande dimension de la fenêtre
- ...

Valeurs de couleur

Référence MDN Color value

- `#RRGGBB` : rouge, vert, bleu (ex: `#FF0000` = rouge)

- `#RGB` : rouge, vert, bleu (ex: `#F00` = rouge)
- `rgb(255,0,0)` : rouge, vert, bleu (ex: `rgb(255,0,0)` = rouge)
- `rgba(255,0,0,1)` : rouge, vert, bleu, alpha (ex: `rgba(255,0,0,1)` = rouge)
- couleurs nommées : `red`, `blue`, `green`, `yellow`, `black`, `white`, ... cf [MDN]https://developer.mozilla.org/en-US/docs/Web/CSS/named-color)
- ...

Valeurs calculées

Référence MDN calc()

- `calc()` : permet de faire des calculs sur les valeurs dimensionnelles (ex: `width: calc(100% - 20px);` ou `width: calc(100vh - 20px);`)
- `clamp()` : permet de fixer une valeur entre deux bornes (ex: `width: clamp(100px, calc(...), 500px);` ou `width: clamp(100px, calc(...), 500px);`)
- `min()` : permet de prendre la valeur la plus petite (ex: `width: min(100px, 50vw);`)
- `max()` : permet de prendre la valeur la plus grande (ex: `width: max(100px, 50vw);`)

Variables CSS

Référence MDN CSS Variables

Les variables CSS permettent de définir des valeurs réutilisables dans le document CSS. Elles sont définies avec le préfixe `--` et peuvent être utilisées avec la fonction `var()`. Par exemple :

```
:root {
  --main-color: #3498db;
  --secondary-color: #2ecc71;
}
h1 {
  color: var(--main-color);
}
p {
  color: var(--secondary-color);
}
```

> **Programme**
>
> - SNT : thème Web

10. Cascade et héritage

10.1. Cascade

Si plusieurs propriétés sont applicables pour un élement, il faut savoir comment choisir la bonne. La cascade est l'ordre de priorité des propriétés CSS. Il y a trois niveaux de priorité :

1. importance (`!important` , par exemple `color: red!important`)
2. sélectivité : du plus spécifique au plus général.
    1. la propriété `style` de l'élément
    2. une propriété définie pour l' `id` d'un élément
    3. une propriété définie par classe ou attribut
    4. une propriété définie pour l'élément
3. ordre d'apparition dans la feuille de style (le plus bas l'emporte)

10.1.1. En pratique

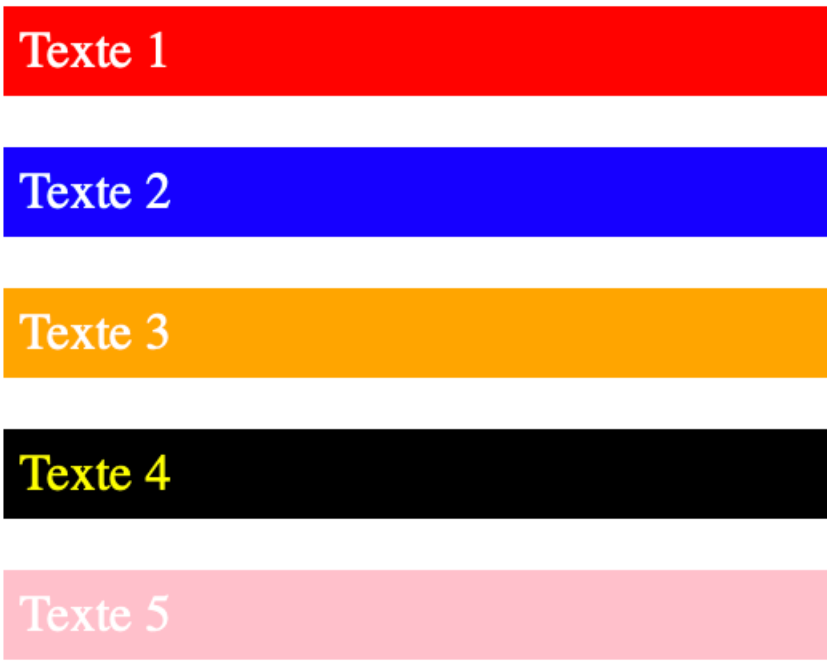
Vu le html et le css suivants, quelle sera la couleur de fond de chaque paragraphe ?

```
<p id="p1" class="c1">Texte 1</p>
<div>
  <p class="c1">Texte 2</p>
  <p class="c2" id="p3">Texte 3</p>
  <p class="c3" style="color: yellow;">Texte 4</p>
  <p>Texte 5</p>
</div>
```

```
p { background-color: green; padding: 5px; color: white }
#p1 { background-color: red; }
.c1 { background-color: blue; }
.c2 { background-color: purple; }
#p3 { background-color: orange; }
div p { background-color: pink; }
.c3 { background-color: black; }
```

10.1.2. Correction

Tester sur codepen.io/jmsolidev/pen/pvvrJpO



10.2. Héritage

Certaines valeurs de propriété appliquées à un élement sont appliquées à ses enfants, d'autres ne le sont pas :

- héritées : `font-family`, `color`, ...
- non héritées : `margin`, `padding`, `border`, ...

developer.mozilla.org/en-US/docs/Web/CSS/Reference

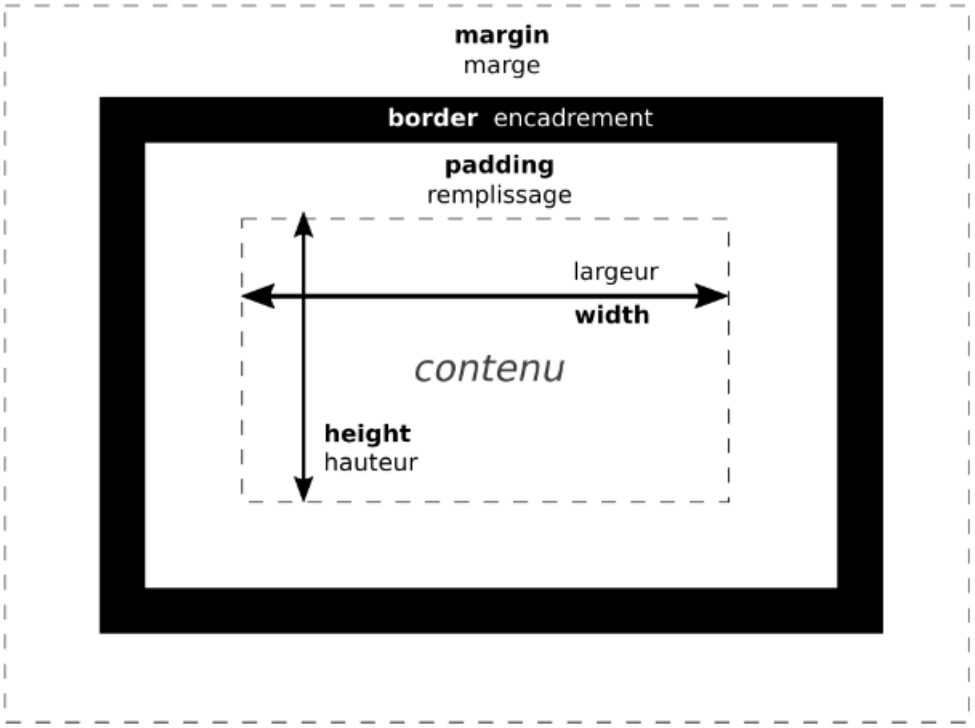
Programme

- SNT : thème Web

11. Flux et positionnement

11.1. Boîte

Documentation : MDN



11.2. Flux & positionnement

- lecture de l'arbre du document => chaque noeud = une boîte
- détermination du style, de la taille & du positionnement de chaque boîte

Propriétés liées :

- `display` : mode d'affichage
  - `block` : bloc, occupe toute la largeur de son parent
  - `inline` : en ligne, occupe la largeur de son contenu
  - `inline-block` : en ligne, mais avec une largeur et une hauteur
  - `flex` : flexbox
  - `grid` : grille
  - `table` : tableau
  - `none` : pas d'affichage
  - ...
- `position` : référentiel de positionnement

    - `static` : position par défaut
    - `relative` : position relative à la position par défaut
    - `absolute` : position absolue, par rapport à l'élément parent le plus proche
    - `fixed` : position fixe, par rapport à la fenêtre du navigateur
    - `sticky` : position collante, entre relative et fixe
- `float` : permet de faire flotter un élément à gauche ou à droite
    - `left` : flottant à gauche
    - `right` : flottant à droite
    - `none` : pas de flottant
- `clear` : suppression des flottants
    - `left` : pas de flottant à gauche
    - `right` : pas de flottant à droite
    - `both` : pas de flottant à gauche ou à droite
- `overflow` : gestion du débordement
    - `visible` : déborde
    - `hidden` : caché
    - `scroll` : barre de défilement
    - `auto` : barre de défilement si nécessaire
- `z-index` : gestion de la superposition : plus le nombre est élevé, plus l'élément est au-dessus
- `visibility` : gestion de la visibilité
    - `visible` : visible
    - `hidden` : caché
    - `collapse` : pour les tableaux, efface la ligne ou la colonne (peu utilisé)
- `opacity` : gestion de l'opacité (0 = transparent, 1 = opaque)

11.3. Exemples & exercices

- codepen.io/jmsolidev/pen/BeJbaj
- solidev.net/tempo/objectif1html.zip
- solidev.net/tempo/objectif2css.zip

Programme

- SNT : thème Web

12. Flexbox, grid

Section non traitée, renvoi vers la documentation MDN pour plus de détails

12.1. Flexbox

Documentation MDN

12.2. Grid

Documentation MDN

Programme

- SNT : thème Web

13. Media queries, responsive

Section non traitée, renvoi vers la documentation MDN pour plus de détails

13.1. Media queries

Documentation MDN

13.2. Responsive

Documentation MDN

Programme

- SNT : thème Web

14. Communication client/serveur

Cette section est consacrée à la communication entre le client (navigateur) et le serveur (site web).

Ils communiquent entre eux en utilisant le *protocole HTTP* qui utilise généralement le *protocole TCP/IP* pour transporter l'information.

On peut observer le détail de ces communications au niveau TCP/IP en utilisant un outil comme Wireshark, et au niveau HTTP en utilisant les outils développeur du navigateur > onglet "Network". Il est également possible d'utiliser des outils comme `Postman` ou `curl` pour tester les requêtes HTTP.

15. HTTP

Le protocole **HTTP** (Hypertext Transfer Protocol) est l'épine dorsale de la communication sur le World Wide Web. Il définit les règles qui permettent aux navigateurs web (clients) d'interagir avec les serveurs web pour récupérer et transmettre des informations, rendant possible la navigation sur les sites web, le téléchargement de fichiers et l'utilisation d'applications web.

15.1. Historique

15.1.1. HTTP/0.9

Comme vu pécédemment, l'histoire de HTTP commence avec Tim Berners-Lee au CERN, en 1990, lors de la création des 3 piliers du **web** (HTML, URL, HTTP).

HTTP, plus précisément **HTTP/0.9**, est implémenté en 1991. Il était extrêmement simple, ne supportant qu'une seule méthode de requête, `GET`, et ne transmettait que le contenu HTML des pages. Il n'y avait pas d'en-têtes de requête ou de réponse, et la connexion TCP était fermée après chaque requête.

15.1.2. HTTP/1.0 et HTTP/1.1 (1996-1997)

L'adoption croissante du web a nécessité des améliorations significatives du protocole HTTP. En 1996, **HTTP/1.0** a indroduit des changements clés :

- **nouvelles méthodes de requête:** `POST` et `HEAD` ont été ajoutées, permettant non seulement de récupérer des ressources (GET), mais aussi d'envoyer des données au serveur (POST) et de demander uniquement les en-têtes d'une ressource sans son contenu (HEAD).
- **en-têtes de requête et de réponse:** des en-têtes ont été introduits pour fournir des informations supplémentaires sur la requête et la réponse, comme le type de contenu, le navigateur utilisé, etc.

Cependant, HTTP/1.0 restait inefficace car chaque requête nécessitait l'établissement d'une nouvelle connexion TCP. En 1997, **HTTP/1.1** a apporté des améliorations majeures pour pallier à ce problème :

- **connexions persistantes (Keep-Alive):** permet de réutiliser la même connexion TCP pour plusieurs requêtes, réduisant la latence et améliorant les performances.
- **pipelining des requêtes:** permet d'envoyer plusieurs requêtes sur la même connexion sans attendre la réponse à la première.
- **Chunked Transfer Encoding:** permet d'envoyer des données de réponse en

morceaux, sans avoir à connaître la taille totale à l'avance.
- **mise en cache:** des mécanismes de cache plus sophistiqués ont été introduits pour réduire la charge sur les serveurs et améliorer la vitesse de chargement des pages.

HTTP/1.1 est resté le protocole dominant pendant plus de 15 ans, mais ses limitations en termes de performances sont devenues de plus en plus évidentes avec l'explosion du contenu web et l'augmentation du nombre d'utilisateurs.

15.1.3. HTTP/2 (2015)

HTTP/2 a été publié en 2015, basé sur le protocole Spdy de Google, et représentait une refonte majeure du protocole. Les principaux objectifs de HTTP/2 étaient d'améliorer les performances et de réduire la latence. Les innovations clés incluaient :

- **multiplexage:** possibilité d'envoyer plusieurs requêtes et réponses simultanément sur une seule connexion TCP, éliminant le problème du "head-of-line blocking" inhérent à HTTP/1.1.
- **compression des en-têtes (HPACK):** réduit la taille des en-têtes, minimisant le volume de données transférées.
- **priorisation des requêtes:** permet au client de spécifier l'ordre dans lequel il souhaite recevoir les ressources.
- **server push:** permet au serveur d'envoyer des ressources au client avant même que celui-ci ne les demande, anticipant les besoins du client.

HTTP/2 est un protocole binaire, plus complexe à analyser manuellement qu'HTTP/1.1, mais beaucoup plus efficace en termes de performances. Il a rapidement été adopté par les principaux navigateurs web et les fournisseurs de contenu.

Les futures versions, comme HTTP/3, continuent d'explorer de nouvelles approches pour optimiser la communication web.

15.2. Fonctionnement de HTTP

15.2.1. Rappel sur TCP/IP

Avant de plonger dans HTTP, il est important de comprendre sa base : TCP/IP. TCP/IP est une suite de protocoles qui régit la communication sur Internet. HTTP, lui, s'appuie sur TCP pour établir une connexion fiable entre le client et le serveur.

- **IP (Internet Protocol)** : responsable de l'adressage et du routage des paquets de données.
- **TCP (Transmission Control Protocol)** : assure une transmission fiable des données, avec vérification d'erreurs et réassemblage des paquets dans l'ordre correct. HTTP utilise TCP car il a besoin de garantir que les données arrivent intégralement et dans l'ordre.

En bref, HTTP "parle" à travers TCP/IP. TCP/IP gère le *transport* des données, HTTP gère le *sens* de ces données.

15.2.2. Format d'un message HTTP (Requête ou Réponse)

Un message HTTP (que ce soit une requête ou une réponse) est un bloc de texte brut

structuré. Il comprend les éléments suivants, séparés par des lignes vides :

1. **ligne de début :**
    - Pour une requête : `MÉTHODE URI Version` (ex: `GET /index.html HTTP/1.1`)
    - Pour une réponse : `Version CodeStatut Phrase` (ex: `HTTP/1.1 200 OK`)
2. **en-têtes (Headers) :** une série de paires clé-valeur, chacune sur une ligne distincte, séparées par un deux-points (`:`). Les en-têtes fournissent des métadonnées sur le message. Exemple:

    ```
    Content-Type: text/html
    Content-Length: 1234
    User-Agent: Mozilla/5.0
    ```

3. **ligne vide :** sépare les en-têtes du corps du message.
4. **Corps (Body) :** optionnel, contient les données réelles de la requête (ex: données de formulaire POST) ou de la réponse (ex: code HTML).

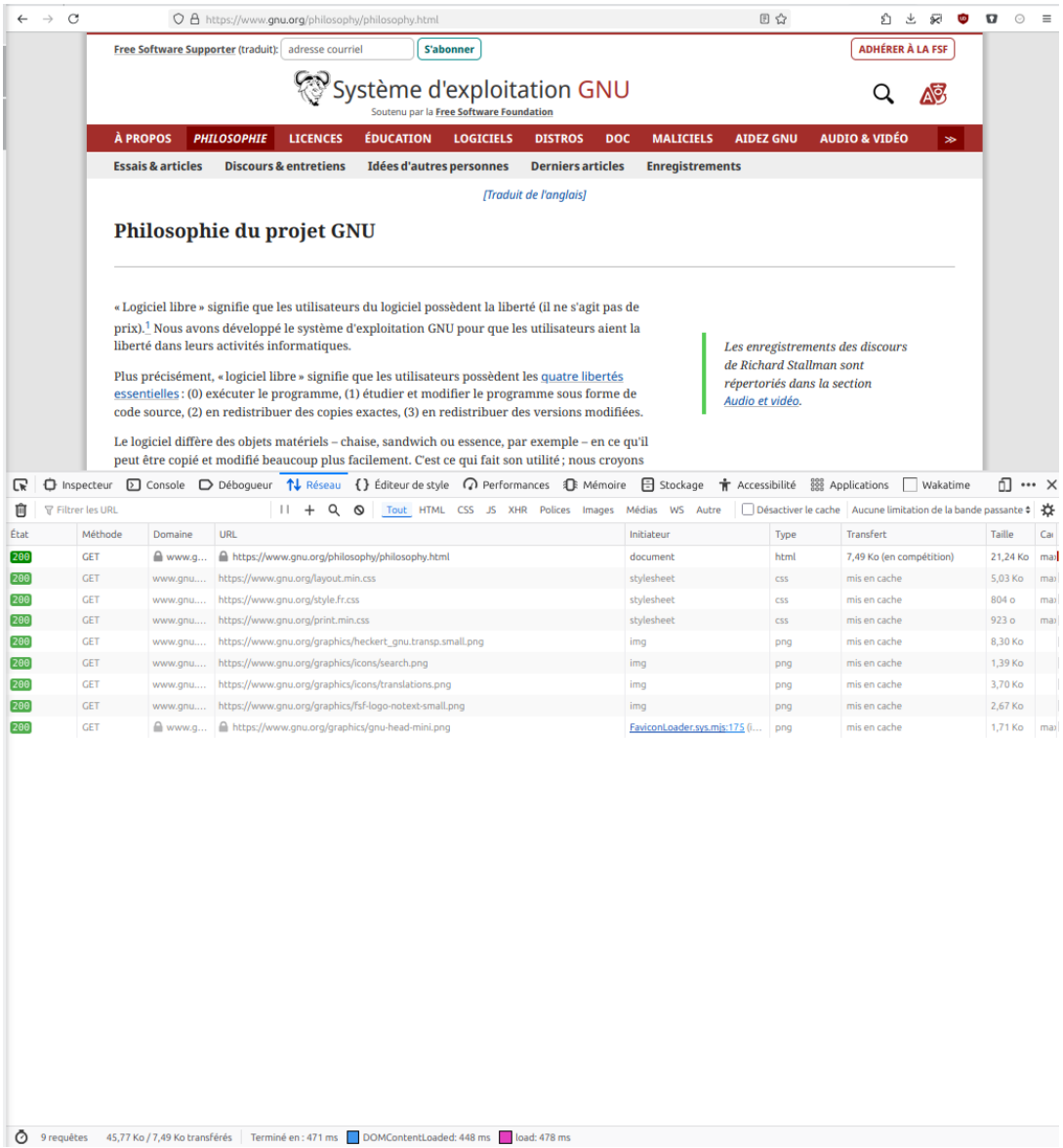
Programme

- SNT : thème Internet
- NSI 1ère : intéraction client/serveur, requêtes HTTP, réponses du serveur

16. Requêtes

16.1. Chargement d'une page et inspection des requêtes

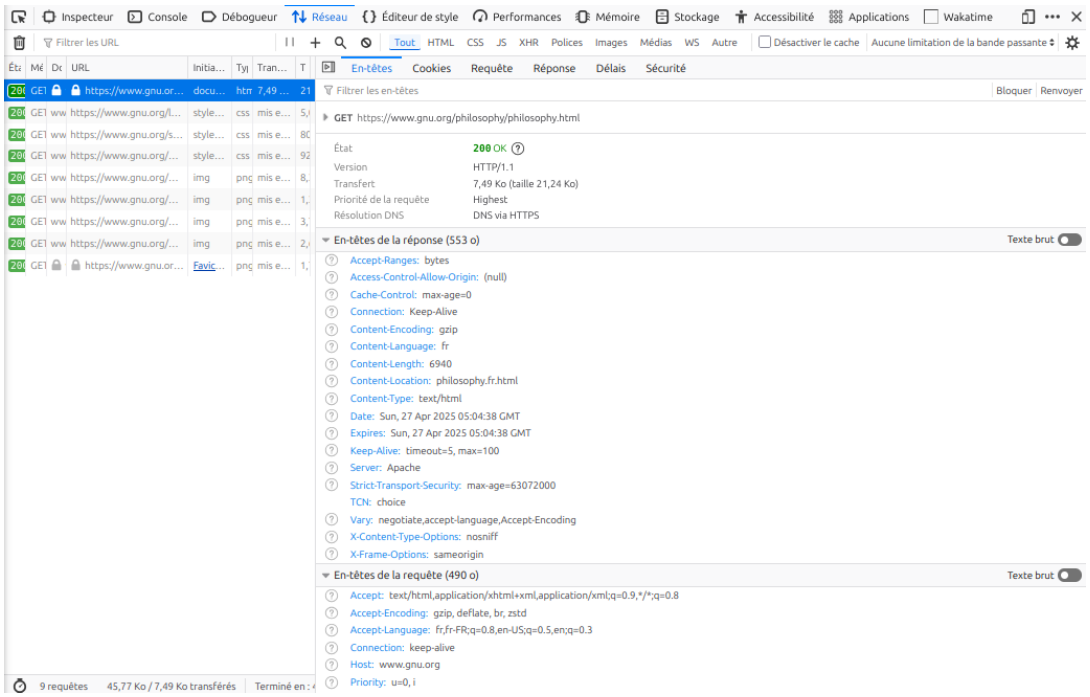
En prenant un exemple sur un chargement de page https://www.gnu.org/philosophy/philosophy.html, nous allons voir comment fonctionne une requête HTTP en utilisant les outils d'inspection du navigateur : F12 > Network.



On peut observer que le chargement de cette page nécessite plusieurs requêtes successives ou simultanées.

16.2. Observation d'une requète

Nous allons nous intéresser plus particulièrement à la première requète.



L'affichage indique à la fois la requète et la réponse.

Dans le protocole HTTP, la requète est faite sous forme d'un texte "spécial". Une première partie (jusqu'à la première ligne vide) constitue les en-têtes, et la deuxième partie constitue les données.

Cette première partie d'en-têtes est recopiées ci-dessous :

```
GET /philosophy/philosophy.html HTTP/1.1
Host: www.gnu.org
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:136.0) Gecko/20100101
Firefox/136.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate, br, zstd
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Priority: u=0, i
```

16.2.1. Première ligne

C'est une requète du **client**, qui part donc de notre navigateur, en direction de son destinataire. Mais à qui est envoyée cette requète ?

16.2.2. URL

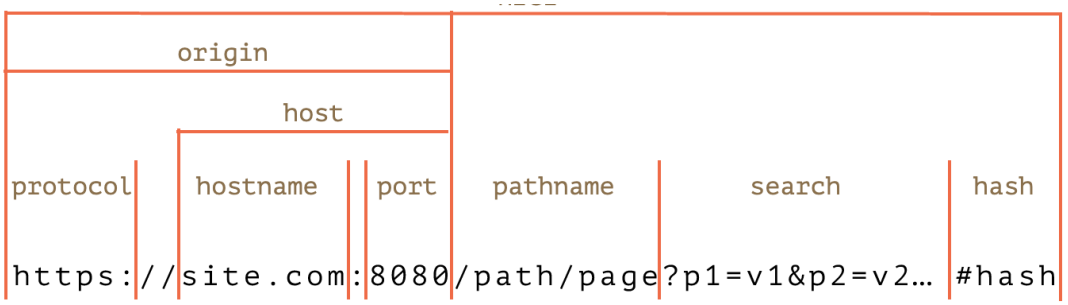
Petit rappel sur les URL (vu précédemment):

l'URL https://www.gnu.org/philosophy/philosophy.html peut être décomposée en

plusieurs partie :

- `https://` indique le protocole (protocol), sécurisé ou non (le s), utilisé pour se connecter au site
- `www.gnu.org` est le nom de domaine (hostname), que le **DNS** (Domain Name System) traduit en une adresse IP
- `/philosophy/philosophy.html` est le chemin (path / pathname)

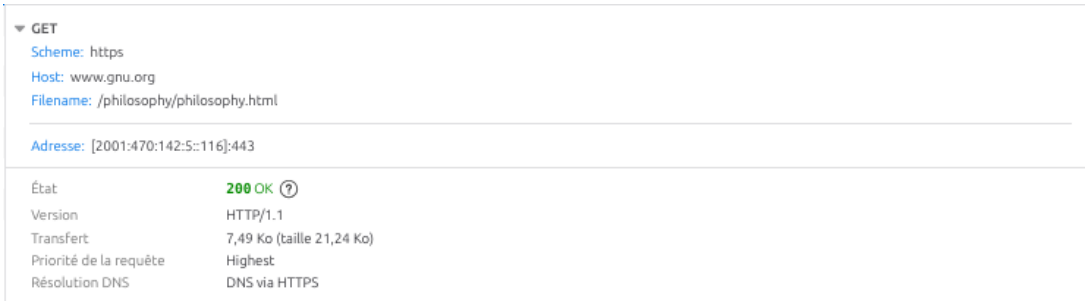
Une url peut aussi contenir d'autres éléments : le port, une partie "search" ou "query string", un "hash"...



La requête est donc envoyé vers l'adresse IP correspondant au *hostname*, sur le port précisé dans l'url, et si aucun port n'est précisé (ce qui est le cas la plupart du temps), vers le port 80 pour le http et 443 pour le https.

Ici, la requête est donc envoyée vers l'IPv6 [2001:470:142:5::116] sur le port 443 (on peut le voir dans la partie General du détail) :

```
Addresse: [2001:470:142:5::116]:443
```



La première ligne de la requête :

```
GET /philosophy/philosophy.html HTTP/1.1
```

indique

- la **méthode** : GET
- le chemin (pathname) : /philosophy/philosophy.html
- le protocole utilisé et sa version (ici HTTP version 1.1)

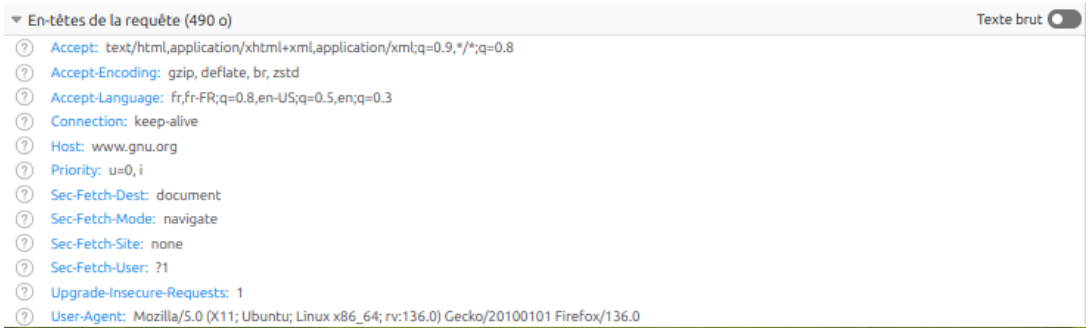
16.2.3. Méthode

La méthode est en fait une commande spécifiant un type de requête, c'est-à-dire qu'elle demande au serveur d'effectuer une action sur la ressource indiquée juste ensuite. Les différentes méthodes disponibles sont GET, POST, HEAD, OPTIONS, DELETE, PATCH, PUT et plus rarement CONNECT et TRACE.

Une requête avec la méthode **GET** est une demande pour récupérer une ressource; de manière générale c'est une méthode qui doit correspondre à une opération *idempotente* (qui peut être répétée plusieurs fois sans que cela aie un effet)

16.2.4. En-têtes

D'autres informations sont transmises au serveur dans les en-têtes; toutes ces informations sont optionnelles, la seule obligatoire est cette première ligne.



On peut reconnaître :

- **Host** : quel était l'hôte (nom de domaine et port) à qui la requête a été envoyé
- **Accept** : quel(s) types de document le navigateur accepte de recevoir en réponse à cette requête, éventuellement avec un ordre de priorité (le `q=`)
- **Accept-Encoding** : le navigateur accepte (ou non) les données compressées
- **Accept-Language** : quelles sont la ou les langues préférées par l'utilisateur de ce navigateur (avec un ordre de priorité donné par le `q=`)
- **Cache-Control** : indications pour garder ou non la réponse en "cache" (mémoire locale ou sur le réseau)
- **User-Agent** : une chaîne de caractères indiquant le système d'exploitation de la machine faisant tourner le navigateur, des infos sur le navigateur et sa version.
- ...

Programme

- SNT : thème Internet
- NSI 1ère : intéraction client/serveur, requêtes HTTP, réponses du serveur

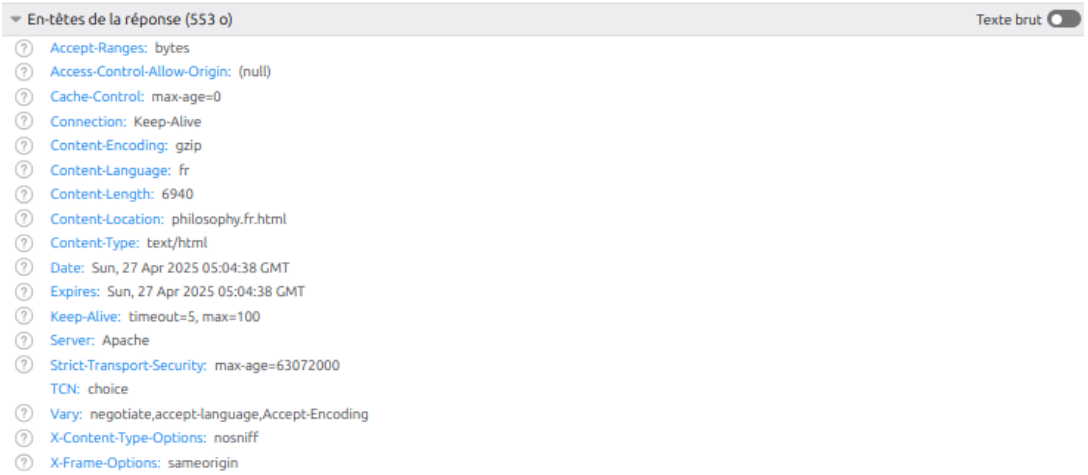
17. Réponse du serveur

Le serveur est une machine qui fait tourner un programme (service) qui écoute sur une adresse IP et un port, et qui répond aux demandes des clients.

La réponse commence par un ou plusieurs en-têtes, puis une ligne blanche et enfin les données de la réponse.

17.1. Observaton d'une réponse

On peut observer cette réponse dans l'inspecteur :



ou sous sa forme brute :

```
HTTP/1.1 200 OK
Date: Sun, 27 Apr 2025 05:04:38 GMT
Server: Apache
Content-Location: philosophy.fr.html
Vary: negotiate,accept-language,Accept-Encoding
TCN: choice
Strict-Transport-Security: max-age=63072000
X-Frame-Options: sameorigin
X-Content-Type-Options: nosniff
Access-Control-Allow-Origin: (null)
Accept-Ranges: bytes
Cache-Control: max-age=0
Expires: Sun, 27 Apr 2025 05:04:38 GMT
Content-Encoding: gzip
Content-Length: 6940
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
Content-Language: fr
```

17.1.1. Code de statut

La première ligne indique la version du protocole utilisé (HTTP version 1.1 ici), un code (ici 200) et un état (OK)

- Les codes **1xx** indiquent une information
- Les codes **2xx** indiquent un succès
- Les codes **3xx** indiquent une redirection
- Les codes **4xx** et **5xx** indiquent une erreur

Liste détaillée sur https://fr.wikipedia.org/wiki/Liste_des_codes_HTTP

17.1.2. Autres en-têtes

Le reste des en-tête donne des informations sur le contenu renvoyé et la manière dont il doit être traité :

- **Content-Type**: type de contenu
- **Date**: date et heure de la réponse
- **Content-Length**: taille du contenu en octets
- **Content-Language**: information sur la langue du contenu
- ...

17.2. Modes de fonctionnement

On peut distinguer deux "modes" de fonctionnement :

- soit le serveur renvoie le contenu de fichiers présents "physiquement" sur la machine
- soit le serveur renvoie un contenu qu'il génère au vol, dynamiquement

17.2.1. Serveur de fichiers

Un serveur de fichiers va associer un *chemin* à un *répertoire* :

Exemple : sur un serveur écoutant sur le port http par défaut (80) d'une machine dont l'ip est correspond au domaine `example.com`, on associe le chemin `/static/` est associé au répertoire `/var/www/html/`

Dans ce cas

- la requête `http://example.com/static/image.jpg` cherche et renvoie l'image `/var/www/html/image.jpg`
- la requête `http://example.com/static/test/toto.txt` cherche et renvoie le fichier texte `/var/www/html/test/toto.txt`

si un fichier n'existe pas avec le chemin correspondant, le serveur renvoie un code d'erreur 404 (Not found - Non trouvé)

17.2.2. Contenu dynamique

Le serveur va associer un morceau de programme (une fonction, par exemple) à un

chemin ou groupe de chemins donnés, et renverra le résultat de ce programme.

Exemple : sur un serveur écoutant sur le port http par défaut (80) d'une machine dont l'ip est correspond au domaine `example.com`, on associe le chemin `/bonjour` à une fonction bonjour() qui retourne la chaîne de caractères "hello world"

- la requête `http://example.com/bonjour` renverra donc "hello world"
- la requête `http://example.com/boujour` renverra une erreur 404

Nous allons voir plus tard comment réaliser ces deux fonctions (serveur de fichiers, contenu dynamique) en utilisant une librairie python appelée Bottle.

18. Architecture

Une application web peut être développé en utilisant plusieurs architectures :

- rendu côté serveur (Server-Side Rendering - SSR)
- rendu côté client (Client-Side Rendering - CSR) - application à page unique (Single Page Application - SPA)
- application hybride : hydratation (SSR + CSR)

Le choix de l'architecture dépend de plusieurs facteurs.

18.1. Rendu côté serveur (SSR)

Modèle "traditionnel" : le serveur génère l'intégralité du HTML pour chaque requête utilisateur. Le navigateur reçoit une page HTML complète et l'affiche directement. Chaque intéraction avec le serveur renvoie une nouvelle page HTML.

Avantages :

- **SEO Amélioré** : Les moteurs de recherche peuvent facilement indexer le contenu HTML.
- **Temps d'affichage initial rapide** (First Contentful Paint) : Le navigateur reçoit directement du contenu affichable.
- **Compatibilité** : Fonctionne sur tous les navigateurs, même anciens, sans nécessiter JavaScript.

Inconvénients :

- **Charge serveur plus élevée** : Le serveur doit générer le HTML pour chaque requête, ce qui peut être coûteux en ressources.
- **Moins interactif** : Chaque interaction nécessite une nouvelle requête au serveur, ce qui peut entraîner des rechargements de page.
- **Moins flexible** : La logique de présentation est étroitement liée au serveur, données et présentation sont mélangées.

Technologies courantes :

- PHP (Laravel, Symfony)
- Python (Django, Flask, Bottle)
- Ruby (Ruby on Rails)
- Java (Spring)

18.2. Rendu côté client (CSR) - Single Page Application (SPA) + API

Modèle plus "moderne" : le navigateur fait le travail de génération de la page, en fonction

des données récupérées sur le serveur via une **API** (application programming interface). Présentation et données sont séparées.

Fonctionnement : le navigateur télécharge une page HTML unique qui contient le squelette de l'application et le code JavaScript nécessaire. Les interactions de l'utilisateur sont gérées par JavaScript, qui met à jour dynamiquement le contenu de la page en utilisant des appels API au serveur pour récupérer ou enregistrer des données. Avantages :

- **expérience utilisateur fluide** : pas de rechargement de page pour les interactions, ce qui offre une expérience plus réactive et agréable, et la création d'interfaces plus dynamiques.
- **rapidité** : une fois la page initiale chargée, les requêtes API sont généralement plus rapides que les rechargements de page complets.
- **séparation des logiques** : la logique de présentation (JavaScript) est séparée de la logique métier (API), ce qui facilite le développement et la maintenance.
- **facilité de maintenance** : l'API et l'interface utilisateur peuvent être développées et maintenues indépendamment.

Inconvénients :

- **SEO (Search Engine Optimization) potentiellement plus difficile :** les moteurs de recherche peuvent avoir du mal à indexer le contenu généré dynamiquement par JavaScript (bien que certains moteurs de recherche puisse indexer les pages dynamiques).
- **temps d'affichage initial plus long :** le téléchargement initial de la page unique et de tout le code JavaScript peut prendre du temps, impactant le temps d'affichage initial, l'expérience utilisateur et le SEO.
- **dépendance JavaScript :** nécessite JavaScript activé dans le navigateur.

Technologies Courantes :

- Frameworks JavaScript/Typescript (React, Angular, Vue.js) pour le frontend
- Tout langage serveur pour l'API (Python, Node.js, Java, PHP, etc.)

18.3. Hydratation (SSR + Client-Side Rendering)

Fonctionnement : combine les avantages du rendu côté serveur (SSR) et du rendu côté client (SPA). Le serveur génère initialement le HTML de la page, comme dans le SSR. Ensuite, le code JavaScript est téléchargé et "hydrate" le HTML, prenant le contrôle de l'interaction avec l'utilisateur et gérant les mises à jour dynamiques comme dans une SPA.

Avantages :

- **meilleur SEO (Search Engine Optimization) :** le contenu est initialement indexable par les moteurs de recherche.
- **temps d'affichage initial rapide :** le navigateur reçoit du contenu affichable rapidement grâce au SSR.
- **expérience utilisateur fluide :** les interactions sont gérées de manière réactive par le JavaScript côté client.

Inconvénients :

- **complexité :** plus complexe à mettre en œuvre que le SSR ou la SPA.
- **charge serveur :** nécessite une capacité serveur pour le rendu initial.
- **temps d'hydratation :** un certain temps est nécessaire pour que JavaScript prenne

le contrôle de l'application, ce qui peut être perceptible pour l'utilisateur.

Technologies Courantes :

- La plupart des frameworks JavaScript modernes prennent en charge l'hydratation, y compris React, Vue.js et Angular.

18.4. Choix

Il n'y a pas de "meilleure" approche universelle. Le choix dépend du contexte. Pour les applications axées sur le contenu et nécessitant un bon référencement, le SSR ou l'hydratation sont de bons choix. Pour les applications très interactives où l'expérience utilisateur est primordiale, une SPA + API peut être plus appropriée. L'hydratation offre un bon compromis entre les deux, mais au prix d'une complexité accrue.

Programme

- NSI 1ère : modalités d'intéraction entre l'homme et la machine
- NSI 1ère : intéraction client/serveur, requêtes HTTP, réponses du serveur

19. REST

Références :

- https://fr.wikipedia.org/wiki/Representational_state_transfer

Les API **REST** (Representational State Transfer) sont capitales dans le développement web moderne, en facilitant l'interconnexion entre différentes applications et services. Elles permettent à des applications d'échanger des données de manière standardisées, même si elles sont écrites dans des langages différents ou fonctionnent sur des plateformes distinctes.

19.1. Concept : Ressources et Représentations

Au cœur d'une API REST réside le concept de "ressource". Une ressource peut être n'importe quoi identifiable : un utilisateur, un produit, un article de blog, une commande, etc. Chaque ressource est identifiée par une **URL unique**, agissant comme son adresse sur le réseau. Exemple : `https://api.example.com/utilisateurs/123` où `123` est l'identifiant de l'utilisateur.

Une API REST ne transmet pas directement les données elles-mêmes. Elle transmet une *représentation* de ces données. Cette représentation peut prendre différentes formes, les plus courantes étant **JSON** (JavaScript Object Notation) et **XML**. Le choix du format de représentation est généralement spécifié dans les en-têtes de la requête et de la réponse.

19.2. Méthodes HTTP = Verbes d'Action

Les interactions avec une API REST s'effectuent à l'aide des méthodes HTTP standard. Ces méthodes définissent l'action à effectuer sur la ressource :

- **GET :** Récupérer une ressource. Par exemple, `GET /utilisateurs/123` pour obtenir les informations de l'utilisateur avec l'ID 123.
- **POST :** Créer une nouvelle ressource. Par exemple, `POST /utilisateurs` pour créer un nouvel utilisateur. Les données du nouvel utilisateur sont incluses dans le corps de la requête.
- **PUT :** Mettre à jour une ressource existante en remplaçant complètement son contenu. Par exemple, `PUT /utilisateurs/123` pour mettre à jour l'ensemble des informations de l'utilisateur 123.
- **PATCH :** Mettre à jour une ressource existante en modifiant partiellement son contenu. Par exemple, `PATCH /utilisateurs/123` pour modifier uniquement le nom de l'utilisateur 123.
- **DELETE :** Supprimer une ressource. Par exemple, `DELETE /utilisateurs/123` pour supprimer l'utilisateur 123.

19.3. Actions non REST

Bien que la méthode **POST** soit principalement associée à la création de nouvelles ressources, elle peut également être utilisée pour déclencher des actions spécifiques sur une ressource ou effectuer des opérations qui ne s'inscrivent pas dans les verbes standard. Par exemple, `POST /utilisateurs/123/messages` pourrait être utilisé pour envoyer un message à l'utilisateur 123, sans pour autant créer une nouvelle ressource "message" indépendante.

Cela permet de réaliser des opérations personnalisées ou complexes, ou de lancer des processus asynchrones liés à une ressource existante, en exploitant le corps de la requête pour transmettre les paramètres nécessaires à l'action. Il est important de documenter clairement ces utilisations non conventionnelles de POST pour assurer la clarté de l'API.

Dans la pratique, cette approche permet de gérer des opérations à la manière d'une API RPC (Remote Procedure Call), où l'on peut appeler des fonctions distantes en utilisant des requêtes HTTP. Même si "c'est pas bien", cela peut être indispensable dans beaucoup de situations.

19.4. Principes Clés de REST

L'architecture REST s'appuie sur plusieurs principes pour garantir son efficacité et sa scalabilité :

- **Stateless (Sans État)** : chaque requête du client au serveur doit contenir toutes les informations nécessaires pour être traitée. Le serveur ne conserve aucune information sur les requêtes précédentes.
- **Cacheable (Mise en Cache)** : les réponses du serveur doivent être explicitement définies comme pouvant être mises en cache ou non, ce qui permet d'améliorer les performances.
- **Uniform Interface (Interface Uniforme)** : l'utilisation des méthodes HTTP standard (GET, POST, PUT, DELETE) et d'une structure d'URL cohérente permet de simplifier l'interaction avec l'API.

19.5. Avantages des APIs REST

- **simplicité** : facile à comprendre et à utiliser grâce à l'utilisation des méthodes HTTP standard.
- **documentation** : les API REST peuvent être facilement documentées (voire auto-documentées) à l'aide d'outils comme Swagger ou OpenAPI, facilitant la compréhension et l'utilisation par les développeurs.
- **scalabilité** : l'absence d'état du serveur permet une scalabilité horizontale facile.
- **flexibilité** : supporte différents formats de données et peut être utilisée avec différents langages de programmation.
- **interopérabilité** : permet l'intégration facile entre différentes applications et services.

Les APIs REST constituent une approche puissante et largement adoptée pour la création d'applications web modernes, favorisant l'interopérabilité et la modularité. Leur utilisation simplifie l'intégration de services tiers et permet la construction d'architectures distribuées robustes et évolutives.

Programme

- NSI 1ère : intéraction client/serveur, requêtes HTTP, réponses du serveur

20. Créer un serveur (bottle)

Dans ce chapitre, nous allons apprendre à créer et programmer un micro serveur web.
Nous allons utiliser pour cela le micro framework python bottle (http://bottlepy.org/docs/stable/), qui a l'avantage d'être assez simple à mettre en oeuvre et à utiliser.

20.1. Prérequis

Obligatoire :

- python 3
- éditeur de texte

Recommandé :

- IDE (vscode, pycharm, autre..)
- `pip` ou `uv`

Si besoin, un IDE en ligne est disponible à l'adresse https://xxxxx-code.formation.devel.space, `xxxxx` devant être remplacé par votre id.

20.2. Étapes

Nous découperons notre premier projet de découverte en plusieurs étapes :

- Installation de l'environnement de développement (si nécessaire)
- Premier serveur "hello world"
- Notion de routes
- Fichiers statiques
- Utilisation de templates
- Gestion des formulaires

21. Prélude : environnement de développement

21.1. Window$

Installation de `python` + `uv` + `vscode` + `extensions vscode python`.

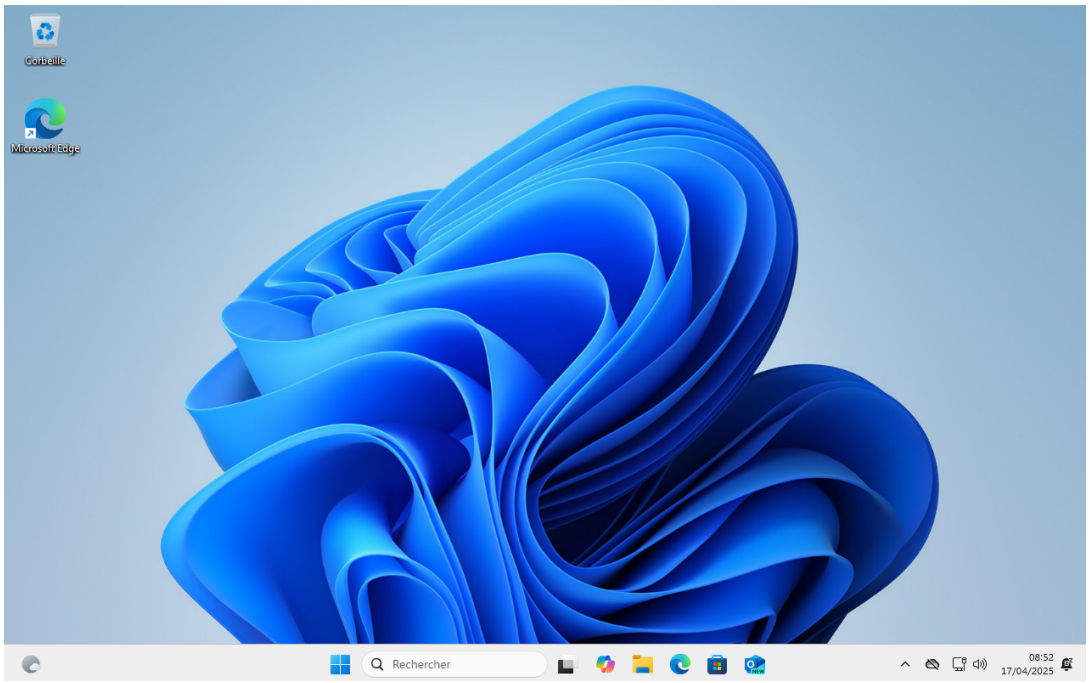
21.1.1. Installation de python

Ouvrir powershell et taper `python`. Si python n'est pas installé, ça lancera le M$ Store pour l'installer.

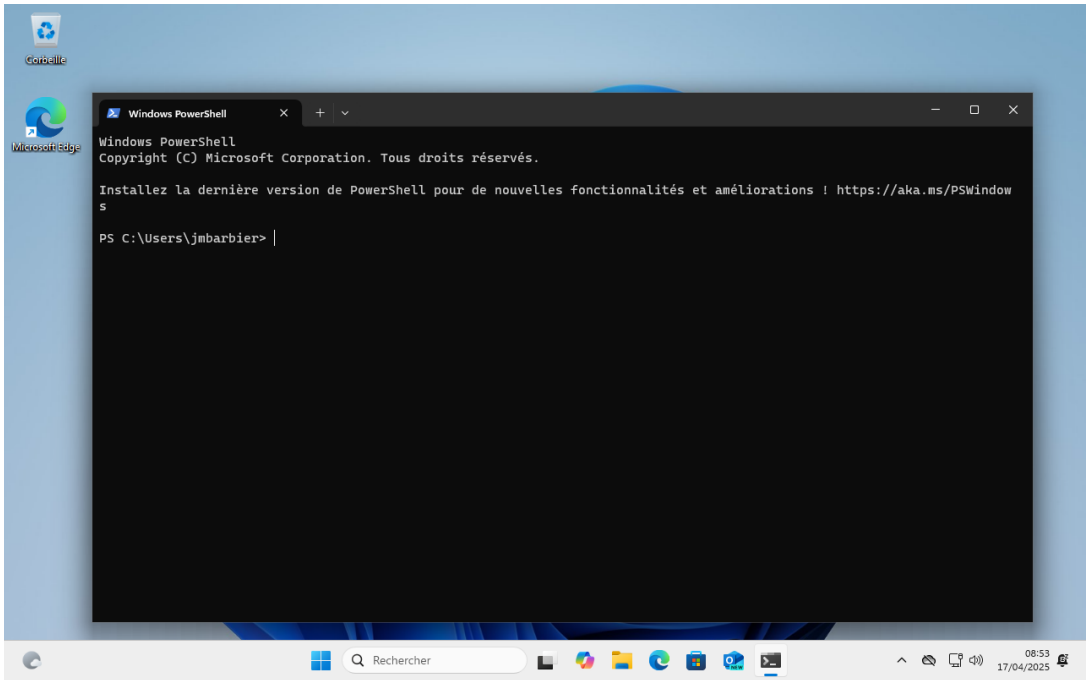
Une fois installé, vérifier l'installation avec la commande `python --version`.

Les étapes :

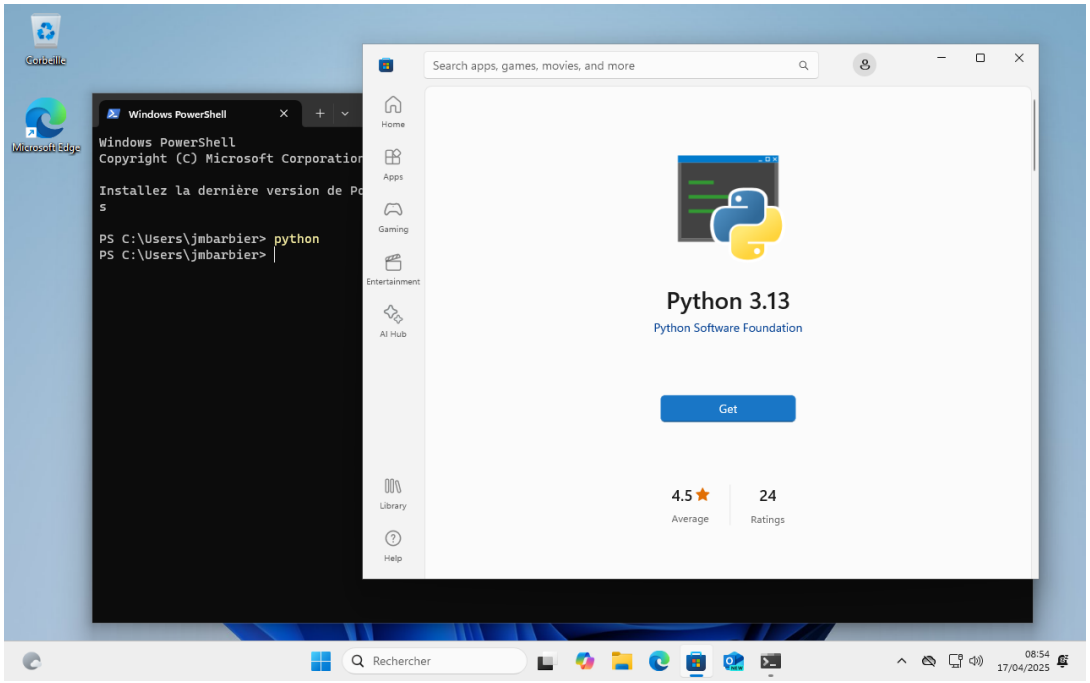
Lancement powershell



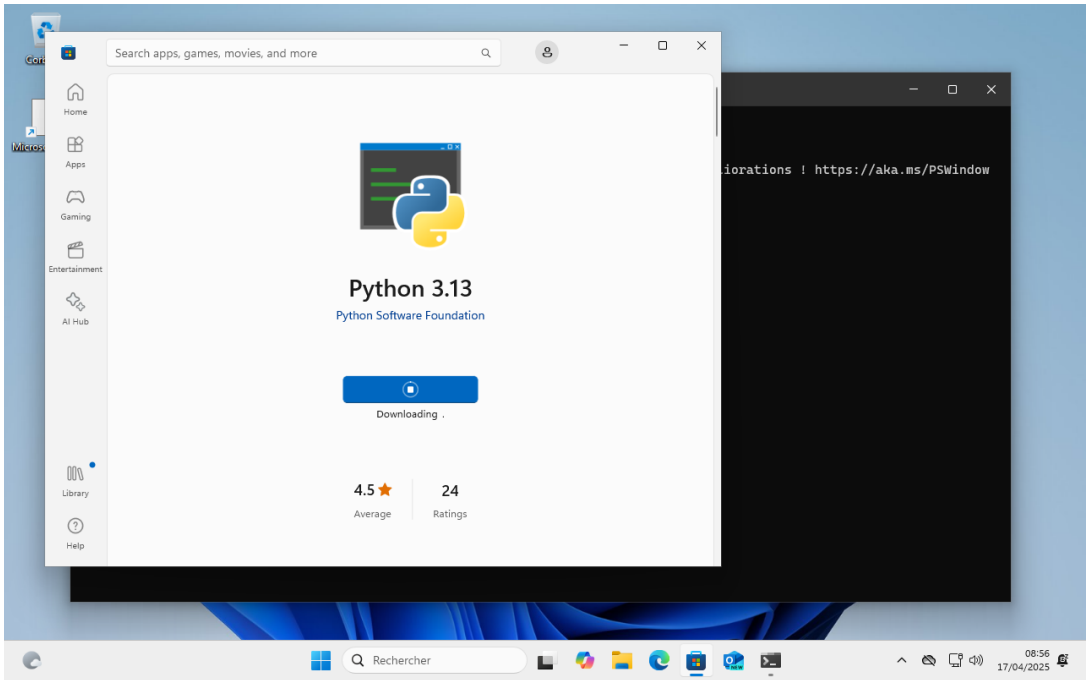
Commande `python`



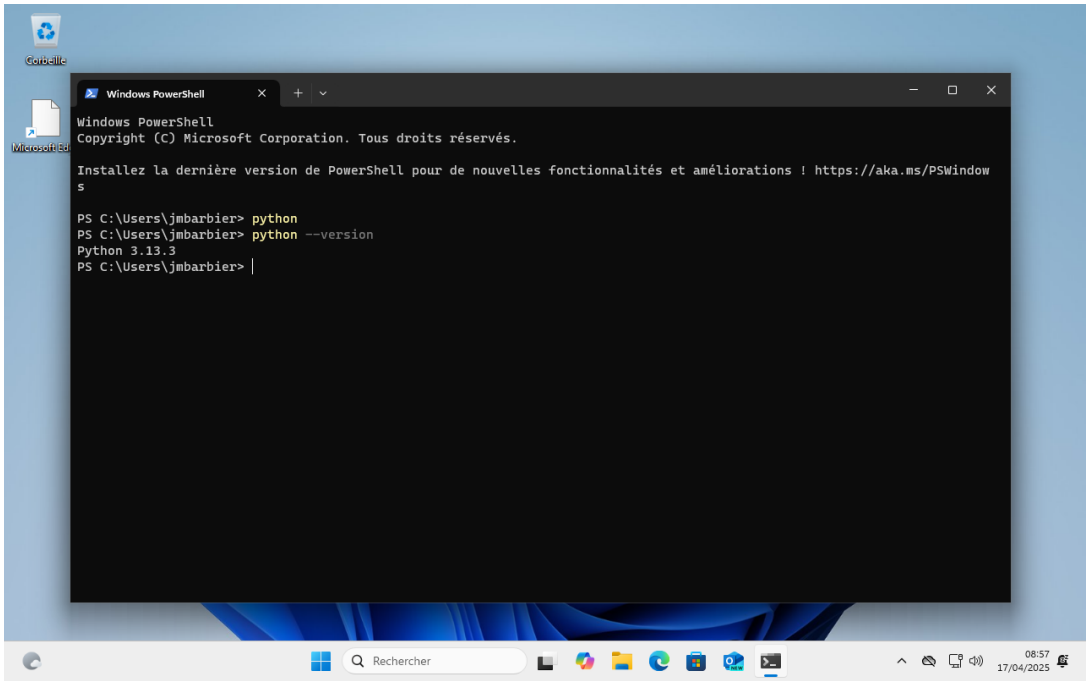
Ouverture du M$ Store



Installation de python



Vérification de l'installation



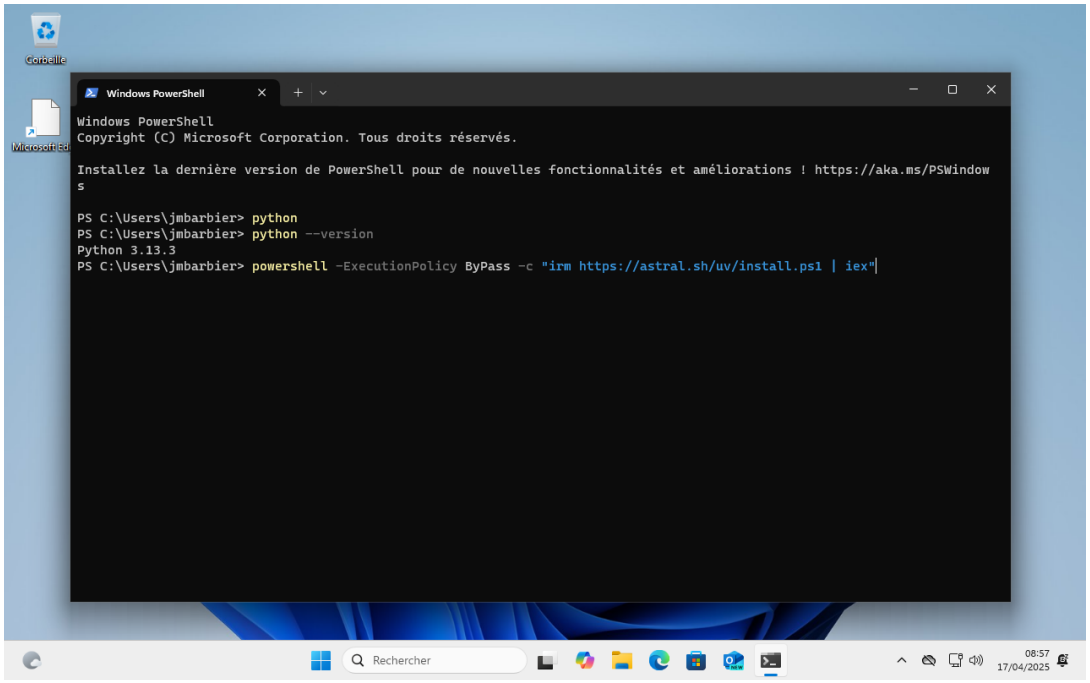
21.1.2. Installation de uv

`uv` est un gestionnaire de paquets pour python. Il permet de gérer l'ensemble des dépendances d'un projet python, ainsi que les différentes versions de python possibles. Il est très rapide, et permet de créer des environnements virtuels pour chaque projet.

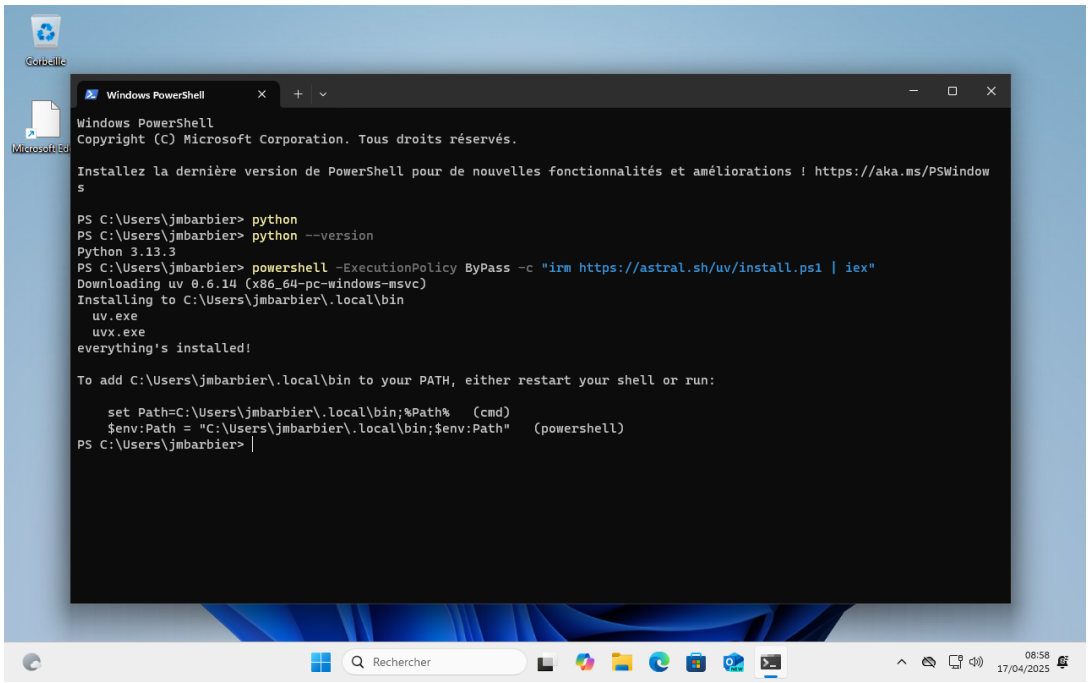
La commande d'installation est sur https://docs.astral.sh/uv/getting-started/installation/, à copier/coller dans le terminal.

```
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 |
iex"
```

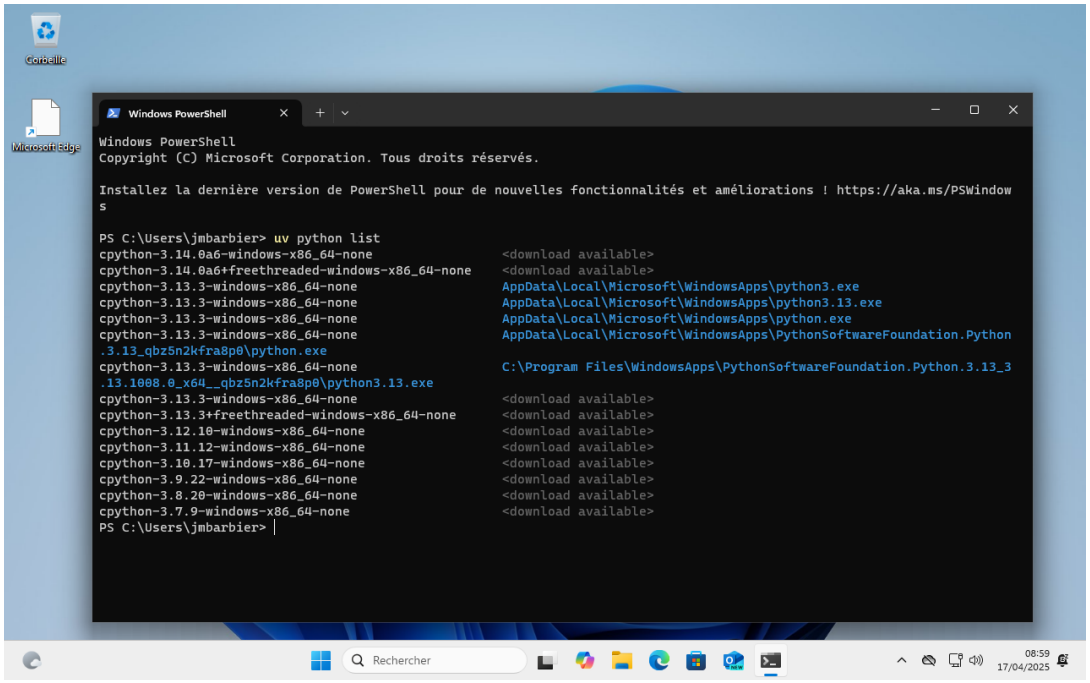
Lancement de la commande



Installation réussie. Relancer ensuite un powershell pour que les changements soient pris en compte.



Vérification de l'installation : `uv python list` liste les versions de python installées sur le système (et permet de les installer avec `uv python install ...`).



Nous pouvons maintenant créer un premier projet pour tester si tout fonctionne :

```
mkdir serveur
cd serveur
uv init
ls
```

ou bien encore

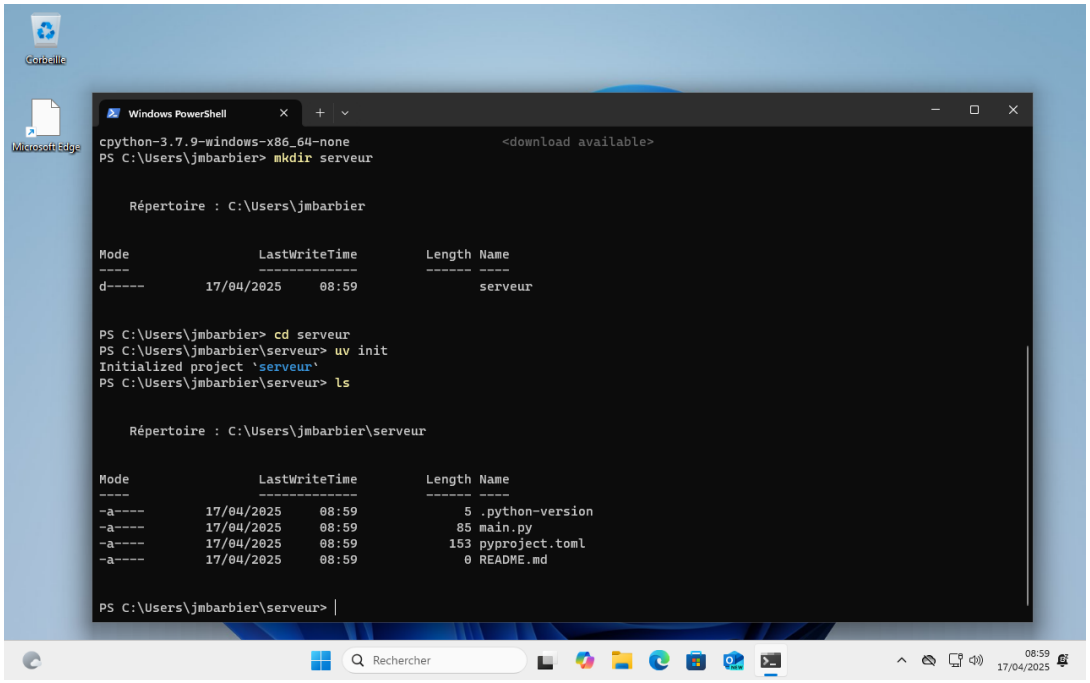
```
uv init serveur
cd serveur
ls
```

La commande `uv init` va créer les fichiers :

- `pyproject.toml` : fichier de configuration du projet
- `.python-version` : fichier de version de python
- `README.md` : fichier de documentation du projet
- `main.py` : fichier principal du projet

Si `git` est détecté sur le système, un dépôt git sera créé et un fichier `.gitignore` sera ajouté.

Fichiers créés par `uv init` sur un window$ sans `git` :

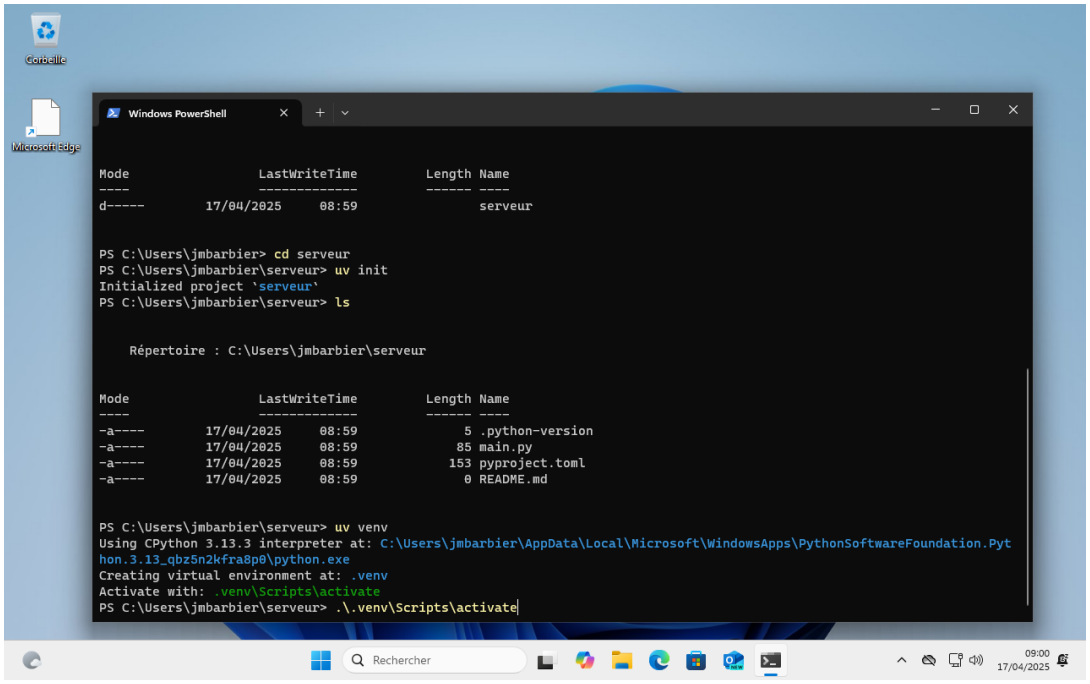


Nous pouvons maintenant utiliser `uv venv` pour créer un environnement virtuel.

```
uv venv
```

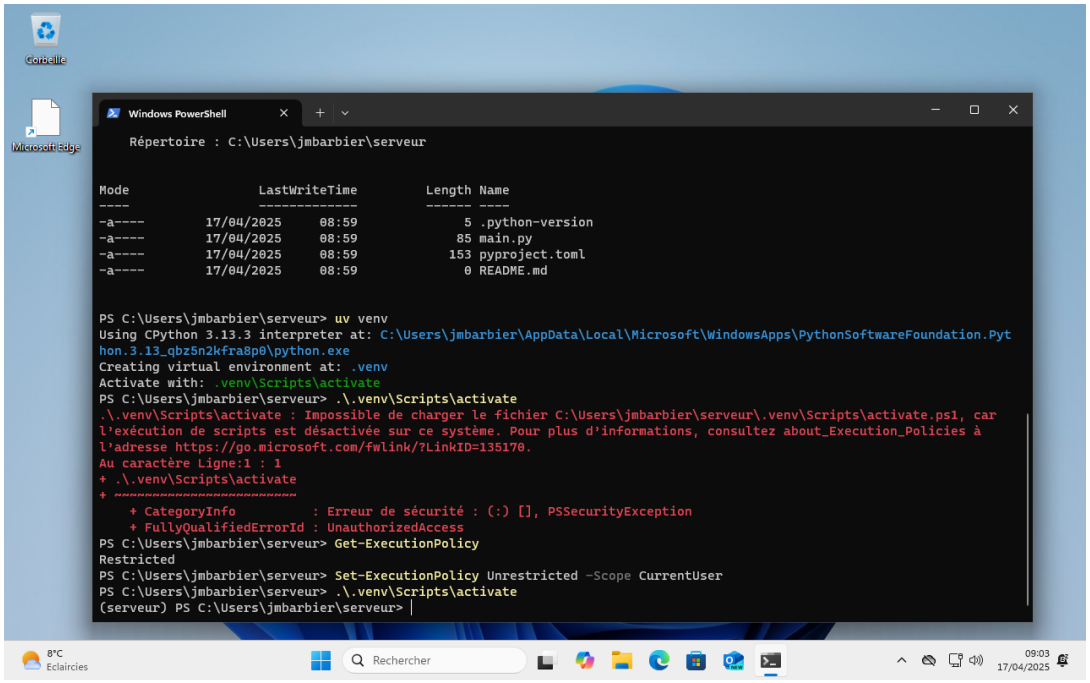
Un dossier `.venv` sera créé dans le répertoire du projet. Ce dossier contiendra l'environnement virtuel, avec les dépendances du projet.

Création de l'environnement virtuel



L'activation de l'environnement virtuel se fait avec la commande :

```
.\.venv\Scripts\activate
```



Selon la configuration des droits d'exécution de votre système, il se peut que la commande ne fonctionne pas et qu'un message d'erreur apparaisse, parlant d'ExecutionPolicy.

Vous pouvez vérifier quelle est votre politique d'exécution avec la commande suivante :

```
Get-ExecutionPolicy
```

Si la politique d'exécution est `Restricted`, vous ne pourrez pas exécuter de script. Il faudra donc la changer avec la commande suivante :

```
Set-ExecutionPolicy Unrestricted -Scope CurrentUser
```

ou bien (plus bourrin, moins sécurisé, mais recommandé par M$ (???))

```
Set-ExecutionPolicy Unrestricted -Force
```

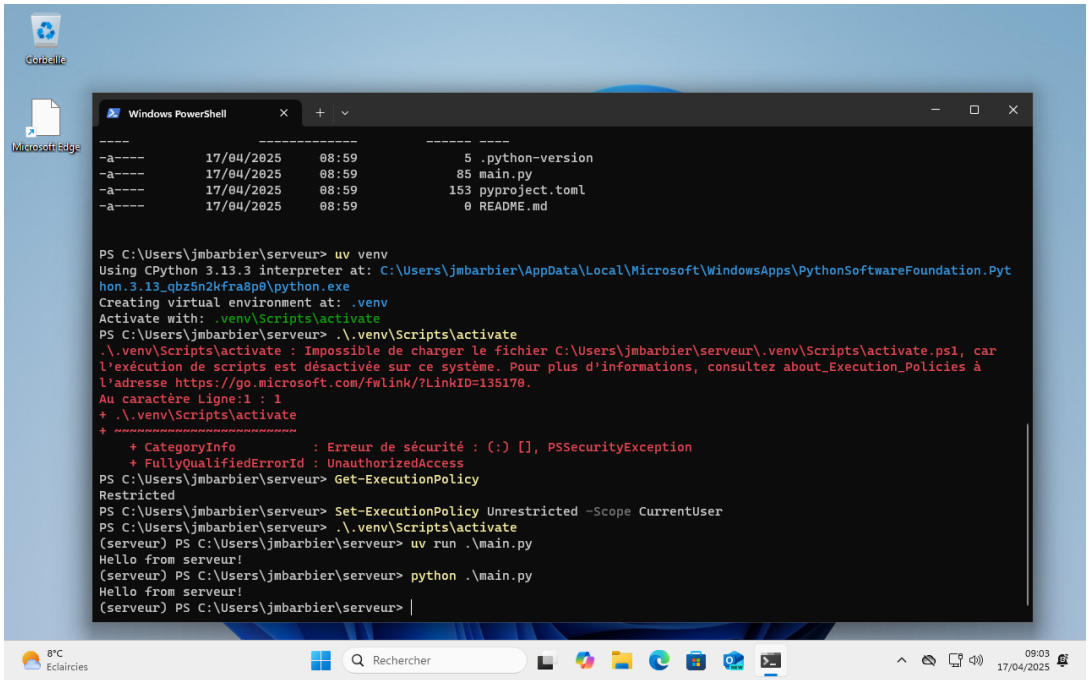
L'environnement virtuel peut alors être activé.

Pour lancer le programme, deux possibilités :

```
uv run main.py
```

ou bien

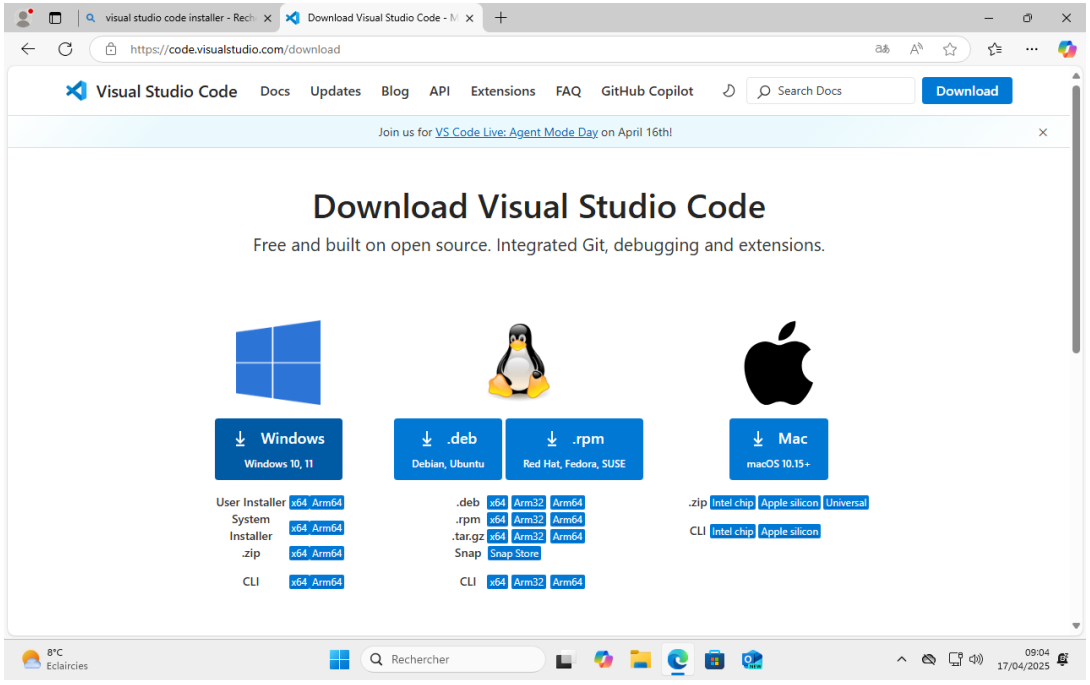
```
python main.py
```



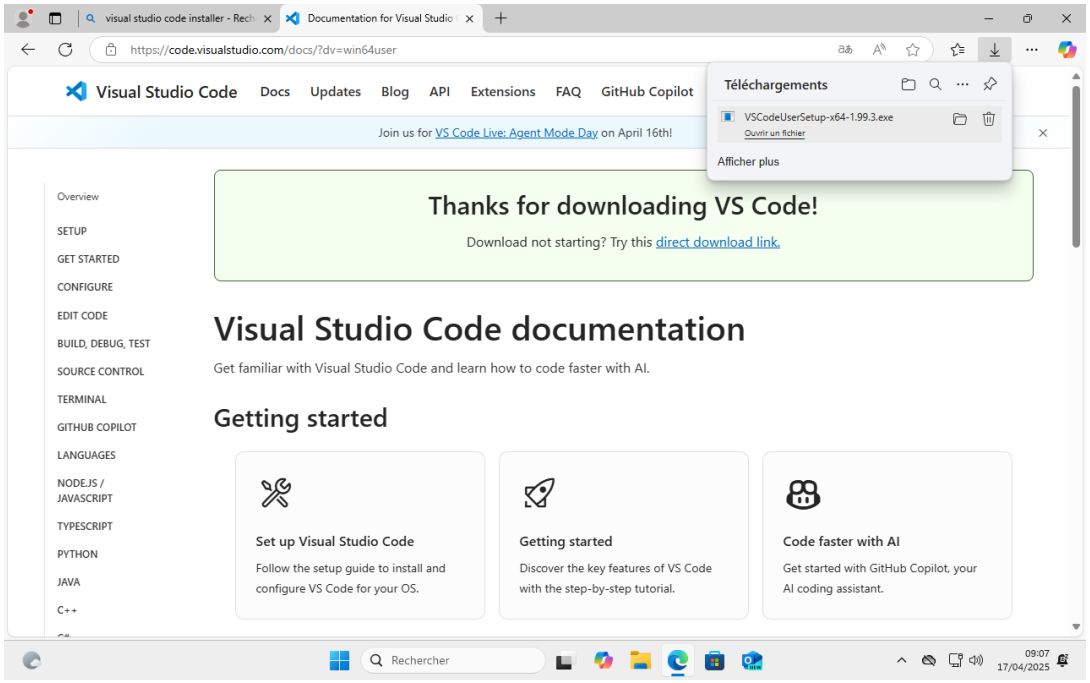
21.1.3. Installation de vscode

Télécharger et installer vscode sur https://code.visualstudio.com/. Lancer l'installation et suivre les étapes.

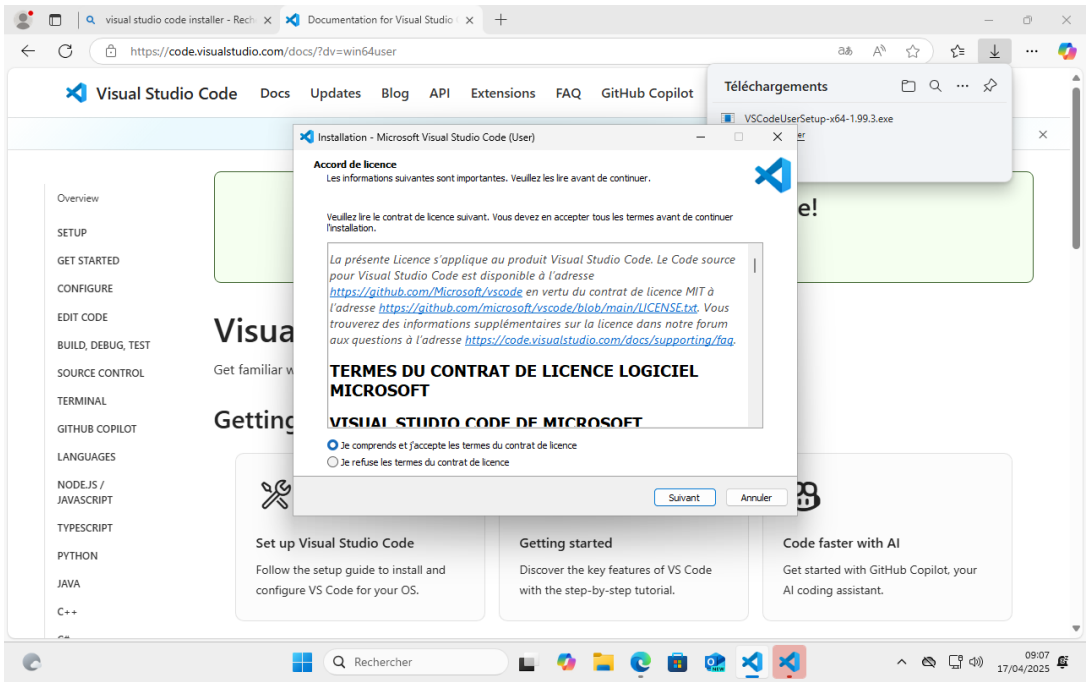
Téléchargement de vscode

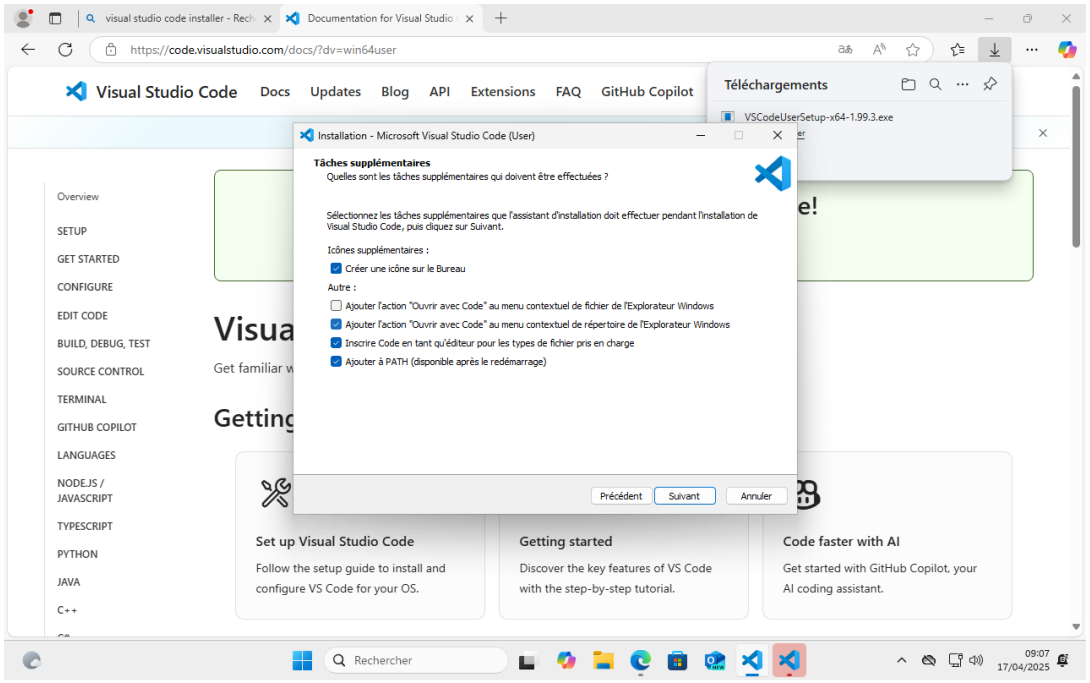


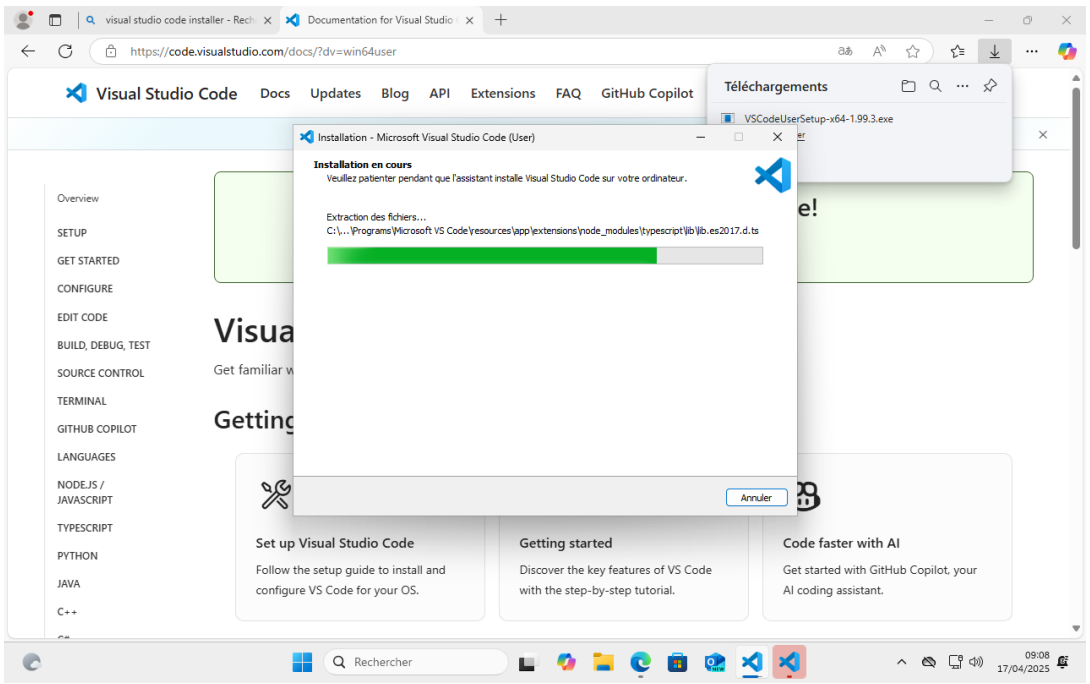
Lancement de l'installation

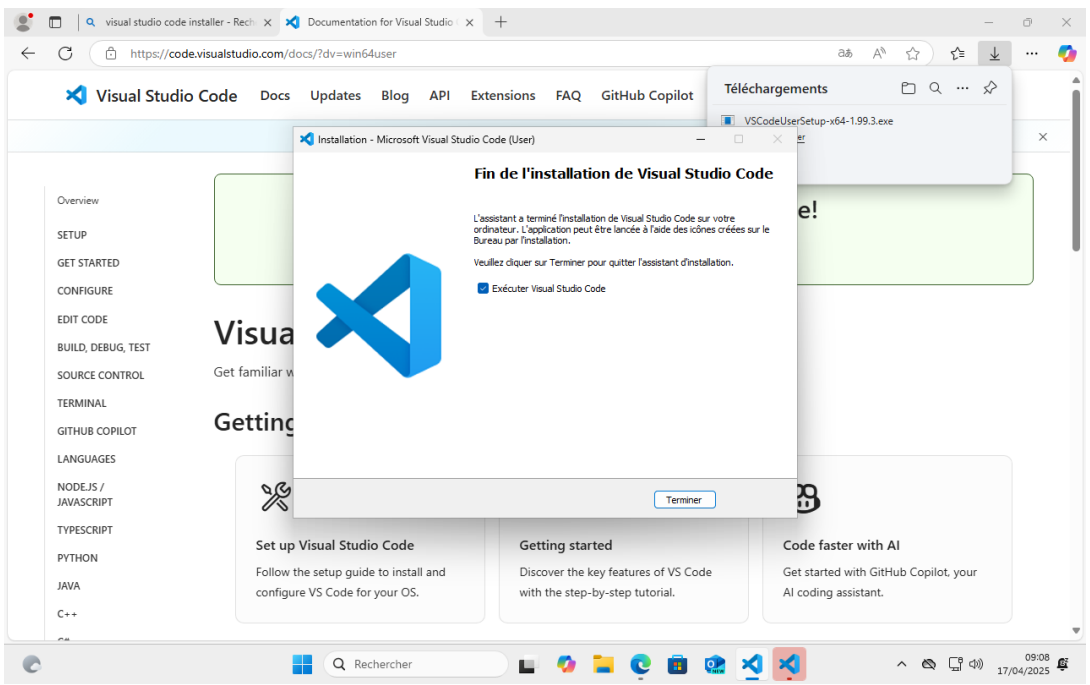


Paramètres recommandés : "Ajouter l'option "Open with Code" au menu contextuel de l'explorateur de fichiers pour les dossiers".

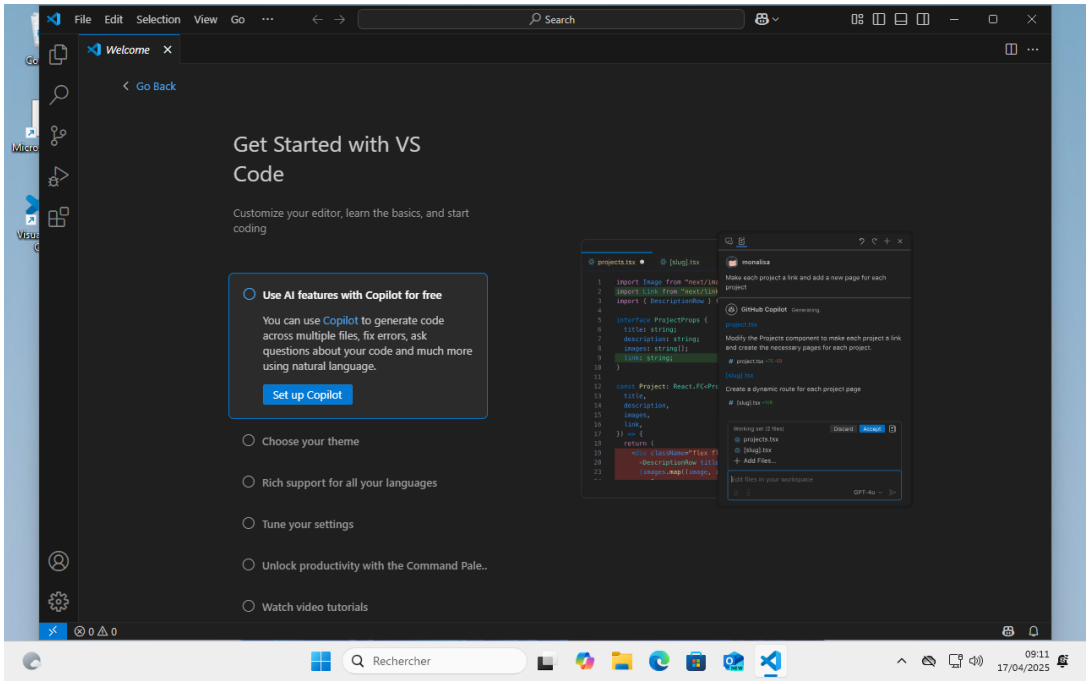



visual studio code installer - Rech
Documentation for Visual Studio
https://code.visualstudio.com/docs/?dv=win64user
Visual Studio Code
Docs
Updates
Blog
API
Extensions
FAQ
GitHub Copilot
Téléchargements
VSCodeUserSetup-x64-1.99.3.exe
Installation - Microsoft Visual Studio Code (User)
Tâches supplémentaires
Quelles sont les tâches supplémentaires qui doivent être effectuées ?
Sélectionnez les tâches supplémentaires que l'assistant d'installation doit effectuer pendant l'installation de Visual Studio Code, puis cliquez sur Suivant.
Icônes supplémentaires :
Créer une icône sur le Bureau
Autre :
Ajouter l'action "Ouvrir avec Code" au menu contextuel de fichier de l'Explorateur Windows
Ajouter l'action "Ouvrir avec Code" au menu contextuel de répertoire de l'Explorateur Windows
Inscrire Code en tant qu'éditeur pour les types de fichier pris en charge
Ajouter à PATH (disponible après le redémarrage)
Précédent
Suivant
Annuler
Overview
SETUP
GET STARTED
CONFIGURE
EDIT CODE
BUILD, DEBUG, TEST
SOURCE CONTROL
TERMINAL
GITHUB COPILOT
LANGUAGES
NODEJS / JAVASCRIPT
TYPESCRIPT
PYTHON
JAVA
C++
Set up Visual Studio Code
Follow the setup guide to install and configure VS Code for your OS.
Getting started
Discover the key features of VS Code with the step-by-step tutorial.
Code faster with AI
Get started with GitHub Copilot, your AI coding assistant.
Rechercher
09:07
17/04/2025

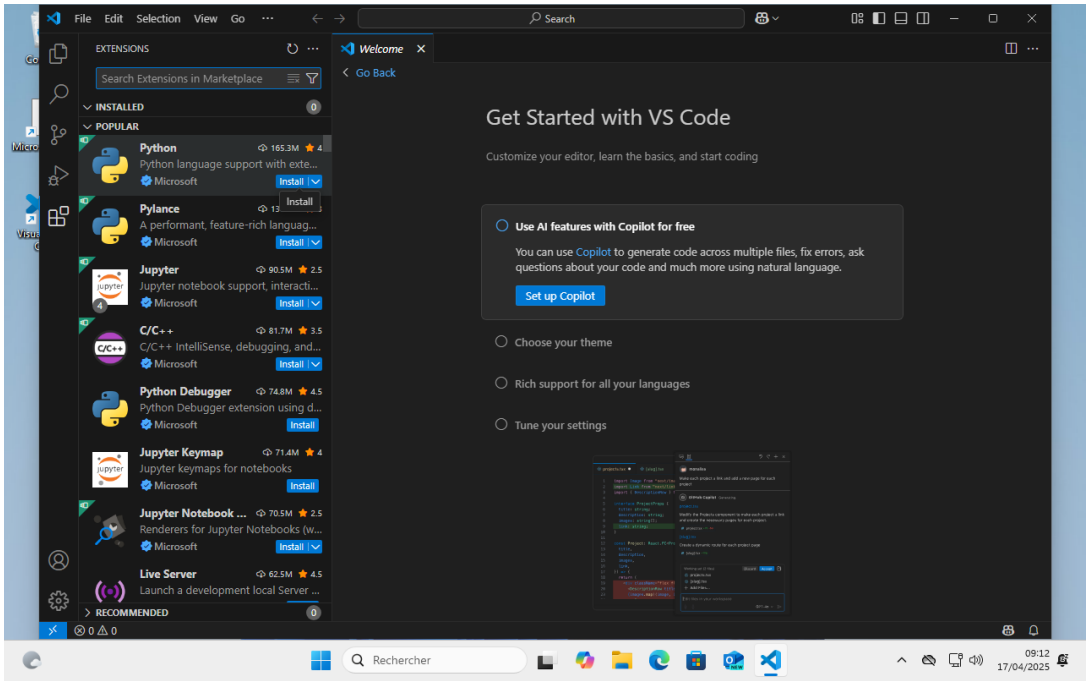

visual studio code installer - Rech
Documentation for Visual Studio
https://code.visualstudio.com/docs/?dv=win64user
Visual Studio Code
Docs
Updates
Blog
API
Extensions
FAQ
GitHub Copilot
Téléchargements
VSCodeUserSetup-x64-1.99.3.exe
Installation - Microsoft Visual Studio Code (User)
Installation en cours
Veuillez patienter pendant que l'assistant installe Visual Studio Code sur votre ordinateur.
Extraction des fichiers...
C:\...\Programs\Microsoft VS Code\resources\app\extensions\node_modules\typescript\lib\lib.es2017.d.ts
Annuler
Overview
SETUP
GET STARTED
CONFIGURE
EDIT CODE
BUILD, DEBUG, TEST
SOURCE CONTROL
TERMINAL
GITHUB COPILOT
LANGUAGES
NODEJS / JAVASCRIPT
TYPESCRIPT
PYTHON
JAVA
C++
Set up Visual Studio Code
Follow the setup guide to install and configure VS Code for your OS.
Getting started
Discover the key features of VS Code with the step-by-step tutorial.
Code faster with AI
Get started with GitHub Copilot, your AI coding assistant.
Rechercher
09:08
17/04/2025


visual studio code installer - Rech
Documentation for Visual Studio
https://code.visualstudio.com/docs/?dv=win64user
Visual Studio Code
Docs
Updates
Blog
API
Extensions
FAQ
GitHub Copilot
Téléchargements
VSCodeUserSetup-x64-1.99.3.exe
Installation - Microsoft Visual Studio Code (User)
Fin de l'installation de Visual Studio Code
L'assistant a terminé l'installation de Visual Studio Code sur votre ordinateur. L'application peut être lancée à l'aide des icônes créées sur le Bureau par l'installation.
Veuillez cliquer sur Terminer pour quitter l'assistant d'installation.
Exécuter Visual Studio Code
Terminer
Overview
SETUP
GET STARTED
CONFIGURE
EDIT CODE
BUILD, DEBUG, TEST
SOURCE CONTROL
TERMINAL
GITHUB COPILOT
LANGUAGES
NODEJS / JAVASCRIPT
TYPESCRIPT
PYTHON
JAVA
C++
Set up Visual Studio Code
Follow the setup guide to install and configure VS Code for your OS.
Getting started
Discover the key features of VS Code with the step-by-step tutorial.
Code faster with AI
Get started with GitHub Copilot, your AI coding assistant.
Rechercher
09:08
17/04/2025

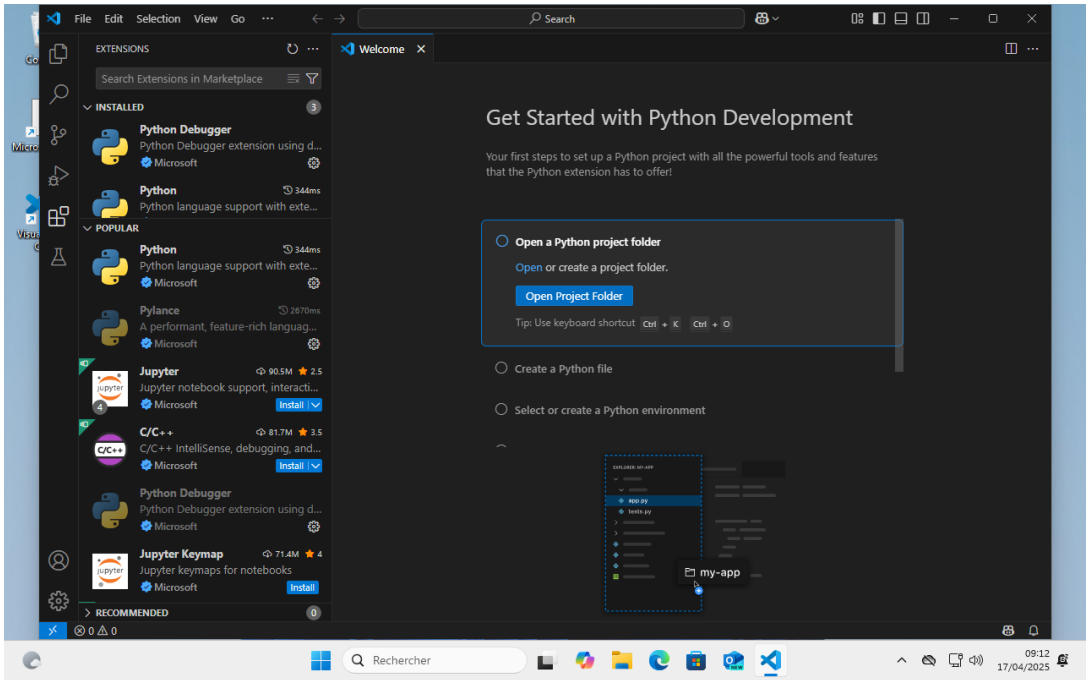
Lancement de VSCode.



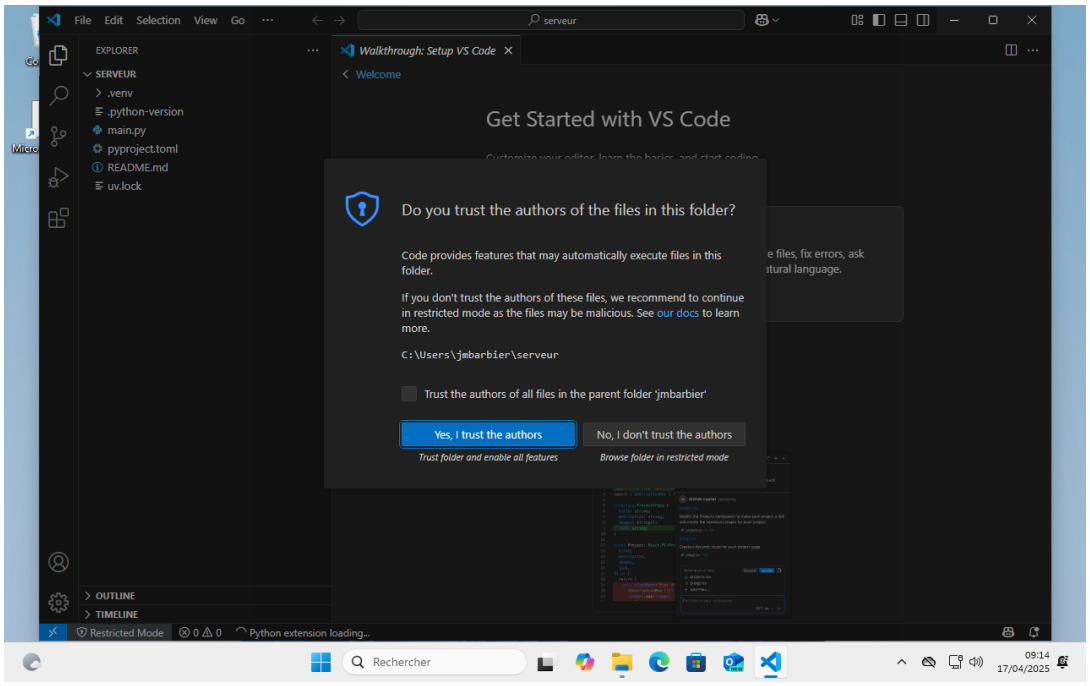
Dans les extensions, installer l'extension python de Microsoft.



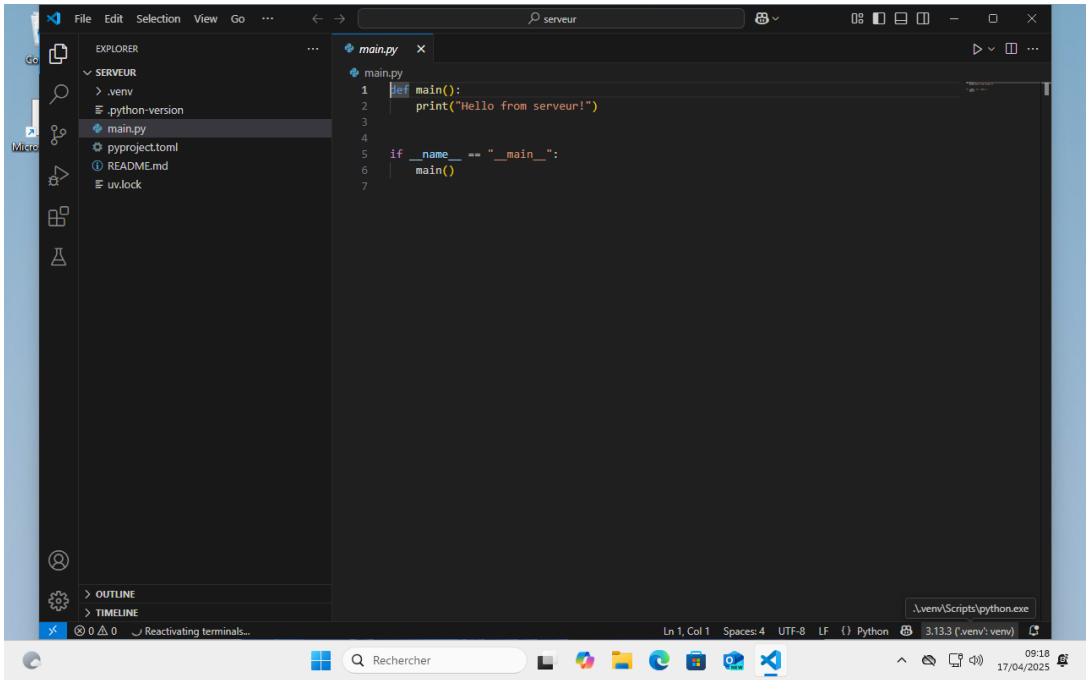
Ouvrir le projet `serveur` créé précédemment avec `uv init`.



Important : "Faire confiance aux auteurs des fichiers du dossier". Sinon vous ne pourrez pas exécuter de script python.



Ouvrir le fichier "main.py". Si tout se passe bien, l'interpréteur python et l'environnement virtuel sont détectés et activés. Cliquer sur le bouton "Play" pour exécuter le programme.



21.2. Linux et OSX

Python est déjà installé sur la plupart des distributions Linux et OSX, et il peut aussi être installé avec `uv` ou `brew` sur OSX.

Uv peut être installé avec `curl -LsSf https://astral.sh/uv/install.sh | sh`.

Vscode peut être installé via :

- https://code.visualstudio.com/docs/setup/linux pour linux
- https://code.visualstudio.com/docs/setup/mac pour OSX

Le reste de l'installation est identique à celle de windows, avec moins de bugs ou de surprises :P

22. Serveur

Le micro-framework `bottle` tient en un fichier unique, il est donc possible de l'utiliser sans `pip` ou autre gestionnaire de paquets. Il suffit de télécharger le fichier `bottle.py` dans le répertoire de votre projet et de l'importer dans votre code python. Cependant, il est recommandé d'utiliser un gestionnaire de paquets pour pouvoir ensuite ajouter proprement d'autres dépendances.

22.1. Installation de bottle

La page d'accueil de bottle (https://bottlepy.org/docs/dev/) donne les liens pour télécharger le fichier `bottle.py` :

`https://raw.githubusercontent.com/bottlepy/bottle/master/bottle.py`

Sinon, avec `uv`, dans le répertoire du projet :

```
uv add https://github.com/bottlepy/bottle.git
```

> **Quelle version ?**
>
> Bottle a pas mal d'années d'existence, et est d'une grande stabilité. La version stable actuelle est la `0.13.3`, mais elle comporte quelques soucis liés à l'encodage des caractères. La version de développement est donc conseillée, ou d'ici peu, la version `0.14` (ou `1.0.0`)

Nous allons ensuite suivre plus ou moins le tutoriel de bottle : https://bottlepy.org/docs/dev/tutorial.html#hello-world

22.2. Premier serveur "hello world"

Dans le fichier `main.py`, copier le code suivant (code) :

```python
from bottle import route, run

@route('/hello')
def hello():
    return "Hello World!"

if __name__ == '__main__':
    run(host='localhost', port=38083, debug=True, reloader=True)
```

Le programme ci-dessus ne comporte que **6** lignes de code, et c'est pourtant déjà un serveur web !!!

Lancer le serveur

```
python3 main.py
# ou bien uv run main.py
# ou bien bouton "play" de l'IDE
```

Le site est alors disponible à l'adresse suivante : http://localhost:38083/hello, ou sur https://xxxxx-app.formation.devel.space/hello si vous utilisez l'IDE en ligne de la formation.

- Le paramètre `reloader=True` permet de relancer le serveur automatiquement à chaque modification du code.
- Le paramètre `debug=True` permet d'afficher les erreurs dans le navigateur.

Pour comprendre un peu mieux ce petit programme, il faut bien se rappeler les points suivants, vus précédemment :

- la communication entre un client et un serveur se fait en TCP/IP (transport)
- le client envoie une requète au serveur (une machine identifiée par son adresse IP ou son nom de domaine), sur un port donné; dans notre cas, la requète est une requète utilisant le protocole HTTP
- le serveur écoute le port sur l'IP, et répond à la requète.

Ici, le port d'écoute est `38083`, l'IP est l'ip locale (`localhost` = `127.0.0.1`, qui ne sort pas de la machine), et le serveur est configuré pour renvoyer le message "Hello World!" à une requète spécifique.

Il n'y a pas un poil de HTML dans la réponse, mais c'est une réponse tout à fait valide.

22.3. Tester le serveur

Lors du lancement du serveur, vous devriez voir un message du type :

```
Bottle v0.13.3 server starting up (using WSGIRefServer())...
Listening on http://localhost:38083/
Hit Ctrl-C to quit.

...
```

Traduction : je suis un serveur utilisant Bottle v0.13.3, j'écoute sur localhost, sur le port 38083 (notation IP:PORT); pour terminer mon exécution, appuyez sur Ctrl-C.

Prenez un navigateur, et tapez l'adresse de votre serveur dans la barre d'adresse (pas dans google ni dans le champ de recherche !!)... votre navigateur doit afficher le message "Hello World", et votre programme, dans son terminal, a lui affiché une ou plusieurs lignes du type

```
127.0.0.1 - - [17/Apr/2025 12:32:00] "GET /hello HTTP/1.1" 200 12
127.0.0.1 - - [17/Apr/2025 12:32:00] "GET /favicon.ico HTTP/1.1" 404 766
```

qui sont affichées grâce au paramètre de debug, et qui précisent :

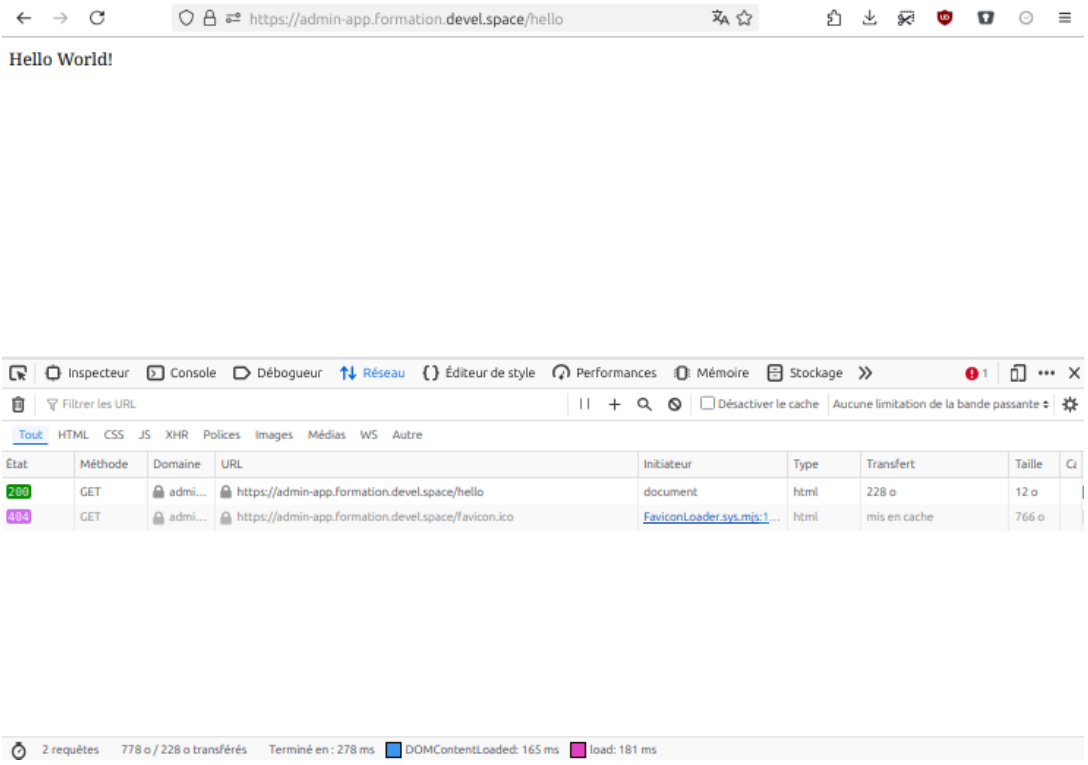
- l'adresse IP du client : ici 127.0.0.1
- la page qui fait référence à la page actuelle (lorsqu'on suit un lien sur un site, ce qui n'est pas le cas ici, le champ est donc vide : c'est l'espace entre les - -)
- la date et l'heure de la requète
- la requète HTTP
- le code de réponse

- le nombre d'octets de la réponse

Ici, la méthode de la requête est GET, le client demande l'URL `/hello` en utilisant le protocole HTTP/1.1. Il y a aussi des en-têtes (non affichés ici), au minimum un en-tête "Host:" indiquant quel est la partie "hôte" de la demande (http://host/url).

Le serveur renvoie une réponse comprenant entre autres un code-réponse : ici 200 indique un succès, et 404 indique "page non trouvée" (le navigateur demande automatiquement la favicon, l'icône pour les favoris, que notre serveur n'est pas programmé pour envoyer : aucune route ne lui correspond).

Pour mieux saisir cet échange, nous pouvons l'espionner un peu plus : dans votre navigateur, ouvrez les outils de développement, activez l'onglet réseau, et rechargez votre page http://localhost:38083/hello ou https://xxxx-app.formation.devel.space/hello ... Vous voyez apparaître une ligne, indiquant qu'une requête a été faite. En développant cette ligne, vous avez toutes les informations sur la requête et la réponse, formatté sympathiquement, qui ressemblent à celles vues précédemment



Programme

- NSI 1ère : intéraction client/serveur, requêtes HTTP, réponses du serveur

23. Routes

Le serveur "attend la commande du client" et va répondre à sa demande. La création d'un serveur consiste donc à définir les différentes commandes que le serveur va pouvoir honorer, et la manière de le faire.

On peut voir le serveur comme un "restaurant", où le client (le navigateur) va passer une commande (une requête HTTP) et le serveur transmettre la commande à la cuisine (le code python) pour la préparer. Le serveur va ensuite transmettre la réponse au client (le plat préparé).

La demande est faite par le client par l'envoi d'une requête HTTP. L'**URL** de la requête, la méthode HTTP utilisée (GET, POST, etc.), et les données envoyées (le cas échéant) sont des éléments clés pour déterminer comment le serveur va répondre.

23.1. Notion de `route`

Définir des **routes** permet d'associer un ou plusieurs méthodes et une ou plusieurs URLs à une fonction python. Le framework va alors utiliser cette fonction pour créer la réponse HTTP à envoyer au client.

Bottle (comme la plupart des autres frameworks) utilise un décorateur pour définir une route. Un `décorateur` est une fonction qui va modifier le comportement d'une autre fonction. En l'occurrence ici, ce décorateur va ajouter à la fonction `hello` la capacité à répondre à une requête http précise (lire les en-têtes et les données de la requête, rajouter les codes / en-têtes de la réponse, etc...).

Le décorateur `@route` utilisé ici effectue la liaison entre une URL et une fonction python, et enregistre cette association dans le routeur global de l'application.

```
@route('/hello')
def hello():
    return "Hello World!"
```

La fonction `hello()` est appelée lorsque le serveur reçoit une requête sur cette URL dont le `chemin` est exactement `/hello` et la méthode `GET`. La fonction doit retourner une chaîne de caractères qui sera envoyée au client dans la réponse HTTP.

Il est possible d'associer plusieurs routes à la même fonction, en spécifiant plusieurs URL dans le décorateur `@route` :

```
@route('/hello')
@route('/salut')
def hello():
    return "Hello World!"
```

Il est aussi possible de spécifier plusieurs méthodes HTTP pour une même route en précisant la liste des méthodes dans le décorateur `@route` :

```
@route('/hello', method=['GET', 'POST'])
def hello():
    return "Hello World!"
```

Des raccourcis existent :

- `@get('/hello')` : équivalent à `@route('/hello', method=['GET'])`
- `@post('/hello')` : équivalent à `@route('/hello', method=['POST'])`
- `@put('/hello')` : équivalent à `@route('/hello', method=['PUT'])`
- `@delete('/hello')` : équivalent à `@route('/hello', method=['DELETE'])`
- ...

Il faut penser à les importer avant de les utiliser.

```
from bottle import get, post, put, delete #...
```

23.2. Routes dynamiques

23.2.1. Jokers

Une route **dynamique** est une route qui permet d'associer plusieurs URLs à une même fonction, en utilisant des "jokers" dans l'URL. Ces jokers sont écris sous la forme `<nom>` et "capturent" dans un paramètre du même nom la valeur correspondante dans l'URL, jusqu'au prochain slash. Par exemple, la route `/hello/<name>` va accepter les demandes `/hello/alice` et `/hello/bob`, mais pas `/hello/`, ni `/hello`, ni `/hello/alice/bob`.

Le ou les "jokers" sont passés par nom à la fonction python.

```
@route('/hello/<name>')
def hello(name):
    return f"Hello {name}!"
```

Plusieurs "jokers" peuvent être utilisés dans la même route : `@route('/hello/<action>/<user>')`

23.2.2. Filtres

Des **filtres** peuvent être ajoutés aux jokers pour restreindre les valeurs qui vont "matcher" :

- `:int` correspond uniquement aux chiffres (signés) et convertit la valeur en entier.
- `:float` similaire à `:int` mais pour les nombres décimaux.
- `:path` correspond à tous les caractères, y compris le caractère slash, et peut être utilisé pour correspondre à plus d'un segment de chemin.
- `:re` vous permet de spécifier une expression régulière[1] personnalisée dans le champ de configuration. La valeur appariée n'est pas modifiée.

Exemples de filtres :

- `@route('/plus/<a:int>/<b:int>')` : le joker `a` et `b` doivent être des entiers, et il répondra à la requête `/plus/2/3` avec `a=2` et `b=3` (nombres entiers) mais pas à la requête `/plus/2.5/3.5` ni à la requête `/plus/toto/titi`
- `@route('/plus/<a:float>/<b:float>')` : le joker `a` et `b` doivent être des flottants, et il répondra à la requête `/plus/2.5/3.5` avec `a=2.5` et `b=3.5` (nombres flottants) mais pas à la requête `/plus/toto/3`.
- `@route('/file/<name:path>')` : le joker `name` peut contenir des slashes, il répondra à la requête `/file/mon/fichier.txt` avec `name='mon/fichier.txt'`
- `@route('/code/<id:re:[A-Z0-9]+>')` : le joker `id` doit être un code ne comportant que des lettres majuscules et des chiffres, et il répondra à la requête `/code/ABC123` avec `id='ABC123'`, mais pas à la requête `/code/abc123` ou `/code/AB.CD`

[1] Une expression régulière (ou expression rationnelle ou expression normale ou motif) est une chaîne de caractères qui décrit, selon une syntaxe précise, un ensemble de chaînes de caractères possibles. Voir fr.wikipedia.org/wiki/Expression_régulière et https://docs.python.org/fr/3/howto/regex.html pour plus de détails. https://regex101.com/ est un site très pratique pour tester les expressions régulières.

23.3. En pratique

23.3.1. Bonjour personnalisé

Dans notre serveur, rajouter une route de bonjour personnalisé qui répond à une requête du type `/hello/bob` en renvoyant un message du type "Hello bob !"

Correction ›

23.3.2. Calcul

Ajouter aussi une route de calcul qui répond à une requête du type `/calcul/add/3/2.5` et qui renvoie le résultat sous la forme "resultat = 5.5" (on utilisera "add", "sub", "mul" et "div" pour addition, soustraction, multiplication et division), l'opération étant `1er arg opérateur 2ème arg` (ex : `3 add 2.5` = `3 + 2.5` = `5.5`). Si l'opérateur n'est pas reconnu, renvoyer "opérateur inconnu".

Correction ›

Programme

- NSI 1ère : intéraction client/serveur, requêtes HTTP, réponses du serveur

24. Templates

Nos pages sont pour l'instant encore un peu tristounettes et basiques. Si nous voulons plus de richesse, il va falloir du HTML, voire du CSS. Mais générer tout ce contenu en python dans des f-strings peut vite devenir compliqué et tout aussi illisible qu'un programme en PHP :P.

Bottle (comme tous les autres frameworks) fournit des outils pour faciliter la création de pages : ce sont les **templates**.

Un **template** est un fichier qui contient du HTML et des balises spéciales pour intégrer des données dynamiques.

Plusieurs moteurs de templates existent, mais nous allons utiliser le moteur intégré de bottle, qui est très simple à utiliser et suffisant pour nos besoins.

24.1. Moteur de template SimpleTemplate

Bottle utilise le moteur de template `SimpleTemplate` (ou `stpl`). Le principe du template est de pouvoir être "rendu" avec des données dynamiques, en remplaçant les balises spéciales par les valeurs correspondantes.

Télécharger le code

```
from bottle import SimpleTemplate
tpl = SimpleTemplate('Hello {{name}}!')
print(tpl.render(name='World')) # affiche u'Hello World!'
assert tpl.render(name='World') == 'Hello World!'
```

ou bien

```
from bottle import SimpleTemplate
tpl = SimpleTemplate('Hello {{name}}!')
dico = {'name': 'World'}
print(tpl.render(**dico)) # affiche 'Hello World!'
assert tpl.render(**dico) == 'Hello World!'
```

Référence : https://bottlepy.org/docs/dev/stpl.html

En résumé, un template est une sorte de `f-string` boostée.

24.2. Utilisation

Pour utiliser les templates, il faut d'abord importer la fonction `template` de bottle.

```
from bottle import template #...
```

On peut alors utiliser les templates de deux manières :

- soit en les définissant dans des chaînes de caractères
- soit en les définissant dans des fichiers séparés, par défaut placés dans le répertoire `views` [1].

24.2.1. Templates dans le code

Télécharger le code

```
HOMEPAGE = """
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Page d'accueil</title>
</head>
<body>
    <h1>Page d'accueil</h1>
    <p>Hello {{ name }}</p>
</body>
</html>
"""

@route('/')
def hello():
    return template(HOMEPAGE, name="World")
```

24.2.2. Templates dans des fichiers

Fichier `views/index.html` (télécharger le code) :

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Page d'accueil</title>
  </head>
  <body>
    <h1>Page d'accueil</h1>
    <p>Hello {{ name }}</p>
  </body>
</html>
```

Fichier `main.py` (télécharger le code) :

```
@route('/')
def index():
    return template('index.html', name="World")
```

24.2.3. Syntaxe

Expressions en ligne

Les expressions en ligne sont entourées de `{{` et `}}`. Elles peuvent contenir n'importe quelle expression python (des variables, des opérations arithmétiques, des appels de fonctions, etc...) à partir du moment où cette expression renvoie une chaîne de caractères ou quelque chose qui peut être converti en chaîne de caractères.

```
<p>Hello {{ name }}</p>
<p>2 + 2 = {{ 2 + 2 }}</p>
<p>Le résultat est {{ "positif" if a > 0 else "negatif" }}</p>
```

L'expression est évaluée lors du rendu et a accès à tous les arguments passés au template (le **contexte**). Les caractères HTML spéciaux sont échappés pour éviter les injections XSS. Par exemple, si `name` contient `<script>alert("Hello")</script>`, le code rendra `<script>alert("Hello")</script>` et non pas le script.

Si vous voulez afficher le code HTML tel quel (sans échappement), il faut utiliser la balise `{{!` et `}}` :

```
<p>Hello {{! name }}</p>
```

24.2.4. Code python inclus

Il est possible d'inclure des blocs de python dans le template. Les lignes de code commencent par `%`, ou bien `<%` et `%>` pour les blocs de code. Le code python "inclus" suit la même syntaxe que le python normal, avec une exception notable : **l'indentation est ignorée**, pour permettre d'écrire du html joliment, tous les blocs qui nécessiteraient une indentation en python doivent être fermés explicitement avec un mot clef `end` :

```
% name = "Bob" # une ligne de code python
<p>Du contenu HTML</p>
<%
# Un bloc de code python
name = name.lower().strip()
%>
<p>Encore du html</p>
<p>Bonjour {{ name }}</p>
```

```
% # Conditions
% if name == "Bob":
  <p>Bonjour Bob</p>
% else:
  <p>Bonjour autre</p>
% end
```

```
% # Boucle for range
<ul>
% for i in range(1, 11):
  <li>{{ i }}</li>
% end
</ul>
```

```
% # Boucle sur un dictionnaire
% # à appeler avec `template('index.html', dico={'a': 1, 'b': 2})`
<ul>
% for key, value in dico.items():
  <li>{{ key }} : {{ value }}</li>
% end
</ul>
```

```
% # Fonctions, bloc de code ...
<%
def add(a, b):
    return a + b
end
%>
<p>2 + 2 = {{ add(2, 2) }}</p>
```

24.2.5. Fonctions de template

Quelques fonctions utiles sont directement utilisables dans les templates :

- `include(sub_template, **context)` : inclut un sous-template dans le template courant. Le contexte est passé au sous-template.
- `setdefault(name, default)` : définit une valeur par défaut pour `name` dans le contexte. Si `name` n'est pas défini, il sera défini avec la valeur de `default`.
- `defined(name)` : renvoie `True` si le nom est défini dans le contexte, `False` sinon.
- `get(name, default=None)` : renvoie la valeur de `name` dans le contexte, ou `default` si `name` n'est pas défini.

`include` est très utile pour éviter de répéter du code dans plusieurs templates. Par exemple, si vous avez un template `header.html` qui contient le code HTML d'en-tête de votre site, vous pouvez l'inclure dans tous vos templates.

`setdefault` est aussi très utile pour définir des valeurs par défaut pour les variables dans le contexte, de manière à pouvoir appeler les templates en fournissant juste les variables qui vous intéressent.

```
% setdefault('page_title', "Page d'accueil")
% setdefault('erreur', None)
% # ...
<h1>{{page_title}}</h1>
% if erreur:
  <p class="erreur">{{ erreur }}</p>
% end
```

```
# ... appels possibles de ce template
    return template('tpl08.html', page_title="Page d'accueil", erreur="Erreur
de saisie")
    return template('tpl08.html', page_title="Page d'accueil")
    return template('tpl08.html')
```

Liste complète des fonctions de template : https://bottlepy.org/docs/dev/stpl.html#template-functions

24.3. En pratique

24.3.1. Vue index / vue hello_user

Créer une vue `index` qui affiche une page d'accueil HTML. Vous pouvez utiliser du CSS pour "décorer" votre page, le css devant pour l'instant être inclus directement dans le HTML. Ajoutez un lien vers la route `/hello/Bob` (ou autre) pour tester la route `hello_user`.

```
<head>
  <!-- .... -->
  <style>
    body {
      background-color: #f0f0f0;
      color: #333;
      font-family: Arial, sans-serif;
    }
    h1 {
      color: #007bff;
    }
  </style>
</head>
```

Modifier la vue `hello_user` (route `/hello/<name>`) pour qu'elle affiche un message de bienvenue joliment présenté (en utilisant un template `hello_user.html`). Rajouter un lien vers la page d'accueil.

Correction

24.3.2. Table de multiplication

Créer une route `/tabmul/<nb:int>`, associée à un template `tabmul.html`, qui affiche la table de multiplication de `nb`, et qui propose une liste de liens permettant d'afficher les tables de multiplication de 1 à 10.

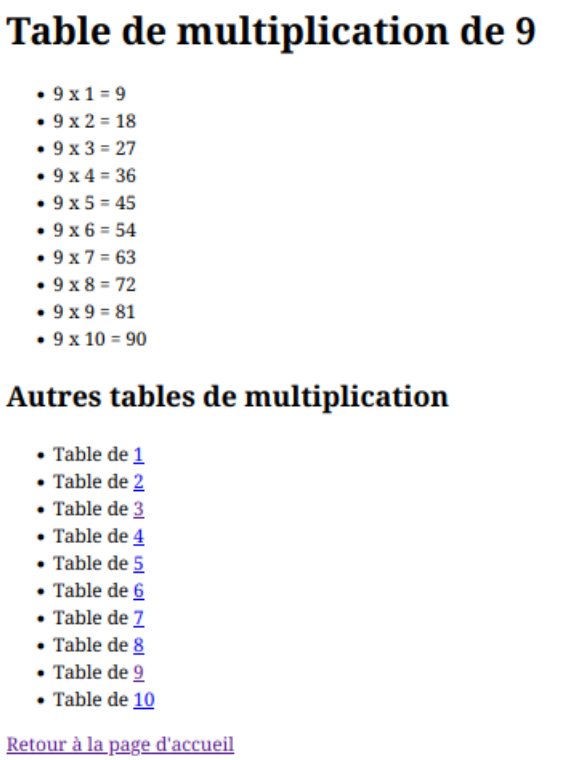


Table de multiplication de 9

- 9 x 1 = 9
- 9 x 2 = 18
- 9 x 3 = 27
- 9 x 4 = 36
- 9 x 5 = 45
- 9 x 6 = 54
- 9 x 7 = 63
- 9 x 8 = 72
- 9 x 9 = 81
- 9 x 10 = 90

Autres tables de multiplication

- Table de 1
- Table de 2
- Table de 3
- Table de 4
- Table de 5
- Table de 6
- Table de 7
- Table de 8
- Table de 9
- Table de 10

Retour à la page d'accueil

Correction

---

[1] On peut modifier le chemin de recherche des templates en modifiant la variable `TEMPLATE_PATH` (`from bottle import TEMPLATE_PATH`), qui contient une liste de chemins dans lesquels bottle cherchera les templates. Par défaut, les chemins sont : "./" et "./views", c'est à dire le répertoire courant (celui dans lequel on a lancé le serveur qui n'est pas forcément celui du fichier python du serveur) et le sous-répertoire "views" du répertoire courant. Pour être ne plus dépendre du répertoire courant, il est pratique de modifier `TEMPLATE_PATH` pour lui indiquer comme chemin de recherche le répertoire du fichier python du serveur. Cela peut se faire à l'aide de la ligne

```
from os.path import dirname
TEMPLATE_PATH.insert(0, join(dirname(__file__), "xxxxx")
```

Cette ligne récupère le répertoire de `__file__` (une variable spéciale de python qui contient le chemin absolu vers le fichier dans lequel elle se trouve), et insère cette valeur au début de `TEMPLATE_PATH` (= ce sera le premier chemin recherché), en ayant joint ce répertoire avec le répertoire "xxxxx" dans lequel on veut chercher les templates.

25. Serveur de fichiers

En utilisant les routes dynamiques, il est possible de créer un **serveur de fichier** en quelques lignes.

Bottle dispose d'une fonction `static_file` qui permet de renvoyer la réponse correspondant à un fichier donné.

Cette fonction prend au moins 2 arguments obligatoires :

- `filename` : un nom de fichier ou un chemin vers un fichier
- `root` : un paramètre root qui indique à partir de quel répertoire de la machine on va chercher le fichier ci-dessus

Références :

- https://bottlepy.org/docs/dev/tutorial.html#serving-assets
- https://bottlepy.org/docs/dev/api.html#bottle.static_file

25.1. Ajouter une route pour les fichiers statiques

Il faut d'abord créer le répertoire de base pour les fichiers. Traditionnellement, il s'appelle `static` ou `assets` ou `public` mais on peut l'appeler comme on veut. Nous allons l'appeler `static`, dans le répertoire `serveur`, et nous allons y ajouter un sous-répertoire `images` pour y placer des images.

```
mkdir -p static/images
```

Si vous êtes en manque d'inspiration, vous pouvez télécharger quelques `Tux` (il y en a 946) à partir de l'adresse suivante : https://solidev.net/tux/tux1.png et les placer dans le répertoire `static/images`.

```
cd static/images
wget https://solidev.net/tux/tux1.png
wget https://solidev.net/tux/tux2.png
wget https://solidev.net/tux/tux3.png
wget https://solidev.net/tux/tux4.png
```

Ce répertoire `static` doit être précisé dans le paramètre `root` de la fonction `static_files`. On peut le définir de plusieurs manières :

- "en dur" : `root=/home/user/serveur/static` mais ce n'est pas "portable".
- "en relatif" : `root=./static` mais ce n'est pas "portable" non plus, le chemin relatif est pris à partir du répertoire courant d'exécution du script, qui peut être différent selon l'endroit où on lance le script.
- "calculé" : `root=join(dirname(__file__), "static")` qui permet de calculer le chemin à partir du répertoire du script en cours d'exécution. C'est la méthode la plus

portable (il faut au préalable importer `join` et `dirname` de `os.path`).

```
from os.path import join, dirname
from bottle import static_file #...
# ...

@route('/static/<filename:path>')
def server_static(filename):
    return static_file(filename, root=join(dirname(__file__), "static"))
```

25.2. En pratique

25.2.1. Ajouter la route `static`

Rajouter la route `/static/<filename:path>` dans le fichier `main.py` pour servir les fichiers statiques.

Correction

25.2.2. Afficher une image dans la page d'accueil

Dans le template de la page d'accueil, rajouter l'affichage d'une image provenant de notre serveur de fichier statique au dessus du message d'accueil.

Correction

25.2.3. Créer un fichier CSS et l'utiliser

Créer un fichier `style.css` et y déplacer les styles définis dans les templates, et lier ce fichier CSS dans le template de la page d'accueil avec `<link rel="stylesheet" href="/static/style.css">`.

Correction

Programme

- SNT : thème Internet
- NSI 1ère : intéraction client/serveur, requêtes HTTP, réponses du serveur

26. Formulaires

Pour l'instant, toutes les intérations avec l'utilisateur passent par la saisie d'une URL différente ou le clic sur un lien. Les formulaires permettent de créer un niveau supérieur d'intéractivité avec les utilisateurs.

Dans une page HTML, un formulaire (défini par la balise `<form>`) est un ensemble de balises permettant :

- à l'utilisateur de saisir des informations
- au navigateur de transmettre ces informations au serveur

Un formulaire peut comporter différents **champs de formulaire** :

- zones de saisie de texte (une ou plusieurs lignes)
- liste déroulantes
- cases à cocher
- ..

Le serveur va recevoir les données saisies par l'utilisateur et pourra les traiter

Nous allons voir un peu plus en détail chacune de ces étapes.

27. Côté HTML

27.1. La balise `<form>`

Tous les éléments du formulaire doivent être à l'intérieur d'une balise `<form>`. Cette balise a plusieurs attributs :

- `action` : l'URL à laquelle le formulaire doit être envoyé
- `method` : la méthode HTTP utilisée pour envoyer le formulaire (GET ou POST)

Lorsque l'utilisateur soumet le formulaire, le navigateur envoie une requête HTTP à l'URL spécifiée dans l'attribut `action`. Les données du formulaire sont envoyées dans le corps de la requête (pour POST) ou dans l'URL (pour GET).

On utilise `GET` lorsque :

- les données sont peu volumineuses
- elles ne contiennent pas d'informations sensibles
- elles peuvent être mises en cache et bookmarkées
- elles n'ont pas d'effet secondaire sur le serveur

On utilise `POST` lorsque :

- les données sont volumineuses
- elles contiennent des informations sensibles
- elles ne doivent pas être mises en cache
- elles ont un effet secondaire sur le serveur

En résumé : on utilise `GET` pour les opérations **idempotentes**, cachables, non sensibles et de petite taille, et `POST` dans tous les autres cas.

L'action peut être une URL absolue ou relative. Si elle est relative, elle est relative à l'URL de la page contenant le formulaire. Par exemple, si l'URL de la page est `http://localhost:5000/formulaire`, et que l'action du formulaire est `../contact`, le formulaire sera envoyé à `http://localhost:5000/contact`.

L'URL relative est souvent utilisée pour envoyer le formulaire à la même page que celle qui l'affiche : `action=""`. Dans le reste des cas, on préfère souvent utiliser l'URL absolue, qui est plus explicite et évite les erreurs de chemin, sans la partie nom de domaine (pour permettre de d'utiliser le même code en développement et en production, entre autres).

```
<!-- Formulaire de recherche, non sensible, => GET -->
<form action="/search" method="GET">
  <label for="query">Rechercher :</label>
  <input type="text" id="query" name="query">
  <button type="submit">Rechercher</button>
</form>
```

```html
<!-- Formulaire de contact, envoie un mail = non idempotente => POST -->
<form action="/contact" method="POST">
  <label for="nom">Nom :</label>
  <input type="text" id="nom" name="nom">
  <label for="message">Message :</label>
  <textarea id="message" name="message"></textarea>
  <button type="submit">Envoyer</button>
</form>
```

Référence : Documentation de la balise form en HTML

27.2. Les champs de formulaire

Pour que les données d'un champ de formulaire soient envoyée lors de la soumission du formulaire, il faut que chaque champ ait un attribut `name`.

Plusieurs balises de formulaire sont disponible selon ce que l'on veut saisir comme données :

- `<input>` : pour saisir du texte, un mot de passe, une case à cocher, un bouton radio, une date, une couleur, un fichier, etc. Référence : Documentation de la balise input en HTML
  - `name` : le nom du champ, utilisé pour identifier la valeur envoyée au serveur
  - `id` : l'identifiant unique du champ, utilisé pour le lier à une étiquette `<label>`
  - `type` : l'attribut `type` permet de définir le type de champ de saisie
    - `text` : pour saisir du texte
    - `password` : pour saisir un mot de passe (les caractères sont masqués)
    - `checkbox` : pour une case à cocher
    - `radio` : pour un bouton radio (un choix parmi plusieurs)
    - `date` : pour choisir une date
    - `color` : pour choisir une couleur
    - `file` : pour télécharger un fichier
  - `placeholder` : un texte d'aide qui s'affiche dans le champ avant la saisie
  - `value` : la valeur par défaut du champ
  - `checked` : pour pré-cocher une case à cocher ou un bouton radio
  - `disabled` : pour désactiver un champ
  - `readonly` : pour rendre un champ en lecture seule
  - `required` : pour rendre un champ obligatoire
  - `minlength` et `maxlength` : pour définir une longueur minimale et maximale (pour les champs de type `text`, `password`, etc.)
  - `pattern` : pour définir une expression régulière que la valeur doit respecter (pour les champs de type `text`, `password`, etc.)
  - `size` : pour définir la taille du champ (en nombre de caractères)
  - `step` : pour définir l'incrément de la valeur (pour les champs de type `number`, `date`, etc.)
  - `min` et `max` : pour définir une valeur minimale et maximale (pour les champs de type `number`, `date`, etc.)
- `<textarea>` : pour saisir un texte sur plusieurs lignes. Référence : Documentation de la balise textarea en HTML
  - `name` : le nom du champ, utilisé pour identifier la valeur envoyée au serveur
  - `id` : l'identifiant unique du champ, utilisé pour le lier à une étiquette `<label>`

    - `rows` : le nombre de lignes visibles
    - `cols` : le nombre de colonnes visibles
    - `placeholder` : un texte d'aide qui s'affiche dans le champ avant la saisie
    - `value` : la valeur par défaut du champ
    - `disabled` : pour désactiver un champ
    - `readonly` : pour rendre un champ en lecture seule
    - `required` : pour rendre un champ obligatoire
    - `minlength` et `maxlength` : pour définir une longueur minimale et maximale

- `<select>` : pour choisir une valeur dans une liste déroulante. Référence : Documentation de la balise select en HTML

    - `name` : le nom du champ, utilisé pour identifier la valeur envoyée au serveur
    - `id` : l'identifiant unique du champ, utilisé pour le lier à une étiquette `<label>`
    - `size` : le nombre d'options visibles (par défaut 1)
    - `multiple` : pour permettre de sélectionner plusieurs options
    - `disabled` : pour désactiver un champ
    - `required` : pour rendre un champ obligatoire
    - `<option>` : pour définir une option dans la liste déroulante
        - `value` : la valeur envoyée au serveur si cette option est sélectionnée
        - `selected` : pour pré-sélectionner une option
        - `disabled` : pour désactiver une option
        - `label` : pour définir un texte d'aide qui s'affiche dans la liste déroulante
        - `hidden` : pour cacher une option

27.3. Envoi du formulaire

L'envoi du formulaire est déclenché par l'utilisateur :

- avec un clic sur un champ de type `submit`
- avec un clic sur un bouton (`<button>`)
- en appuyant sur la touche `Entrée` dans un champ de saisie de texte si un bouton un champ de type `submit` est présent dans le formulaire (si plusieurs boutons ou champs submit sont présents, l'appui sur entrée est équivalent à un clic sur le premier bouton de type `submit` ou sur le premier champ de type `submit` du formulaire)

Exemple : https://codepen.io/jmsolidev/pen/NWqLORY

```
<form method="POST" action="https://echo.solidev.net/formulaire">
  <label for=" inputnom">Votre nom</label>
  <input type="text" name="nom" id="inputnom" />
  <br />
  <label for="inputpass">Votre mot de passe</label>
  <input type="password" name="pass" id="inputpass" />
  <br />
  <button>Envoyer 1</button>
  <input type="submit" value="Envoyer 2" name="res" />
  <input type="submit" value="Envoyer 3" name="res" />
  <button>Envoyer 4</button>
</form>
```

Le formulaire ci-dessus envoie les données à l'URL `https://echo.solidev.net/formulaire` (echo.solidev.net est un service qui affiche comme un écho le détail de la requête envoyée - vous pouvez l'utiliser pour tester vos formulaires ou pour débugger vos

applications).

```
POST /formulaire HTTP/1.1

Host: echo.solidev.net
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Content-Length: 27
Content-Type: application/x-www-form-urlencoded
Origin: https://cdpn.io
Priority: u=4
Referer: https://cdpn.io/
Sec-Fetch-Dest: iframe
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
Sec-Fetch-User: ?1
Te: trailers
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:136.0) Gecko/20100101
Firefox/136.0
X-Forwarded-Proto: https

nom=machin&pass=supersecret
```

Si on modifie le formulaire pour qu'il envoie les données avec la méthode `GET` :

```
GET /formulaire?nom=machin&pass=supersecret HTTP/1.1

Host: echo.solidev.net
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Priority: u=4
Referer: https://cdpn.io/
Sec-Fetch-Dest: iframe
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
Sec-Fetch-User: ?1
Te: trailers
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:136.0) Gecko/20100101
Firefox/136.0
```

Programme

- SNT : thème Internet
- NSI 1ère : intéraction client/serveur, requêtes HTTP, réponses du serveur
- NSI 1ère : formulaire d'une page web, interaction avec l'utilisateur dans une page web

28. Côté serveur

Les données envoyées par le formulaire arrivent au serveur. Bottle fournit l'objet `request`, qui contient toutes les informations sur la requête, et en particulier les données du formulaire.

28.1. Classe `Request`

Référence de la classe `Request` : https://bottlepy.org/docs/dev/api.html#bottle.BaseRequest

Il faut importer `request` depuis `bottle` :

```
from bottle import request
```

Les données de formulaire sont accessible via :

- `request.query` ou `request.GET` pour les données envoyées par la méthode `GET`
- `request.forms` et `request.files`, ou `request.POST` qui combine les deux pour les données envoyées par la méthode `POST`

Ces données sont rendues disponibles sous forme de dictionnaire (plus précisément un `MultiDict`, qui est un type de données permettant de gérer plusieurs valeurs pour une même clé. Il est utile pour traiter les formulaires où plusieurs valeurs peuvent être envoyées pour un même champ).

28.2. Accéder aux données

Pour accéder aux données, on peut utiliser la syntaxe `request.POST['nom']` ou `request.forms.get('nom')`. La méthode `get` permet de spécifier une valeur par défaut si la clé n'existe pas, par exemple `request.forms.get('nom', default=None)`.

Il est aussi possible d'utiliser directement un accès par attribut, par exemple `request.forms.nom`.

Si plusieurs valeurs sont envoyées pour un même champ (exemple avec une requête `GET /formulaire?choix=1&choix=2&choix=3`), on peut les récupérer avec `request.GET.getall('choix')` ou `request.GET.getlist('choix')`, qui renvoient une

liste de toutes les valeurs associées à la clé `choix`.

`.keys()`, `.values()` et `.items()` fonctionnent aussi sur les objets `request.GET` et `request.POST`, ce qui permet de les parcourir comme un dictionnaire classique.

Lorsqu'on envoie des fichiers via un formulaire (voir ci-dessous), on peut les récupérer avec `request.files`, qui est un dictionnaire associant le nom du champ à un objet `FileUpload`. Cet objet contient des méthodes pour accéder au contenu du fichier, à son nom, à son type MIME, etc. Par exemple, pour récupérer le nom du fichier téléchargé, on peut utiliser `request.files['fichier'].filename`.

La méthode `save` de l'objet `FileUpload` permet d'enregistrer le fichier sur le serveur. Elle prend en paramètre le chemin où le fichier doit être enregistré. Par exemple, pour enregistrer le fichier dans le dossier `/tmp`, on peut utiliser (sauf sous Windows) :

```
request.files['fichier'].save('/tmp/' + request.files['fichier'].filename)
```

28.3. Routage

Le traitement des données du formulaire se fait dans une route; deux options sont possibles :

- utiliser la même route que celle qui affiche le formulaire, en spécifiant la méthode `POST` dans le décorateur de la route. C'est souvent la méthode la plus simple pour pouvoir gérer les erreurs de saisie et afficher le formulaire avec les données déjà saisies pour correction.

```
from bottle import route, run, template, request

@route('/formulaire', method=['GET', 'POST'])
def formulaire():
    if request.method == 'POST':
        # Traitement des données du formulaire
        nom = request.forms.get('nom', None)
        message = request.forms.get('message', None)
        # ... traitement des données
        if erreur_dans_le_formulaire:
            return template("formulaire.html",
                            erreur=erreur,
                            nom=nom,
                            message=message)
        # Vérification des données OK
        # ... traitement des données (envoi de mail, etc..)
        return template("merci_message.html")
    else:
        return template("formulaire.html")
```

- utiliser une route différente pour traiter les données du formulaire, en spécifiant l'URL de la route dans l'attribut `action` du formulaire

```python
from bottle import route, run, template, request

@route('/formulaire', method='GET')
def formulaire():
    return template("formulaire.html")

@route('/traitement', method='POST')
def traitement():
    # Traitement des données du formulaire
    nom = request.forms.get('nom')
    message = request.forms.get('message')
    # ...
    return template("merci_message.html")
```

28.4. Exemples

28.4.1. Formulaire de contact basique

Télécharger le code

```html
<body>
    <form action="/contact" method="POST">
        <label for="nom">Nom :</label>
        <input type="text" id="nom" name="nom" />
        <label for="message">Message :</label>
        <textarea id="message" name="message"></textarea>
        <button type="submit">Envoyer</button>
    </form>
</body>
```

Télécharger le code

```python
from bottle import route, request, template, run

def envoyer_email(nom, message):
    # Simuler l'envoi d'un email
    print(f"Envoi d'un email à {nom} avec le message : {message}")

@route('/contact', method=['GET', 'POST'])
def contact():
    if request.method == 'GET':
        return template("contact_formulaire.html")
    # Récupération des données du formulaire
    nom = request.forms.get('nom', default=None)
    message = request.forms.get('message', default=None)

    # Traitement des données (par exemple, envoi d'un email)
    envoyer_email(nom, message)

    return template("merci.html")
```

28.4.2. Formulaire de contact avec vérification des erreurs

Télécharger le code

```
% setdefault("erreur", None)
% setdefault("nom", "")
% setdefault("message", "")
<!-- ... -->
<body>
    <form action="/contact" method="POST">
    % if erreur is not None:
        <p class="erreur">{{ erreur }}</p>
    % end
    <label for="nom">Nom :</label>
    <input type="text" id="nom" name="nom" value="{{ nom }}" />
    <label for="message">Message :</label>
    <textarea id="message" name="message">{{ message }}</textarea>
    <button type="submit">Envoyer</button>
    </form>
</body>
```

Télécharger le code

```
from bottle import route, request, template, run

def envoyer_email(nom, message):
    # Simuler l'envoi d'un email
    print(f"Envoi d'un email à {nom} avec le message : {message}")

@route('/contact', method=['GET', 'POST'])
def contact():
    if request.method == 'POST':
        # Récupération des données du formulaire
        nom = request.forms.get('nom', default=None)
        message = request.forms.get('message', default=None)

        # Vérification des erreurs
        if not nom or not message:
            erreur = "Tous les champs sont obligatoires."
            return template("contact_formulaire_check.html",
                            erreur=erreur, nom=nom, message=message)
        if len(message) < 10:
            erreur = "Le message doit contenir au moins 10 caractères."
            return template("contact_formulaire_check.html",
                           erreur=erreur, nom=nom, message=message)
        if "spam" in message:
            erreur = "Le message contient un mot interdit."
            return template("contact_formulaire_check.html",
                            erreur=erreur, nom=nom, message=message)
        # Traitement des données (par exemple, envoi d'un email)
        envoyer_email(nom, message)
        return template("merci.html")
    else:
        return template("contact_formulaire_check.html")
```

28.4.3. Choix d'options multiples

Télécharger le code : choix_formulaire.html (avec les 2 versions et l'affichage des choix)

```
% setdefault("choix", list())
<!DOCTYPE html>
<html lang="fr">

Avec un `select[multiple]` :

```html
% setdefault("choix", list())

    <form action="/choix" method="POST">
        <label for="choix">Choisissez une ou plusieurs options :</label>
        <select id="choix" name="choix" multiple>
          <option value="choix1" {{"selected" if "choix1" in choix else ""}}
>Option 1</option>
          <option value="choix2" {{"selected" if "choix2" in choix else ""}}
>Option 2</option>
          <option value="choix3" {{"selected" if "choix3" in choix else ""}}
>Option 3</option>
        </select>
        <button type="submit">Envoyer</button>
    </form>
```

ou bien avec des `input[type=checkbox]` :

```
% setdefault("choix", list())

    <form action="/choix" method="POST">
        <input type="checkbox" id="choix1" name="choix"
         value="choix1" {{"checked" if "choix1" in choix else ""}}>
        <label for="choix1">Option 1</label><br>
        <input type="checkbox" id="choix2" name="choix"
         value="choix2" {{"checked" if "choix2" in choix else ""}}>
        <label for="choix2">Option 2</label><br>
        <input type="checkbox" id="choix3" name="choix"
          value="choix3" {{"checked" if "choix3" in choix else ""}}>
        <label for="choix3">Option 3</label><br>
        <button type="submit">Envoyer</button>
    </form>
```

Dans les deux cas, affichage des options choisies :

```
    % if len(choix) > 0:
        <hr>
        <h2>Vous avez choisi :</h2>
        <ul>
        % for c in choix:
            <li>{{ c }}</li>
        % end
        </ul>
    % end
```

Télécharger le code du serveur

```
@route('/choix', method=['GET', 'POST'])
def choix():
    if request.method == 'POST':
        # Récupération des choix
        choix = request.POST.getall('choix')
        print(choix)
        return template("choix_formulaire.html", choix=choix)
    else:
        return template("choix_formulaire.html")
```
```

28.5. Upload de fichiers

Pour permettre à l'utilisateur d'uploader un fichier, il faut utiliser l'attribut `enctype="multipart/form-data"` dans la balise `<form>`. Cela permet d'envoyer des fichiers binaires au serveur. `request.files` contiendra alors le fichier uploadé.

Télécharger le code : upload.html

```
    <form action="/upload" method="POST" enctype="multipart/form-data">
        <label for="fichier">Choisissez un fichier :</label>
        <input type="file" id="fichier" name="fichier" />
        <button type="submit">Envoyer</button>
    </form>
```

Télécharger le code du serveur

```
from bottle import route, request, run, template
from os.path import join, dirname

@route('/upload', method=['GET', 'POST'])
def upload():
    if request.method == 'GET':
        return template("upload.html")

    # Récupération du fichier
    fichier = request.files.get('fichier')
    # Traitement du fichier (par exemple, enregistrement sur le serveur
    # dans un dossier "uploads" situé dans le même répertoire que le script)
    if fichier:
        fichier.save(join(dirname(__file__), "uploads", fichier.filename))
    return template("merci.html")
```

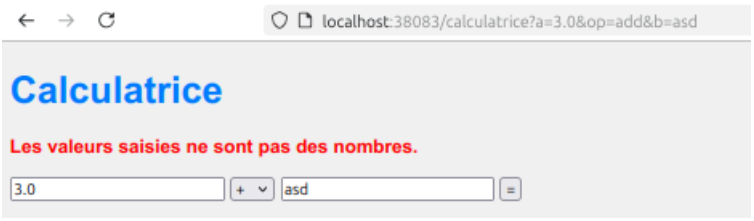
Référence : Documentation de la balise input type="file" en HTML et Documentation de la méthode save() de bottle

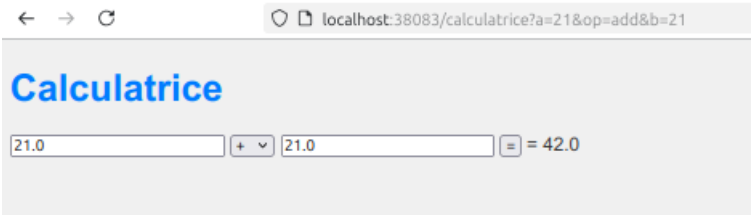
28.6. En pratique

28.6.1. Calculatrice

Créer un formulaire de calcul qui permet de saisir deux nombres et de choisir (avec une liste déroulante) une opération (addition, soustraction, multiplication, division), et qui affiche le résultat. La route du formulaire et du résultat est la même (`/calculatrice`), la méthode est `GET`, et la page doit afficher

- un message d'erreur si au moins un des deux nombres n'est pas un nombre ou si l'opération n'est pas reconnue
- le résultat du calcul juste après le formulaire (forme "[a] [op] [b] = [resultat]")





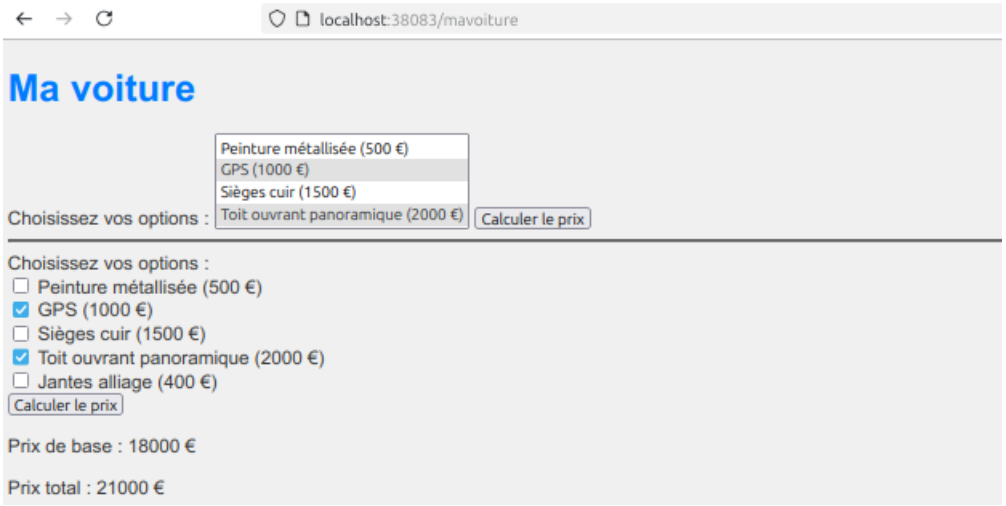
Correction

28.6.2. Options de voiture

Créer un formulaire à l'adresse `/mavoiture`, permettant de choisir des options pour une voiture, et qui affiche un résumé des choix ainsi que le prix de la voiture (prix de base sans options = 18000 euros) lorsque le formulaire est soumis (méthode POST). Les options sont disponibles dans le dictionnaire suivant :

```
{
    "pmetal": {
        "titre": "Peinture métallisée",
        "prix": 500
    },
    "gps": {
        "titre": "GPS",
        "prix": 1000
    },
    "sieges": {
        "titre": "Sièges cuir",
        "prix": 1500
    },
    "toit": {
        "titre": "Toit ouvrant panoramique",
        "prix": 2000
    },
    "jantes": {
        "titre": "Jantes alliage",
        "prix": 400
    }
}
```

Vous pouvez choisir d'afficher les options dans une liste déroulante `<select>` (avec l'attribut `multiple`) ou avec des cases à cocher `<input type="checkbox">`. Dans les deux cas, le prix total doit être affiché sous le formulaire.



Correction

Programme

- NSI 1ère : intéraction client/serveur, requêtes HTTP, réponses du serveur
- NSI 1ère : formulaire d'une page web, interaction avec l'utilisateur dans une page web

29. API

Jusqu'à présent, nous avons utilisé notre serveur pour servir des pages HTML ou des fichiers statiques. Cependant, nous pouvons aussi l'utiliser pour créer une API (Application Programming Interface) qui permet de communiquer différemment avec le client. Selon le type renvoyé par la route, Bottle peut aussi renvoyer des données sous d'autres formats. Par exemple, si on renvoie un dictionnaire, le contenu du dictionnaire est renvoyé au format JSON. Plus de détails dans la note [1].

29.1. Exemple

Dans cette partie, nous allons :

- côté serveur, créer une API qui renvoie des données au format JSON
- côté client, utiliser JavaScript pour faire des requêtes à cette API et afficher les données reçues dans la page HTML

29.1.1. Côté serveur

Nous allons créer un point d'entrée `/api/pwgen` qui renvoie une liste de caractères aléatoires.

Télécharger le code

```
@route('/api/pwgen')
def api_pwgen():
    """
    Renvoie une liste de caractères aléatoires.
    """
    # On génère une liste de 10 caractères aléatoires
    import random
    import string
    chars = [random.choice(string.ascii_letters + string.digits) for _ in
range(10)]
    # On renvoie un dictionnaire pour que Bottle le convertisse en JSON
    return {'password': ''.join(chars)}
```

29.1.2. Côté client

Le point d'entréé `/api/pwgen` peut être appelé en javascript dans une page, et être utilisé pour afficher le mot de passe généré dans la page HTML, sans rechargement de la page.

Télécharger le code

```html
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Exemple API</title>
    <script>
        const fetchPassword = async () => {
            const response = await fetch('/api/pwgen');
            const data = await response.json();
            document.getElementById('password').innerText = data.password;
        }
    </script>
</head>
<body>
    <h1>Générateur de mot de passe</h1>
    <button onclick="fetchPassword()">Générer un mot de passe</button>
    <code id="password"></code`>
</body>
</html>
```

Dans ce code, nous avons ajouté un bouton qui, lorsqu'il est cliqué, appelle la fonction `fetchPassword()`. Cette fonction utilise l'API Fetch pour faire une requête GET à notre point d'entrée `/api/pwgen`. La réponse est ensuite convertie en JSON et le mot de passe généré est affiché dans la page.

Programme

- NSI 1ère : intéraction client/serveur, requêtes HTTP, réponses du serveur
- NSI 1ère : évènements
- NSI 1ère : intéraction avec l'utilisateur dans une page web

---

[1] Si la fonction renvoie :

- un dictionnaire, Bottle le convertit automatiquement en JSON et le renvoie au client avec le `Content-Type` approprié (`application/json`)
- une chaîne de caractères vide, `False` , `None` : réponse vide (`204 No Content`)
- une lise de chaînes de caractères : elles sont concaténées et renvoyées comme une seule chaîne de caractères
- une chaîne de caractères unicode : elle encodée avec le codec approprié spécifié dans l'en-tête `Content-Type` de la réponse (`utf-8` par défaut) et traitéee ensuite comme une liste d'octets (byte string).
- un liste d'octets (byte string) : elle est renvoyée telle quelle, sans traitement.
- une instance de `HTTPError` ou `HTTPResponse` : elle est utilisée directement pour générer la réponse, en ignorant les modifications faites sur `response`
- un fichier ou un objet de type fichier (n'importe quel objet qui a une méthode `read()`) : il est envoyé en listant les octets via `read()` ; `Content-Type` et `Content-Length` ne sont pas générés automatiquement. Pour plus de sécurité, il est recommandé d'utiliser `static_file()` pour servir des fichiers.
- un iterable ou un générateur : le contenu est envoyé au fur et à mesure qu'il est produit, en mode `streaming` . Le `Content-Length` n'est pas généré dans ce cas.

Référence : https://bottlepy.org/docs/dev/tutorial.html#generating-content.

30. Sécurité

Principe de base : **le serveur ne doit pas faire confiance au client**. Il faut donc vérifier toutes les données envoyées par le client avant de les utiliser. Cela inclut les données envoyées par les formulaires, mais aussi éventuellement les données récupérées par une API ou dans une capture de paramètre d'URL.

Donc :

- ne jamais utiliser `eval` sur des données envoyées par le client (eval is evil...)
- ne jamais utiliser de données envoyées par le client pour exécuter directement des commandes système (ex. : `os.system()`, `subprocess.run()`, etc.) (**shell injection**)
- ne jamais utiliser de données envoyées par le client pour exécuter des requêtes sans les protéger au préalable (**SQL injection**)
- ne jamais afficher des donnéés envoyers par le client sans les échapper au préalable (ex. : `return f"{nom}"` ou `{{! xxxx }}` dans un template) (**XSS**)
- ne jamais faire confiance à une validation des données effectuée côté client (ex: `required`, `pattern`, `minlength`, etc..), il faut toujours valider les données côté serveur. ...

D'autres attaques existent :

- **CSRF** (Cross-Site Request Forgery) : une attaque qui consiste à faire exécuter une action sur un site web par un utilisateur sans son consentement (remède = token CSRF)
- **Clickjacking** : une attaque qui consiste à faire cliquer un utilisateur sur un élément d'une page web en le superposant à un autre élément (remède = `X-Frame-Options`)
- ...

Nous reviendrons plus particulièrement sur :

- les injections SQL dans la partie sur les bases de données
- les XSS dans la partie sur l'authentification

Plus (beaucoup plus) de lecture (en anglais) : https://cheatsheetseries.owasp.org/index.html

31. Authentification / persistence

Le protocole HTTP est un protocole sans état. Cependant, les communications entre le client et le serveur nécéssitent parfois un "historique" lié à l'utilisateur, au navigateur ou à la session de navigation. Par exemple, l'ajout d'articles dans un panier d'achat nécessite de conserver l'état du panier entre chaque page. Il existe plusieurs moyens de conserver cet état : côté client ou côté serveur.

- Stockage côté client
  - **Cookies** : petits fichiers stockés sur le client, envoyés au serveur à chaque requête. Ils peuvent être utilisés pour stocker des données persistentes.
  - **Web Storage** : stockage local ou de session, accessible via JavaScript. Le stockage local persiste entre les sessions, tandis que le stockage de session est supprimé lorsque l'onglet est fermé.
  - **IndexedDB** : base de données côté client, accessible via JavaScript. Elle permet de stocker des données structurées et persistantes.
- Stockage côté serveur
  - **Sessions** : données stockées côté serveur, identifiées par un identifiant unique envoyé au client à chaque requète. Elles nécessitent aussi un stockage côté client (généralement un cookie) pour identifier la session/l'utilisateur.

Nous allons détailler ces techniques dans les sections suivantes.

31.1. Sécurité (2ème couche :-)

Que ce soit côté client ou côté serveur, il est important de se dire qu'un jour ou l'autre, les données stockées peuvent être compromises. Il est donc important de

1. ne stocker que les données nécessaires (RGPD)
2. s'assurer que les données sensibles soient chiffrées (ex: on ne stocke **JAMAIS** de mot de passe en clair, mais seulement un hash, permettant de vérifier l'authenticité du mot de passe sans avoir à le connaître)
3. s'assurer que les transmissions soient sécurisées (HTTPS)

32. Cookies

Référence : MDN

Un cookie HTTP (également appelé cookie web ou cookie de navigateur) est une donnée de *petite taille* envoyée par le serveur au navigateur web de l'utilisatrice ou de l'utilisateur. Le navigateur peut alors enregistrer le cookie et le renvoyer au serveur lors des requêtes ultérieures.

Généralement, un cookie HTTP sert à indiquer que plusieurs requêtes proviennent du même navigateur où une personne est connectée. Il permet de **mémoriser des informations d'état** alors que le protocole HTTP est sans état.

Les utilisations principales des cookies HTTP sont :

- la **gestion de session** (authentification, panier d'achat, etc.)
- la **personnalisation** (préférences de l'utilisateur, thèmes, etc.)
- le **tracking utilisateur** (analytics, publicité ciblée, etc.)

Les cookies étant renvoyés à chaque requête, il est important de ne pas les surcharger d'informations inutiles. D'autres mécanismes de stockage côté client existent, comme le **Web Storage** (localStorage et sessionStorage) et **IndexedDB**, qui permettent de stocker des données plus volumineuses et plus structurées, qui ne sont pas transmises au serveur à chaque requête.

32.1. Création d'un cookie

Côté serveur, un cookie est créé en ajoutant un en-tête `Set-Cookie` à la réponse HTTP. Par exemple, en utilisant le framework Bottle, et la méthode `set_cookie` de l'objet `response` (référence : response.set_cookie) :

```
from bottle import response, #...
@route('/set_cookie')
def set_cookie():
    # création d'un cookie nommé "hello" avec la valeur "world"
    response.set_cookie("hello", "world")
    return "Cookie créé !"
```

La réponse HTTP ressemblera à ceci :

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Set-Cookie: hello=world
```

Ensuite, le cookie sera envoyé par le navigateur à chaque requête vers le serveur, jusqu'à ce qu'il expire ou soit supprimé. Par exemple :

```
GET /set_cookie HTTP/2.0
Host: localhost:38083
Cookie: hello=world
```

32.2. Options de cookie

32.2.1. Durée de vie

Un cookie peut avoir une durée de vie définie par l'attribut `Max-Age` ou un date d'expiration définie par l'attribut `Expires`. Si aucun de ces attributs n'est défini, le cookie sera supprimé lorsque la session du navigateur sera fermée (cookie de session).

```
# cookie valable 1 heure
response.set_cookie("hello", "world", max_age=3600)
# cookie valable jusqu'au 1er janvier 2026
response.set_cookie("hello", "world", expires="2026-01-01T00:00:00Z")
# cookie de session (supprimé à la fermeture du navigateur)
response.set_cookie("hello", "world", expires=None, max_age=None)
```

32.2.2. Chemin et domaine

Un cookie peut être limité à un chemin ou un domaine spécifique. Par défaut, le cookie est valable pour le chemin de la requête et le domaine du serveur. On peut modifier ces valeurs avec les attributs `Path` et `Domain`.

```
# cookie valable pour le chemin /app et le domaine example.com
response.set_cookie("hello", "world", path="/app", domain="example.com")
# cookie valable pour tous les chemins et tous les sous-domaines de
example.com
response.set_cookie("hello", "world", path="/", domain=".example.com")
```

On ne peut pas définir un cookie pour un sous-domaine d'un domaine différent. Par exemple, on ne peut pas définir un cookie pour `app.example.com` depuis `example.org`. En revanche, on peut définir un cookie pour `example.com` depuis `app.example.com` ou `www.example.com`.

Par défaut, le cookie est valable pour le chemin de la requête et le domaine du serveur qui l'a créé.

32.2.3. Secure et HttpOnly

Un cookie peut être marqué comme `Secure` pour indiquer qu'il ne doit être transmis que sur des connexions HTTPS. Cela permet de protéger le cookie contre une interception par un tiers lors de la transmission sur le réseau (ex: réseau Wi-Fi public).

```
# cookie valable uniquement sur HTTPS
response.set_cookie("hello", "world", secure=True)
```

Un cookie peut également être marqué comme `HttpOnly` pour indiquer qu'il ne doit pas être accessible par JavaScript. Cela permet de protéger le cookie contre les attaques de type cross-site scripting (XSS), où un script malveillant pourrait essayer de lire le cookie et d'en voler les informations (ex: session, identifiant de l'utilisateur, etc.).

```
# cookie non accessible par JavaScript
response.set_cookie("hello", "world", httponly=True)
```

32.2.4. SameSite

Un cookie peut être marqué comme `SameSite` pour indiquer qu'il ne doit pas être transmis avec des requêtes cross-site (requêtes faites depuis un autre domaine que celui qui a créé le cookie). Cela permet de protéger le cookie contre les attaques de type cross-site request forgery (CSRF), où un site malveillant pourrait essayer de faire une requête vers un autre site en utilisant les cookies de l'utilisateur

Valeurs possibles :

- `Strict` : le cookie n'est pas envoyé avec les requêtes cross-site.
- `Lax` : similaire à `Strict`, mais le navigateur envoie le cookie lorsque la personne **navigue** vers le site (ex: en cliquant sur un lien).
- `None` : le cookie est envoyé avec toutes les requêtes, y compris les requêtes cross-site. Dans ce cas, le cookie doit être marqué comme `Secure`.

Par défaut, la valeur de `SameSite` est `Lax`.

```
# cookie non envoyé avec les requêtes cross-site
response.set_cookie("hello", "world", samesite="strict")
# cookie envoyé avec les requêtes cross-site de navigation
response.set_cookie("hello", "world", samesite="lax")
# cookie envoyé avec toutes les requêtes cross-site
response.set_cookie("hello", "world", samesite="none")
```

32.3. Accès côté serveur

Pour accéder à un cookie côté serveur, on peut utiliser l'objet `request` du framework Bottle. Par exemple, pour accéder au cookie `hello` créé précédemment :

```
from bottle import request
hello = request.get_cookie("hello")  # accès au cookie "hello"
print(hello)  # affiche "world"
```

On peut également accéder à tous les cookies en utilisant la méthode `request.cookies`, qui renvoie un dictionnaire contenant tous les cookies du domaine courant.

```
from bottle import request
cookies = request.cookies  # accès à tous les cookies
print(cookies)  # affiche tous les cookies sous forme de dictionnaire
```

32.4. Accès en JavaScript

Pour accéder à un cookie en JavaScript, on peut utiliser la propriété `document.cookie`. Cette propriété renvoie une chaîne de caractères contenant tous les cookies du domaine courant, sous la forme `nom=valeur; nom=valeur; ...`.

```
console.log(document.cookie); // affiche tous les cookies du domaine courant
```

Les cookies avec les attributs `HttpOnly` ne sont pas accessibles en JavaScript. Cela permet de protéger les cookies sensibles (ex: session, identifiant de l'utilisateur, etc.) contre les attaques de type cross-site scripting (XSS).

32.5. Pistage et vie privée

Une page contenant des images ou des scripts provenant d'autres domaines (ex: page sur `example.com` et image sur `images.com`) peut recevoir des cookies de ces domaines. C'est ce qu'on appelle des `cookies tiers`. Ils peuvent permettre (en définissant `SameSite` à None), de suivre l'utilisateur sur tous les sites qui utilisent des ressources provenant de `images.com`. Ceci est utilisé par les régies publicitaires pour enregistrer les visites de l'utilisateur sur différents sites, et pour lui soumettre des publicités ciblées. Ces cookies peuvent être bloqués par certains navigateurs.

32.6. En pratique

32.6.1. Compteur

Écrire une vue `/compteur` qui affiche le nombre de fois que la page a été visitiée par le navigateur. Le compteur doit être stocké dans un cookie nommé `compteur`. Le cookie doit être valable uniquement pour le chemin `/compteur`. Le cookie doit expirer à la fin de la session, ne doit pas être accessible en JavaScript.

Correction ›

Programme

- NSI 1ère : interaction client/serveur, requêtes HTTP, réponses du serveur : distinguer ce qui est mémorisé dans le client et retransmis au serveur

33. Sessions

Le principe des sessions est de stocker des données côté serveur, et d'envoyer un identifiant au client pour permettre au serveur de retrouver les données associées à cet identifiant. Cela permet :

- de stocker des données plus volumineuses que celles pouvant être stockées dans un cookie
- de ne pas exposer les données sensibles au client

Ce mécanisme passe par le partage d'un identifiant de session entre le client et le serveur. Cet identifiant est généralement stocké dans un cookie, mais il peut également être transmis dans l'URL (très bôf point de vue sécurité) ou dans les en-têtes HTTP (mieux).

33.1. Fonctionnement

1. Lorsqu'un utilisateur envoie une requête au serveur, le serveur vérifie si la requête inclut un identifiant de session.
2. Si l'identifiant de session est présent, le serveur récupère les données de la session associée à cet identifiant et peut les utiliser pour personnaliser la réponse.
3. Si l'identifiant de session n'est pas présent, le serveur génère un nouvel identifiant de session et envoie cet identifiant au client dans un cookie, et associe cet identifiant à un stockage de données côté serveur.
4. Le client envoie cet identifiant de session dans les requêtes suivantes, ce qui permet au serveur de retrouver les données de la session associée.

33.2. Exemple basique

On peut créer un mécanisme de session basique, sans réelle persistance côté serveur, en utilisant un dicitonnaire pour stocker les données de session (= perte des données de session à chaque redémarrage du serveur).

Télécharger le code

```python
from bottle import request, response, route, run
from typing import Tuple, Union

SESSIONS = {}

def get_session() -> Union[dict, None]:
    """
    Renvoie la session associée à l'identifiant de session dans le cookie, ou
None
    si la session n'existe pas.
    """
    # on récupère l'identifiant de session dans le cookie
    session_id = request.get_cookie("session_id")
    if session_id in SESSIONS:
        return SESSIONS[session_id]
    return None

def create_session_id() -> str:
    """
    Renvoie un identifiant de session aléatoire.
    """
    # on génère une chaîne aléatoire pour l'identifiant de session
    import random
    import string
    return ''.join(random.choices(string.ascii_letters + string.digits, k=16))


def create_session() -> Tuple[str, dict]:
    """
    Renvoie un identifiant de session et un dictionnaire vide pour la session.
    """
    # on génère un nouvel identifiant de session
    session_id = create_session_id()
    SESSIONS[session_id] = {}
    return session_id, SESSIONS[session_id]

@route("/compteur")
def compteur():
    # on récupère la session
    session = get_session()
    if session is None:
        # si la session n'existe pas, on en crée une nouvelle
        session_id, session = create_session()
        response.set_cookie("session_id", session_id, path="/", httponly=True)
    # on incrémente le compteur
    session["compteur"] += 1
    return f"Compteur : {session['compteur']}"
```

33.3. Stockage persistant

Le code ci-dessus ne permet pas de conserver les sessions entre les redémarrages du serveur. Pour cela, il faut stocker les sessions. Pour l'instant, nous allons stocker chaque session dans un fichier JSON, mais nous verrons plus tard d'autres solutions plus fiables avec une base de données. En effet, le stockage dans des fichiers est une solution simple, mais pas très performante (accès disque à chaque requête), et elle ne permet pas une mise à l'échelle horizontale (plusieurs serveurs).

On stockera chaque session dans un répertoire dédié `sessions` situé dans le même répertoire que le script, en utilisant la bibliothèque `json` de Python.

Télécharger le code

```python
from os import path
from bottle import request, response, route, run
from typing import Tuple
import json
# Le répertoire où seront stockées les sessions, il doit exister.
SESSIONS_PATH = path.join(path.dirname(__file__), "sessions")

def create_session() -> Tuple[str, dict]:
    """
    Renvoie un identifiant de session aléatoire et un dictionnaire vide
    pour la session.
    """
    # on génère une chaîne aléatoire pour l'identifiant de session
    import random
    import string
    session_id = ''.join(random.choices(string.ascii_letters + string.digits,
k=16))
    response.set_cookie("session_id", session_id, path="/", httponly=True)
    return session_id, dict()


def get_or_create_session() -> Tuple[str, dict, bool]:
    """
    Récupère la session associée à l'identifiant de session dans le cookie,
    ou en crée une nouvelle si elle n'existe pas.
    Renvoie l'identifiant de session, la session et un booléen indiquant
    si la session a été créée ou non.
    """
    # on récupère l'identifiant de session dans le cookie
    session_id = request.get_cookie("session_id", None)
    # on vérifie que l'identifiant de session ne contient que des caractères
    # alphanumériques
    if session_id is None or not session_id.isalnum():
        session_id, session = create_session()
        return session_id, session, True
    session_file = path.join(SESSIONS_PATH, session_id + ".json")
    try:
        with open(session_file, "r") as f:
            session = json.load(f)
            return session_id, session, False
    except FileNotFoundError:
        pass
    session_id, session = create_session()
    return session_id, session, True

def save_session(session_id: str, session: dict) -> None:
    """
    Enregistre la session dans un fichier JSON.
    A appeler après chaque modification de la session.
    """
    session_file = path.join(SESSIONS_PATH, session_id + ".json")
    with open(session_file, "w") as f:
        json.dump(session, f)


@route("/compteur")
def compteur():
    # on récupère la session
    session_id, session, created = get_or_create_session()
    # on incrémente le compteur
    if "compteur" not in session:
        session["compteur"] = 0
    session["compteur"] += 1
    # on enregistre la session
    save_session(session_id, session)
    return f"Compteur : {session['compteur']}"
```

Attention

ATTENTION : cette méthode induit une concurrence (race condition) lorsqu'un utilisateur accède à la session en même temps depuis plusieurs requêtes. Il faut donc utiliser un mécanisme de verrouillage pour éviter d'écrire dans le fichier en même temps, et on se retrouve vite avec un code compliqué.

Il est préférable d'utiliser une base de données et des opérations atomiques ou des transactions pour gérer les sessions.

De toutes façons, étant donné que les sessions sont stockées dans un fichier, cette méthode n'est pas adaptée pour une application en production, où on peut être amené à utiliser plusieurs serveurs (répartition de charge) ou à déployer l'application dans un conteneur avec un stockage non persistant.

33.4. En pratique

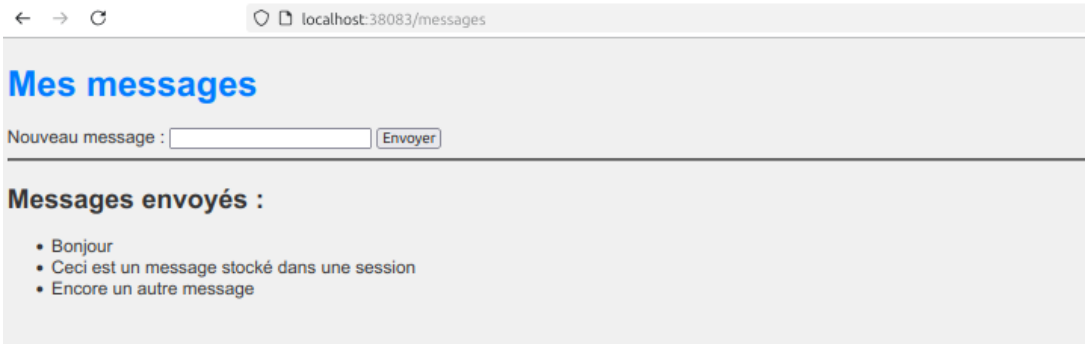
33.4.1. Messages

Créer une vue `/messages` qui affiche un formulaire de saisie de texte, et qui enregistre chaque message saisi dans une session. La page doit afficher l'historique des messages saisis. Le formulaire doit être envoyé en POST.

Vous pouvez télécharger et utiliser le module `sessions` qui reprend le code de l'exemple précédent pour la gestion des sessions utilisateur, ou le copier-coller ci-dessous.

Module sessions

Résultat attendu :



Correction

Programme

- NSI 1ère : interaction client/serveur, requêtes HTTP, réponses du serveur : distinguer ce qui est mémorisé dans le client et retransmis au serveur

34. Authentification

34.1. Procédure d'authentification par login/mot de passe

L'authentification permet de vérifier l'identité d'un utilisateur à l'origine d'une requête. Elle est traditionnellement réalisée en utilisant un nom d'utilisateur et un mot de passe.

1. L'utilisateur envoie ses identifiants (nom d'utilisateur et mot de passe) au serveur (HTTPS recommandé voire obligatoire).
2. Le serveur vérifie les identifiants en les comparant à ceux stockés dans une base de données (même nom d'utilisateur, et comparaison des hashs des mots de passe).
3. Si les identifiants sont valides, le serveur crée une session pour l'utilisateur, enregistre l'identifiant de l'utilisateur dans la session, et renvoie un cookie de session au navigateur.
4. Le navigateur envoie le cookie de session avec chaque requête suivante, permettant au serveur d'identifier l'utilisateur.

Le fait d'enregistrer l'association entre l'utilisateur dans les données de session (donc côté serveur) permet d'avoir confiance dans l'authentification (le client ne peut pas modifier ces données). Reste le fait qu'une personne tierce en possession de l'identifiant de session peut usurper l'identité de l'utilisateur. Il est donc important de sécuriser la transmission du cookie de session (HTTPS) et de le protéger contre le vol (voir section sécurité : HTTPOnly, Secure, SameSite..).

34.2. Exemple d'authentification - basique

En reprenant le module `sessions` de la section précédente, nous allons créer un système d'authentification basique, en utilisant une base d'utilisateurs stockée "en dur" dans le code.

34.2.1. Hashage de mot de passe

Le mot de passe est un élément sensible, et il est important de ne pas le stocker en clair. Pour cela, on utilise un algorithme de hashage qui permet de transformer le mot de passe en une chaîne de caractères fixe, appelée hash. Le hash est une représentation unique du mot de passe, et il est impossible de retrouver le mot de passe à partir du hash.

Pour un hashage sécurisé, il est recommandé d'utiliser des algorithmes de hashage

résistants aux attaques par force brute, comme `bcrypt` ou `argon2`. Ces algorithmes sont plus lents et nécessitent des ressources supplémentaires, mais ils rendent les attaques par force brute beaucoup plus difficiles.

Pour éviter les attaques par dictionnaire, il est également recommandé d'utiliser un "sel" (salt) : une chaîne de caractères aléatoire ajoutée au mot de passe avant le hashage. Cela rend le hash unique même si deux utilisateurs ont le même mot de passe. Le sel doit être stocké avec le hash, mais il n'est pas nécessaire de le rendre secret.

Nous utiliserons ici simplement l'algorithme `sha256` pour le hashage, qui ne répond pas à ces critères de sécurité, mais qui est simple à mettre en oeuvre.

Télécharger le code

```python
# Création d'un hash de mot de passe
import hashlib
def hash_password(password: str) -> str:
    """
    Renvoie le hash sha256 du mot de passe.
    """
    return hashlib.sha256(password.encode()).hexdigest()
```

Liste des utilisateurs :

Télécharger le code

```python
# Un dictionnaire par utilisateur, contenant nom, prénom et sha256 du mot de
passe
USERS = {
    "alice": {
        "nom": "Alice",
        "prenom": "Dupont",
        # évidement aucun intérêt ici d'avoir le hash si le mdp est dans le
code
        # c'est juste pour l'exemple :)
        "password_hash": hash_password("password")
    },
    "bob": {
        "nom": "Bob",
        "prenom": "Martin",
        # idem !
        "password_hash": hash_password("supersecret")
    }
}
```

34.2.2. Vue de login

Télécharger le code

```python
from bottle import route, run, request, redirect

@route("/login", method=["GET", "POST"])
def login():
    """
    Authentifie l'utilisateur et crée une session.
    Reçoit le nom d'utilisateur et le mot de passe en POST.
    La page de redirection est spécifiée dans le paramètre "next" de l'URL.
    Rajouter un template et une logique pour afficher les erreurs.
    """
    if request.method == "POST":
        # on récupère le nom d'utilisateur et le mot de passe
        username = request.forms.get("username")
        password = request.forms.get("password")
        next = request.query.get("next", "/")

        # on vérifie si l'utilisateur existe et si le mot de passe est correct
        if (username in USERS
            and USERS[username]["password_hash"] == hash_password(password)):
            # on crée une session pour l'utilisateur
            session_id, session_data, created = get_or_create_session()
            session_data["user"] = username
            session_data["nom"] = USERS[username]["nom"]
            session_data["prenom"] = USERS[username]["prenom"]
            save_session(session_id, session_data)
            return redirect(next)
        else:
            return "Nom d'utilisateur ou mot de passe incorrect."

    return '''
        <form action="" method="post">
            Nom d'utilisateur: <input name="username" type="text" />
            Mot de passe: <input name="password" type="password" />
            <input value="Se connecter" type="submit" />
        </form>
    '''
```

34.2.3. Page protégée - utilisation des données utilisateur

Télécharger le code

```python
@route("/")
def index():
    """
    Page protégée, accessible uniquement si l'utilisateur est authentifié.
    """
    _, session, _ = get_or_create_session()
    if "user" not in session:
        return redirect("/login?next=/")
```

34.3. En pratique

34.3.1. Vue de login, protection de la calculatrice

Créez (à partir de l'exemple ci-dessus) une vue de login `/login` qui utilise un template, et qui affiche le formulaire de login, et un messages d'erreur si le login ou le mot de passe est incorrect.

Protégez la page `/calculatrice` pour qu'elle ne soit accessible qu'aux utilisateurs authentifiés.

Correction ›

34.3.2. Déconnexion

Ajoutez une vue de déconnexion `/logout` qui :

- affiche un message de confirmation et un bouton si la page est appelée en `GET`
- détruit la session de l'utilisateur et le redirige vers la page d'accueil si la page est appelée en `POST`

Pour détruire la session, vous pouvez utiliser le code ci-dessous, à rajouter dans le module `sessions` :

```
def delete_session() -> None:
    """
    Supprime la session associée à l'identifiant de session.
    """
    # on récupère l'identifiant de session dans le cookie
    session_id = request.get_cookie("session_id", None)
    # on vérifie que l'identifiant de session ne contient que des caractères
    # alphanumériques
    if session_id is None or not session_id.isalnum():
        return
    session_file = path.join(SESSIONS_PATH, session_id + ".json")
    response.delete_cookie("session_id")
    try:
        os.remove(session_file)
    except FileNotFoundError:
        pass
```

Correction ›

Programme

- NSI 1ère : interaction client/serveur, requêtes HTTP, réponses du serveur : distinguer ce qui est mémorisé dans le client et retransmis au serveur
- NSI terminale : sécurisation des communications (un peu)

35. Web tokens

Beaucoup de sites utilisent des "JSON Web Tokens" (JWT) pour gérer l'authentification des utilisateurs.

Références : https://jwt.io/introduction

36. Bases de données

Dans une application web, la persistance des données est généralement assurée par une base de données. Nous allons faire un tour d'horizon des bases de données et de leur utilisation dans une application web en python.

37. SQLite

SQLite est un système de gestion de base de données relationnelle (SGBDR) léger et **intégré**. Il est souvent utilisé pour des applications embarquées ou des projets de petite à moyenne envergure. SQLite est écrit en C et est disponible sous une licence libre. Il est très populaire en raison de sa simplicité, de sa rapidité **et de sa capacité à fonctionner sans serveur**. SQLite stocke les données dans un fichier unique sur le disque, ce qui le rend facile à déployer et à gérer. Inconvénient : le revers de la médaille de ses avantages : il n'est pas conçu pour gérer des applications à fort volume de transactions ou des environnements multi-serveurs / multi-utilisateurs. Il est donc à réserver pour des applications légères ou des prototypes.

37.1. Installation

SQLite est compilé par défaut dans la plupart des distributions Python. Vous pouvez vérifier si SQLite est installé en exécutant la commande suivante dans un terminal :

```
python -c "import sqlite3; print(sqlite3.sqlite_version)"
```

37.2. Utilisation en python

Le module `sqlite3` de Python permet d'intéragir avec les bases de données SQLite. La documentation officielle de Python fournit des informations détaillées sur l'utilisation de ce module : sqlite3 documentation.

37.2.1. Connexion

La fonction `connect` permet de se "connecter" à une base de données SQLite. Si la base de données n'existe pas, elle sera créée. Si vous précisez ":memory:", une base de données temporaire sera créée en mémoire vive (RAM) et sera détruite à la fermeture de la connexion.

```
import sqlite3
conn = sqlite3.connect(":memory:")
# conn = sqlite3.connect("data.db")
```

Par défaut, les bases de données SQLite ne gèrent pas les contraintes d'intégrité référentielle (foreign key). Pour activer cette fonctionnalité **pour la connexion courante**, vous devez exécuter `PRAGMA foreign_keys = ON` :

```
conn.execute('PRAGMA foreign_keys = ON')
```

De manière générale, l'utilisation de la connexion est faite à travers des curseurs (`cursor` - un curseur est un objet qui permet d'exécuter des requêtes SQL et de récupérer les résultats), y compris lorsque la requête semble être faite via la connexion comme ci-dessus (dans ce cas, un curseur est automatiquement créé). Vous pouvez créer un curseur en utilisant la méthode `cursor()` de la connexion :

```
cursor = conn.cursor()
```

37.2.2. Exécution de requêtes

Ce curseur peut alors être utilisé pour **exécuter** des requêtes SQL. Par exemple, pour créer une table :

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS ingredients (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL
);
""")
```

Pour insérer des données dans la table :

```
cursor.execute("""
INSERT INTO ingredients (name)
VALUES ('Mozzarella'),('Tomate'),('Basilic'),
       ('Jambon'),('Olive'),('Oignon'),('Oeuf');
""")
conn.commit()
```

La méthode `commit()` **de la connexion** est utilisée pour valider les modifications apportées à la base de données, car "INSERT" ouvre automatiquement une *transaction*. Si vous ne l'appelez pas, les modifications ne seront pas enregistrées.

Autre manière de faire : utiliser le `manager de contexte` (le mot-clé `with`) pour gérer la connexion, la création du curseur et le commit automatiquement :

```
with conn:
    conn.execute('''INSERT INTO ingredients (name) VALUES ('Merguez');''')
```

37.2.3. Récupération de données

Pour récupérer des données, vous pouvez utiliser la méthode `fetchall()` ou `fetchone()` du curseur. Par exemple, pour récupérer tous les ingrédients :

```
cursor.execute('SELECT id,name FROM ingredients')
ingredients = cursor.fetchall()
for ingredient in ingredients:
    print(ingredient) # (1, 'Mozzarella'), ...
```

`fetchall()` renvoie une liste de tuple, chaque tuple représentant une ligne de résultat. D'autres méthodes de récupération existent :

- `fetchone()` : récupère une seule ligne de résultat. Si aucune ligne n'est trouvée, renvoie `None`.
- `fetchmany(size)` : récupère un nombre spécifié de lignes de résultat. Si aucune

ligne n'est trouvée, renvoie une liste vide.

37.2.4. Insertion de données avec des paramètres

Il est impératif de ne pas construire des requêtes SQL en concaténant des chaînes de caractères, car cela peut entraîner des failles de sécurité (injection SQL). Utilisez plutôt des paramètres pour insérer des données :

```
cursor.execute('INSERT INTO ingredients (name) VALUES (?)', ('Viande
hachée',))
```

Chaque `?` est remplacé par la valeur correspondante dans le tuple passé en second argument (pensez à la virgule finale pour le tuple à un seul élément).

Pour insérer plusieurs lignes, vous pouvez utiliser `executemany()` :

```
cursor.executemany('INSERT INTO ingredients (name) VALUES (?)',
                   [('Viande hachée',), ('Chèvre',)])
```

Programme

- NSI terminale : Système de gestion de bases de données relationnelles
- NSI terminale : Langage SQL : requêtes d'interrogation et de mise à jour d'une base de données

38. Utilisation serveur

38.1. Connexion globale

Pour utiliser sqlite dans un serveur bottle, il est préférable de créer une connexion **globale** à la base de données et de l'utiliser dans les différentes routes.

Création de la connexion, création et remplissage de la table `ingredients` (télécharger le code) :

```
import sqlite3

# Connexion à la base de données
db_conn = sqlite3.connect('data.db')
db_conn.execute('PRAGMA foreign_keys = ON')

# Préparation de la base de données
db_conn.execute('''
CREATE TABLE IF NOT EXISTS ingredients (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL UNIQUE
);''')
try:
    with db_conn:
        db_conn.execute('''
        INSERT INTO ingredients (name)
        VALUES ('Mozzarella'),('Tomate'),('Basilic'),
               ('Jambon'),('Olive'),
                ('Oignon'),('Oeuf');
        ''')
except sqlite3.IntegrityError:
    # Si la table a déjà été remplie, on ne fait rien
    pass
```

38.2. Utilisation de la connexion

La connexion globale est ensuite utilisée dans les différentes routes. Exemple : affichage de la liste des ingrédients (télécharger le code) :

```
@route('/ingredients')
def ingredients():
    # utilisation de la référence à la connexion globale
    cursor = db_conn.cursor()
    cursor.execute('SELECT * FROM ingredients')
    rows = cursor.fetchall()
    out = "<h1>Liste des ingrédients</h1>"
    out += "<ul>"
    for row in rows:
        out += f"<li>{row[0]} : {row[1]}</li>"
    out += "</ul>"
    return out
```

38.3. En pratique

Modifiez le code ci-dessus pour :

- utiliser un template pour la page `/ingredients`
- en incluant un formulaire pour ajouter un nouvel ingrédient (via la méthode `POST` vers `/ingredients`)
- en incluant un formulaire pour filtrer les ingrédients par nom (via la méthode `GET` vers `/ingredients?q=Tomate` par exemple)
- en affichant la liste des ingrédients dans un tableau HTML
- en affichant un message d'erreur si l'ingrédient existe déjà (vous pouvez utiliser un `try/except` pour gérer l'erreur d'intégrité)

Correction ›

38.4. Précautions

Par défaut, en mode développement, Bottle ne tourne que dans un seul thread et un seul processus. Ceci ne pose aucun problème pour SQLite. En revanche, pour un usage en production, notre application peut être lancée dans plusieurs threads voire processus. Dans ce cas, il faut faire attention à la gestion de la base de données. En effet, SQLite ne gère pas forcément bien les accès concurrents.

Pour vérifier le support des accès concurrents dans SQLITE (ref : https://docs.python.org/3/library/sqlite3.html#sqlite3.threadsafety, vous pouvez exécuter la commande suivante dans un terminal :

```
python -c "import sqlite3; print(sqlite3.threadsafety)"
```

Selon la valeur retournée, la gestion des accès concurrents peut être :

- `0` : pas de gestion des accès concurrents, pas de partage du module, ni des connexions, ni des curseurs
- `1` : gestion des accès concurrents par le module sqlite3, mais pas de partage des connexions ni des curseurs
- `3` : gestion des accès concurrents par le module sqlite3 et partage des connexions et des curseurs

Si le `threadsafety` est à `0` , il vaut mieux réfléchir à une autre solution (changer de base de données, ou changer de distribution Python).

Programme

- NSI terminale : Système de gestion de bases de données relationnelles
- NSI terminale : Langage SQL : requêtes d'interrogation et de mise à jour d'une base de données

39. Injection SQL

39.1. Principe

Une injection SQL est une attaque qui consiste à insérer un code SQL malveillant dans une requête SQL afin de manipuler la base de données. Cela peut permettre à un attaquant d'accéder à des données sensibles, de modifier ou de supprimer des données, ou même de prendre le contrôle total de la base de données.

Pour éviter les injections SQL, il est important de se rappeler qu'il ne faut **jamais faire confiance aux données fournies par l'utilisateur**. Par conséquent, il ne faut **jamais** construire de requêtes SQL en concaténant des chaînes de caractères. Au lieu de cela, il est préférable d'utiliser des **paramètres** dans les requêtes SQL.

39.2. Exemple / demo

Si dans l'exemple précédent, au lieu d'utiliser des paramètres dans la requête SQL, on avait construit la requête SQL en concaténant des chaînes de caractères, on aurait pu avoir quelque chose comme :

```
@route('/ingredients')
def ingredients():
    q = request.query.get('q', '')
    #...
    cursor.execute(f"""
        SELECT id, name
        FROM ingredients
        WHERE UPPER(name) LIKE '%{q.upper()}%'
    """)
    #...
```

Dans ce cas, si l'utilisateur entre `Tomate' OR 1=1 --`, la requête exécutée sera :

```
SELECT id, name FROM ingredients WHERE UPPER(name) LIKE '%TOMATE' OR 1=1 --%'
```

Ce qui va renvoyer tous les ingrédients de la base de données, car `1=1` est toujours vrai. Le `--` permet de commenter le reste de la requête, ce qui évite les erreurs de syntaxe.

Avec des injections SQL plus complexes, il est possible d'accéder à l'ensemble de la base de données, à son schéma, de la modifier, de la supprimer, etc.. Un outil permettant d'exploiter les injections SQL est sqlmap. Pour l'installer, il suffit de cloner le dépôt :

```
git clone --depth 1 https://github.com/sqlmapproject/sqlmap.git sqlmap-dev
cd sqlmap-dev
```

Puis de l'exécuter avec l'url de l'application web en argument :

```
python sqlmap.py 'http://localhost:8080/ingredients?q=Tomate' # ...
```

Sqlmap va alors chercher les injections SQL possibles et essayer de les exploiter. Une fois l'injection trouvée, il est possible de l'utiliser pour extraire les données de la base.

Exemple en "video" sur notre application : screencast session sqlmap. On ajoute le `--delay 0.1` pour éviter de surcharger le serveur avec trop de requêtes simultanées, en mode `debug` ça ne passe pas terriblement.

```
[*] starting @ 11:05:25 /2025-04-21/

[11:05:25] [INFO] resuming back-end DBMS 'sqlite'
[11:05:25] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: q (GET)
    Type: boolean-based blind
    Title: SQLite AND boolean-based blind - WHERE, HAVING, GROUP BY or HAVING clause (JSON)
```

En quelques secondes, sqlmap a trouvé et exploité l'injection SQL. Il a pu extraire les données de la base de données, y compris le schéma de la base de données, les tables et les colonnes.

Pour info, les logs serveur associés à cette session.

Programme

- NSI terminale : Mise au point des programmes, gestion des bugs (éventuellement)

40. SQL / ORM

Dès qu'une application web devient un peu complexe, il est nécessaire de passer par une "vraie" base de données (sans enlever aucun mérite à SQLite qui fait parfaitement bien ce pour quoi il a été conçu).

Au niveau des bases de données relationnelles **libres** (on oublie tout de suite Oracle et SQL Server), il y a principalement deux SGDB :

- **PostgreSQL** : base de données relationnelle extrêmement puissante et riche en fonctionnalités. Elle est souvent considérée (avis perso partagé par pas mal de monde..) comme la meilleure base de données relationnelle libre. Elle est très bien intégrée avec Python (et donc Bottle).
- **MySQL** : base de données relationnelle extrêmement populaire, mais qui a perdu de sa superbe depuis le rachat par Oracle. Un "fork" existe, créé par le concepteur original de MySQL: **MariaDB**. Elle est souvent utilisée pour des applications web, mais elle est moins riche en fonctionnalités que PostgreSQL. Elle est également très bien intégrée avec Python (et donc Bottle).

Dans l'idéal, le code de l'application devrait être indépendant du SGBD utilisé. Vu l'ampleur des différences entre les différents SGDB, il est extrêmement difficile d'écrire "à la main" du code réellement indépendant du SGBD, sauf à utiliser un ORM (Object Relational Mapping) qui va se charger de traduire le code python en code SQL (ou en code de l'ORM). Il existe plusieurs ORM pour Python, mais le plus populaire est **SQLAlchemy**.

SQLAlchemy sait "parler" plusieurs "dialectes" SQL (PostgreSQL, MySQL, SQLite, Oracle, M$ SQL Server) : https://docs.sqlalchemy.org/en/21/dialects/index.html.

D'autres frameworks web (comme Django) intègrent directement un ORM extrêmement puissant et complet.

41. PostgreSQL

41.1. Installation

L'installation de PostgreSQL sort du cadre de ce document. On va supposer qu'une instance de PostgreSQL est déjà installée et accessible sur le port 5432 (port par défaut de PostgreSQL).

Pour cette formation, `pg.solidev.net` est disponible, avec des utilisateurs `userXX` associés à une base de données `dbuserXX` (où `XX` est le numéro de l'utilisateur). https://pg.solidev.net est une instance de PgAdmin, un outil d'administration de PostgreSQL. Des comptes `userXX@nomail.com` ont été créés pour permettre d'administrer les bases de données en mode "graphique".

Nous allons utiliser `sqlalchemy` pour interagir avec PostgreSQL, ainsi que le driver `psycopg` pour PostgreSQL. Pour installer ces deux paquets, il suffit d'utiliser `pip` ou `uv` :

```
pip install sqlalchemy psycopg[binary,pool]
```

ou

```
uv add sqlalchemy psycopg[binary,pool]
```

Référence : https://docs.sqlalchemy.org/en/21/dialects/postgresql.html

41.2. Connexion à PostgreSQL - utilisation en mode "direct"

La connexion à PostgreSQL se fait en créant un "Moteur" SQLAlchemy. Le moteur permet de se connecter à la base de données et d'exécuter des requêtes SQL. La chaîne de connexion permet de spécifier le type de base de données, le driver utilisé, le nom d'utilisateur, mot de passe, nom de la base de données, etc..

Référence : https://docs.sqlalchemy.org/en/20/core/engines.html#engine-creation-api

```
from sqlalchemy import create_engine
engine = create_engine(
    'postgresql+psycopg://userXX:password@pg.solidev.net/dbuserXX',
    echo=True
)
```

Le paramètre `echo=True` permet d'afficher les requêtes SQL exécutées dans la console. Il est très utile pour le débogage, mais à éviter en production.

On peut tester la connexion en exécutant une requête SQL simple. Par exemple, pour vérifier que la connexion fonctionne, on peut exécuter une requête qui renvoie `1` :

```
from sqlalchemy import text
with engine.connect() as sqa_conn:
    result = sqa_conn.execute(text("SELECT 1"))
    for row in result:
        print(row[0])
```

Il est alors possible d'exécuter des requêtes SQL directement sur le moteur en utilisant la méthode `execute()` : (reference : https://docs.sqlalchemy.org/en/20/tutorial/dbapi_transactions.html#getting-a-connection)

```
from sqlalchemy import text, exc as sa_exc
with engine.connect() as sqa_conn:
    sqa_conn.execute(text("""
    CREATE TABLE IF NOT EXISTS ingredients (
        id BIGSERIAL PRIMARY KEY,
        name VARCHAR NOT NULL UNIQUE
    );
    """))
    sqa_conn.commit()

    try:
        sqa_conn.execute(text("""
        INSERT INTO ingredients (name)
        VALUES ('Mozzarella'),('Tomate'),('Basilic'),
                ('Jambon'),('Olive'),
                    ('Oignon'),('Oeuf');
        """))
        sqa_conn.commit()
    except sa_exc.IntegrityError:
        # Si la table a déjà été remplie, on ne fait rien
        pass
```

La récupération de données se fait en utilisant aussi la méthode `execute()`. Le résultat renvoyé est un objet de type `Result` qui permet de récupérer les lignes de la requète, et d'accèder aux colonnes par leur nom ou leur index. Par exemple, pour récupérer les ingrédients de la table `ingredients` :

```
with engine.connect() as sqa_conn:
    result = sqa_conn.execute(text("SELECT id, name FROM ingredients"))
    for row in result:
        print(row['id'], row['name'])
```

ou

```
with engine.connect() as sqa_conn:
    result = sqa_conn.execute(text("SELECT id, name FROM ingredients"))
    for row in result:
        print(row[0], row[1])
```

Le paramètrage des requètes SQL se fait en utilisant des paramètres nommés :

```
q = request.query.get('q', '')
with engine.connect() as sqa_conn:
    result = sqa_conn.execute(text("SELECT id, name FROM ingredients WHERE
name ILIKE :name"), {'name': f"%{q}%"})
    for row in result:
        print(row['id'], row['name'])
```

Référence : SQLAlchemy - PostgreSQL dialect

41.3. Connexion à PostgreSQL - utilisation en mode "ORM"

Hors du cadre de la formation, voir le reste du tutoriel SQLAlchemy : https://docs.sqlalchemy.org/en/20/tutorial/index.html

---

Programme

- NSI terminale : Système de gestion de bases de données relationnelles

42. MySQL

Référence : https://docs.sqlalchemy.org/en/21/dialects/mysql.html

Ce qui a été fait précédemment pour `postgresql` est également valable pour `MySQL` et `MariaDB`. Il faut juste changer le nom du dialecte dans la chaîne de connexion. Par exemple, pour `MySQL` :

```
pip install sqlalchemy pymysql
```

ou

```
uv add sqlalchemy pymysql
```

```
from sqlalchemy import create_engine
engine = create_engine('mysql+pymysql://user:password@host/db', echo=True)
```

43. NoSQL

Selon les besoins, il est possible d'utiliser des bases de données NoSQL (base de données non relationnelles).

Dans les raisons qui peuvent pousser à utiliser une base de données NoSQL, on peut citer :

- besoin de scalabilité horizontale (ajout de serveurs pour augmenter la capacité de la base de données) :
  - MongoDB, CouchDB, Cassandra, etc.
- besoin de flexibilité dans le schéma de la base de données :
  - MongoDB, CouchDB, etc.
- besoin de performances élevées pour des opérations de lecture/écriture simples (cache, etc..) :
  - Redis, etc.

43.1. Redis

Exemple d'utilisation de Redis comme cache pour une application web en python.

```
import redis

# Connexion à Redis
rconn = redis.Redis(host='localhost', port=6379, db=0)


@route('/vue_lourde_a_afficher')
def vue_lourde():

    # Vérification si le résultat est déjà en cache
    result = rconn.get('clef_cache_vue_lourde')
    if result is None:
        # Si le résultat n'est pas en cache, on effectue le calcul
        result = calcul_lourd()
        # On met le résultat en cache pour 10 minutes
        rconn.setex('clef_cache_vue_lourde', 600, result)
    return result
```

44. Stockage de sessions

On peut utiliser une base de données pour stocker les sessions. Cela permet de gérer la concurrence et éventuellement de partager les sessions entre plusieurs serveurs. On peut par exemple utiliser une table `sessions` avec les colonnes suivantes :

- `session_id` : identifiant de session (clé primaire)
- `key` : clé de la valeur de session
- `value` : valeur de la clef de session
- `created_at` : date de création de la session
- `updated_at` : date de dernière mise à jour de la session

Avec un index et une contrainte d'unicité sur la combinaison `session_id` et `key` (ou une clef primaire composite).

```
CREATE TABLE sessions (
    session_id TEXT NOT NULL,
    key TEXT NOT NULL,
    value TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (session_id, key)
);
```

Pour lire la valeur d'une clef de la session, on peut faire une requête SQL suivante :

```
SELECT value FROM sessions WHERE session_id = ? AND key = ?
```

Pour écrire la valeur d'une clef de la session, on peut faire une requête SQL suivante (insertion ou mise à jour si conflit, permet de rendre la requête plus atomique) :

```
INSERT INTO sessions (session_id, key, value) VALUES (?, ?, ?)
    ON CONFLICT (session_id, key) DO
        UPDATE SET value = ?, updated_at = CURRENT_TIMESTAMP
```

Pour supprimer une session, on peut faire une requête SQL suivante :

```
DELETE FROM sessions WHERE session_id = ?
```

Un autre choix possible est de stocker toutes les données de session dans une seule colonne, en utilisant un format de sérialisation comme JSON. `Redis` est souvent utilisé pour stocker les sessions.

45. Déploiement

Pour l'instant, notre application tourne en local. Pour la rendre accessible à tous (et en https, on est en 2025..), il faut généralement sortir la CB.

Il y a besoin de plusieurs choses pour qu'un site soit accessible sur le web :

- un nom de domaine (payant, une dizaine d'euros par an)
- une machine qui tourne 24/7, avec une adresse IP publique (la plupart du temps payant, mais on peut éventuellement bricoler avec une machine chez soi)
- certificat SSL (gratuit, avec Let's Encrypt)

Nous allons prendre un exemple complet de déploiement d'une application web, en utilisant les services de chez Scaleway, de la création du compte à la mise en ligne de l'application en https avec son nom de domaine.

46. Créer une clef SSH

Nous allons communiquer avec notre serveur via SSH. Pour cela, il faut créer une clef SSH qui servira à s'authentifier sur le serveur. La clef publique sera transférée sur le serveur, et la clef privée restera sur votre machine.

46.1. Sous linux et OSX

```
ls ~/.ssh
# Si une clef (id_xxxxx.pub) existe déjà, on peut l'utiliser.
# Sinon, on en crée une nouvelle
ssh-keygen # Laisser toutes les options par défaut
cat ~/.ssh/id_ed25519.pub # ou id_rsa.pub
```

46.2. Sous Windows

SSH n'est pas installé par défaut sur Windows. Il faut l'installer (ou utiliser WSL). Vous pouvez faire d'une pierre deux coups en installant `git` sur Windows, qui l'installe aussi[1].

Voir ici pour plus d'infos sur git.

Télécharger et installer git.

Une fois installé, ouvrez `Git Bash` et exécutez les mêmes commandes que pour linux et OSX :

```
ssh-keygen # Laisser toutes les options par défaut
cat ~/.ssh/id_ed25519.pub # ou id_rsa.pub
```

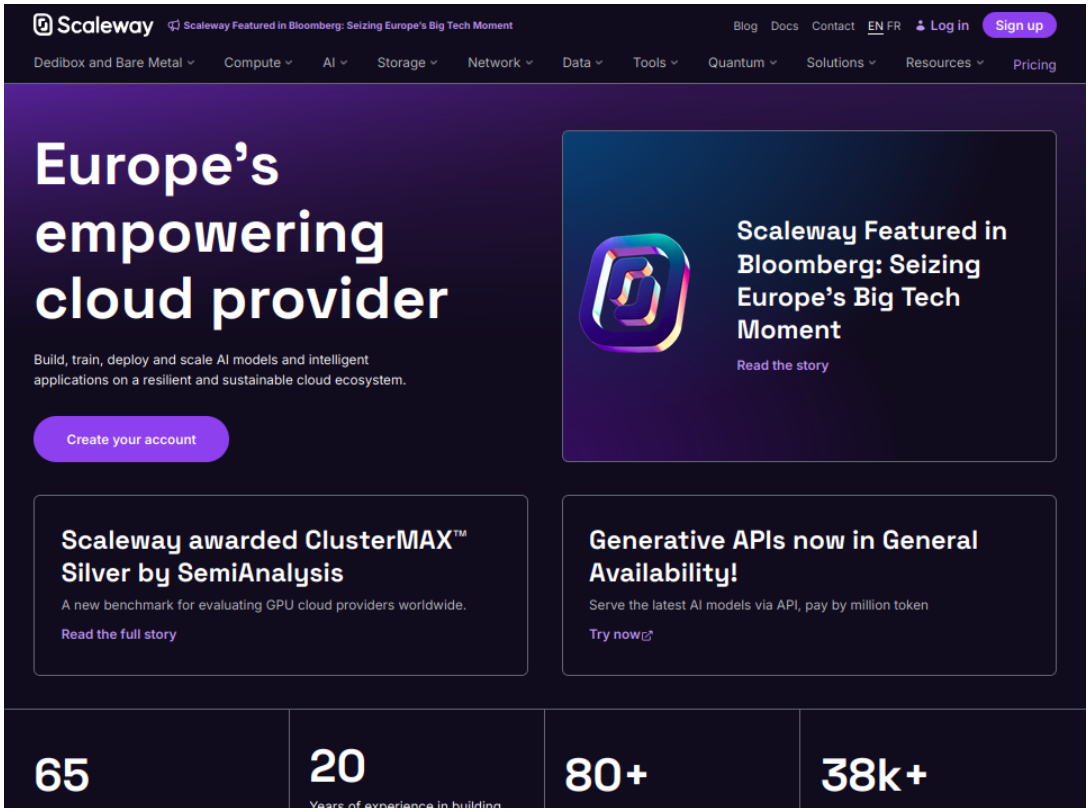
[1] Vous pouvez aussi utiliser `PuTTY` pour vous connecter en SSH à une machine distante. Il faut installer `PuTTY` et `PuTTYgen` (qui est inclus dans l'installation de `PuTTY`). Vous pouvez le télécharger ici. Une fois installé, ouvrez `PuTTYgen` et cliquez sur `Generate`. Suivez les instructions pour générer une nouvelle clef SSH. Une fois la clef générée, vous pouvez la sauvegarder en cliquant sur `Save private key`. Vous pouvez aussi copier la clef publique dans le presse-papier en cliquant sur `Copy public key`. Dans ce document, nous utiliserons plutôt `Git Bash` pour que toutes les commandes soient identiques sur tous les systèmes d'exploitation.

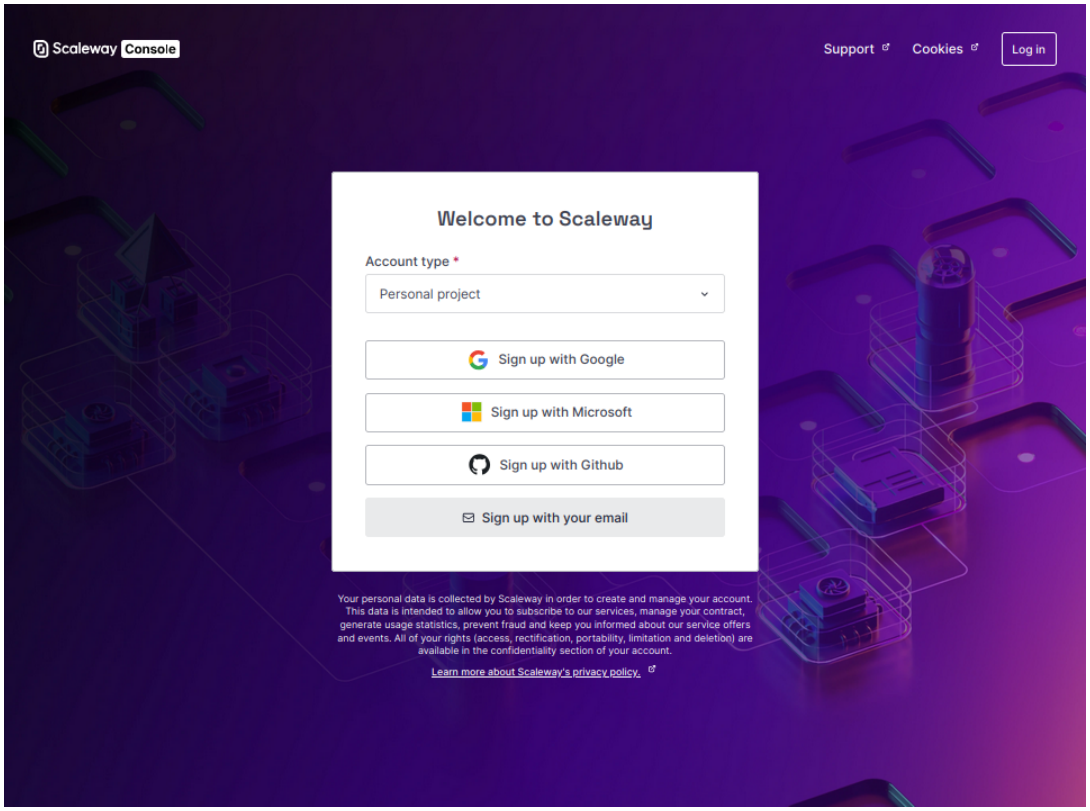
47. Scaleway

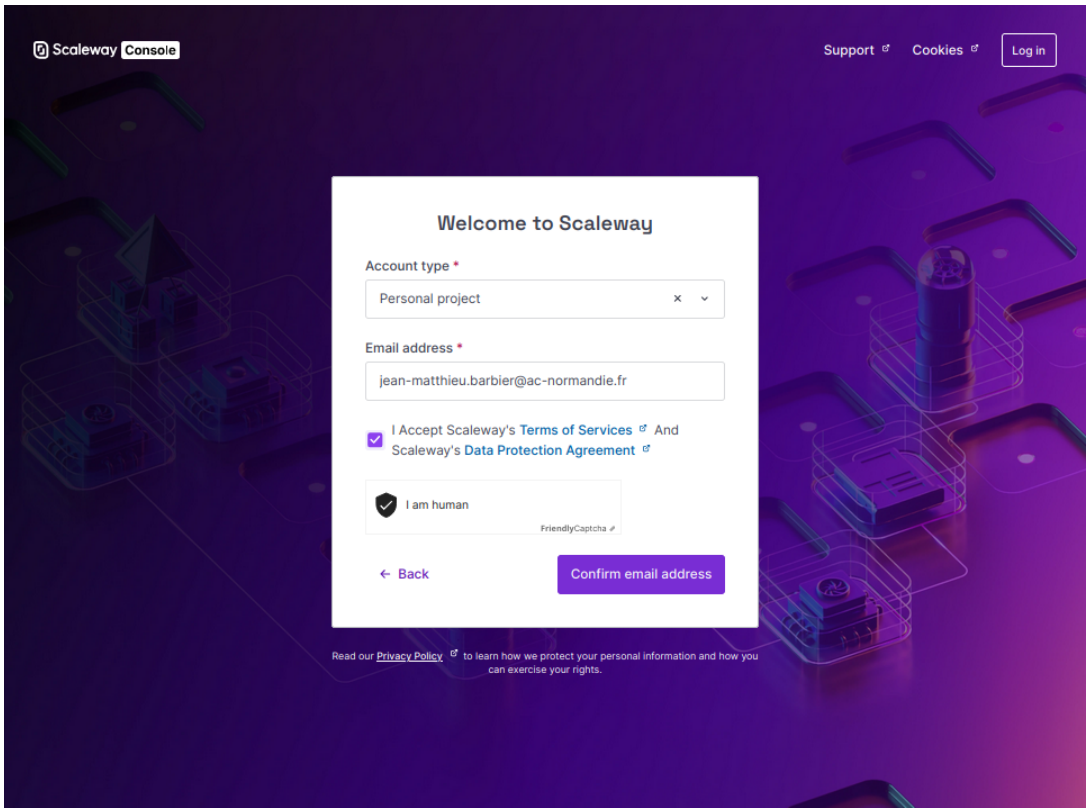
Le choix de Scaleway est assez personnel, vous pouvez aussi aller voir chez OVH ou d'autres fournisseurs de cloud. L'avantage de Scaleway par rapport à OVH est d'avoir une interface d'administration plus simple et plus intuitive. Au niveau des prix, c'est à peu près équivalent.

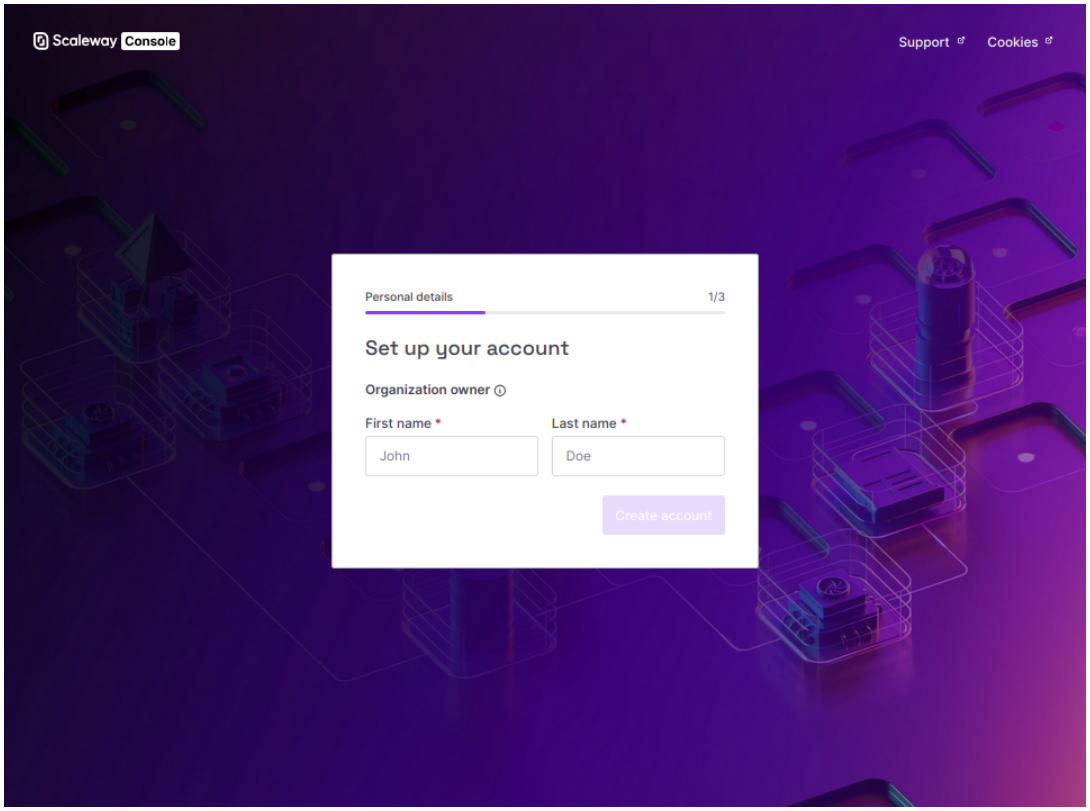
Nous allons commencer par créer un compte sur scaleway. Il faut une CB pour pouvoir finaliser la création du compte.

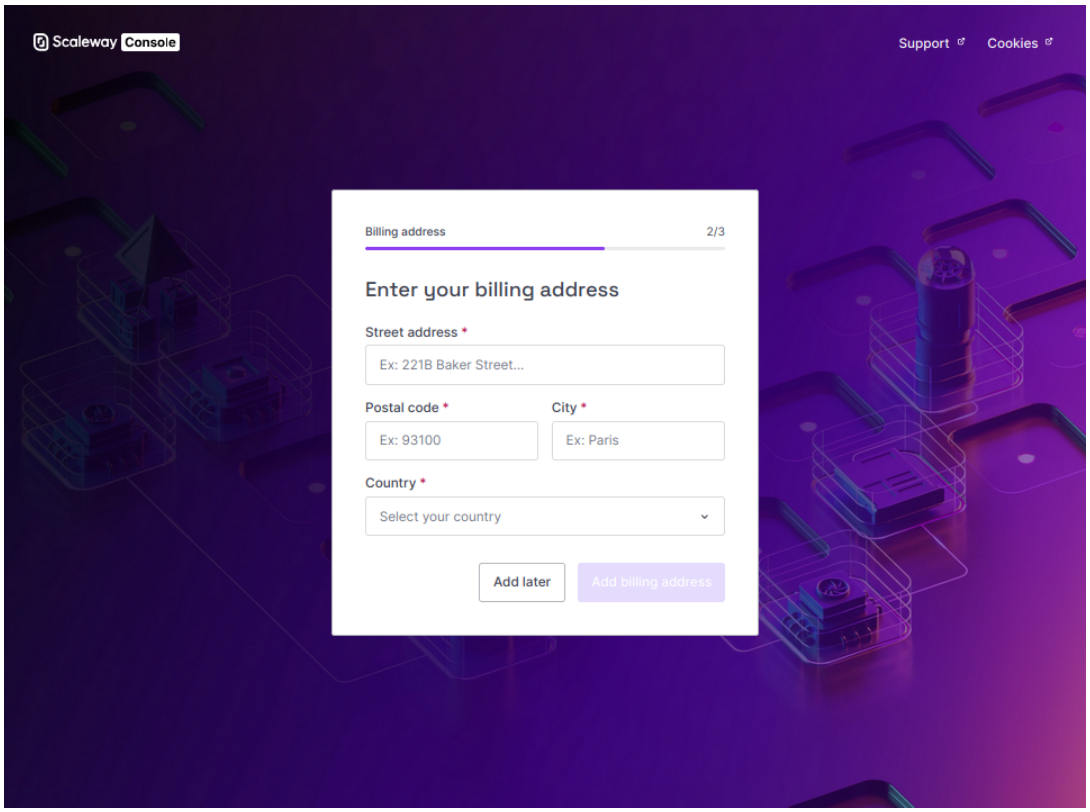
47.1. Étapes de création de compte

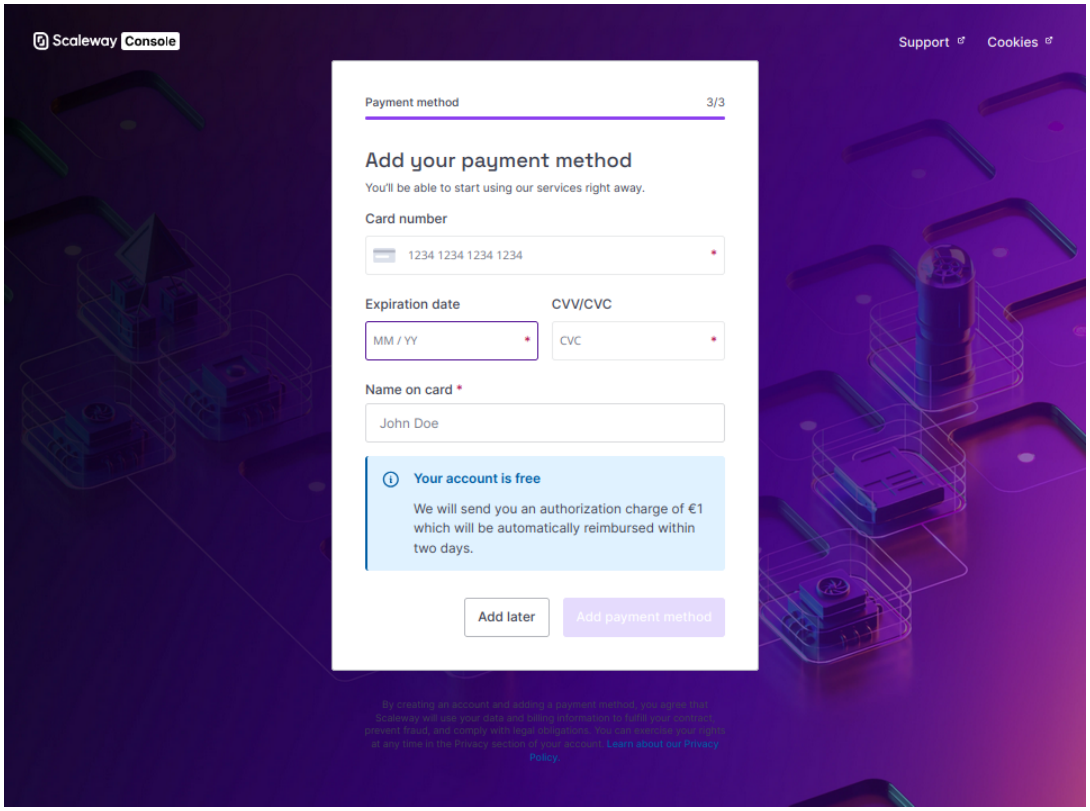



Scaleway Console
Support
Cookies
Log in
Welcome to Scaleway
Account type *
Personal project
Sign up with Google
Sign up with Microsoft
Sign up with Github
Sign up with your email
Your personal data is collected by Scaleway in order to create and manage your account. This data is intended to allow you to subscribe to our services, manage your contract, generate usage statistics, prevent fraud and keep you informed about our service offers and events. All of your rights (access, rectification, portability, limitation and deletion) are available in the confidentiality section of your account.
Learn more about Scaleway's privacy policy.


Scaleway Console
Support
Cookies
Log in
Welcome to Scaleway
Account type *
Personal project
Email address *
jean-matthieu.barbier@ac-normandie.fr
I Accept Scaleway's Terms of Services And Scaleway's Data Protection Agreement
I am human
FriendlyCaptcha
Back
Confirm email address
Read our Privacy Policy to learn how we protect your personal information and how you can exercise your rights.


Scaleway Console
Support
Cookies
Personal details
1/3
Set up your account
Organization owner
First name *
John
Last name *
Doe
Create account

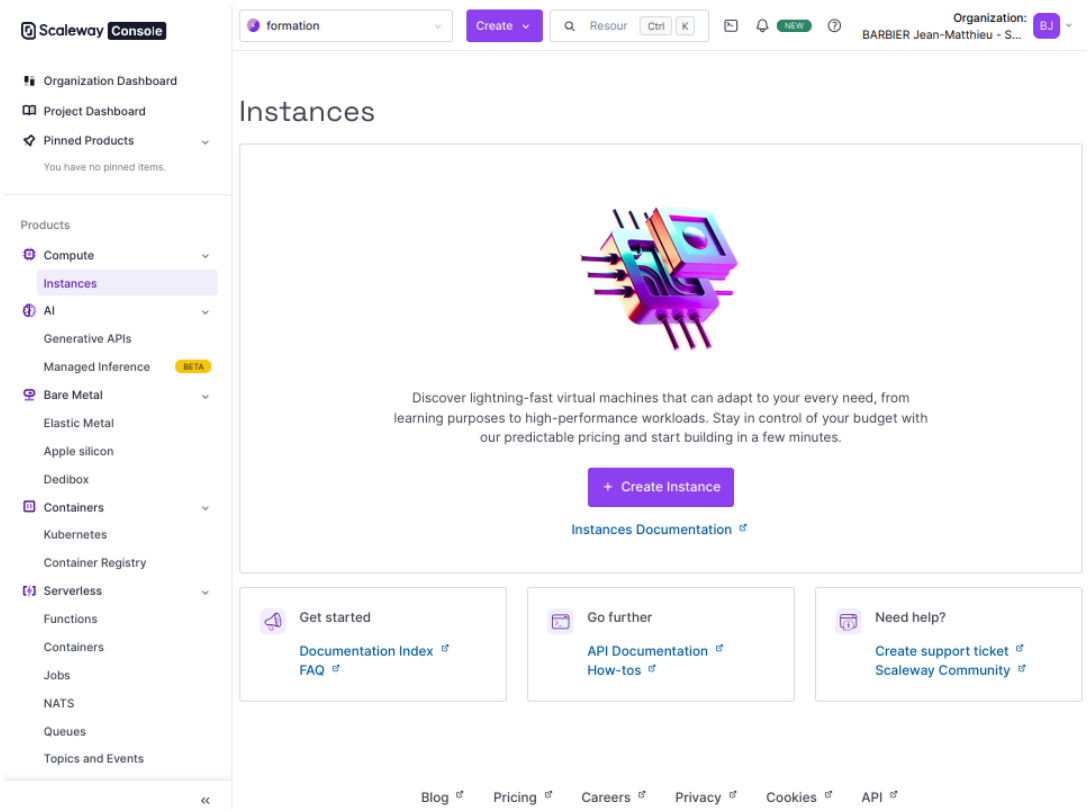

Scaleway Console
Support
Cookies
Billing address
2/3
Enter your billing address
Street address *
Ex: 221B Baker Street...
Postal code *
Ex: 93100
City *
Ex: Paris
Country *
Select your country
Add later
Add billing address



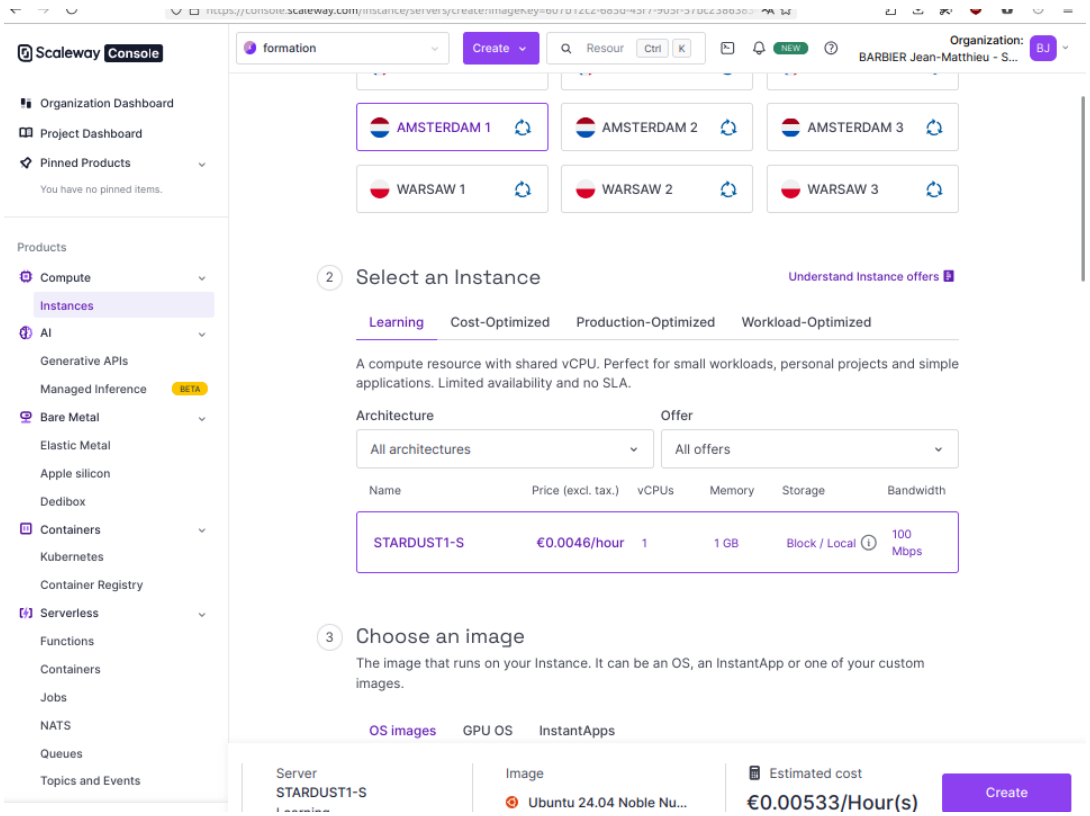
47.2. Création d'une instance

Scaleway fournit une instance de test appelée `Stardust`, disponible à moins de 4 euros par mois. Il faut parfois la chercher un peu dans la liste des "Availability Zones". En ce moment, est est disponible dans la zone `AMSTERDAM1`.

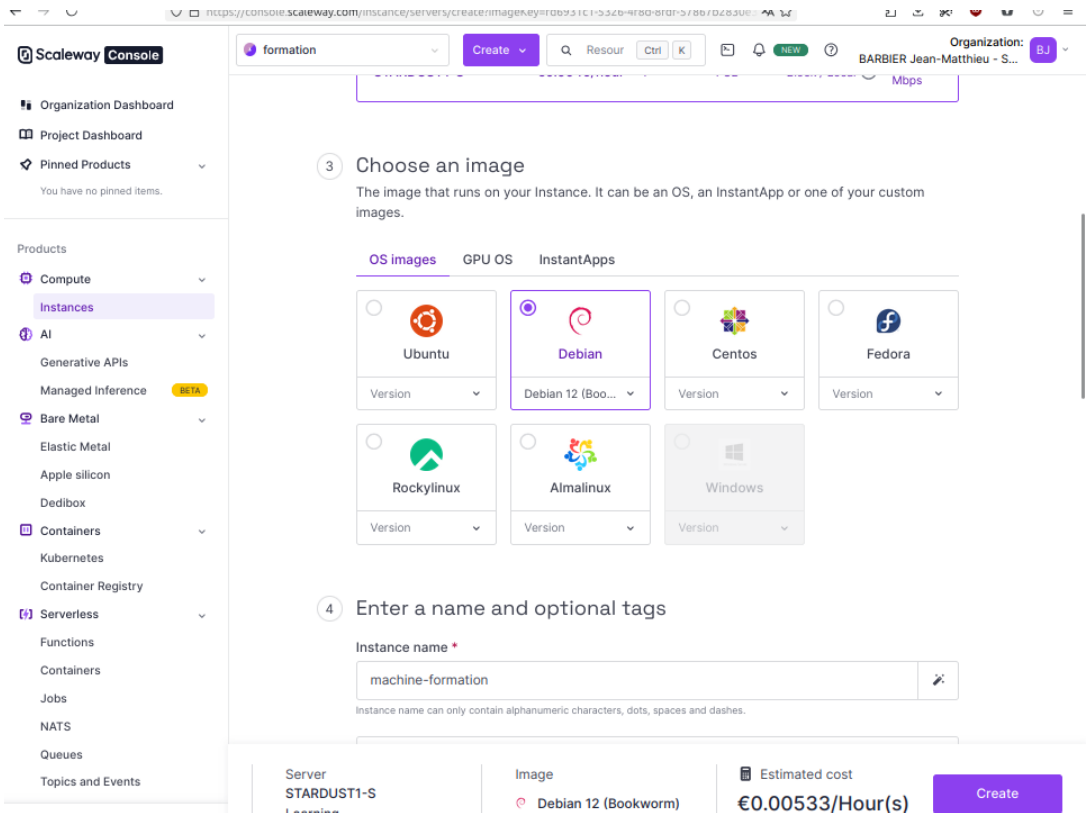
Aller dans le menu `Compute` puis `Instances` et cliquer sur le bouton `Create an instance`.



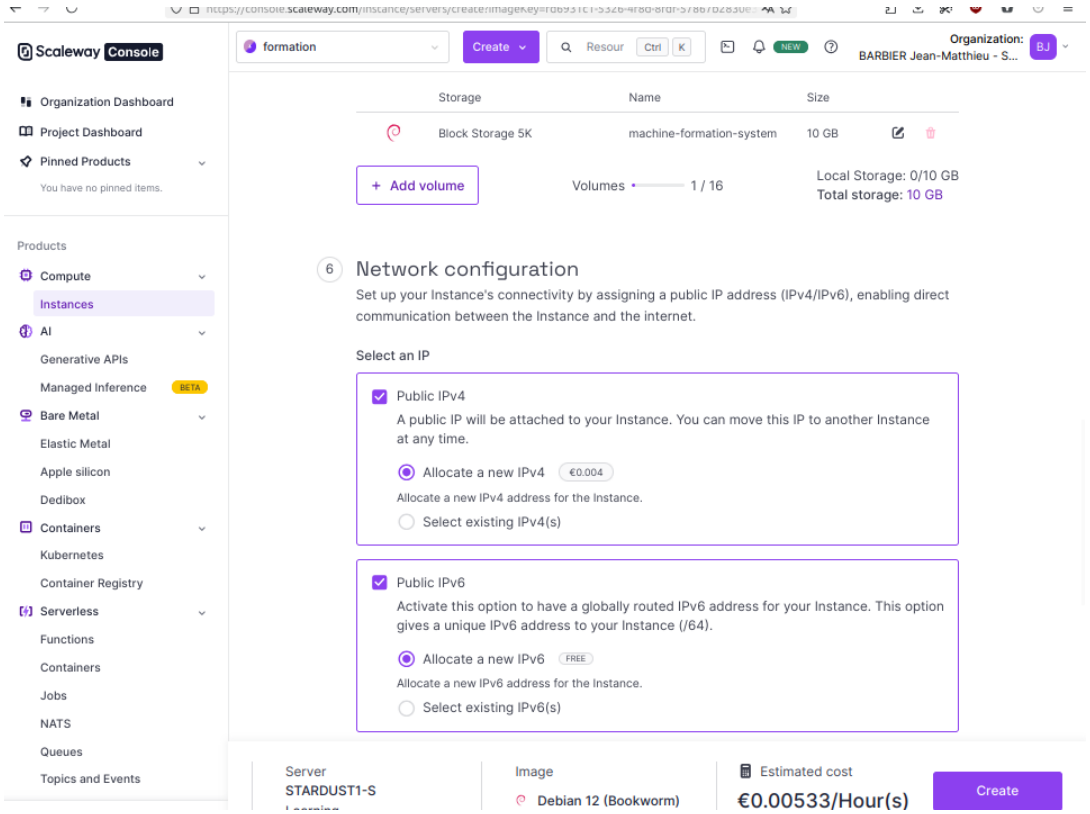
Sélectionner une instance `STARDUST` dans une zone où elle est disponible (par exemple `AMSTERDAM1`).



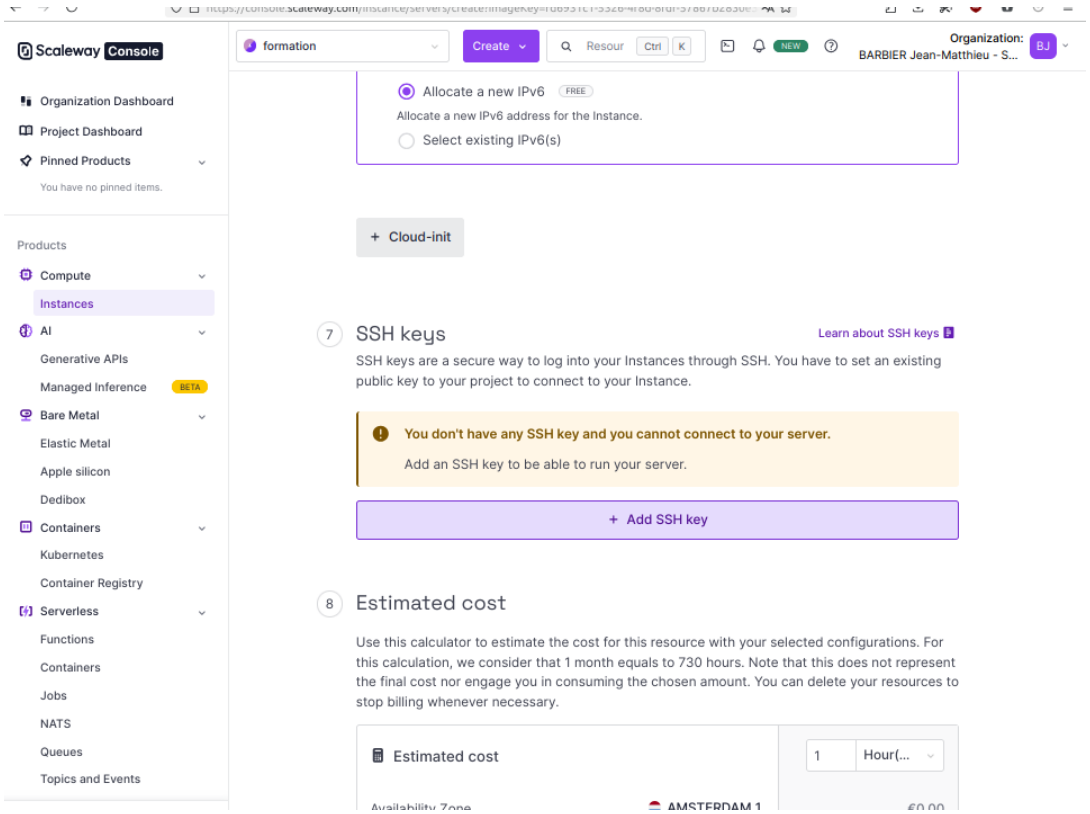
Sélectionner une distribution (`Ubuntu 24.04` ou `Debian 12`) et donnez un nom à votre instance.

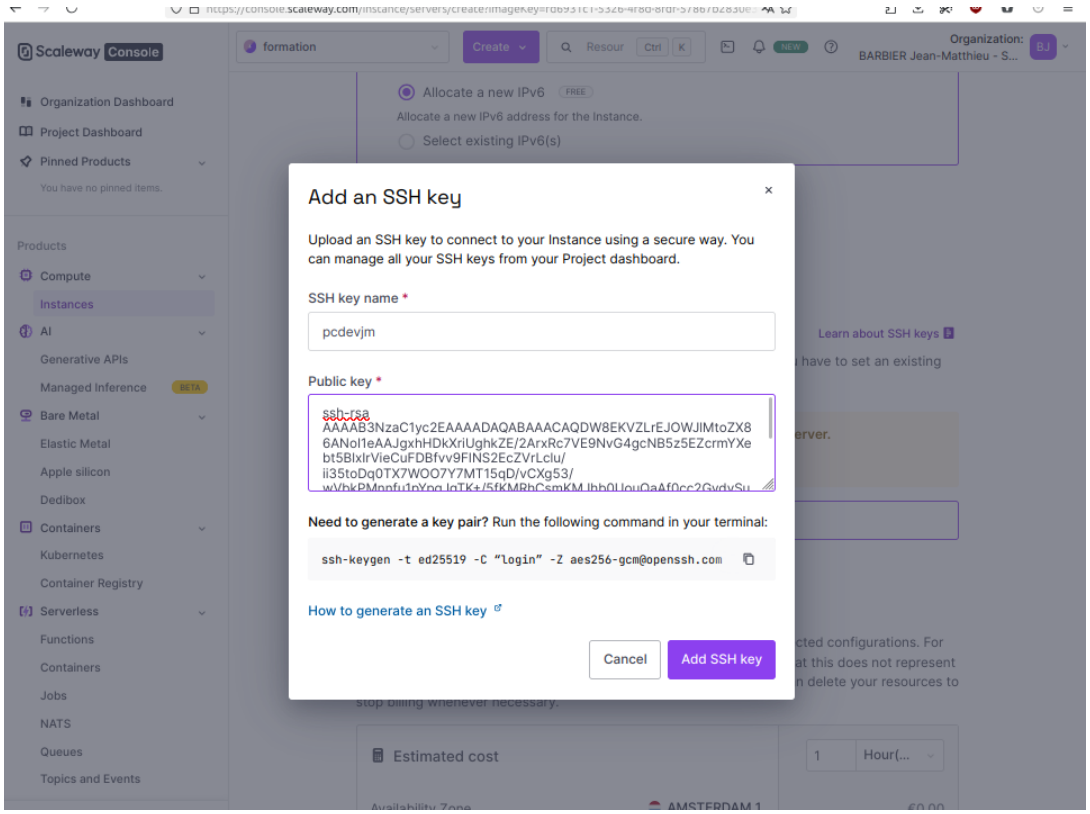


On garde le disque par défaut (10 Go) et on garde les champs par défaut pour le réseau[1].

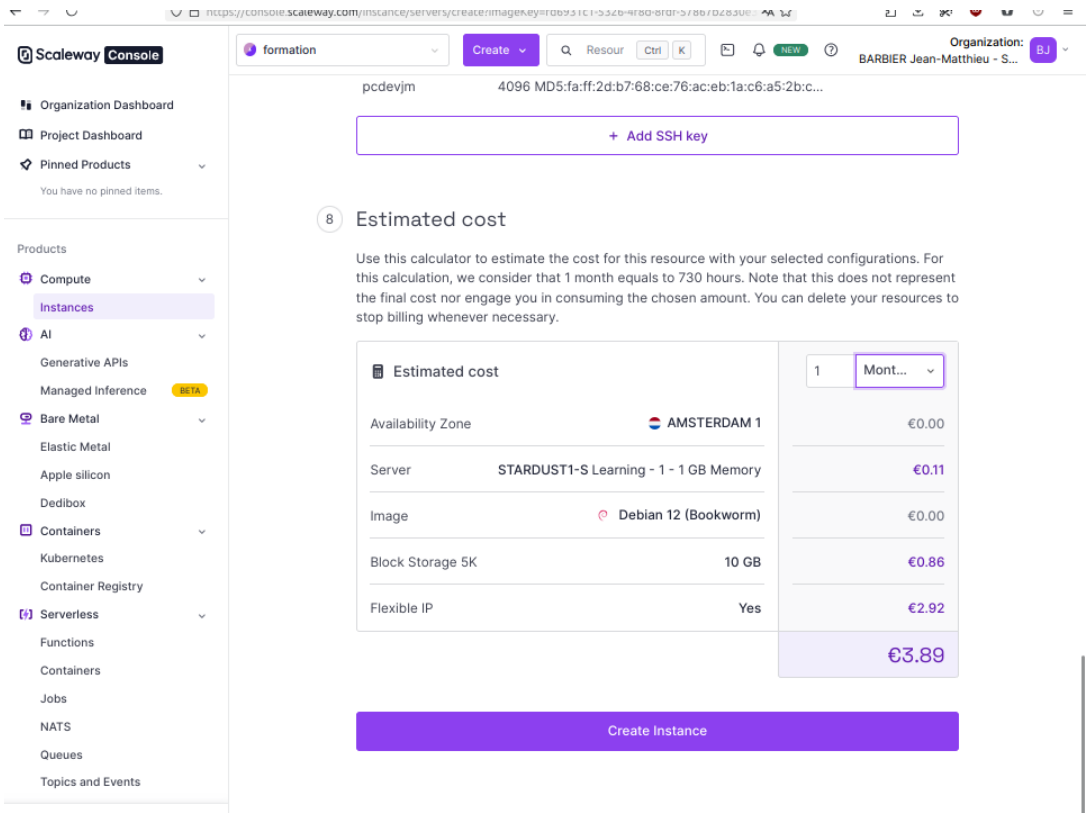


Ajouter votre clef SSH. Elle est dans le fichier `~/.ssh/id_ed25519.pub` (ou `id_rsa.pub` si vous avez utilisé une autre méthode pour créer la clef SSH). Il faut copier le contenu de ce fichier dans le champ `SSH Key` .

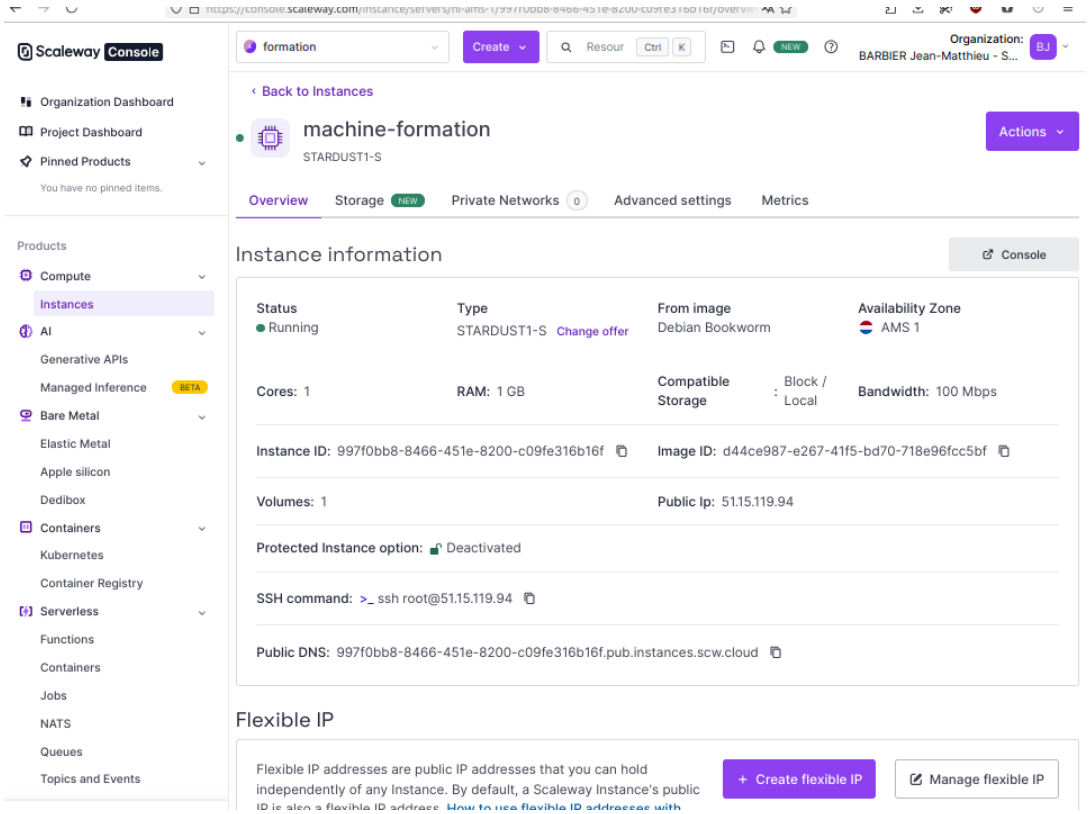




Puis créer l'instance.

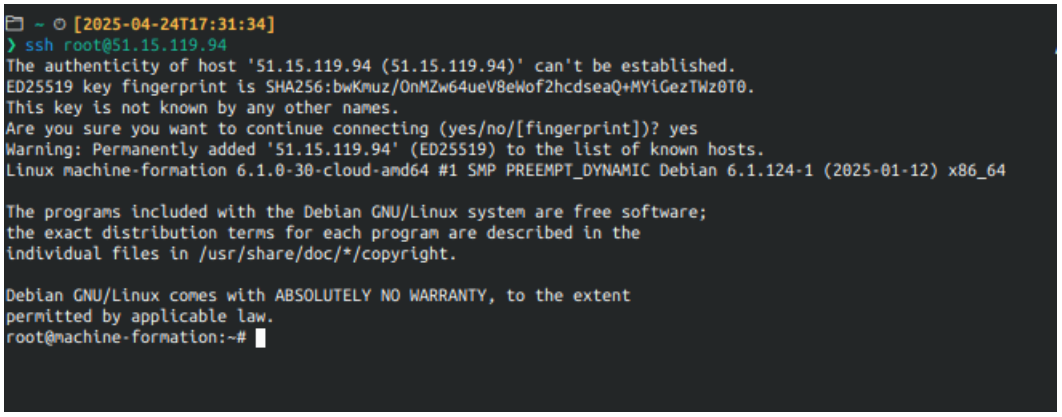


Au bout de quelques secondes, l'instance est créée.



Vous pouvez alors vous connecter dessus en SSH :

```
ssh root@<adresse_ip_publique>
```



[1] Si on veut **vraiment** faire des économies, on peut choisir de ne pas prendre d'IPv4 publique (c'est ce qui coûte le plus cher dans avec une instance stardust). Mais le reste est un peu plus compliqué, donc on va rester en IPv4 publique.

48. Installation

À partir du moment où vous êtes connecté à votre instance, vous pouvez administrer votre serveur en SSH.

On va commencer par a mise à jour de la machine, ça ne fait jamais de mal.

```
apt update
apt upgrade
```

48.1. Hello world non sécurisé

On va juste vérifier que tout fonctionne bien en créant un premier serveur, **non sécurisé**, qui tournera sur le port 80 (HTTP) avec l'utilisateur `root` (à ne **pas faire** en vrai...).

```
apt install python3 python3-bottle
mkdir test
cd test
nano hello.py # ou vim hello.py
```

Rappels :

- `nano` est un éditeur de texte en ligne de commande, très simple à utiliser. Il faut juste faire `CTRL+X` pour quitter et valider les modifications.
- `vim` est un éditeur de texte en ligne de commande, plus puissant mais plus compliqué à utiliser. Il faut faire `i` pour entrer en mode insertion, puis `ESC` pour sortir du mode insertion et taper `:wq` pour enregistrer et quitter.

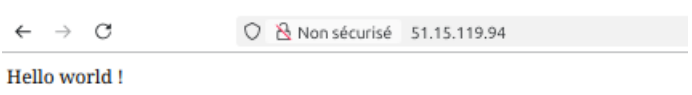
Dans `hello.py`, taper ou copier-coller le code suivant:

```
from bottle import route, run
@route('/')
def hello():
    return "Hello world !"
if __name__ == '__main__':
    run(host='0.0.0.0', port=80)
```

Puis on lance le serveur :

```
python3 hello.py
```

Vous pouvez alors tester en ouvrant un navigateur et en tapant l'adresse IP publique de votre instance. Vous devriez voir le message "Hello world !".



Une fois le test terminé, vous pouvez arrêter le serveur en appuyant sur `CTRL+C` dans le terminal où il tourne, et supprimer le dossier `test` :

```
cd ..
rm -rf test
```

48.2. Hello world sécurisé

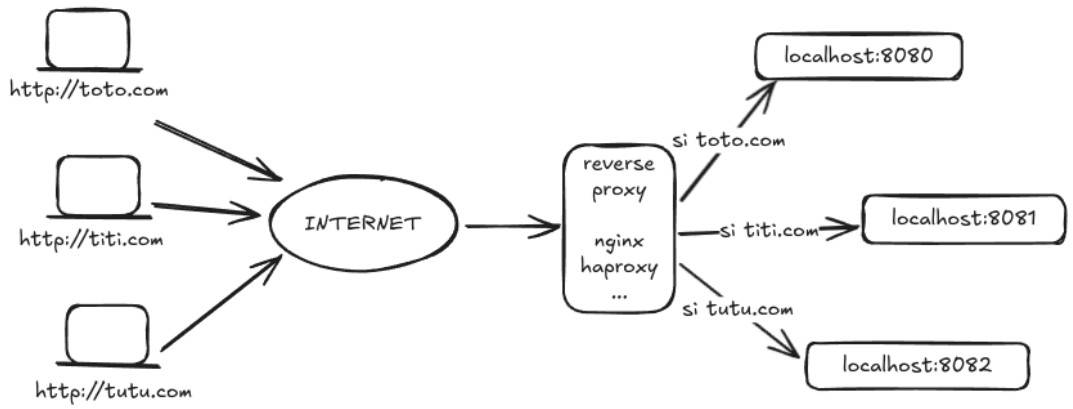
Un serveur HTTP **ne doit pas tourner en root**. Sauf que le protocole HTTP nécessite d'écouter sur le port 80, qui est un port privilégié.

Autre problème : nous avons une machine qui peut faire tourner plusieurs applications, mais une seule peut écouter sur le port 80.

Enfin, pour l'instant, notre application ne tourne pas en HTTPS, ce qui est un gros problème de sécurité.

Pour résoudre tous ces problèmes, on va utiliser un "vrai" serveur HTTP, qui va écouter sur le port 80, et rediriger les requêtes vers notre application, qui elle tournera sur un port non privilégié (au dessus de 1024, par exemple 8080), et sous un utilisateur non priviliégié (typiquement, un utilisateur différent pour chaque application : `appli1`, `appli2`, `eleve1`, ...).

Cette configuration est appelée "reverse proxy". Le serveur HTTP va simplement relayer les requètes vers notre application.



On va utiliser `nginx`, qui est un serveur HTTP très répandu et très performant. Il est aussi capable de faire du reverse proxy, c'est à dire de rediriger les requêtes vers une autre application qui tourne sur un autre port.

```
apt install nginx
```

On va aussi créer un utilisateur `webapp` qui va être l'utilisateur sous lequel va tourner notre application. Il n'aura pas de droits d'administration, donc il ne pourra pas faire de bêtises sur le serveur.

```
adduser webapp
```

Il faut maintenant créer un fichier de configuration pour nginx, qui va lui dire quoi faire. On va créer un fichier `/etc/nginx/sites-available/webapp` et y mettre la configuration suivante :

```
server {
    listen 80;
    server_name <adresse_ip_publique>;
    location / {
        proxy_pass http://localhost:8080;
        proxy_set_header Host $host;
    }
}
```

On crée ensuite un lien symbolique vers ce fichier dans le dossier `/etc/nginx/sites-enabled/`, qui est le dossier où nginx va chercher les fichiers de configuration actifs.

```
ln -s /etc/nginx/sites-available/webapp /etc/nginx/sites-enabled/webapp
```

et on redémarre nginx pour prendre en compte la nouvelle configuration :

```
systemctl restart nginx
```

Si vous allez voir dans votre navigateur à l'adresse IP publique de votre instance, vous devriez avoir une page d'erreur 502 Bad Gateway. C'est normal, car on n'a pas encore lancé notre application.

On va maintenant relancer notre application, mais cette fois-ci sous l'utilisateur `webapp` et sur le port 8080.

```
su webapp # On passe à l'utilisateur webap p
cd # On va dans le dossier de l'utilisateur webapp
mkdir test
cd test
nano hello.py
```

Puis on copie le code de tout à l'heure, mais en changeant le port et l'adresse IP d'écoute :

```
from bottle import route, run
@route('/')
def hello():
    return "Hello world !"
if __name__ == '__main__':
    run(host='localhost', port=8080)
```

Puis on lance le serveur :

```
python3 hello.py
```

Là, si vous allez sur l'adresse IP publique de votre instance, vous devriez voir le message "Hello world !".

Il ne manque plus que quelques "petites" choses :

- un site avec un vrai nom de domaine (et pas une adresse IP publique)
- un certificat SSL pour que le site soit en HTTPS
- servir l'application avec `gunicorn` (ou `uwsgi`) pour qu'elle soit plus performante et qu'elle puisse tourner en arrière-plan.
- un moyen de démarrer automatiquement l'application au démarrage du serveur (avec `systemd` par exemple)
- et savoir comment déployer une nouvelle version de notre application ou la mettre à jour notre application sans devoir tout taper à la main dans un terminal.

48.3. Envoyer des fichier sur le serveur

48.3.1. Accès SSH

L'utilisateur `webapp` a été crée avec un login/mdp, mais les accès SSH par login/mdp sont désactivés par défaut sur les serveurs (pour des raisons de sécurité, l'authentification par clé SSH est plus sécurisée).

Il faut donc, pour pouvoir se connecter en SSH

- soit ajouter une clef publique SSH à l'utilisateur `webapp` (ce qui est recommandé), dans le fichier `~/.ssh/authorized_keys` (il faut créer le dossier `.ssh` et le fichier `authorized_keys` s'ils n'existent pas déjà).
- soit modifier la configuration de SSH pour autoriser les connexions par login/mdp (ce qui n'est pas recommandé).

Pour ajouter une clef publique SSH à l'utilisateur `webapp`, il faut d'abord générer une clef SSH sur la machine depuis laquelle on va se connecter au serveur (vous pouvez utiliser la clef créée dans les étapes précédentes si c'est vous qui faites la manipulation, mais vous pouvez aussi donner un accès à vos élèves pour leur permettre d'envoyer leurs fichiers sur le serveur).

```
ssh-keygen
cat ~/.ssh/id_ed25519.pub # ou cat ~/.ssh/id_rsa.pub
```

Il faut ensuite copier le contenu de la clef publique (la ligne qui commence par `ssh-ed25519` ou `ssh-rsa`) et la coller dans le fichier `~/.ssh/authorized_keys` de l'utilisateur `webapp` sur le serveur.

```
su webapp
mkdir -p ~/.ssh
nano ~/.ssh/authorized_keys
```

Puis coller la clef publique dans le fichier `authorized_keys` et sauvegarder avec `CTRL+X`, `Y`, `ENTRER`. Il faut aussi donner les bons droits au dossier `.ssh` et au fichier `authorized_keys` :

```
chmod 700 ~/.ssh
chmod 600 ~/.ssh/authorized_keys
```

48.3.2. Envoi de fichiers

Il est possible d'envoyer des fichiers ou des dossiers sur le serveur avec `scp` (secure copy).

```
# Pour un fichier
scp <fichier local> <utilisateur>@<adresse_ip_publique>:<fichier distant>
# Pour un dossier (le dossier distant sera écrasé si il existe déjà)
scp -r <dossier local> <utilisateur>@<adresse_ip_publique>:<dossier distant>
```

Par exemple, pour envoyer le dossier `test` sur le serveur :

```
scp -r test webapp@<adresse_ip_publique>:/home/webapp/test
```

Si vous êtes sous linux ou osx, vous pouvez aussi utiliser `rsync`, qui est plus rapide et plus efficace pour synchroniser des fichiers ou des dossiers entre deux machines.

```
rsync -avz <dossier local> <utilisateur>@<adresse_ip_publique>:<dossier
distant>
```

Par exemple, pour envoyer le dossier `test` sur le serveur :

```
rsync -avz test webapp@<adresse_ip_publique>:/home/webapp/test
```

Vous pouvez aussi utiliser un client FTP/SFTP comme `FileZilla` ou `WinSCP` pour envoyer des fichiers sur le serveur.

48.4. Utiliser `git` pour déployer

Il est aussi possible d'utiliser `git` pour déployer une application sur un serveur : `[local] -> [gitlab] -> [serveur]`. CF formation `forge`.

En bref, il faut (en utilisant un `gitlab`, celui de la forge `forge.apps.education.fr`) :

- créer / activer son compte sur gitlab

Sur votre machine locale :

- associer votre clef SSH publique à votre compte gitlab (dans les paramètres de votre compte)
- créer un projet gitlab (ou utiliser un projet existant)
- utiliser les commandes fournies à la création du projet pour cloner le projet sur votre machine locale (ou ajouter git à un projet existant)
- enregistrer vos modifications locales (`commit`) et les envoyer (`push`) sur le serveur gitlab

Sur votre serveur :

- créer une clef SSH pour l'utilisateur qui est ou sera propriétaire du projet
- ajouter cette clef SSH publique à votre compte gitlab (dans les paramètres de votre compte)
- cloner le projet gitlab sur le serveur

A chaque modification, il suffira :

- sur votre machine locale : d'enregistrer vos modifications (`commit`) et de les envoyer (`push`) sur le serveur gitlab
- sur le serveur : de récupérer les modifications (`pull`) et de relancer l'application

49. Nom de domaine

Pour que votre site soit accessible depuis un nom de domaine, il faut **acheter** un nom de domaine. Pas de solution "gratuite" ici, il faut passer par un "registrar" (une société qui gère les noms de domaine). Il y a foultitude de possibilités, donc les choix sont forcément personnels. Si vous voulez éviter les comptes multiples, vous pouvez acheter le nom de domaine chez le même fournisseur que votre serveur (scaleway ou OVH. La partie "domaine" est globalement assez pourrie chez OVH, avis perso).

Sinon, dans les recommandations du chef :

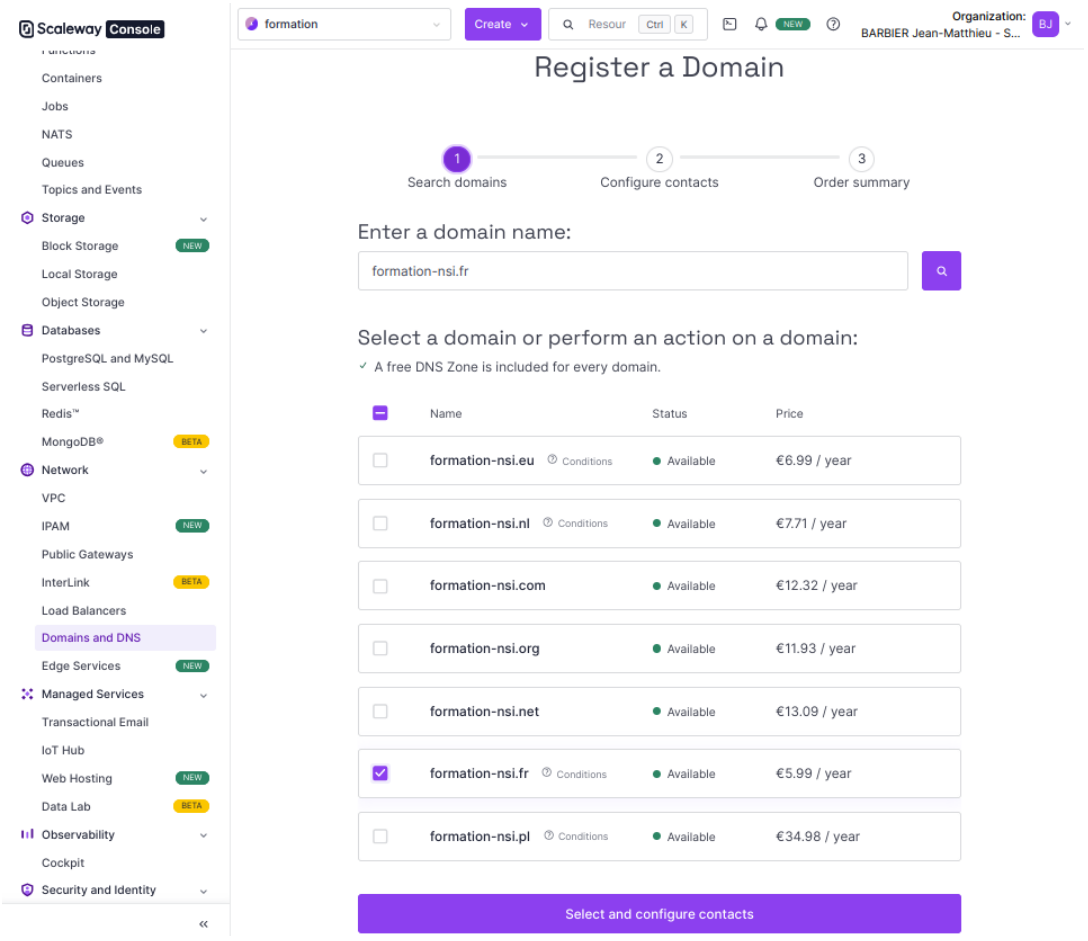
- https://porkbun.com/, pas cher, qui ne fait que du domaine et qui le fait bien, mais pas français.
- AWS Route 53 est un peu plus cher, et globalement plus compliqué à utiliser, mais puissant.

Gandi, ancienne pépite française, a complètement perdu son âme et est désormais à éviter.

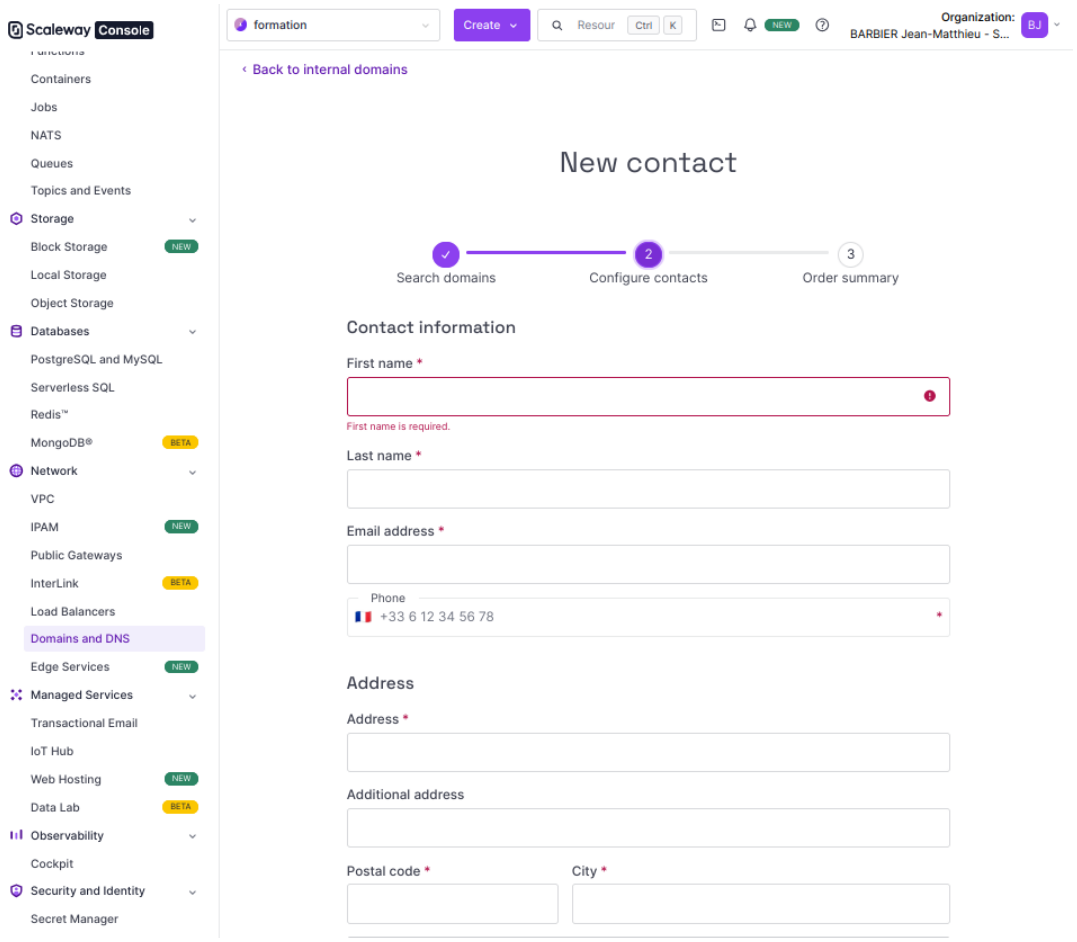
Évidemment ces avis sont personnels et n'ont pas vocation à être objectifs.

49.1. Exemple chez scaleway

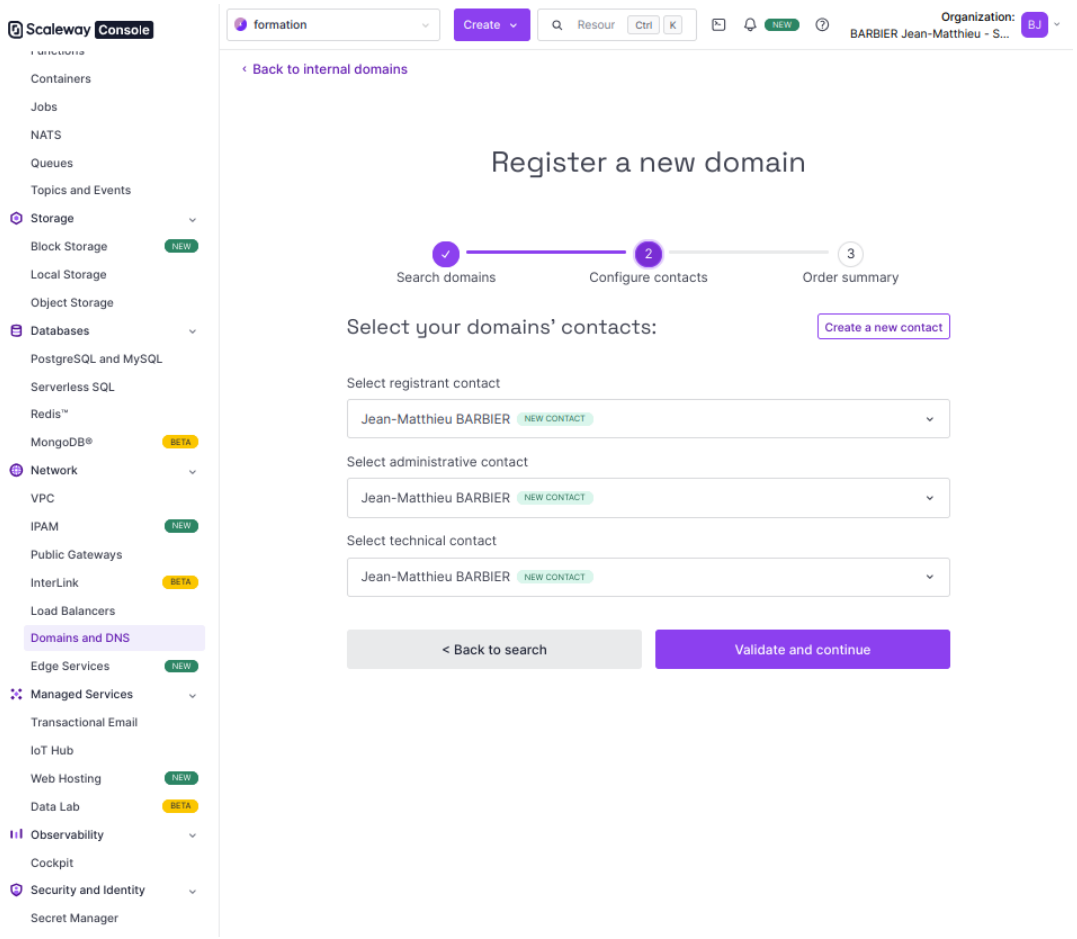
Pour acheter un nom de domaine chez scaleway, il faut aller dans le menu `Domains and DNS`. Cliquer sur "Search domain name" et entrer le nom de domaine que vous voulez acheter.



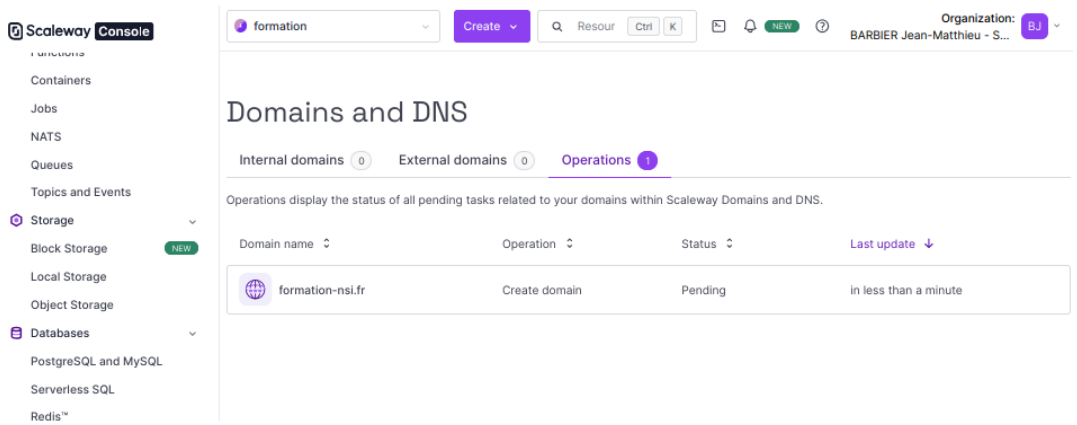
Il faut ensuite entrer les informations de contact.



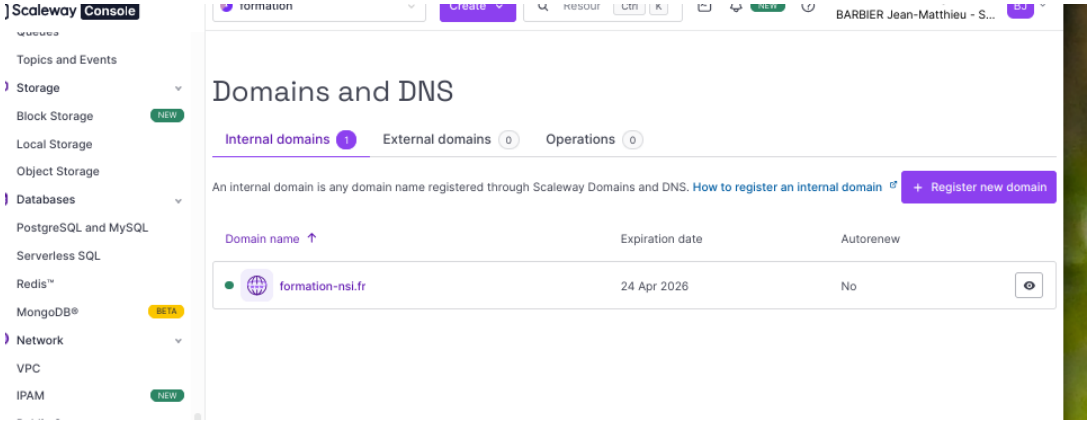
et associer ce contact pour le contact propriétaire, administratif et technique. Puis cliquer sur "Validate and continue".



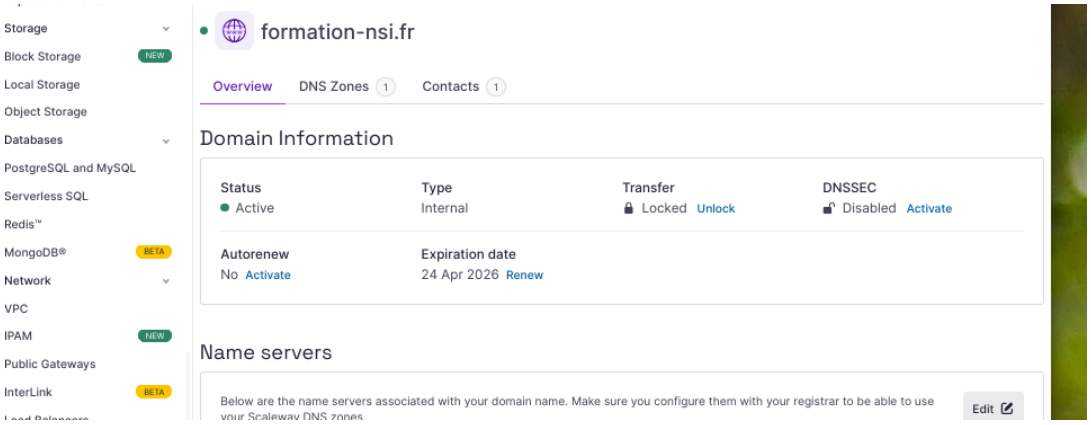
Une page de confirmation s'affiche, cliquer sur "Continuer". La page de liste des domaines apparaît, il faut attendre quelques minutes le temps que le paiement se fasse et que le domaine soit créé.

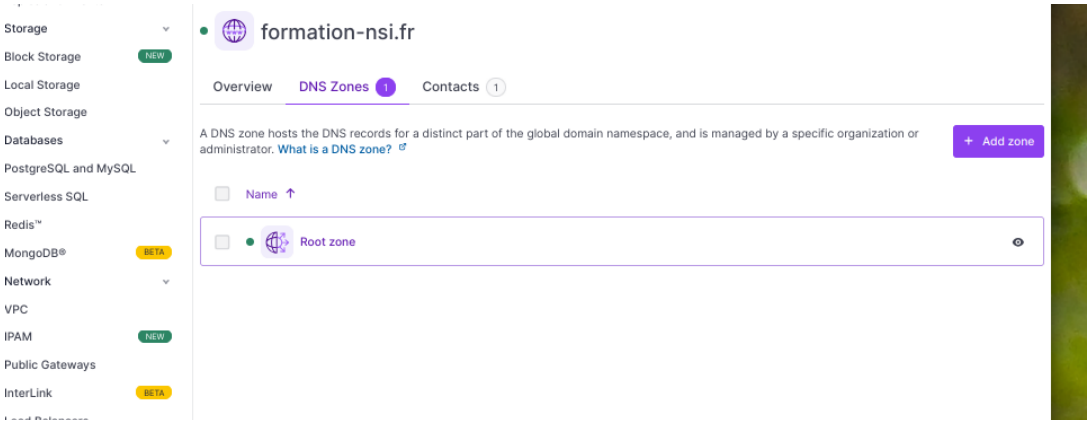


Une fois le domaine créé, il faut faire pointer le domaine vers l'adresse IP publique de votre instance. Pour cela, cliquez sur le nom de domaine dans la liste "Internal domains".

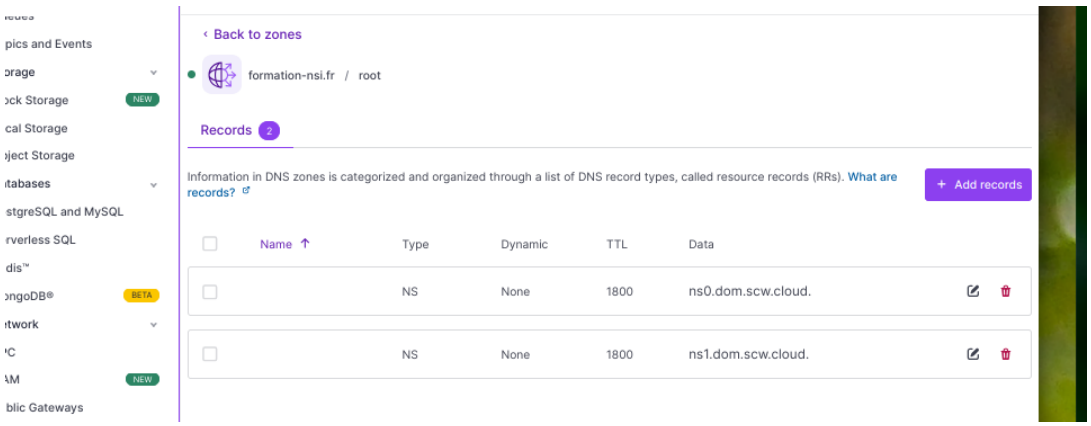


Cliquer sur l'onglet "DNS Zones" et sur "Root zone".

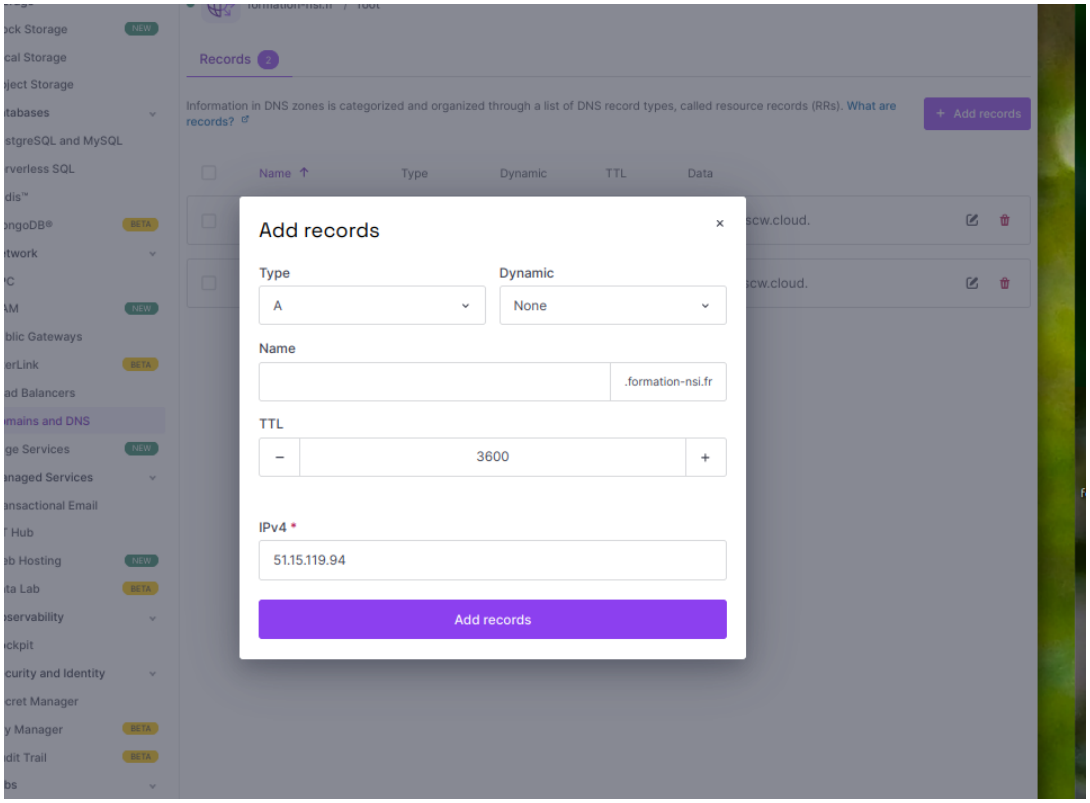




Il faut ensuite ajouter un enregistrement de type A qui pointe vers l'adresse IP publique de votre instance. Pour cela, cliquer sur "Add records".



Sélectionner le type A et entrer l'adresse IP publique de votre instance. Puis cliquer sur "Add records".



Attendre quelques minutes le temps que les DNS se mettent à jour. Vous pouvez vérifier que le domaine pointe bien vers l'adresse IP publique de votre instance en utilisant la commande `dig` (si elle n'est pas installée, vous pouvez l'installer avec `apt install dnsutils`).

```
dig @ns0.dom.scw.cloud <votre_nom_de_domaine>
```

Si une `ANSWER SECTION` apparaît avec l'adresse IP de votre instance, c'est que tout est bon. Sinon, attendez un peu et réessayez.

Vous pouvez créer autant de sous-domaines que vous voulez, il suffit de créer un enregistrement de type A pour chaque sous-domaine.

Chez les autres registrars, c'est à peu près les mêmes étapes, mais les interfaces sont différentes. Vous n'avez pas besoin d'acheter des packages d'hébergement ou autre, juste le nom de domaine.

49.2. Utiliser le nom de domaine pour votre application

Une fois que le nom de domaine et créé et qu'un enregistrement `A` pointe vers l'IP publique de votre instance, vous pouvez utiliser le nom de domaine pour accéder à votre application. Il faut juste modifier la configuration du reverse proxy pour qu'il envoie les requêtes destinées à votre nom de domaine (en-tête `Host`) vers votre application.

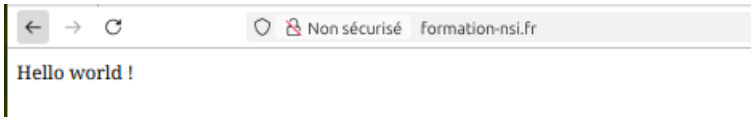
Pour cela, on va modifier le fichier `/etc/nginx/sites-available/webapp` créé précédemment et changer la ligne `server_name` pour mettre le nom de domaine désiré. Si vous souhaitez que le site soit accessible sur plusieurs domaines, il suffit de les séparer par un espace (`server_name mondomaine.com www.mondomaine.com`). Pour l'instant on va juste mettre le nom de domaine principal.

```
server {
    listen 80;
    server_name mondomaine.com;
    location / {
        proxy_pass http://localhost:8080;
        proxy_set_header Host $host;
    }
}
```

et redémarrer nginx :

```
systemctl restart nginx
```

À partir de maintenant, vous pouvez accéder à votre application avec le nom de domaine.



50. Certificat SSL

Il ne nous manque plus que le https. Pour cela, on va utiliser Let's Encrypt, qui est un service gratuit qui permet de générer des certificats SSL.

50.1. Installation de certbot

On va utiliser `certbot`, qui est un client Let's Encrypt. Il permet de générer des certificats SSL et de les renouveler automatiquement. Il sait aussi configurer nginx pour utiliser automatiquement le certificat.

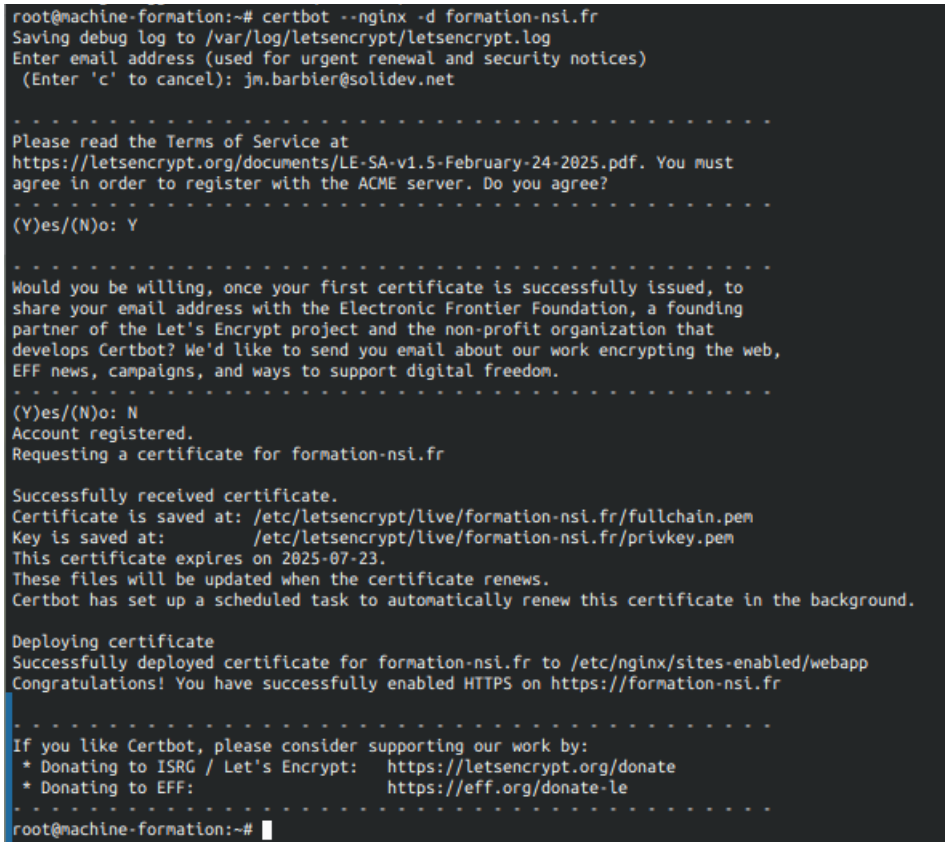
```
apt install certbot python3-certbot-nginx
```

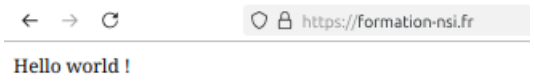
50.2. Génération du certificat

On va générer un certificat pour le nom de domaine `www.monsite.fr`. Il faut que le nom de domaine pointe vers l'adresse IP publique de la machine. Si ce n'est pas le cas, le certificat ne pourra pas être généré.

```
certbot --nginx -d www.monsite.fr
```

Il vous est demandé d'entrer un email de contact, d'accepter les conditions d'utilisation, de partager ou non votre adresse avec l'EFF et ses partenaire. Et c'est tout ! Le certificat est généré et installé automatiquement, nginx est configuré et redémarré, vous pouvez accéder à votre site en https.





50.3. Renouvellement du certificat

Le certificat est valable 3 mois. Il faut le renouveler tous les 3 mois. Pour cela, il suffit de lancer la commande suivante :

```
certbot renew
```

Cette commande va vérifier si le certificat est toujours valide et le renouveler si besoin. Il est possible de l'automatiser en ajoutant une tâche cron qui va lancer cette commande tous les jours. Pour cela, il faut éditer le fichier `/etc/crontab` et ajouter la ligne suivante :

```
0 0 * * * root certbot renew --quiet
```

Cette ligne va lancer la commande `certbot renew` tous les jours à minuit. Le `--quiet` permet de ne pas afficher de message si tout se passe bien. Si le certificat est renouvelé, nginx sera redémarré automatiquement.

51. Déploiement

Notre site n'est toujours pas complètement déployé :

- nous avons dû le lancer à la main avec `python hello.py`, et il ne tourne que tant que la fenêtre de terminal est ouverte et la commande est lancée.
- nous avons créé le code du serveur directement sur la machine, il serait beaucoup plus confortable de le faire sur notre machine de développement et de le transférer sur le serveur

51.1. gunicorn

Référence : https://bottlepy.org/docs/dev/deployment.html

Nous allons utiliser un serveur HTTP plus performant que le serveur de développement de bottle, qui est prévu pour le développement et pas pour la production. Nous allons utiliser `gunicorn`, qui est un serveur HTTP WSGI (Web Server Gateway Interface) pour Python. Il est capable de gérer plusieurs requêtes simultanément, ce qui le rend plus adapté pour un environnement de production.

Pour l'installer, en tant que `root` :

```
apt install gunicorn
```

Une petite modification de notre code `hello.py` est nécessaire pour que `gunicorn` puisse le lancer. Il faut ajouter une variable `app` qui contient l'application bottle, et qui permet d'exporter l'application pour un serveur externe.

```
from bottle import default_app, #...

app = default_app()
```

Pour lancer le serveur avec `gunicorn`, il faut se placer dans le répertoire où se trouve le fichier `hello.py` et lancer la commande suivante :

```
gunicorn hello:app
```

Cela va lancer le serveur sur le port 8000 par défaut. On peut changer ce port et beaucoup d'autres options avec des arguments de la ligne de commande.

- pour changer le port, on peut utiliser l'option `-b` (bind) : `-b localhost:8080` pour le port 8080 par exemple.
- pour changer le nombre de workers (processus) à lancer, on peut utiliser l'option `-w` (workers) : `-w 4` pour 4 workers par exemple.
- pour lancer le serveur en arrière-plan, on peut utiliser l'option `--daemon` : `--daemon` pour le lancer en arrière-plan.
- ... (`man gunicorn` pour les xxx autres options)

```
gunicorn -b localhost:8080 -w 4 hello:app
```

Il est possible de mettre toutes ces options dans un fichier de configuration `gunicorn.conf.py` :

```
bind = 'localhost:8080'
workers = 4
daemon = True
```

On peut ensuite lancer le serveur avec la commande :

```
gunicorn -c gunicorn.conf.py hello:app
```

51.2. Lancement automatique

Pour que le serveur se relance automatiquement après un redémarrage de la machine, on peut utiliser `systemd`, qui est un système d'initialisation et de gestion de services installé par défaut sur la plupart des distributions Linux.

Il faut créer un fichier de configuration pour le service dans le répertoire `/etc/systemd/system/`. En tant que `root`, on peut créer un fichier `hello.service` :

```
[Unit]
# Description du service
Description=Hello World service
# On attend que le réseau soit démarré avant de lancer le service
After=network.target
# On attend que le service nginx soit démarré avant de lancer le service (pas
obligatoire)
After=nginx.service
[Service]
# La commande à exécuter pour lancer le service (elle doit rester au premier
plan)
ExecStart=/usr/bin/gunicorn -b localhost:8080 -w 4 hello:app
# On indique que le service doit être lancé dans le répertoire où se trouve le
fichier hello.py
WorkingDirectory=/home/webapp/test
# On indique l'utilisateur sous lequel le service doit s'exécuter
User=webapp
# On indique le groupe sous lequel le service doit s'exécuter
Group=webapp
# On indique que le service doit être redémarré automatiquement dans tous les
cas
Restart=always
```

Il faut ensuite recharger la configuration de `systemd` :

```
systemctl daemon-reload
```

Puis on peut démarrer le service :

```
systemctl start hello
```

On peut vérifier que le service est bien démarré avec la commande :

```
systemctl status hello
```

On peut aussi activer le service pour qu'il se lance automatiquement au démarrage de la machine :

```
systemctl enable hello
```

52. Bricolage chez soi

Avec une vieille machine ou un Raspberry Pi, il est possible d'auto-héberger un ou plusieurs sites. La condition principale est que l'adresse IP de votre box soit une IP fixe, c'est normalement le cas chez Free, mais pas chez tous les FAI; si votre IP est dynamique et que vous ne pouvez pas la rendre fixe, vous pouvez utiliser un service de DNS dynamique, ce qui complique encore les choses.

Les étapes en gros sont :

1. configurer une machine comme serveur (voir partie installation)
2. réserver une adresse IP fixe pour cette machine (configuration du DHCP de la box - réserver une adresse IP pour l'adresse MAC de la machine)
3. rediriger le port 80 et 443 de la box vers la machine (configuration du NAT de la box)
4. faire pointer le nom de domaine vers l'adresse IP publique de la box (d'où l'intérêt d'avoir une IP fixe)