Appli Web NSI

HTML/CSS

Piliers du web

Mots clés

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

WWW

WWW = “World Wide Web” = “Toile mondiale”

- **système hypertexte public**
- inventé par Tim Berners-Lee en 1989 au CERN
- consultable ici

HTML

Histoire

HTML : HyperText Markup Language

- GML - SGML : documentation technique
- HTML3.2 - HTML4 - XHTML/CSS2.1
- HTML5/CSS2-3 living standard

Balises

Décrit la **sémantique** du contenu.

```
<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>
```

Figure 1: Antisèche courte

URL

Décomposition

URL : Uniform Resource Locator

`https://sub.dom.fr/path/to/pg.html?p1=v1#fra`

- protocole
- nom de domaine
- chemin
- nom de la ressource
- paramètres
- fragment

Écriture

URLs :

- complètes : **`https://exemple.com/site/page.html`**
- absolues : **`/site/page.html`**
- relatives : **`./page.html`** ou **`page.html`** ou **`../page.html`**

DOM

Document Object Model

DOM : Document Object Model = arborescence

- représentation de la page
- interface de programmation

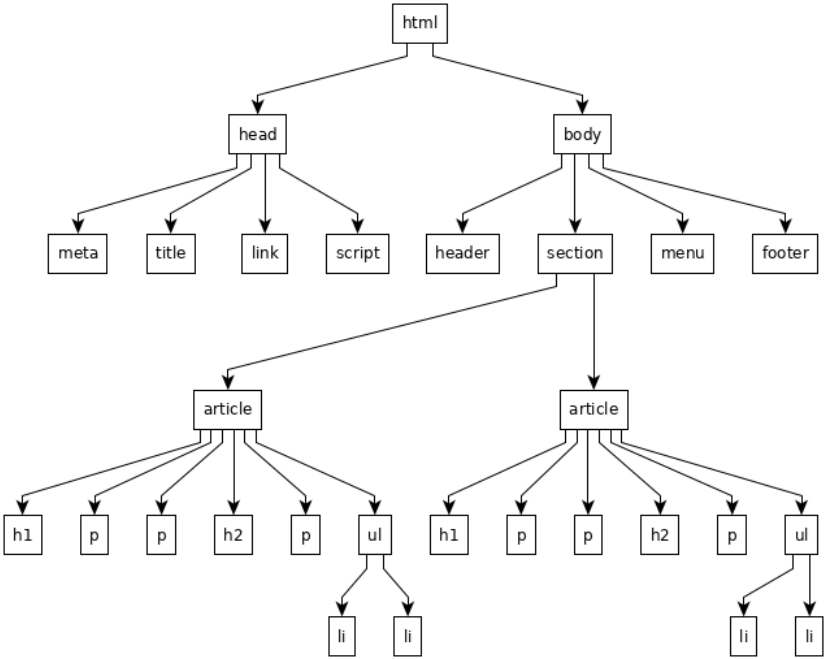


Figure 2: Arborescence d'une page HTML

Exploration

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

En python

```python
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")
    affiche_dom(soup.html) # parcours récursif
```

CSS

Concept

Séparation entre la structure et la présentation

- HTML : structure
- CSS : présentation

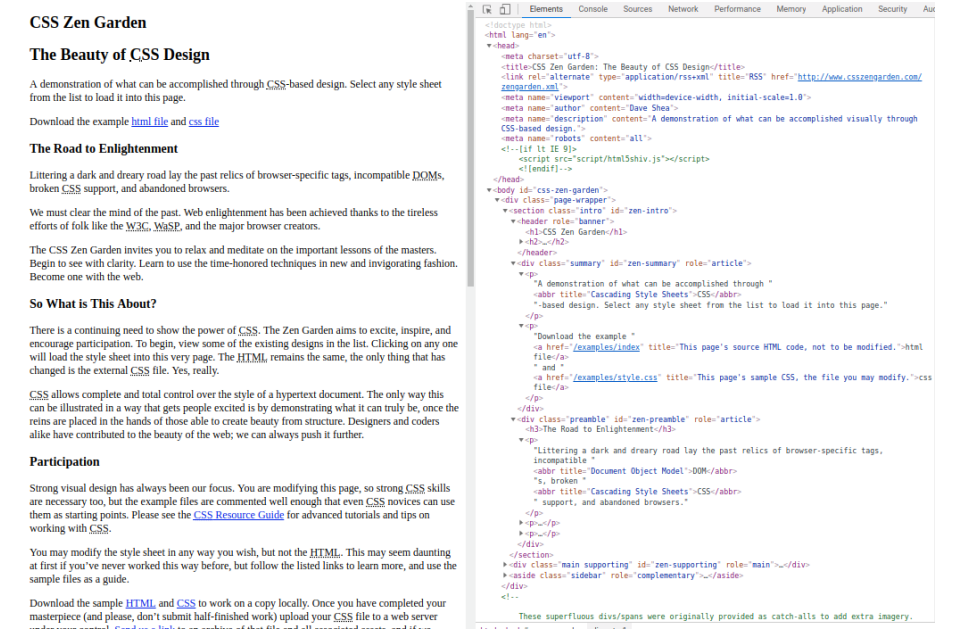


Figure 3: CSS Zen Garden - sans css

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

Figure 4: CSS Zen Garden - avec css

Syntaxe

CSS = règles de représentation

- **règle de représentation** : **sélecteur(s)** + **déclaration(s)**
- **déclaration** : **propriété** + **valeur**

Exemple :

```
article li {
  background-color: yellow;
}
```

Sélecteurs simples, hiérarchie

- **`li`** : tous les li du document
- **`article li`** : tous les li descendants d'un article
- **`article > ul`** : tous les ul enfants directs d'un article
- **`article > ul > li`** : tous les li enfants directs d'un ul descendant direct d'un article
- **`h1 + p`** : tous les p qui suivent immédiatement un h1
- **`h1 ~ p`** : tous les p qui suivent (immédiatement ou non) un h1 et qui ont le même parent

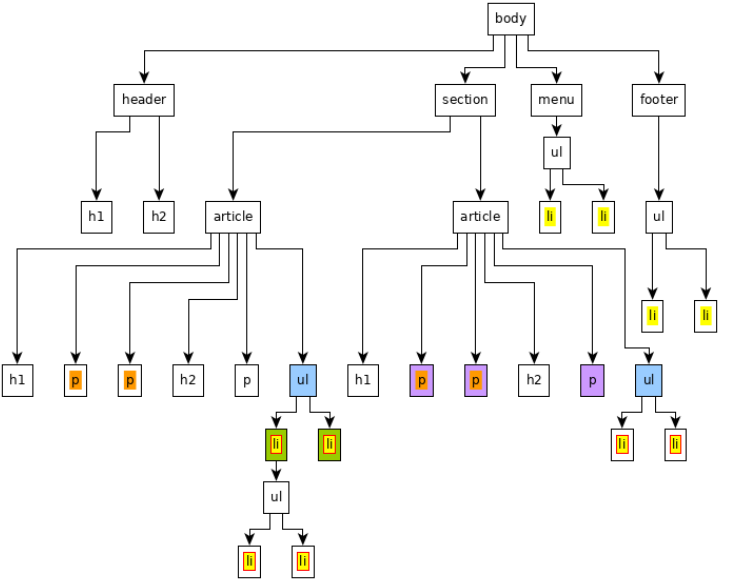


Figure 5: Sélecteur simples & hiérarchie

Sélecteurs de classe, id et attribut

- **`.rouge`** : balises avec la classe “rouge”
- **`h1.rouge`** : balises h1 avec la classe rouge
- **`#article-1`** : balise avec l’id article-1
- **`a[href="http://example.com"]`** : balises a avec l’attribut href égal à “http://example.com”

Variantes :

- **`a[href^="http://"]`** : balises a avec l’attribut href commençant par “http://”
- **`a[href$=".html"]`** : balises a avec l’attribut href finissant par “.html”
- **`a[href*="example"]`** : balises a avec l’attribut href contenant “example”
- ...

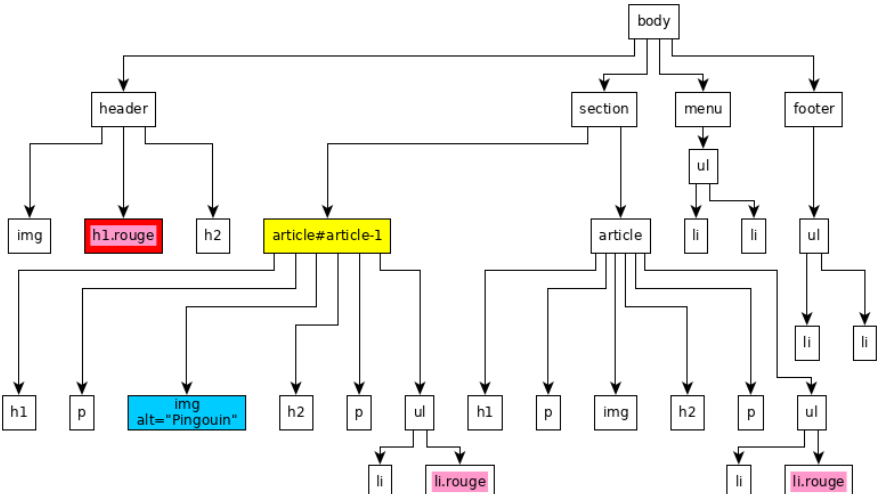


Figure 6: Sélecteurs de classe, d'id et d'attribut

Sélecteurs de pseudo-classes

- **`a:hover`** : balises a au survol
- **`a:visited`** : balises a déjà visitées
- **`a:before`** : avant le contenu de a
- **`a:after`** : après le contenu de a
- **`a:first-child`** : premier enfant de a
- **`a:last-child`** : dernier enfant de a
- **`a:nth-child(n)`** : n-ième enfant de a
- ...

Propriétés

- Liste “antisèche” de 5 pages, très complète
- Référence MDN CSS

Exemples :

- **`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’)
- ...

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
- ...

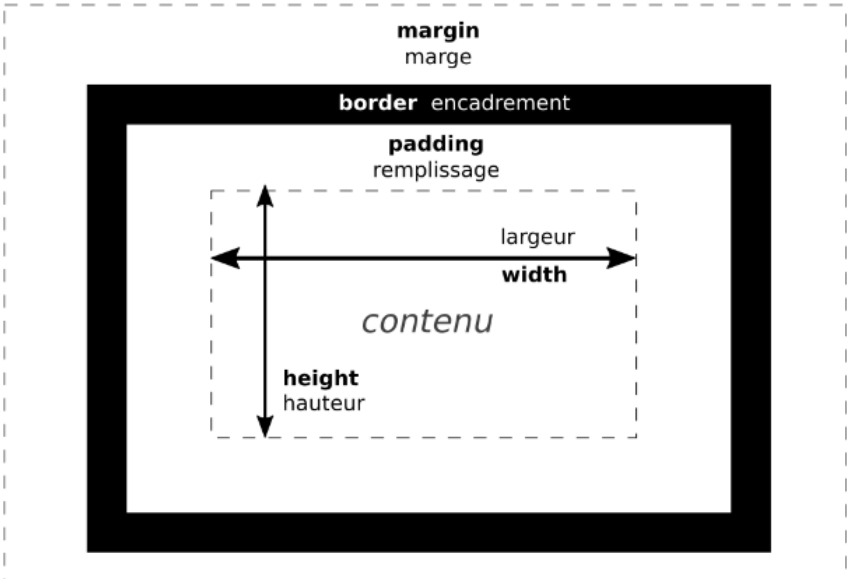
Variables CSS et valeurs calculées

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

Calculs :

- `calc()` : `width: calc(100% - 20px);`
- `min()` : `width: min(100%, 50px);`
- `max()` : `width: max(100%, 50px);`
- `clamp()` : `width: clamp(100px, 50%, 200px);`

Boîte


margin
marge
border encadrement
padding
remplissage
largeur
width
contenu
height
hauteur

Flux et 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`** : block, inline, flex, grid, none
- **`position`** : static, relative, absolute, fixed
- ...

Cascade

Si plusieurs propriétés sont applicables pour un élement, laquelle appliquer ?

1. importance (**`!important`**, par exemple **`color: red!important`**)
2. sélectivité (style > id > classe/attribut > élément)
3. ordre d'apparition dans la feuille de style (le plus bas l'emporte)

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`**, ...

Client/serveur

HTTP

Visualisation

HTTP : HyperText Transfer Protocol

Visualisation : outils de développement du navigateur > onglet “Network”

Histoire

- **`HTTP/0.9`** : 1990, simple, pas de headers, pas de méthode
- **`HTTP/1.0`** : 1996, ajout de headers, méthodes GET/POST/HEAD
- **`HTTP/1.1`** : 1997, ajout de headers, méthodes PUT/DELETE, keep-alive, caching
- **`HTTP/2`** : 2015, multiplexage, compression des headers, server push
- **`HTTP/3`** : 2022, ...

Format d'un message HTTP

- ligne de début:
  - requête : **`GET /index.html HTTP/1.1`**
  - réponse : **`HTTP/1.1 200 OK`**
- headers : paires clé/valeur (ex: **`Content-Type: text/html`**)
- ligne vide
- corps : données (ex: code HTML, image, etc.)

Requêtes et réponses

Requête

```
GET /philosophy/philosophy.html HTTP/1.1
Host: www.gnu.org
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:136.0) Gec
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*
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
```

Réponse

```
HTTP/1.1 200 OK
Date: Sun, 27 Apr 2025 05:04:38 GMT
Server: Apache
Content-Location: philosophy.fr.html
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
```

Modes de fonctionnement

- serveur de fichiers : renvoie le contenu de fichiers présents sur la machine
- serveur dynamique : génère le contenu au vol

Architecture

Types d'architecture

- 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)

REST

Concept

REST : REpresentational State Transfer / **API** : Application Programming Interface

- interface entre client et données
- 1 API, plusieurs clients
- simplicité et documentation

Concept

- représentation d'une ressource : **URL**
- méthodes HTTP = verbes d'action
  - **GET** = récupérer une ressource ou une collection de ressources
  - **POST** = créer une ressource
  - **PUT** = mettre à jour une ressource
  - **PATCH** = mettre à jour une ressource partiellement
  - **DELETE** = supprimer une ressource

Exemple

- **`GET /utilisateurs/123`** = récupérer l'utilisateur 123
- **`POST /utilisateurs`** = créer un nouvel utilisateur
- **`DELETE /utilisateurs/123`** = supprimer l'utilisateur 123
- **`PATCH /utilisateurs/123`** = mettre à jour l'utilisateur 123

Actions non REST

Pour les cas complexes ou spécifiques, mode RPC (Remote Procedure Call)

- **`POST /utilisateurs/123/messages`** = envoyer un message à l'utilisateur 123
- **`POST /paniers/123/paiement`** = procéder au paiement du panier 123
- **`POST /utilisateurs/123/notifications`** = envoyer une notification à l'utilisateur 123

Premier serveur

Préparation

Prélude

Prérequis :

- python
- IDE

Conseillé :

- **`vscode`** ou **`pycharm`**
- **`uv`** ou **`pip`**

IDE en ligne : https://xxxxx-code.formation.devel.space

Installation

Bottle : micro-framework web en python, simple et léger. Un seul fichier **`bottle.py`**.

- **`uv add https://github.com/bottlepy/bottle.git`**
- téléchargement fichier **`bottle.py`** sur **`https://bottlepy.org/`**

Hello world

```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)
```

Lancer le serveur

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

Site : **http://localhost:38083/hello**[1]

[1] ou https://xxxxx-app.formation.devel.space/hello si ide en ligne.

Logs

```
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" 4
```

Routes

Concept

Route = association entre une URL et une fonction python

Recette de cuisine permettant de répondre à la requête du client.

Définition

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

Méthodes et multiples routes

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

Raccourcis

```
from bottle import get, post, put, delete, patch #...
@get('/hello')
@post('/hello')
@put('/hello')
@delete('/hello')
@patch('/hello')
```

Routes dynamiques

Utilisation de paramètres dans l'URL (**jokers**) = paramètres capturés pour la fonction.

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

Filtres

- **<name:int>** : entier
- **<name:float>** : flottant
- **<name:path>** : chemin incluant les **/** (non “gourmand”)
- **<name:re:[a-z]+>** : expression régulière

En pratique

Dans notre serveur, rajouter :

- une route de bonjour personnalisé qui répond à une requète du type **`/hello/bob`** en renvoyant un message du type “Hello bob !”
- une route de calcul qui répond à une requète 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”.

Templates

Concept

Séparation entre le code et la présentation : logique dans le python, présentation dans le HTML.

Template = fichiers texte permettant de générer du HTML dynamique.

Utilisation

- Répertoire **`views`** contenant les fichiers **`html`**.
- Importer **`template`** et utiliser **`template('nomdufichier.html', param=value)`** pour générer le HTML.

Page d'accueil - template

```
<!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>
```

Code python - route /

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

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

Syntaxe - inline

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

Échappement automatique. Sans échappement : **{{! name }}**

Syntaxe - bloc python

- **%** en début de chaque ligne
- **<%** et **%>** pour un block
- indentation ignorée => mot clef **end** pour terminer le bloc

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

Syntaxe - boucles

```
% for i in range(10):
<p>{{ i }}</p>
% end
```

Syntaxe - full python

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

Fonctions de template

- **`include(sub_template, **context)`** : inclure un sous-template
- **`setdefault(key, value)`** : définir une valeur par défaut pour une variable
- **`defined(key)`** : vérifier si une variable est définie
- **`get(key, default)`** : obtenir la valeur d'une variable

Utilisation

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

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

En pratique

1. 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`**.
2. 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.
3. 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.

Serveur de fichiers

Préparation

Créer un répertoire **static**, avec un sous-répertoire **images**, y mettre des images.

```
mkdir -p 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
```

Fonction **`static_file`**

`static_file(filename, root)` :

- **`filename`** : nom du fichier à servir (relatif au répertoire **`root`**)
- **`root`** : répertoire racine à partir duquel servir les fichiers

Code

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

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

En pratique

1. Rajouter la route **`/static/<filename:path>`** dans le code du serveur pour servir les fichiers statiques.
2. 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.
3. Créer un fichier **`style.css`** dans le répertoire **`static`** et y déplacer les styles définis jusqu'à présent dans les templates, et lier ce fichier CSS dans les templates avec **`<link rel="stylesheet" href="/static/style.css">`**.

Formulaires - HTML

Balise form

```
<form action="" method="">
  <!-- champs de saisie -->
</form>
```

- **action** : URL de destination
- **method** : méthode HTTP (GET ou POST)

method : GET ou POST ?

- **GET** :
  - paramètres dans l'URL
  - visible dans l'historique du navigateur
  - requêtes idempotentes (sans effet de bord)
- **POST** :
  - paramètres dans le corps de la requête
  - pas visible dans l'historique du navigateur
  - requêtes non idempotentes
  - données volumineuses ou sensibles

Champs de saisie

- **`input`** : champ de saisie (texte, mot de passe, email, etc.)
- **`textarea`** : zone de texte
- **`select`** : liste déroulante
- **`button`** : bouton

Propriétés globales :

- **`name`** : nom du champ (envoyé au serveur)
- **`disabled`** : champ désactivé
- **`readonly`** : champ en lecture seule

Input

- **type** : type de champ (text, password, email, number, date, file, etc.)
- **value** : valeur initiale
- **placeholder** : texte d'aide si champ vide
- **required** : champ obligatoire
- ...

```
<label for="username">Nom d'utilisateur</label>
<input
  type="text"
  id="username"
  name="username"
  required
  minlength="3"
  maxlength="20"
  placeholder="Entrez votre nom d'utilisateur"
/>
```

Texarea

- **rows** : nombre de lignes
- **cols** : nombre de colonnes

```
<label for="message">Message</label>
<textarea
  id="message"
  name="message"
  rows="4"
  cols="50"
  placeholder="Entrez votre message"
></textarea>
```

Select

- **option** : élément de la liste
- **selected** : option sélectionnée par défaut
- **multiple** : sélection multiple
- **size** : nombre d'options visibles (si multiple)

```
<label for="langue">Langue</label>
<select id="langue" name="langue">
  <option value="fr" selected>Français</option>
  <option value="en">Anglais</option>
  <option value="es">Espagnol</option>
</option>
```

Envoi du formulaire

- clic sur input de type submit
- clic sur bouton
- appui sur entrée dans un champ de saisie (= appui sur 1er bouton)

Exemple :

codepen.io/jmsolidev/pen/NWqLORY : envoi du formulaire vers **`echo.solidev.net`**

Formulaires - serveur

Request

Dictionnaire (**`MultiDict`**) contenant les paramètres de la requête.

- **`request.query`** : paramètres de la requête (GET)
- **`request.GET`** : alias de **`request.query`**
- **`request.forms`** : paramètres du formulaire (POST)
- **`request.files`** : fichiers envoyés (POST)
- **`request.POST`** : alias de **`request.forms`** + **`request.files`**

Récupération de valeurs

```
GET /formulaire?choix=1&choix=2&user=toto
```

```
request.query.get('choix') # 1
request.query.getall('choix') # ['1', '2']
request.query.get('user', None) # 'toto' ***
request.query.user # 'toto'
```

Routage

- une seule route pour **GET** et **POST**
  - plus simple pour gérer les erreurs
- une route pour chaque méthode

Exemple

```
@route('/formulaire', method=['GET', 'POST'])
def formulaire():
    if request.method == 'POST':
        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)
        # ... traitement des données (envoi de mail, etc..)
        return template("merci_message.html")
    else:
        return template("formulaire.html")
```

En pratique

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]")

API - javascript

Réponse json

Bottle : **`return <dictionnaire>`** = réponse format **`json`**, utilisable en dynamique (javascript).

Côté serveur

```python
@route('/api/pwgen')
def api_pwgen():
    import random
    import string
    chars = [random.choice(
        string.ascii_letters + string.digits
    ) for _ in range(10)]
    return {'password': ''.join(chars)}
```

Côté client

```
<script>
    const fetchPassword = async () => {
        const response = await fetch('/api/pwgen');
        const data = await response.json();
        document
          .getElementById('password')
          .innerText = data.password;
    }
</script>
<h1>Générateur de mot de passe</h1>
<button onclick="fetchPassword()">
  Générer un mot de passe
</button>
<code id="password"></code`>
```

Sécurité

Principe de base

IL NE FAUT JAMAIS FAIRE CONFIANCE AU CLIENT !

- validation des données **côté serveur**
- jamais de **`eval`** ou **`exec`** sur des données non vérifiées
- ...

Attaques

Principalement :

- XSS (Cross Site Scripting)
- shell injection (avec **`os.system`** ou **`exec`**)
- injection SQL

Mais aussi :

- CSRF (Cross Site Request Forgery)
- Clickjacking
- ...

Persistance / authentification

Cookies

Concept

Cookie : petit fichier texte stocké sur le client

- créé par le serveur (en-tête **`Set-Cookie`**)
- envoyé par le client (en-tête **`Cookie`**) à chaque requête

Permet de stocker *sur le client* des informations retransmises au serveur.

Utilisation

- **gestion de session**
- tracking utilisateur
- personnalisation

Pour du stockage pur, **`localStorage`**, **`sessionStorage`** et **`IndexedDb`** sont préférables.

Création de 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 !"
```

Options de cookie

- durée de vie : **`max_age`** (en secondes) ou **`expires`** (date d'expiration)
- domaine : **`domain`**
- chemin : **`path`**
- sécurité : **`secure`** (HTTPS uniquement)
- HttpOnly : **`httponly`** (non accessible par javascript)
- SameSite : **`samesite`** (protection CSRF)

En pratique

Écrire une vue **`/compteur`** qui affiche le nombre de fois que la page a été visitiée. 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.

Sessions

Concept

- stockage de données côté serveur associées à un *identifiant de session*
- transmission de l'identifiant de session au client via un cookie
- cookie de session envoyé à chaque requête => récupération des données de session

Sans persistance (1)

```
SESSIONS = {}

def get_session() -> Union[dict, None]:
    session_id = request.get_cookie("session_id")
    if session_id in SESSIONS:
        return SESSIONS[session_id]
    return None

def create_session() -> Tuple[str, dict]:
    session_id = create_session_id()
    SESSIONS[session_id] = {}
    return session_id, SESSIONS[session_id]
```

Sans persistance (2)

```python
@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']}"
```

Avec persistance

- stockage de chaque session dans un fichier **json**

```
def save_session(session_id: str, session: dict) -> None:
    session_file = path.join(SESSIONS_PATH, session_id + ".jso
    with open(session_file, "w") as f:
        json.dump(session, f)
```

- chargement de la session depuis le fichier

```
def get_or_create_session() -> Tuple[str, dict, bool]:
    session_id = request.get_cookie("session_id", None)
    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
```

- création de la session

```
def create_session() -> Tuple[str, dict]:
    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()
```

En pratique

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`**.

Figure 8: Résultat attendu

Authentification

Procédure

- envoi des identifiants (login + mot de passe) au serveur
- vérification des identifiants (login + correspondance hash mot de passe)
- création d'une session et enregistrement de l'authentification
- envoi d'un cookie de session au client

Hachage bien fait

ON NE STOCKE JAMAIS DE MOT DE PASSE EN CLAIR !

- algorithmes de hashage (fonction injective)
- attaque brute force = fonctions de hashage lentes
- rainbow tables = salage

Utilisé : **`bcrypt`** ou **`argon2`**

Hachage basique

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

Vue de login

```
USERS = {
    "alice": {
        "nom": "Alice",
        "prenom": "Dupont",
        # évidement aucun intérêt ici d'avoir le hash si le mdp
        # c'est juste pour l'exemple :)
        "password_hash": hash_password("password")
    },
    #...
}
```

```python
@route("/login", method=["GET", "POST"])
def login():
    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."
```

```
# Si la méthode est GET, on affiche le formulaire de login
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>
'''
```

En pratique

1. 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.
2. 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

Bases de données

SQLite

Connexion et utilisation en python

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

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

Curseur et requêtes

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS ingredients (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL
);
""")
cursor.execute("""
INSERT INTO ingredients (name)
VALUES ('Mozzarella'),('Tomate'),('Basilic'),
       ('Jambon'),('Olive'),('Oignon'),('Oeuf');
""")
conn.commit()
with conn:
    conn.execute('''INSERT INTO ingredients (name)
    VALUES ('Merguez');''')
```

Récupération de données, paramètres

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

Paramétrisation

```
cursor.execute('''INSERT INTO ingredients (name)
  VALUES (?)', ('Viande hachée',))''')
cursor.execute('''SELECT id,name FROM ingredients
  WHERE name LIKE ?''', ('%o%',))
```

Dans un serveur

Connexion

La connexion est **globale** et partagée entre toutes les requêtes.

```
db_conn = sqlite3.connect("data.db")
db_conn.execute('PRAGMA foreign_keys = ON')
# ... initialisation
```

```
# 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
```

Utilisation

```python
@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
```

En pratique

Modifiez le précédent 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)

Injection SQL

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

Recherche automatique et exploitation de la faille