Réalisation du site de compte rendu de recherche

I. Présentation du projet

Ce projet vise à produire un **site web statique professionnel**, élégant et entièrement automatisé à partir d'une arborescence de documents (PDF, images, textes sources).

Le site est généré localement dans le dossier `/html`, puis déployable immédiatement sur GitHub Pages, Netlify, Vercel, ou tout serveur statique.

Objectifs

- Génération 100 % automatique de la structure du site à partir des dossiers et fichiers sources.
- Navigation fluide, cohérente et dynamique avec fil d'Ariane.
- Table des matières dynamique intelligente, pliable, avec filtrage personnalisé.
- Contrôle fin et granulaire via fichiers `structure.py` pour chaque dossier (visibilité, ordre, titres personnalisés, etc.).
- Design épuré, responsive, accessible, avec support de mises en forme Markdown-like.
- Zéro dépendance externe lourde (Python standard + http-server pour le local).
- Gestion des entêtes, pieds de page globaux et locaux.
- Personnalisation complète via fichiers de configuration (`options.py`, `config.py`, `style.css`).
- Log détaillé et nettoyage automatique des caches.

Le système est conçu pour être extensible : ajout de nouveaux dossiers/fichiers régénère automatiquement le site sans effort.

Le temps investi a permis de créer un système robuste, avec gestion des erreurs, logging, et personnalisation avancée, garantissant une maintenabilité à long terme et une scalabilité pour des projets similaires.

II. Structure du dossier de génération

La structure globale du projet est conçue pour une clarté maximale, avec séparation claire entre sources, scripts et output.

```
hebreu4.0/
├── documents/              ← Sources principales : dossiers, sous-dossiers, fichiers
(PDF, images, etc.).
│   ├── Annexes/            ← Exemple de dossier avec fichiers secondaires.
│   ├── Dossier Principal/  ← Dossier racine des contenus principaux, avec sous-dossiers
imbriqués.
│   │   ├── Dossier secondaire1/
│   │   ├── Dossier secondaire2/
│   │   │   └── SousDossier secondaire21/
│   ├── Références/         ← Dossier pour bibliographie ou références.
│   ├── nppBackup/          ← Sauvegardes automatiques (ignorées par le générateur).
│   └── TDM/                ← Dossier spécial pour la table des matières (géré
automatiquement, non affiché).
├── prog/                   ← Scripts de génération et librairies.
│   ├── lib1/               ← Modules partagés.
│   │   ├── config.py       ← Configuration globale (titres, classes CSS, etc.).
│   │   ├── options.py      ← Chemins des dossiers sources/output.
│   │   └── style.css       ← Fichier CSS central pour tout le site.
│   ├── genere_site.py      ← Script principal : génère la structure HTML et copie les
fichiers.
│   ├── cree_table_des_matieres.py ← Script dédié à la génération de la TDM dynamique.
│   ├── lancer.cmd          ← Batch Windows pour lancer la génération + serveur local.
│   └── methode.py          ← Ce script : génère le rapport méthodologique.
├── html/                   ← Output : site statique généré, prêt à déployer.
│   ├── index.html          ← Page d'accueil.
│   ├── style.css           ← Copie du CSS.
│   ├── TDM/                ← Dossier de la table des matières.
│   │   └── index.html      ← Page TDM dynamique.
│   ├── annexes/            ← Dossiers générés avec leurs index.html et fichiers copiés.
│   └── ...                 ← Tous les autres dossiers et fichiers.
└── MÉTHODOLOGIE_SITE.md    ← Ce rapport (généré par methode.py).
```

Chaque dossier dans `documents/` peut contenir un `structure.py` personnalisé, un `entete.html` et/ou `pied.html` local. Le temps passé à structurer ainsi assure une organisation scalable, facile à maintenir pour des extensions futures.

III. Contenu des fichiers de génération

Dossier `documents/`

- Contient tous les fichiers sources : PDF, images, textes.
- Arborescence libre : tout dossier/sous-dossier est reproduit dans `html/`.
- Fichiers spéciaux par dossier :
  - `structure.py` : Configuration locale (visibilité, ordre, titres).
  - `entete.html` : Contenu ajouté en haut de la page index.html locale.
  - `pied.html` : Contenu ajouté en bas de la page index.html locale.
  - `entete_general.html` et `pied_general.html` à la racine : Appliqués à tout le site.

Dossier `prog/lib1/`

- `options.py` : Définit les chemins (DOSSIER_DOCUMENTS, DOSSIER_HTML).
- `config.py` : Paramètres globaux (titre_site, classe_dossier, ignorer, etc.).

- `style.css` : Définit le style global du site (voir détail ci-dessous).

Dossier `prog/`

- Scripts Python principaux pour la génération.
- Batch `lancer.cmd` pour exécution facile.

Dossier `html/`

- Généré automatiquement : ne pas modifier manuellement (régénéré à chaque lancement).

Le temps investi permet un contenu exhaustif, avec documentation interne (docstrings) et commentaires pour une compréhension immédiate.

IV. Les programmes

1. `genere_site.py` (version 15.7)

- **Description détaillée**
  - Suppression et recréation du dossier `html/` pour une génération propre.
  - Création immédiate du dossier `TDM` pour éviter les erreurs de chemin.
  - Nettoyage automatique des `__pycache__`.
  - Génération/mise à jour récursive des `structure.py` dans chaque dossier source.
  - Parcours récursif des dossiers : copie des fichiers, génération des `index.html`.
  - Support des mises en forme Markdown-like dans les noms affichés.
  - Gestion des paramètres globaux/locaux via `structure.py`.
  - Log détaillé (console/fichier).
  - Temps investi : optimisation pour une exécution rapide même sur de grandes arborescences.

2. `cree_table_des_matieres.py` (version 6.4)

- **Description détaillée**
  - Parcours récursif de `html/` pour construire l'arborescence.
  - Génération d'une TDM pliable avec `<details><summary>`.
  - Filtrage intelligent : ignore les éléments si `"affiché_TDM": False` dans le `structure.py` du parent.
  - Liens URL propres, noms normalisés.
  - Style CSS embarqué pour un arbre visuel élégant (lignes, marqueurs +/−).
  - Ne montre jamais TDM dans la TDM.
  - Temps investi : algorithme efficace, gestion d'erreurs robuste.

3. `lancer.cmd` (version 2.1)

- **Description détaillée**
  - Activation de l'environnement virtuel si nécessaire.
  - Exécution séquentielle : `genere_site.py` puis `cree_table_des_matieres.py`.
  - Lancement d'un serveur local avec `npx http-server` (port 3500, CORS, no-cache).
  - Gestion des erreurs avec pause.
  - Temps investi : simplification pour une utilisation one-click.

4. `methode.py` (version 3.0)

- **Description détaillée**
  - Génère ce rapport exhaustif au format Markdown.
  - Utilise caractère spécial § pour formatage parfait.
  - Temps investi : rendu complet, sans omission, pour une documentation de qualité.

V. Le dossier `html`

Le dossier `html/` est le résultat final de la génération : un site statique autonome.

- **Contenu exhaustif**
  - `index.html` à la racine et dans chaque dossier.
  - `style.css` central.
  - Tous les fichiers sources copiés (PDF, images) dans leurs dossiers respectifs.
  - Dossier `TDM/` avec son `index.html` (table des matières).
- **Caractéristiques**
  - URLs propres et normalisées (minuscules, sans espaces).
  - Responsive et accessible (via CSS).
  - Prêt pour déploiement : pas de dépendances dynamiques.
  - Temps investi : optimisation pour chargement rapide, compatibilité browsers.

VI. Structure des fichiers `index.html`

Chaque `index.html` est généré dynamiquement et suit une structure modulaire pour une personnalisation maximale.

Structure détaillée

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>[Titre du dossier via structure.py ou config]</title>
    <link rel="stylesheet" href="/style.css">
</head>
<body>
    [haut_page global (config.py) : bandeau, logo, etc., si haut_page=True]
    [entete_general.html (racine documents/) : entête site-wide, si entete_general=True]
    [navigation dynamique : fil d'Ariane + lien Sommaire, si navigation=True]
    [entete.html local : contenu spécifique au dossier, si entete=True]
    <table class="dossiers"><tbody>
        [Liste des dossiers/fichiers triés par position, avec liens, styles, et mises en
forme MD]
    </tbody></table>
    [pied.html local : pied spécifique, si pied=True]
    [pied_general.html : pied site-wide, si pied_general=True]
    [bas_page global (config.py) : footer avec date, si bas_page=True]
</body>
</html>
```

Autorisations globales et locales

- **Globales** : `haut_page`, `bas_page` dans `config.py` ; `entete_general.html`, `pied_general.html` dans `documents/`.
- **Locales** : `entete.html`, `pied.html` dans chaque dossier.
- Contrôles dans `structure.py` : True/False pour chaque.

Navigation

- Fil d'Ariane : liens vers parents (Accueil → Dossier → Sous-dossier).
- Lien fixe "Sommaire" vers `/TDM/index.html`.
- Paramètres : `navigation=True/False` dans `structure.py` (par dossier).
- Style : classes `.navigation`, `.monbouton` pour personnalisation CSS.

Liste des éléments

- Tri par `position`.
- Visibilité via `affiché_index`.
- Mises en forme dans `nom_affiché` (MD-like).
- Classes CSS personnalisables via `config.py`.

Temps investi

Structure flexible, facile à déboguer, pour une évolution rapide.

VII. Le fichier `style.css` et l'action de ses items

`prog/lib1/style.css` définit le style global, copié dans `html/`.

Contenu exhaustif

```
body { font-family: Arial, sans-serif; margin: 20px; background-color: #f9f9f9; }  ← Base : police, marges, fond clair pour lisibilité.

.navigation { display: flex; justify-content: space-between; margin-bottom: 20px; background: #eee; padding: 10px; }  ← Barre : flexbox pour alignement gauche/droite, fond gris pour séparation visuelle.

.gauche, .droite { display: flex; align-items: center; }  ← Groupes : alignement vertical des boutons.

.monbouton { margin-right: 10px; padding: 5px 10px; background: #007bff; color: white; text-decoration: none; border-radius: 5px; }  ← Boutons : bleu, arrondis, sans soulignement pour modernité.
.monbouton:hover { background: #0056b3; }  ← Hover : couleur foncée pour feedback interactif.

.dossiers { width: 100%; border-collapse: collapse; }  ← Table : pleine largeur, sans bordures internes.
.dossiers td { padding: 10px; border-bottom: 1px solid #ddd; }  ← Cellules : espacement, lignes séparatrices pour clarté.

.dossier { font-weight: bold; color: #333; }  ← Dossiers : gras, sombre pour hiérarchie.
.fichier { color: #666; }  ← Fichiers : gris pour distinction.

a { text-decoration: none; }  ← Liens : sans soulignement par défaut.
a:hover { text-decoration: underline; }  ← Hover : soulignement pour interactivité.

.tdm-content { max-width: 800px; margin: auto; }  ← TDM : centrée, largeur limitée pour lecture.
```

Actions des items

- Améliore l'UX : responsive (flex), feedback (hover), hiérarchie (gras/gris).
- Personnalisable : modifie pour thèmes (dark mode, etc.).
- Temps investi : CSS minimal mais puissant, optimisé pour performance.

VIII. Structure du fichier table des matières (`/TDM/index.html`)

Page dédiée, générée par `cree_table_des_matieres.py`.

Structure détaillée

```
[haut_page]
[navigation]
<h1>Sommaire du site</h1>
<div class="tdm-content">
  <ul class="tree">
    <li><details><summary><a href="...">[Dossier]</a></summary>[Sous-arbo]</details>
</li>
    <li><a href="...">[Fichier]</a></li>
  </ul>
</div>
[bas_page]
<style>[CSS arbre]</style>
```

Filtrage

Seulement `affiché_TDM=True` (parent).

Temps investi

Algorithme récursif efficace, style immersif.

IX. Les procédures

Liste exhaustive avec description détaillée :

Procédure	Description détaillée
`log(msg)`	Log console/fichier avec flush. Utilisé pour debug, tracing complet.
`normaliser_nom(nom)`	Convertit en URL-safe. Ex : "Dossier Principal" → "dossier_principal".
`appliquer_style(texte)`	Parse MD-like : **gras**, *italic*, ~~barré~~, [rouge] [/rouge], {grand} {/grand}. Regex avancées pour couleurs.
`_creer_structure_py(dossier)`	Génère `structure.py` récursif : liste, positions, visibilité. Gestion effacés.
`_lire_structure(dossier)`	Parse `structure.py` : retourne dict filtré. Gestion exceptions.
`_lire_fichier(chemin)`	Lit entete/pied. Retourne vide si absent.
`_generer_navigation(chemin_relatif)`	Fil d'Ariane + Sommaire. Flexible pour chemins longs.
`_generer_page(src, dst, chemin)`	Assemble HTML : globals/locaux, table triée avec styles.
`_traiter_dossier(src, dst, chemin)`	Récursif : mkdir, copie, génération pages.
`doit_afficher_dans_tdm(entry)`	Vérifie `affiché_TDM` dans parent. Robustesse erreurs.
`construire_arbo(dossier, prefixe)`	HTML récursif TDM : filtrage, liens normalisés.

Procédures courtes (<25 lignes), modulaires, pour maintenance facile.

X. Syntaxe dans `nom_affiché` de structure.py

Syntaxe	Résultat
`**texte**`	**gras**
`_texte_`	*italique*
`**_texte_**`	***gras italique***
`~~texte~~`	~~barré~~
`[rouge]texte[/rouge]`	texte en rouge

Syntaxe	Résultat
`[bleu]texte[/bleu]`	texte en bleu
`[couleur:#ff3366]texte[/couleur]`	couleur personnalisée
`{grand}texte{/grand}`	texte très gros
`{taille:2.5em}texte{/taille}`	taille personnalisée

XI. Contenu des fichiers générés et modification possible

Fichiers générés

- `index.html` : Pages navigables, modifiables via sources.
- `structure.py` : Auto-générés, mais éditables (ajoute MD dans `nom_affiché`).
- `TDM/index.html` : TDM filtrée.

Modifications

- Sources : ajout → régénération.
- `nom_affiché` : MD-like pour rich text.
- Visibilité : `affiché_index/TDM` = False masque.

XII. Fichiers de configuration

`options.py`

```
# options.py – Version 1.1

version = ("options.py", "1.1")
print(f"[Import] {version[0]} - Version {version[1]} chargé")

# Chemins relatifs ou absolus selon ton environnement
DOSSIER_RACINE = r"C:\SiteGITHUB\Hebreu4.0"
DOSSIER_DOCUMENTS = f"{DOSSIER_RACINE}\\documents"
DOSSIER_HTML = f"{DOSSIER_RACINE}\\html"

# Base path pour les liens (GitHub Pages vs local)
# "" pour test local (serveur sur /html)
# "/hebreu4.0" pour GitHub Pages (repo à la racine du user site)
BASE_PATH = "/Hebreu4.0/html"  # ← Modifier ici : "" pour local, "/Hebreu4.0" pour
GitHub

# fin du "options.py" version "1.1"
```

`config.py`

```
# Début de "config.py" version "2.2"

CONFIG = {
    "version": "2.2",
    "titre_site": "Hébreu Biblique v4.0",
    "fichier_index": "index.html",
    "classe_dossier": "dossier-item",
    "classe_fichier": "fichier-pdf",
    "lien_souligné_index": False,    # False = pas de soulignement dans les pages index
    "lien_souligné_TDM": False,      # False = pas de soulignement dans la TDM
    "haut_page": [
        "<div><!-- debut haut_page -->",
#        "<div class=\"monTitre\">Hébreu biblique</div>",
#        "<div class=\"monSousTitre\">Mes dossiers partagés</div>",
        "<!-- fin haut_page --></div>",
    ],
    "bas_page": [
        "<div><!-- debut bas_page -->",
#        "<div>bas de la page</div>",
        '<footer><a href="mailto:fraboulanger@orange.fr" class="btn-mail">Pour me
joindre</a></footer>',
        "<!-- fin bas_page --></div>",
    ],
    "navigation": {
        "sous_dossiers_position": "gauche",
        "afficher_sommaire": True
    },
    "affichage": {
        "afficher_icones": True,
        "table_align": "centre",
        "table_largeur_max": "80%"
    },
    "couleurs": {
        "dossiers": "#0066cc",
        "pdf": "#c0392b",
        "images": "#27ae60"
    },
    "ignorer": [
        "nppBackup", ".git", ".gitignore", "Thumbs.db",
        "entete_general.html", "pied_general.html",
        "entete.html", "pied.html", "structure.json",
        "index.html", "style.css","__pycache__"
    ],
    "extensions_acceptees": ["pdf"],
    "dossier_tdm": "TDM",
    "ajout_affichage": ["□ ", "", "□ ", ""],
    "logging": ["console", "generation.log"]  # "console", "fichier.log", ou les deux
}

# Fin de "config.py" version "2.2"
```

`structure.py` exemple

```
STRUCTURE = {
    "entete_general": True,
    "pied_general": True,
    "entete": True,
    "pied": True,
    "navigation": True,
    "haut_page": True,
    "bas_page": True,
    "ajout_affichage": True,
    "dossiers": [
        {"nom_document": "Secondaire1",
         "nom_html": "Secondaire1",
         "nom_affiché": "**Secondaire** [rouge]1[/rouge]",
         "nom_TDM": "Secondaire 1",
         "affiché_index": True,
         "affiché_TDM": True,
         "position": 2
        }
    ],
    "fichiers": [
        {"nom_document": "Introduction sujet.pdf",
         "nom_html": "introduction_sujet.pdf",
         "nom_affiché": "_Introduction_ {grand}PDF{/grand}",
         "nom_TDM": "Introduction",
         "affiché_index": True,
         "ajout_affichage": True,
         "affiché_index": True,
         "affiché_TDM": True,
         "position": 1
         }
    ]
}
```

XIII. genere_site.py

```
# genere_site.py – Version 20.0

version = ("genere_site.py", "20.0")

# Importation des librairies
import os
import json
import shutil
import unicodedata
import re
import psutil
import tempfile
from pathlib import Path
from datetime import datetime
from typing import List, Dict, Any
from bs4 import BeautifulSoup  # Pour prettify des index.html

# Conversion .doc/.docx → .pdf via Microsoft Word (Windows uniquement)
try:
    from win32com.client import Dispatch
    word_app = Dispatch("Word.Application")
    word_app.Visible = False
except ImportError:
```

```
    word_app = None

from lib1.options import DOSSIER_DOCUMENTS, DOSSIER_HTML, BASE_PATH
from lib1.config import CONFIG

print(f"[Version] {version[0]} — {version[1]}")

# Acquisition des constantes
def lire(variable: dict, element: str, defaut: Any) -> Any:
    """Lit une valeur dans un dictionnaire, retourne la valeur par défaut sinon."""
    return variable.get(element, defaut)

STYLE_CSS_SRC = Path(__file__).parent / "lib1" / "style.css"
IGNORER = set(lire(CONFIG, "ignorer", [])) | {"__pycache__", ".pyc", "structure.py",
r"~\$"}
FICHIERS_ENTETE_PIED = {"entete.html", "entete_general.html", "pied.html",
"pied_general.html"}
EXTENSIONS_ACCEPTEES = {".html", ".htm", ".pdf", ".txt"}
DOSSIER_TDM = lire(CONFIG, "dossier_tdm", "TDM")
AJOUT = lire(CONFIG, "ajout_affichage", ["", "", "", ""])
voir_structure = lire(CONFIG, "voir_structure", False)

log_file = Path("generation.log")
log_file.write_text(f"--- DÉBUT GÉNÉRATION — {datetime.now().strftime('%d/%m/%Y
%H:%M:%S')} ---\n", encoding="utf-8")

# Fonctions utilitaires
def log(msg: str) -> None:
    """Écrit un message dans la console et dans generation.log."""
    print(msg)
    with open(log_file, "a", encoding="utf-8") as f:
        f.write(msg + "\n")

def normaliser_nom(nom: str) -> str:
    """Normalise un nom pour URL (minuscules, underscore, sans accent)."""
    nom = unicodedata.normalize('NFD', nom)
    nom = ''.join(c for c in nom if unicodedata.category(c) != 'Mn')
    return nom.replace(" ", "_").lower()

def appliquer_style(texte: str) -> str:
    """Applique les balises Markdown-like au texte pour coloration, gras, etc."""
    texte = re.sub(r'\*\*(.*?)\*\*', r'<strong>\1</strong>', texte)
    texte = re.sub(r'_(.*?)_', r'<em>\1</em>', texte)
    texte = re.sub(r'~~(.*?)~~', r'<del>\1</del>', texte)

    couleurs = {"rouge": "red", "bleu": "blue", "vert": "green", "jaune": "gold",
                "violet": "purple", "orange": "orange", "gris": "gray", "noir": "black"}
    for nom, code in couleurs.items():
        texte = texte.replace(f"[{nom}]", f'<span style="color:{code}">')
        texte = texte.replace(f"[/{nom}]", "</span>")

    texte = re.sub(r'\[couleur:(#[0-9a-fA-F]{6}|rgba?\([^)]+\))\]', lambda m: f'<span
style="color:{m.group(1)}">', texte)
    texte = texte.replace("[/couleur]", "</span>")
    texte = texte.replace("{grand}", '<span style="font-size:1.8em">').replace("
{/grand}", "</span>")
    texte = texte.replace("{petit}", '<span style="font-size:0.8em">').replace("
{/petit}", "</span>")
    texte = re.sub(r'\{taille:([^}]+)\}', lambda m: f'<span style="font-size:
{m.group(1)}">', texte)
    texte = texte.replace("{/taille}", "</span>")
    return texte
```

```
def deb_html(titre: str) -> str:
    """Générateur départ html."""
    return f"""<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="utf-8"/>
    <title>{titre}</title>
    <link href="{BASE_PATH}/style.css" rel="stylesheet"/>
</head>
<body>"""

def fin_html() -> str:
    """Générateur fin html."""
    return """</body>
</html>"""

def plage_html_avec_fallback(dossier: Path, fichier: str, position: str, commun: str) ->
str:
    """Lit un fichier HTML avec fallback à la racine pour entete_general et
pied_general."""
    local = dossier / fichier
    if local.exists():
        modele = local
    else:
        if fichier in ("entete_general.html", "pied_general.html"):
            racine = Path(DOSSIER_DOCUMENTS)
            modele = racine / fichier
            if not modele.exists():
                return ""
        else:
            return ""

    with open(modele, "r", encoding="utf-8") as f:
        h = f.read()

    if voir_structure:
        h = f"<div><!-- début {position}{commun} -->{h}<!-- fin {position}{commun} -->
</div>"
    return h

def _generer_navigation(chemin_relatif: List[str]) -> str:
    """Génère la barre de navigation avec BASE_PATH."""
    nav = f'<nav class="navigation"><div class="gauche"><a href="{BASE_PATH}/index.html"
class="monbouton">Accueil</a>'
    for i in range(len(chemin_relatif) - 1):
        lien_parts = [normaliser_nom(p) for p in chemin_relatif[:i+1]]
        lien = BASE_PATH + "/" + "/".join(lien_parts)
        nav += f' → <a href="{lien}/index.html" class="monbouton">{chemin_relatif[i]}
</a>'
    nav += f'</div><div class="droite"><a href="{BASE_PATH}/TDM/index.html"
class="monbouton">Sommaire</a></div></nav>'
    if voir_structure:
        nav = f"<div><!-- début navigation -->{nav}<!-- fin navigation --></div>"
    return nav

def get_word_processes() -> List[Any]:
    """Retourne la liste des processus Word actifs."""
    return [proc for proc in psutil.process_iter(['pid', 'name']) if proc.info['name']
and proc.info['name'].upper() == 'WINWORD.EXE']

def kill_word_processes(processes: List[Any]) -> None:
    """Ferme proprement les processus Word."""
```

```
    for proc in processes:
        print(".", end=" ")
        try:
            proc.terminate()
            proc.wait(timeout=5)
        except Exception:
            proc.kill()
    print(" !")

def traiter_docx(dossier: Path, temp_dir: Path) -> None:
    """Traite tous les .doc/.docx du dossier : crée PDF au début si nécessaire."""
    log(f"Traitement .doc/.docx dans {dossier}")
    entries = list(dossier.iterdir())
    nb_conversions = 0
    for entry in entries:
        if entry.is_file() and entry.suffix.lower() in (".doc", ".docx"):
            nom_pdf = normaliser_nom(entry.stem + ".pdf")
            cible_pdf = dossier / nom_pdf
            if not cible_pdf.exists() or entry.stat().st_mtime >
cible_pdf.stat().st_mtime:
                log(f"Conversion : {entry.name} → {nom_pdf}")
                cree_pdf(dossier, entry.name, cible_pdf, temp_dir)
                nb_conversions += 1
    if nb_conversions == 0:
        log("Aucune conversion nécessaire")

def cree_pdf(chemin_doc: Path, fichier_doc: str, cible_pdf: Path, temp_dir: Path) ->
None:
    """Convertit un .doc/.docx en .pdf via Word sans modifier la date du .docx
original."""
    processes = get_word_processes()
    if processes:
        log("Fermeture processus Word")
        kill_word_processes(processes)

    temp_doc = temp_dir / fichier_doc
    shutil.copy2(chemin_doc / fichier_doc, temp_doc)

    if word_app is None:
        log("Word non disponible — copie simple")
        shutil.copy2(temp_doc, cible_pdf)
        return

    try:
        full_path = str(temp_doc.resolve())
        doc = word_app.Documents.Open(full_path)
        doc.SaveAs(str(cible_pdf.resolve()), FileFormat=17)
        doc.Close()
        log(f"PDF créé : {cible_pdf.name}")
    except Exception as e:
        log(f"Conversion échouée {fichier_doc} : {e}")
        shutil.copy2(temp_doc, cible_pdf)

def _creer_structure_complete(dossier: Path, temp_dir: Path) -> Dict[str, Any]:
    """Crée ou complète structure.py : source de vérité unique pour index.html et
TDM."""
    log(f"Traitement dossier : {dossier}")
    # Création PDF au début
    traiter_docx(dossier, temp_dir)

    struc = {
        "titre_dossier": dossier.name if dossier != Path(DOSSIER_DOCUMENTS) else
```

```
CONFIG.get("titre_site", "Site"),
        "entete_general": True,
        "pied_general": True,
        "entete": True,
        "pied": True,
        "navigation": True,
        "haut_page": True,
        "bas_page": True,
        "ajout_affichage": True,
        "dossiers": [],
        "fichiers": []
    }

    existing = {}
    p = dossier / "STRUCTURE.py"
    if p.exists():
        try:
            from importlib.machinery import SourceFileLoader
            module = SourceFileLoader("STRUCTURE", str(p)).load_module()
            existing = module.STRUCTURE
        except Exception as e:
            log(f"Erreur lecture STRUCTURE.py : {e}")
    struc.update(existing)

    entries = list(dossier.iterdir())
    for entry in sorted(entries, key=lambda x: x.name.lower()):
        if entry.name in IGNORER or entry.name in FICHIERS_ENTETE_PIED:
            continue

        if entry.suffix.lower() in (".doc", ".docx", ".py"):
            continue

        if entry.suffix.lower() not in EXTENSIONS_ACCEPTEES and not entry.is_dir():
            continue

        nom_html = normaliser_nom(entry.name)

        item_defaults = {
            "nom_document": entry.name,
            "nom_html": nom_html,
            "nom_affiché": entry.stem if entry.is_file() else entry.name,
            "nom_TDM": entry.stem if entry.is_file() else entry.name,
            "ajout_affichage": True
        }

        found = False
        for cat in [struc["dossiers"], struc["fichiers"]]:
            for existing_item in cat:
                if existing_item["nom_document"] == entry.name:
                    existing_item.update({k: v for k, v in item_defaults.items() if k
not in existing_item})
                    found = True
                    break
        if not found:
            max_pos = max((it.get("position", 0) for it in struc["dossiers"] +
struc["fichiers"]), default=0)
            item = item_defaults.copy()
            item.update({
                "affiché_index": True,
                "affiché_TDM": True,
                "position": max_pos + 1
            })
            (struc["dossiers"] if entry.is_dir() else struc["fichiers"]).append(item)
```

```
            (struc["dossiers"] if entry.is_dir() else struc["fichiers"]).append(item)

    # Tri des listes par position croissante avant sauvegarde
    struc["dossiers"].sort(key=lambda x: x.get("position", 9999))
    struc["fichiers"].sort(key=lambda x: x.get("position", 9999))

    # Sauvegarde
    content = f"""# STRUCTURE.py - Généré automatiquement
STRUCTURE = {json.dumps(struc, ensure_ascii=False, indent=4).replace("true",
"True").replace("false", "False")}
"""
    p.write_text(content, encoding="utf-8")
    log(f"STRUCTURE.py mis à jour : {dossier}")

    return struc

def copie_site(temp_dir: Path) -> None:
    """Copie /documents vers /html avec gestion .docx → .pdf."""
    log(f"Création du dossier HTML : {DOSSIER_HTML}")
    if Path(DOSSIER_HTML).exists():
        shutil.rmtree(DOSSIER_HTML)
    Path(DOSSIER_HTML).mkdir(parents=True, exist_ok=True)

    if STYLE_CSS_SRC.exists():
        shutil.copy2(STYLE_CSS_SRC, Path(DOSSIER_HTML) / "style.css")
        log("style.css copié")

    arbre_site = _construire_arbre_complet(Path(DOSSIER_DOCUMENTS), temp_dir)
    tdm_path = Path(DOSSIER_HTML) / DOSSIER_TDM
    tdm_path.mkdir(parents=True, exist_ok=True)
    (tdm_path / "structure_site.json").write_text(json.dumps(arbre_site,
ensure_ascii=False, indent=4), encoding="utf-8")
    log("structure_site.json généré")

    for racine, dirs, files in os.walk(DOSSIER_DOCUMENTS):
        dirs[:] = [d for d in dirs if d not in IGNORER]

        rel_path = Path(racine).relative_to(DOSSIER_DOCUMENTS)
        cible_rel_norm = Path(*(normaliser_nom(part) for part in rel_path.parts))
        cible = Path(DOSSIER_HTML) / cible_rel_norm
        cible.mkdir(parents=True, exist_ok=True)

        _creer_structure_complete(Path(racine), temp_dir)

        for fichier in files:
            if any(re.search(pattern, fichier) for pattern in IGNORER):
                continue

            src_file = Path(racine) / fichier

            if fichier.lower().endswith((".doc", ".docx")):
                nom_pdf = normaliser_nom(Path(fichier).stem + ".pdf")
                cible_pdf_documents = Path(racine) / nom_pdf
                cible_pdf_html = cible / nom_pdf
                if cible_pdf_documents.exists():
                    shutil.copy2(cible_pdf_documents, cible_pdf_html)
                    log(f"PDF copié : {nom_pdf}")
                else:
                    log(f"PDF manquant pour {fichier} — ignoré")
            elif fichier.lower().endswith(".html"):
                nom_html = normaliser_nom(fichier)
                shutil.copy2(src_file, cible / nom_html)
            else:
```

```
                nom_html = normaliser_nom(fichier)
                shutil.copy2(src_file, cible / nom_html)

def _construire_arbre_complet(dossier: Path, temp_dir: Path) -> Dict[str, Any]:
    """Construit l'arbre pour structure_site.json."""
    arbre = {
        "titre_dossier": dossier.name if dossier != Path(DOSSIER_DOCUMENTS) else
CONFIG.get("titre_site", "Site"),
        "nom_html": normaliser_nom(dossier.name) if dossier != Path(DOSSIER_DOCUMENTS)
else "",
        "dossiers": [],
        "fichiers": []
    }

    struc = _creer_structure_complete(dossier, temp_dir)

    for cat in ["dossiers", "fichiers"]:
        for item in struc.get(cat, []):
            arbre[cat].append(item.copy())

    for entry in arbre["dossiers"]:
        entry.update(_construire_arbre_complet(dossier / entry["nom_document"],
temp_dir))

    return arbre

def table_index(liste_fils: List[Dict[str, Any]]) -> str:
    """Génère le HTML de la table des éléments avec <br> après chaque lien et style
appliqué."""
    h = []
    for fils in liste_fils:
        if not fils.get("affiché_index", True):
            continue
        nom_affiché = fils.get("nom_affiché", Path(fils.get("nom_document",
"inconnu")).stem)
        nom_stylé = appliquer_style(nom_affiché)
        if fils.get("genre") == "dossier":
            nom = f"{AJOUT[0]}{nom_stylé}{AJOUT[1]}" if fils.get("ajout_affichage",
True) else nom_stylé
            h.append(f'<a class="dossier-item" href="{fils["nom_html"]}/index.html">
{nom}</a><br>')
        else:
            nom = f"{AJOUT[2]}{nom_stylé}{AJOUT[3]}" if fils.get("ajout_affichage",
True) else nom_stylé
            h.append(f'<a class="dossier-item" href="{fils["nom_html"]}">{nom}</a><br>')
    return "".join(h)

def generer_page_index(dossier: Path, temp_dir: Path) -> None:
    """Génère index.html avec BeautifulSoup prettify et fallback pour entete/pied
general."""
    log(f"Génération page : {dossier}")
    rel_path = dossier.relative_to(DOSSIER_DOCUMENTS)
    cible_rel_norm = Path(*(normaliser_nom(part) for part in rel_path.parts))
    cible = Path(DOSSIER_HTML) / cible_rel_norm
    cible.mkdir(parents=True, exist_ok=True)

    struc = _creer_structure_complete(dossier, temp_dir)

    for item in struc.get("dossiers", []):
        item["genre"] = "dossier"
    for item in struc.get("fichiers", []):
        item["genre"] = "fichier"
```

```
    liste_fils = sorted(struc.get("dossiers", []) + struc.get("fichiers", []),
key=lambda x: x.get("position", 9999))

    html_parts = []
    titre = struc.get("titre_dossier", dossier.name)
    html_parts.append(deb_html(titre))

    # haut_page global (si configuré)
    if struc.get("haut_page", False):
        html_parts.append("".join(CONFIG.get("haut_page", [])))

    # entete_general avec fallback
    if struc.get("entete_general", False):
        html_parts.append(plage_html_avec_fallback(dossier, "entete_general.html",
"début", "_général"))

    # entete local
    if struc.get("entete", False):
        html_parts.append(plage_html_avec_fallback(dossier, "entete.html", "début", ""))

    # navigation
    if struc.get("navigation", False):
        html_parts.append(_generer_navigation(list(rel_path.parts)))

    # table
    html_parts.append(f"<div class=\"table-container\"><table class=\"dossiers\"><tbody>
<tr><td>{table_index(liste_fils)}</td></tr></tbody></table></div>")

    # pied local
    if struc.get("pied", False):
        html_parts.append(plage_html_avec_fallback(dossier, "pied.html", "fin", ""))

    # pied_general avec fallback
    if struc.get("pied_general", False):
        html_parts.append(plage_html_avec_fallback(dossier, "pied_general.html", "fin",
"_général"))

    # bas_page global
    if struc.get("bas_page", False):
        html_parts.append("".join(CONFIG.get("bas_page", [])))

    html_parts.append(fin_html())

    html_brut = "".join(html_parts)
    html_prettify = BeautifulSoup(html_brut, 'html.parser').prettify()
    (cible / "index.html").write_text(html_prettify, encoding="utf-8")
    log(f"Page générée : {cible / 'index.html'}")

def main() -> None:
    """Lance la génération complète du site."""
    log("=== DÉBUT GÉNÉRATION ===")
    with tempfile.TemporaryDirectory() as tmpdirname:
        temp_dir = Path(tmpdirname)
        copie_site(temp_dir)
        for racine, dirs, files in os.walk(DOSSIER_DOCUMENTS):
            dirs[:] = [d for d in dirs if d not in IGNORER]
            generer_page_index(Path(racine), temp_dir)
    processes = get_word_processes()
    if processes:
        log("Fermeture processus Word résiduels")
        kill_word_processes(processes)
    log("=== FIN GÉNÉRATION ===")
```

```
    log("=== FIN GENERATION ===")

if __name__ == "__main__":
    main()

# fin du "genere_site.py" version "20.0"
```

XIV. cree_table_des_matieres.py

```
# cree_table_des_matieres.py — Version 6.15

version = ("cree_table_des_matieres.py", "6.15")
print(f"[Version] {version[0]} — {version[1]}")

import json
from pathlib import Path

from lib1.options import DOSSIER_HTML, BASE_PATH

def log(msg: str) -> None:
    """Affiche un message simple."""
    print(msg)

def construire_arbo(dossier: Path, prefixe: str = "") -> str:
    """Construit récursivement l'arbre HTML de la TDM."""
    html = ""
    structure_path = dossier / "STRUCTURE.py"
    if structure_path.exists():
        try:
            from importlib.machinery import SourceFileLoader
            module = SourceFileLoader("STRUCTURE", str(structure_path)).load_module()
            struc = module.STRUCTURE
        except Exception as e:
            log(f"Erreur lecture STRUCTURE.py dans {dossier} : {e}")
            struc = {"dossiers": [], "fichiers": []}
    else:
        struc = {"dossiers": [], "fichiers": []}

    # Tri par position
    struc["dossiers"].sort(key=lambda x: x.get("position", 9999))
    struc["fichiers"].sort(key=lambda x: x.get("position", 9999))

    for item in struc["dossiers"]:
        if not item.get("affiché_TDM", True):
            continue
        nom_html = item["nom_html"]
        nom_affiché = item.get("nom_affiché", item["nom_document"])
        chemin = f"{prefixe}/{nom_html}/index.html"
        lien = f"{BASE_PATH}{chemin}"
        sous_arbo = construire_arbo(dossier / item["nom_document"], f"
{prefixe}/{nom_html}")
        if sous_arbo:
            html += f'<li><details open><summary><a href="{lien}">{nom_affiché}</a>
</summary><ul>{sous_arbo}</ul></details></li>\n'
        else:
            html += f'<li><a href="{lien}">{nom_affiché}</a></li>\n'

    for item in struc["fichiers"]:
        if not item.get("affiché_TDM", True):
            continue
```

```
            continue
        nom_html = item["nom_html"]
        nom_affiché = item.get("nom_affiché", item["nom_document"])
        chemin = f"{prefixe}/{nom_html}"
        lien = f"{BASE_PATH}{chemin}"
        html += f'<li><a href="{lien}">{nom_affiché}</a></li>\n'

    return html

def generer_tdm() -> None:
    """Génère le fichier TDM/index.html sans style embarqué."""
    log("Génération de la Table des Matières")
    racine = Path(DOSSIER_HTML)
    tdm_path = racine / "TDM"
    tdm_path.mkdir(parents=True, exist_ok=True)

    arbre_html = construire_arbo(racine)

    titre_site = "Site"
    html = f"""<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="utf-8"/>
    <title>Table des matières - {titre_site}</title>
    <link href="{BASE_PATH}/style.css" rel="stylesheet"/>
</head>
<body>
    <h1>Table des matières</h1>
    <ul class="tree">
        {arbre_html}
    </ul>
</body>
</html>"""

    (tdm_path / "index.html").write_text(html, encoding="utf-8")
    log("TDM/index.html généré")

if __name__ == "__main__":
    generer_tdm()

# fin du "cree_table_des_matieres.py" version "6.15"
```

XV `style.css`

```
/* style.css — Version 3.3 – Gamme bleu-vert élégante + table centrée 80% */

* { box-sizing: border-box; margin: 0; padding: 0; }

body {
    font-family: "Segoe UI", Arial, sans-serif;
    background: #e8f4f8;
    color: #2c3e50;
    padding: 20px;
    line-height: 1.6;
}

.monTitre {
    font-size: 58px;
    color: #16a085;
    text-align: center;
    padding: 20px;
```

```
}

.monSousTitre {
    font-size: 32px;
    color: #1abc9c;
    text-align: center;
    margin: 20px 0;
    border-bottom: 4px solid #16a085;
}

.navigation {
    display: flex;
    justify-content: space-between;
    align-items: center;
    background: #1abc9c;
    padding: 15px 30px;
    border-radius: 15px;
    margin: 30px 0;
    box-shadow: 0 4px 15px rgba(26, 188, 156, 0.3);
    flex-wrap: wrap;
    gap: 10px;
}

.navigation .gauche, .navigation .droite {
    display: flex;
    align-items: center;
    gap: 15px;
    flex-wrap: wrap;
}

.monbouton {
    padding: 12px 26px;
    background: #ecf0f1;
    color: #2c3e50;
    border: 3px solid #bdc3c7;
    border-radius: 30px;
    text-decoration: none;
    font-weight: bold;
    box-shadow: 0 4px 10px rgba(0,0,0,0.1);
    transition: all 0.3s ease;
}

.monbouton:hover {
    background: #1abc9c;
    color: white;
    border-color: #16a085;
    transform: translateY(-2px);
}

/* Conteneur centré pour la table */
.table-container {
    display: flex;
    justify-content: center;
    margin: 40px 0;
}

/* Table centrée à 80% de largeur */
table.dossiers {
    width: 80%;                  /* 80% de la largeur du conteneur */
    max-width: 1000px;           /* Largeur maximale raisonnable */
    margin: 0 auto;              /* Centrage horizontal (redondant avec .table-container
mais sûr) */
```

```
    border-collapse: collapse;
    background: white;
    box-shadow: 0 8px 25px rgba(0,0,0,0.1);
    border-radius: 12px;
    overflow: hidden;
}

table.dossiers td {
    padding: 18px 24px;
    border-bottom: 1px solid #ecf0f1;
    text-align: left;
}

table.dossiers tr:last-child td { border-bottom: none; }
table.dossiers tr:hover { background: #f8fcfd; }

.dossier-item a { color: #16a085; font-weight: bold; }
.fichier-pdf a { color: #e74c3c; font-weight: bold; }

footer {
    text-align: center;
    margin-top: 50px;
    color: #7f8c8d;
    font-size: 0.9em;
}

/* Style du bouton mail */
.btn-mail {
    padding: 15px 30px;
    font-size: 16px;
    font-weight: bold;
    color: white;
    background-color: #4CAF50;
    border: 2px solid #388E3C;
    border-radius: 8px;
    cursor: pointer;
    outline: none;
    transition: all 0.2s ease;
    box-shadow: 0 4px #666, 0 -2px 3px rgba(0, 0, 0, 0.2);
}

.btn-mail:active {
    transform: translateY(4px);
    box-shadow: 0 2px #666, 0 -1px 3px rgba(0, 0, 0, 0.2);
}

.btn-mail:hover {
    background-color: #45a049;
    border-color: #388E3C;
}

/* Responsive */
@media (max-width: 768px) {
    table.dossiers {
        width: 95%;
    }
    .monTitre { font-size: 42px; }
    .monSousTitre { font-size: 26px; }
}

/* Styles spécifiques à la TDM */
.tree ul {
    margin-left: 20px;
```

```
    margin-left: 20px;
}
.tree details {
    margin: 5px 0;
}
.tree summary {
    cursor: pointer;
    font-weight: bold;
}
.tree a {
    text-decoration: none;
    color: #0066cc;
}
.tree a:hover {
    text-decoration: underline;
}

/* fin du "style.css" version "3.3" */
```

XVI `lancer.cmd`

- Mettre le dossier courant dans hebreu4.0

```
cd C:\SiteGITHUB\Hebreu4.0
```

- Executer lancer.cmd

```
prog\lancer.cmd
```

```
@echo off
REM Début de "lancer.cmd" version "2.2"
cls
echo.
echo lancer.cmd — Version 2.2
echo.

:: Activation de l'environnement virtuel si nécessaire
where python | findstr /i "virPy13" >nul
if %errorlevel% neq 0 (
    echo Activation de l'environnement virtuel virPy13...
    call C:\virPy13\Scripts\activate.bat
    if %errorlevel% neq 0 (
        echo [ERREUR] Impossible d'activer virPy13
        pause
        exit /b 1
    )
)

:: Aller dans le répertoire du projet
cd /d "%~dp0\.."

echo.
echo === Génération du site réel à partir du dossier documents ===

:: 1. On lance d'abord genere_site.py (il crée html/ et tous les dossiers)
python prog\genere_site.py
if %errorlevel% neq 0 (
    echo [ERREUR] genere_site.py a échoué
    pause
    exit /b 1
)

:: 2. Ensuite seulement on génère la TDM (html/ existe maintenant → plus d'erreur)
python prog\cree_table_des_matieres.py
if %errorlevel% neq 0 (
    echo [ERREUR] cree_table_des_matieres.py a échoué
    pause
    exit /b 1
)
rem créer /Hebreu4.0/html/Hebreu4.0/html/style.css
xcopy html\style.css html\Hebreu4.0\html\*.*

echo.
echo === Démarrage du serveur local ===
npx http-server html -p 3500 --cors -c-1 -o "/index.html"

echo.
echo Site réel disponible sur : http://localhost:3500/index.html
echo.
pause
REM Fin de "lancer.cmd" version "2.2"
```

Généré le 20 Décembre 2025