EXAMEN — PROJET GO

FileOps / WebOps / ProcOps / SecureOps/InfraOps

M1 Devops — Projet individuel – 26-27/03/2026
Notation graduée : 10 / 12 / 14 / 16 / 18 (sur 20)

Nous avons vu ensemble les bases du langage que vous allez utiliser dans ce projet.

Au fil des niveaux, vous allez devoir faire des recherches.

C'est un projet solo, mais je vous invite à communiquer entre vous et à ne pas rester bloquer.

Et…. Posez-moi des questions !!!

Objectif

Développer un outil Go en console qui manipule des fichiers texte, récupère du contenu web (Wikipédia), gère les processus système, et ajoute des mécanismes de sécurité (droits / verrouillage) aux niveaux avancés.

Contraintes générales

- Go uniquement
- Standard library autorisée
- Bibliothèques externes autorisées **uniquement lorsqu'elles sont imposées** (Wikipédia)
- Le programme doit gérer les erreurs proprement
- Les fichiers générés doivent être écrits dans out/

Livrables du projet (<u>idéalement</u> lien vers github/gitlab, sinon zip)

A rendre sur Cesar

- code source Go
- config.txt - config.json
- data/ tous les fichiers d'input nécessaires
- out/ fichiers créés par votre programme
- README.md (capital !) :
    - Procédure d'exécution

- !!!fonctionnalités implémentées
- niveau visé (10/12/14/16/18)
- Description du travail effectué

NIVEAU 10/20 — FileOps (fichiers & données) + config TXT

1) Menu interactif

- menu en boucle + choix utilisateur + quitter

2) Configuration au format .txt

Au démarrage, le programme lit un fichier config.txt.

Format imposé

Fichier texte avec une clé par ligne :

default_file=data/input.txt

base_dir=data

out_dir=out

default_ext=.txt

- les lignes vides ou commençant par # sont ignorées
- si une clé manque, vous utilisez une valeur par défaut

3) Fichier texte courant (fournir un ou plusieurs fichier - lorem ipsum par exemple)

- charger le fichier par défaut depuis la config
- possibilité de choisir un autre fichier via le menu
- vérifier existence et type (fichier)

4) Liste des fonctionnalités – choix menu

Sélectionnable depuis le menu Choix A et B

Chaque choix exécutera toutes les sous-fonctionnalités décrites

Le programme demandera à l'utilisateur un chemin. Si l'utilisateur ne le fourni pas => chemin par défaut dans le fichier de config.

Choix A - Analyse sur fichier courant

L'utilisateur fourni le nom d'un fichier

1. Infos fichier : taille, date création/modif, nb lignes

2. Stats mots : nb mots en ignorant les numériques + longueur moyenne
3. Compter lignes contenant un mot-clé
4. Filtrer lignes contenant un mot-clé demandé → out/filtered.txt
5. Filtrer lignes ne contenant pas le mot-clé demandé → out/filtered_not.txt
6. Head : N premières lignes → out/head.txt
7. Tail : N dernières lignes → out/tail.txt

Choix B - Analyse multi-fichiers

L'utilisateur fourni le nom d'un repertoire

8. Batch : analyser tous les .txt situé dans un emplacement demandé à l'utilisateur
9. Rapport global : générer out/report.txt (format libre mais lisible)
10. Indexation : générer out/index.txt listant (chemin, taille, date)
11. Fusion : fusionner tous les .txt de base_dir → out/merged.txt

---

NIVEAU 12/20 — WebOps : Wikipédia avec goquery

Ajouter au menu :

Choix C - Analyser une page Wikipédia

Bibliothèque conseillée : goquery

Installation :

go get github.com/PuerkitoBio/goquery

Comportement attendu

1. demander un article (ex: Go_(langage))
2. télécharger :
   https://fr.wikipedia.org/wiki/<ARTICLE>
3. extraire le texte des paragraphes
4. appliquer **au moins 2 traitements déjà présents** (mots / filtres / stats)
5. écrire dans out/wiki_<article>.txt (ou afficher)

Bonus : possibilité de traiter plusieurs articles en même temps.

Extraction HTML via regex déconseillée.

NIVEAU 14/20 — ProcOps : gestion des processus Windows + macOS

Ce niveau doit fonctionner **sur Windows et sur macOS**.
Piste : **OS-aware** (détection runtime.GOOS).

Fonctionnalités ajoutée au menu principal :

Choix D . Créer un sous-menu ProcessOps :

1) Lister les processus (top N)

- Windows : utiliser tasklist (par ex. tasklist /FO CSV)
- macOS : utiliser ps -Ao pid,comm

Afficher au minimum : PID + nom.

2) Rechercher / filtrer

- demander un mot (ex: “chrome”, “go”, “code”)
- afficher les processus correspondants

3) Kill sécurisé

- demander un PID
- afficher un récapitulatif (PID + nom si possible)
- demander une confirmation explicite (yes/no)
- exécuter :
    - Windows : taskkill /PID <pid> /T (et éventuellement /F en option)
    - macOS : kill <pid> (option avancée : kill -9)

4) Gestion d’erreurs

- PID invalide
- droits insuffisants
- process déjà terminé
- commande non disponible

NIVEAU 16/20 — SecureOps + config JSON + droits + verrouillage

À ce niveau :

1. la configuration doit être **migrée vers JSON**
2. vous ajoutez une **fonctionnalité de gestion des droits / verrouillage**

A) Configuration en JSON

Remplacer config.txt par config.json.

Exemple :

```
{
  "default_file": "data/input.txt",
```

```
  "base_dir": "data",
  "out_dir": "out",
  "default_ext": ".txt",
  "wiki_lang": "fr",
  "process_top_n": 10
}
```

Le programme doit accepter un flag optionnel :

- --config config.json

Choix E Gestion des droits

Ajouter au menu “SecureOps” une option :

Verrouiller / Déverrouiller un fichier

Objectif : empêcher l’écriture concurrente (ou simuler un verrou).

Deux options acceptées :

Option 1 (recommandée, portable, simple) : Lockfile

- créer un fichier out/<nom>.lock
- si le lock existe → fichier considéré “verrouillé”
- déverrouiller = supprimer le lock
- avantage : marche partout, simple à corriger

Option 2 (avancée, vraie lock OS - Bonus) : File lock OS

- macOS/Linux : syscall.Flock
- Windows : LockFileEx via syscall
- plus complexe...

C) Sécurité (au choix, mais “système”)

Ajouter au moins 2 :

- rendre un fichier “read-only” (si OS le permet) via os.Chmod (macOS) / attribut Windows via commande

- vérifier permissions (macOS) et signaler WARN
- journaliser les actions sensibles (kill, lock) dans out/audit.log
- Toute action destructive (kill, lock) doit être **confirmée** et **loggée** dans out/audit.log (fortement recommandé)

NIVEAU 18/20 — InfraOps : Performance & Orchestration Conteneur

1. Parallélisme & Performance (Goroutines) :

- Transformer le scan de fichiers (Niveau 10) et le scrapping (Niveau 12) pour qu'ils s'exécutent en parallèle.
- Utiliser des sync.WaitGroup et des channels pour récupérer les résultats.
- *Défi :* Si l'utilisateur demande 5 pages Wikipédia, elles doivent être téléchargées simultanément.

2. Supervision Docker (SDK ou Exec) :

- Ajouter une option "ContainerOps" au menu.
- Lister les conteneurs Docker actifs sur la machine (Nom, Image, Statut).
- Afficher l'utilisation CPU/RAM d'un conteneur spécifique (via docker stats --no-stream).

3. Santé Système & Alerting :

- Vérifier l'espace disque restant. Si < 10%, le programme doit lever une alerte visuelle (couleur dans le terminal via code ANSI ou message d'erreur bloquant).