

Le Livre de GrimOS

"La complexité est l'ennemie de la fiabilité."

Bienvenue dans la documentation officielle de **GrimOS**, le système d'exploitation frugal, entièrement codé en Python, conçu pour redonner à l'utilisateur le contrôle absolu de sa machine.

Introduction

À l'heure où les systèmes d'exploitation modernes pèsent des dizaines de gigaoctets et masquent leur fonctionnement derrière d'opaques couches de code incompréhensible, GrimOS propose une approche radicalement différente. Il ne s'agit pas d'un système conçu pour les serveurs d'entreprise ou le cloud, mais d'un outil personnel, intime et transparent.

GrimOS n'utilise pas de démons lourds, pas de gestionnaire de fenêtres complexe, et pas d'animations superflues. Il utilise un Linux minimal, et délègue toute l'interface, la gestion du matériel et l'expérience utilisateur à de simples scripts Python (avec Tkinter) que n'importe qui peut lire, modifier, et casser pour mieux les comprendre.

Ce livre a été écrit pour documenter l'architecture de ce système. Que vous soyez un simple curieux, un hacker cherchant à déployer une borne d'arcade, ou un développeur souhaitant comprendre l'art de l'illusion graphique (le "Simulacre"), vous êtes au bon endroit.

Table des Matières

- [Le Livre de GrimOS](#)
 - [Introduction](#)
 - [Table des Matières](#)
- [Chapitre 1 : Genèse et Philosophie de GrimOS](#)
 - [Introduction](#)
 - [1. Redonner vie aux ordinateurs oubliés](#)

- [2. La Philosophie du Projet : Transparence et Minimalisme](#)
- [3. GrimOS face aux systèmes modernes](#)
 - [L'inflation logicielle \(Bloatware\)](#)
 - [L'illusion du contrôle](#)
 - [Télémétrie et vie privée](#)
- [Chapitre 2 : Panorama des fonctionnalités de GrimOS](#)
 - [1. L'Environnement de Bureau \(Le Desktop\)](#)
 - [2. Les Applications Intégrées](#)
 - [L'Explorateur de Fichiers](#)
 - [L'Éditeur de Texte et de Code](#)
 - [Le Navigateur Web](#)
 - [Les Outils Périphériques](#)
 - [3. Personnalisation et Modularité](#)
- [Chapitre 3 : Architecture du système](#)
 - [1. Vue d'ensemble de l'architecture](#)
 - [2. Le Coeur du Système \(Dossier core/\)](#)
 - [core/desktop.py \(Le Bureau\)](#)
 - [core/window.py \(Le Simulacre de Fenêtre\)](#)
 - [core/app_manager.py \(Le Chargeur d'Applications\)](#)
 - [core/theme.py, wifi.py, audio.py](#)
 - [3. Les Applications \(Dossier apps/\)](#)
 - [Structure d'une application](#)
 - [La fonction magique : start\(window\)](#)
 - [4. La Configuration \(Dossier config/\)](#)
 - [applications.json](#)
 - [5. Le Terminal TTY Natif : Un choix architectural](#)
- [Chapitre 4 : Tutoriel : Créer sa première application \(Calculatrice\)](#)

- [1. Créer le dossier de l'application](#)
- [2. Le code minimal \(Le contrat GrimOS\)](#)
- [3. Déclarer l'application au système](#)
- [4. Tester l'application](#)
- [Conclusion](#)
- [Chapitre 5 : L'installation magique et le déploiement](#)
 - [La Philosophie : Le Workflow "Deux Clés USB".](#)
 - [1. Le Mythe de l'ISO vs La Méthode des Deux Clés](#)
 - [2. Étape 1 : Le Socle Debian Minimal](#)
 - [3. Étape 2 : Le Script d'Installation \(install_grimos.sh\).](#)
 - [A. Téléchargement des dépendances](#)
 - [B. Configuration de l'Autologin](#)
 - [C. Le lancement automatique de X11 \(.profile\)](#)
-  [Guide d'Installation Officiel - GrimOS](#)
 -  [Étape A : Installation du système de base \(Debian\).](#)
 - [1. Télécharger Debian](#)
 - [2. Créer une clé USB d'installation \(Clé n°1\)](#)
 - [3. Installer Debian sur le PC cible](#)
 -  [Étape B : Préparation de la clé USB "GrimOS" \(Clé n°2\)](#)
 -  [Étape C : Déploiement de GrimOS](#)
 - [1. Se connecter](#)
 - [2. Brancher et trouver la clé USB](#)
 - [3. Monter \(connecter\) la clé USB](#)
 - [4. Lancer l'installation magique](#)
 - [5. Laissez la magie opérer !](#)
 - [6. Redémarrage final](#)
- [Chapitre 6 : L'Automagie sous le capot \(Réseau et USB\).](#)

- [1. La gestion du Réseau \(Wi-Fi\) "à la main"](#)
- [2. L'Auto-montage USB "Maison"](#)
 - [Comment ça marche ?](#)
- [3. La Frugalité comme philosophie](#)
- [Chapitre 7 : L'Art du Trompe-l'œil : Le Moteur de Thèmes](#)
 - [1. Séparer le fond de la forme](#)
 - [2. L'Injection globale \(core/theme.py\)](#)
 - [3. Le dictionnaire magique](#)
 - [4. Créez votre propre thème](#)
- [Chapitre 8 : Repenser la sécurité : L'approche Mono-Utilisateur](#)
 - [1. Un ordinateur, Un humain](#)
 - [2. Le secret : L'automatisation du Sudo](#)
 - [3. Les avantages ergonomiques](#)
 - [4. Les compromis assumés \(Les risques\)](#)
- [Chapitre 9 : L'Avenir de GrimOS \(Kiosques, Raspberry Pi et Conclusion\)](#)
 - [1. Au-delà du Bureau : Les Kiosques et Bornes d'arcade](#)
 - [2. Le mariage parfait avec le Raspberry Pi](#)
 - [Conclusion générale](#)
- [Annexe : Comment nous avons développé GrimOS](#)
 - [1. La philosophie du développement à distance](#)
 - [2. Préparation des machines](#)
 - [A. Sur la machine cible \(L'antique ordinateur\)](#)
 - [B. Sur la machine de développement \(L'ordinateur moderne\)](#)
 - [3. Sécuriser et faciliter la connexion \(Clé SSH\)](#)
 - [4. Le Flux de travail quotidien \(Workflow\)](#)
 - [1. Coder \(En local\)](#)
 - [2. Transférer \(SCP\)](#)

- [3. Exécuter et Déboguer \(SSH\)](#)
- [Annexe : Noyau - app_manager.py](#)
 - [Rôle et utilité](#)
 - [Implémentation technique](#)
 - [Pistes de modification](#)
 - [Code Source](#)
- [Annexe : Noyau - audio.py](#)
 - [Rôle et utilité](#)
 - [Implémentation technique](#)
 - [Pistes de modification](#)
 - [Code Source](#)
- [Annexe : Noyau - config.py](#)
 - [Rôle et utilité](#)
 - [Implémentation technique](#)
 - [Pistes de modification](#)
 - [Code Source](#)
- [Annexe : Noyau - desktop.py](#)
 - [Rôle et utilité](#)
 - [Implémentation technique](#)
 - [Pistes de modification](#)
 - [Code Source](#)
- [Annexe : Noyau - main.py](#)
 - [Rôle et utilité](#)
 - [Implémentation technique](#)
 - [Pistes de modification](#)
 - [Code Source](#)
- [Annexe : Noyau - theme.py](#)

- [Rôle et utilité](#)
- [Implémentation technique](#)
- [Pistes de modification](#)
- [Code Source](#)
- [Annexe : Noyau - wifi.py](#)
 - [Rôle et utilité](#)
 - [Implémentation technique](#)
 - [Pistes de modification](#)
 - [Code Source](#)
- [Annexe : Noyau - window.py](#)
 - [Rôle et utilité](#)
 - [Implémentation technique](#)
 - [Pistes de modification](#)
 - [Code Source](#)
- [Annexe : Application - Bloc-notes](#)
 - [Rôle et utilité](#)
 - [Implémentation technique](#)
 - [Pistes de modification](#)
 - [Code Source](#)
- [Annexe : Application - Caméra](#)
 - [Rôle et utilité](#)
 - [Implémentation technique](#)
 - [Pistes de modification](#)
 - [Code Source](#)
- [Annexe : Application - Éditeur de Code](#)
 - [Rôle et utilité](#)
 - [Implémentation technique](#)

- [Pistes de modification](#)
- [Code Source](#)
- [Annexe : Application - Explorateur](#)
 - [Rôle et utilité](#)
 - [Implémentation technique](#)
 - [Pistes de modification](#)
 - [Code Source](#)
- [Annexe : Application - Imprimante](#)
 - [Rôle et utilité](#)
 - [Implémentation technique](#)
 - [Pistes de modification](#)
 - [Code Source](#)
- [Annexe : Application - Navigateur Web](#)
 - [Rôle et utilité](#)
 - [Implémentation technique](#)
 - [Pistes de modification](#)
 - [Code Source](#)
- [Annexe : Application - Paramètres](#)
 - [Rôle et utilité](#)
 - [Implémentation technique](#)
 - [Pistes de modification](#)
 - [Code Source](#)
- [Annexe : Application - Gestionnaire des tâches \(TaskMgr\)](#)
 - [Rôle et utilité](#)
 - [Implémentation technique](#)
 - [Pistes de modification](#)
 - [Code Source](#)

- [Annexe : Application - Terminal](#)
 - [Rôle et utilité](#)
 - [Implémentation technique](#)
 - [Pistes de modification](#)
 - [Code Source](#)
- [Annexe : Application - Visionneuse d'Images](#)
 - [Rôle et utilité](#)
 - [Implémentation technique](#)
 - [Pistes de modification](#)
 - [Code Source](#)

Chapitre 1 : Genèse et Philosophie de GrimOS

Introduction

Dans un monde technologique où la course à la puissance semble sans fin, l'industrie informatique nous pousse inexorablement vers le renouvellement constant de nos équipements. Chaque nouvelle version d'un système d'exploitation grand public exige davantage de mémoire vive, des processeurs plus véloce et des espaces de stockage gigantesques. Face à cette surenchère, une question fondamentale se pose : nos besoins réels ont-ils évolué de manière aussi exponentielle que le matériel que l'on nous impose ?

C'est de cette interrogation qu'est né le projet **GrimOS**.

Le nom **GrimOS** n'a pas été choisi au hasard. C'est l'acronyme de **Graphical Resource-Independent Minimal Operating System** (Système d'Exploitation Graphique Minimal et Indépendant des Ressources). Ce nom porte en lui la promesse centrale du projet : offrir une interface graphique complète et fonctionnelle, tout en s'affranchissant de la course aux ressources matérielles.

1. Redonner vie aux ordinateurs oubliés

L'objectif initial et le moteur principal de GrimOS sont profondément ancrés dans une démarche de lutte contre l'obsolescence programmée et le gaspillage électronique. Aux quatre coins du monde, des millions d'ordinateurs parfaitement fonctionnels dorment dans des placards ou finissent à la décharge. Leur seul tort ? Ne plus être capables de faire tourner les lourdes interfaces graphiques et les services en arrière-plan imposés par les systèmes modernes.

GrimOS a été conçu avec une ambition claire : ressusciter ces machines. En ciblant un environnement minimaliste, capable de s'exécuter sur du matériel très modeste (comme un Raspberry Pi ancien ou un ordinateur portable datant de plus d'une décennie), GrimOS prouve qu'une machine "obsolète" peut encore offrir une expérience fluide, réactive et utile.

Que ce soit pour naviguer sur le web de manière basique, éditer des textes, organiser des fichiers ou s'initier à la programmation, la puissance nécessaire est en réalité bien en deçà des standards actuels de l'industrie.

2. La Philosophie du Projet : Transparence et Minimalisme

La philosophie de GrimOS s'articule autour de trois piliers fondamentaux : **le minimalisme, la lisibilité et l'auto-détermination**.

Le minimalisme : Là où les systèmes traditionnels empilent les couches d'abstraction (gestionnaires de fenêtres complexes, daemons de télémétrie, services de mise à jour automatiques imposés), GrimOS revient à l'essentiel. L'interface est dessinée pixel par pixel via Python et Tkinter, éliminant le besoin de lourds serveurs d'affichage. Chaque widget, de la barre des tâches au moniteur système graphique, a été codé pour n'utiliser que les ressources strictement nécessaires.

La lisibilité (Le "DIY" informatique) : L'une des plus grandes forces de GrimOS est d'être écrit en Python, un langage interprété et reconnu pour sa clarté. Contrairement aux OS classiques, compilés et transformés en boîtes noires hermétiques, le code source de GrimOS est son propre manuel. N'importe quel utilisateur curieux peut ouvrir `core/desktop.py` avec l'éditeur de texte intégré, comprendre comment la barre des tâches est dessinée, la modifier à sa guise et relancer l'interface instantanément. GrimOS n'est pas seulement un système d'exploitation, c'est un bac à sable pédagogique.



```
Éditeur de Code - desktop.py
Nouveau Nouveau GUI Ouvrir Enregistrer
1 import tkinter as tk
2 from tkinter import simpledialog
3 import subprocess
4 from time import strftime
5 import sys
6 import os
7
8 from core.app_manager import AppManager
9 from core.config import load_applications
10
```

Le code de la barre des tâches ouvert directement dans l'éditeur intégré de GrimOS.

L'auto-détermination : GrimOS redonne le pouvoir à l'utilisateur. Rien ne s'exécute sans raison, aucune donnée n'est envoyée à l'insu de l'utilisateur. Le système fait exactement ce qu'on lui demande, ni plus ni moins.

3. GrimOS face aux systèmes modernes

Pour bien comprendre la place de GrimOS, il est essentiel de le comparer aux géants modernes (Windows, macOS, et même certaines distributions Linux grand public comme Ubuntu).

L'inflation logicielle (Bloatware)

Les systèmes modernes souffrent de ce que l'on appelle l'inflation logicielle. Lors d'un démarrage typique, un OS moderne consomme souvent entre 2 et 4 Go de mémoire vive simplement pour afficher le bureau, accaparé par des indexeurs de fichiers, des antivirus, des stores d'applications et des services de synchronisation cloud. **L'approche GrimOS** : Au démarrage (Boot-to-GUI via X11 sans gestionnaire de session lourd), GrimOS consomme à peine quelques dizaines de mégaoctets de RAM. L'interface graphique est directement liée à l'interpréteur Python.

L'illusion du contrôle

Sur les OS modernes, l'utilisateur est devenu un simple "consommateur" de l'interface. Modifier le comportement fondamental de la barre des tâches, ou ajouter une fonction spécifique au cœur du système relève du parcours du combattant ou de la rétro-ingénierie.

L'approche GrimOS : Le système est un ensemble modulaire de scripts Python et de configurations JSON (`applications.json`). L'utilisateur est invité à devenir l'architecte de son propre système. Le bureau n'est qu'une surcouche applicative transparente reposant sur un socle Debian extrêmement solide.



```
1  [
2    {
3      "name": "Bloc-notes",
4      "module": "apps.blocnotes.app",
5      "icon": "icons/blocnotes.png",
6      "category": "Bureautique"
7    },
8    {
9      "name": "Visionneuse",
10     "module": "apps.visionneuse.app",
11     "icon": "icons/visionneuse.png",
12     "category": "Bureautique"
```

Le fichier de configuration JSON qui gère le menu "Démarrer" de GrimOS.

Télémetrie et vie privée

Aujourd'hui, l'ordinateur personnel n'a jamais été aussi connecté... aux serveurs de ses créateurs. Les systèmes d'exploitation modernes récoltent en continu des données d'utilisation sous couvert d'amélioration du service. **L'approche GrimOS** : GrimOS est inerte d'un point de vue réseau tant que l'utilisateur ne le sollicite pas explicitement (via le navigateur ou des requêtes réseau volontaires). Le système respecte l'intimité de manière absolue.

En conclusion, GrimOS n'a pas la prétention de remplacer les systèmes modernes pour le traitement vidéo 8K ou les jeux vidéo de dernière génération. Sa mission est tout autre : prouver qu'une informatique frugale, élégante, transparente et réparable par tous est non seulement possible, mais extrêmement gratifiante. Les chapitres suivants détailleront l'architecture technique qui a permis de donner vie à cette vision.

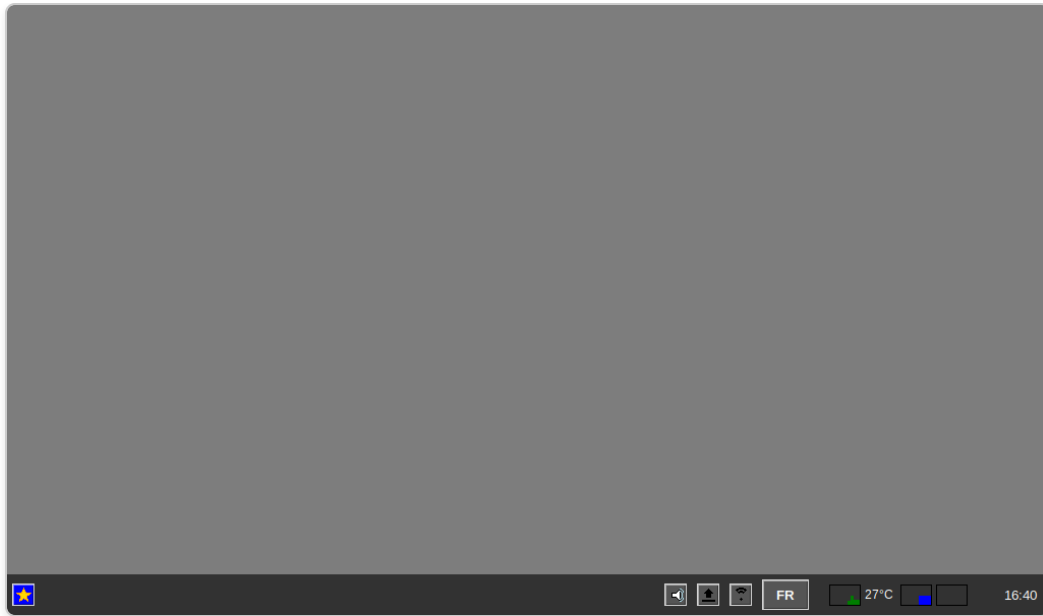
Chapitre 2 : Panorama des fonctionnalités de GrimOS

Bien que GrimOS soit extrêmement léger et repose sur un socle technique minimaliste, il est livré "clés en main" avec tout le nécessaire pour une utilisation quotidienne classique. Ce chapitre dresse un inventaire des fonctionnalités natives de la version actuelle.

1. L'Environnement de Bureau (Le Desktop)

Au démarrage, GrimOS accueille l'utilisateur avec une interface graphique familière, conçue en pur Python (via Tkinter) pour ne dépendre d'aucun gestionnaire de fenêtres lourd (comme GNOME ou KDE).

- **Le Bureau et ses Raccourcis** : L'écran principal permet d'accueillir des raccourcis vers vos applications ou dossiers préférés. Ces raccourcis sont cliquables, déplaçables et persistants.
- **La Barre des Tâches Multifonction** : Située en bas de l'écran, elle centralise le contrôle du système. Outre le menu "Démarrer" et les fenêtres ouvertes, elle intègre un **Moniteur Système Graphique** avancé. Ce dernier affiche en temps réel des graphiques pour l'utilisation du processeur (CPU), de la mémoire vive (RAM), de la mémoire virtuelle (SWAP), la température, ainsi qu'une icône dynamique pour l'état de la batterie.
- **Contrôles Rapides** : D'un simple clic depuis la barre des tâches, l'utilisateur peut modifier le volume sonore, gérer ses réseaux Wi-Fi, ou basculer la disposition de son clavier.



Aperçu du bureau principal avec son moniteur système interactif.

2. Les Applications Intégrées

GrimOS fournit un écosystème d'applications de base, chacune pensée pour consommer le moins de mémoire possible.

L'Explorateur de Fichiers

Véritable centre névralgique, l'explorateur permet de naviguer dans l'arborescence du système. Il intègre des fonctions avancées telles que :

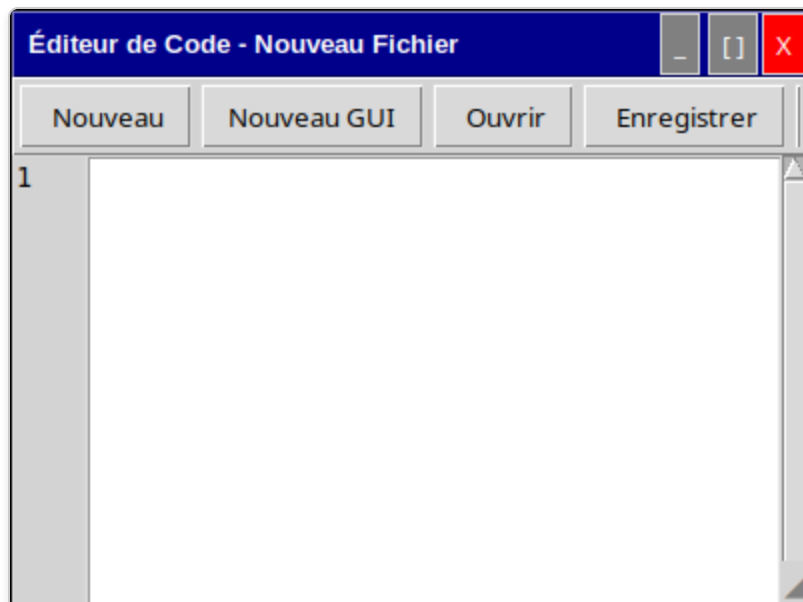
- Le montage et démontage de clés USB en un clic.
- La connexion native aux partages réseau (SMB/Samba).
- Des icônes vectorielles nettes et professionnelles (générées dynamiquement à l'installation).



L'explorateur de fichiers naviguant dans les dossiers systèmes.

L'Éditeur de Texte et de Code

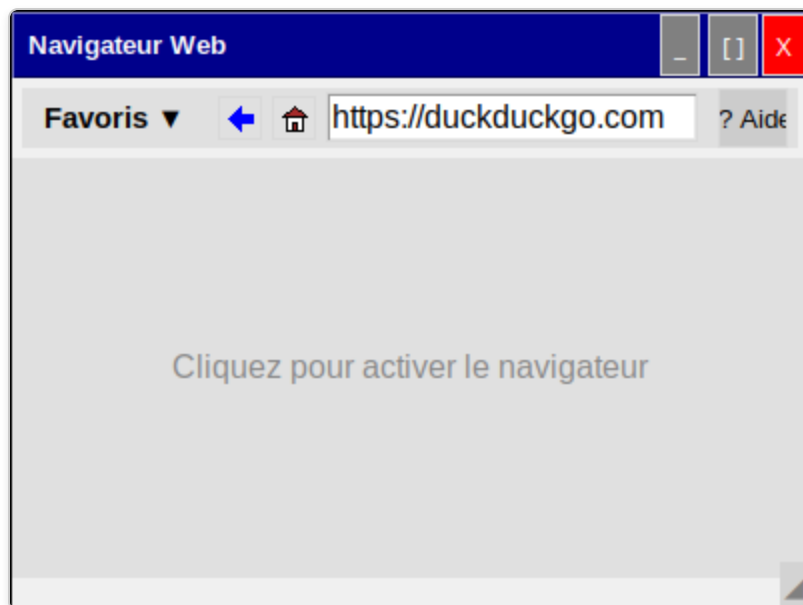
Plus qu'un simple bloc-notes, l'éditeur de GrimOS est un outil idéal pour la productivité et la programmation. Il permet bien sûr de prendre des notes, mais il offre aussi la possibilité de modifier directement le code source du système (fichiers Python, JSON) et d'exécuter des scripts à la volée.



L'éditeur de code intégré avec coloration syntaxique.

Le Navigateur Web

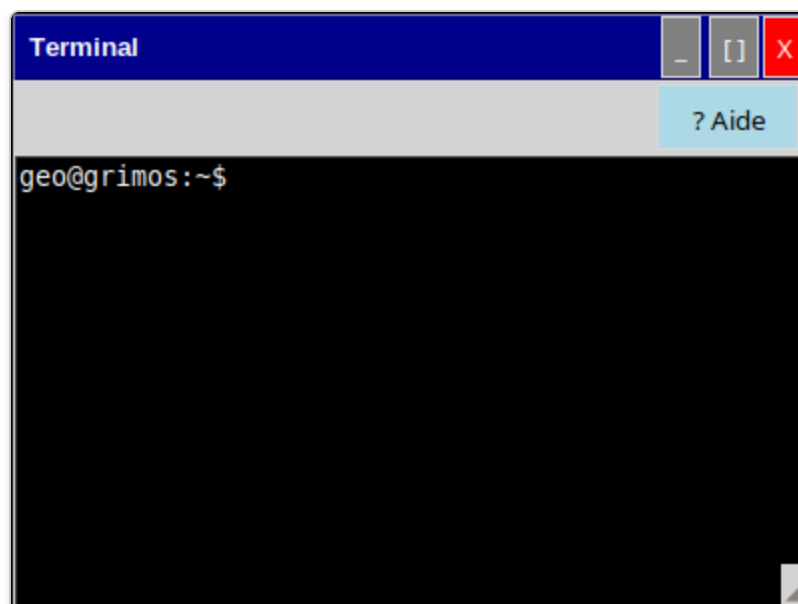
Pour l'accès à Internet, GrimOS intègre une solution d'une frugalité redoutable. Reposant sur un moteur allégé (comme `surf`), le navigateur permet de consulter des pages web complexes sans saturer la mémoire vive de la machine, tout en s'intégrant parfaitement dans le fenêtrage de l'OS.



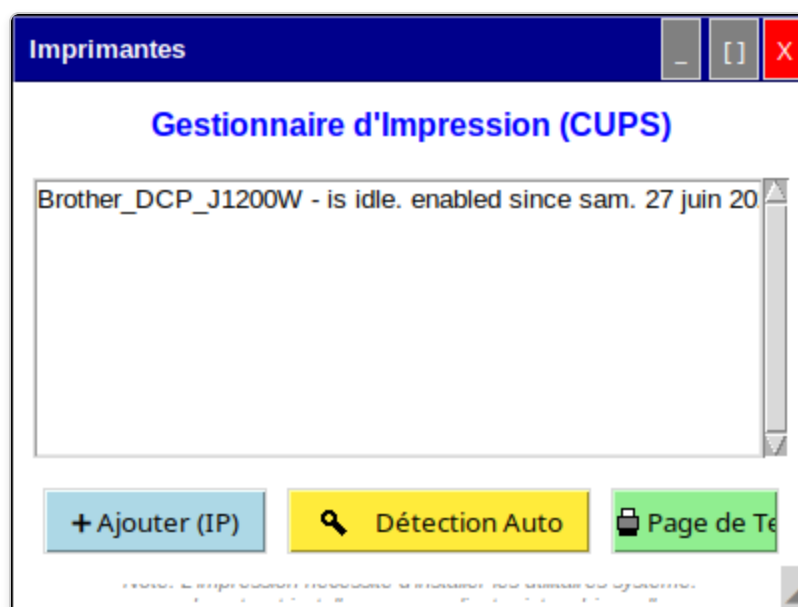
Navigation web ultra-légère directement intégrée.

Les Outils Périphériques

- **Terminal (Xterm)** : Pour les utilisateurs avancés, un émulateur de terminal est directement accessible, parfaitement dimensionné dans l'interface, permettant d'interagir avec le noyau Debian sous-jacent.



- **Imprimante** : Un gestionnaire basé sur CUPS permet de détecter automatiquement les imprimantes locales ou réseau et de lancer des pages de test sans ligne de commande.



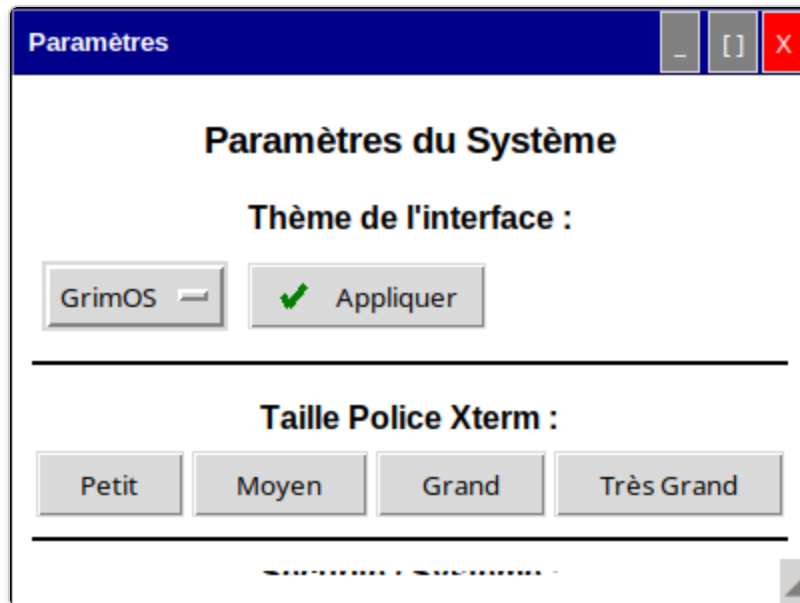
- **Caméra** : Une application légère pour capturer des photos via une webcam connectée.

 Capture d'écran de la Caméra

3. Personnalisation et Modularité

L'un des aspects les plus uniques de GrimOS est sa malléabilité.

- **Le Moteur de Thèmes** : Via l'application **Paramètres**, l'utilisateur peut changer instantanément l'apparence de tout le système. Envie d'un style "Windows 95", "Windows XP", d'une interface très sombre "Hacker" ou de l'élégance par défaut de "GrimOS" ? Un clic suffit à modifier l'intégralité des couleurs et des bordures, sans aucun redémarrage.
- **La Persistance de Session** : GrimOS mémorise intelligemment votre espace de travail. Si vous éteignez la machine avec l'éditeur ouvert sur un coin de l'écran, il s'ouvrira exactement au même endroit et à la même taille lors du prochain démarrage.



L'application Paramètres permet de changer de thème à la volée.

Avec ces outils, un vieil ordinateur équipé de GrimOS redevient immédiatement une station de travail capable de rédiger, de naviguer, de coder et de communiquer, le tout avec une consommation de ressources défiant toute concurrence.

Chapitre 3 : Architecture du système

GrimOS est conçu pour être aussi lisible qu'un livre ouvert. Son architecture repose sur une séparation stricte entre le noyau (la boucle principale et le bureau), le gestionnaire de fenêtres, et les applications tierces.

L'ensemble du système d'exploitation est structuré dans un dossier unique (`grimos_build`), sans fichiers éparpillés dans le système de fichiers linux.

1. Vue d'ensemble de l'architecture

La Fondation de GrimOS est structurée en trois couches principales, emboîtées comme des poupées russes :

1. La Couche d'Exécution (Le Noyau)

2. `main.py` : Point d'entrée de l'application. Initialise la fenêtre Tkinter plein écran.
3. `core/desktop.py` : Construit l'environnement visuel (fond d'écran, barre des tâches, horloge).

4. La Couche de Gestion (Le Chef d'Orchestre)

5. `core/app_manager.py` : Le moteur qui lance et ferme les applications.
6. `core/window.py` : Génère le "simulacre" de fenêtre (cadre, barre de titre, boutons).

7. La Couche Applicative (Les Logiciels)

8. `apps/` : Dossier contenant toutes les applications indépendantes (`editeur` , `terminal` , etc.).
9. `config/applications.json` : Le registre déclarant ces applications pour le menu Démarrer.

L'architecture est construite en 3 couches horizontales :

1. **La Fondation (Core)** : Gère la boucle graphique Tkinter, le fond d'écran, et la barre des tâches.
2. **Le Gestionnaire (App Manager & Window)** : Isole chaque application dans une fausse "fenêtre" (puisque'il n'y a pas de gestionnaire de fenêtres X11 classique).

3. **Les Applications (Apps)** : Des modules Python indépendants qui se branchent sur le système.

2. Le Cœur du Système (Dossier `core/`)

Le dossier `core/` contient le "noyau" de GrimOS. Ce sont les fichiers critiques qui maintiennent l'illusion du système d'exploitation. L'approche de GrimOS prend ce concept à rebours : c'est l'interface graphique qui est le cœur du système.

```
graph TD
    A[Noyau Linux / Shell] -->|Bypass| B[Python / Tkinter main.py]
    B --> C[Gestionnaire de Fenêtres Virtuel]
    C --> D[Services Intégrés]
    D --> E[wpa_cli Wi-Fi]
    E --> F[lsblk USB Mount]
    F --> G[Applications Python apps/]
```

Dès le démarrage, le système Unix lance immédiatement un gigantesque script Python (`main.py`). (À la racine) C'est l'étincelle qui allume le moteur. Ce fichier initialise Tkinter (`root = tk.Tk()`), force l'affichage en plein écran sans bordures, et appelle le `Desktop`. Il ne contient aucune logique métier.

`core/desktop.py` (Le Bureau)

C'est le chef d'orchestre visuel.

- **Rôle** : Il dessine l'environnement (le fond d'écran), crée la barre des tâches en bas de l'écran, gère le bouton "Démarrer", et affiche l'heure ainsi que les widgets système (CPU, RAM).
- **Fonctionnement** : Il charge le fichier `applications.json` pour construire dynamiquement le menu Démarrer.

`core/window.py` (Le Simulacre de Fenêtre)

C'est le secret technique de GrimOS.

- **Rôle** : Étant donné que GrimOS s'exécute sur un serveur X11 "nu" (sans environnement de bureau comme GNOME ou XFCE), **les fenêtres n'existent pas naturellement.**

- **Fonctionnement** : Ce fichier crée un composant (`Frame`) qui ressemble à une fenêtre (avec une barre de titre bleue, un bouton croix rouge pour fermer, et des bordures). Il gère mathématiquement la logique de "cliquer-glisser" pour permettre à l'utilisateur de déplacer ces fausses fenêtres sur l'écran.

`core/app_manager.py` (Le Chargeur d'Applications)

- **Rôle** : C'est lui qui fait le lien entre le Bureau et les Applications.
- **Fonctionnement** : Quand l'utilisateur clique sur une icône, l' `AppManager` va importer dynamiquement le module Python demandé (par exemple `apps.editeur.app`), instancier un conteneur `Window` , et injecter cette fenêtre dans le module de l'application via la fonction `start(window)` .

`core/theme.py` , `wifi.py` , `audio.py`

Ces modules sont des assistants (helpers) :

- `theme.py` : Fournit les codes couleurs et les polices (style "Classic 98" ou "Modern Dark").
- `wifi.py` & `audio.py` : Sont des ponts (wrappers) qui envoient des commandes silencieuses au système Debian sous-jacent (comme `nmcli` pour le réseau ou `amixer` pour le son) et renvoient le résultat à l'interface Python.

3. Les Applications (Dossier `apps/`)

Dans GrimOS, une application n'est pas un fichier exécutable binaire (`.exe` ou fichier ELF). C'est un simple dossier contenant un fichier `app.py` .

Structure d'une application

Si vous explorez `apps/editeur/` OU `apps/explorateur/` , vous y trouverez toujours la même architecture minimale :

- `app.py` : Le fichier principal.
- (*optionnel*) Des sous-fichiers Python ou des ressources locales.

La fonction magique : `start(window)`

Toutes les applications GrimOS obéissent à un "contrat" (une interface). Le fichier `app.py` **doit** contenir une fonction nommée `start` qui prend au moins un argument :

```
def start(window, app_manager=None, **kwargs):  
    # La magie opère ici
```

L'argument `window` n'est pas la fenêtre principale de l'ordinateur, mais le "panneau intérieur" de la fausse fenêtre générée par `window.py`. L'application n'a plus qu'à dessiner ses boutons et ses textes à l'intérieur de ce panneau, sans jamais se soucier de savoir comment elle peut être déplacée ou fermée.

4. La Configuration (Dossier `config/`)

Pour rester facilement modifiable, GrimOS extrait sa configuration du code source.

`applications.json`

C'est le registre du système. Il contient une liste de toutes les applications installées sous forme de tableau JSON. Pour installer une nouvelle application, il suffit de copier son dossier dans `apps/`, puis d'ajouter quelques lignes dans `applications.json` :

```
{  
  "name": "Éditeur de Code",  
  "module": "apps.editeur.app",  
  "icon": "icons/editeur.png",  
  "category": "Développement"  
}
```

Au prochain démarrage, l'application apparaîtra automatiquement dans le menu Démarrer sous la bonne catégorie, avec sa propre icône. C'est cette simplicité absolue qui différencie l'architecture de GrimOS de celle des monstres logiciels de notre époque.

5. Le Terminal TTY Natif : Un choix architectural

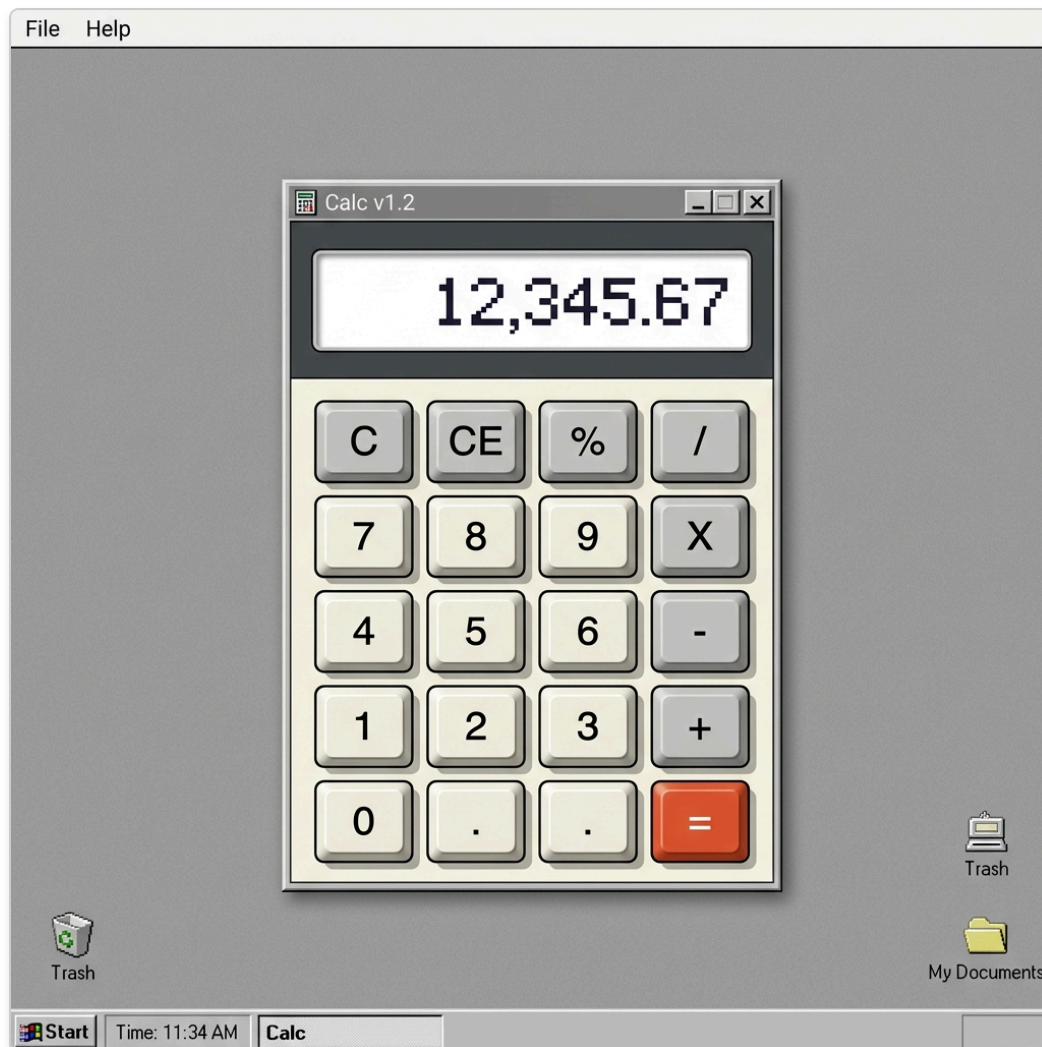
L'une des particularités de GrimOS est son traitement du "Terminal" (la ligne de commande administrateur). Plutôt que de s'enfermer dans un terminal virtuel instable à l'intérieur de l'interface graphique, GrimOS joue la carte de l'architecture Unix pure.

Dans le menu Démarrer, le bouton "**Fermer la session**" ne fait pas qu'éteindre l'ordinateur. Il met fin au processus Python (`main.py`) et tue le serveur graphique X11.

L'écran noir redescend alors brutalement, révélant le véritable terminal maître de Debian (le TTY). Dans ce terminal natif, l'utilisateur possède un accès absolu et inébranlable (`sudo`) pour mettre à jour son système ou installer des paquets complexes (sans aucun bug d'affichage).

Une fois son administration système terminée, il lui suffit de taper `startx` pour relancer instantanément la machine dans le monde coloré et fenêtré de GrimOS, ou de taper `exit` pour déclencher une reconnexion automatique.

Chapitre 4 : Tutoriel : Créer sa première application (Calculatrice)



Dans le chapitre précédent, nous avons vu que GrimOS ne possède pas de "bureau" au sens Unix du terme. Chaque fenêtre n'est en fait qu'un petit composant (un `Frame` Tkinter) dessiné à l'intérieur de l'application principale.

Maintenant que vous avez compris l'architecture de GrimOS (Chapitre 4) et que votre environnement de développement à distance est prêt (Chapitre 3), il est temps de mettre les mains dans le code !

Dans ce chapitre, nous allons créer ensemble une application de A à Z : une petite **Calculatrice**.

1. Créer le dossier de l'application

Toutes les applications de GrimOS vivent de manière indépendante dans le dossier `apps/`. Pour commencer, créez un nouveau dossier nommé `calculatrice` dans ce répertoire.

Terminal :

```
cd grimos_build/apps/  
mkdir calculatrice
```

À l'intérieur de ce dossier, créez un fichier Python nommé obligatoirement `app.py`. C'est le point d'entrée que le moteur de GrimOS cherchera à lancer.

2. Le code minimal (Le contrat GrimOS)

Ouvrez le fichier `app.py`. Pour qu'une application soit reconnue par le système, elle **doit** posséder une fonction nommée `start(window, app_manager=None, **kwargs)`.

L'argument `window` correspond au "panneau intérieur" de la fenêtre. C'est un simple cadre (Frame) Tkinter dans lequel nous avons le droit de dessiner.

Copiez le code suivant :

```
import tkinter as tk  
from tkinter import messagebox  
  
def start(window, app_manager=None, **kwargs):  
    # Changement de la couleur de fond du panneau  
    window.configure(bg="#f0f0f0")  
  
    # Titre de l'application  
    titre = tk.Label(window, text="Calculatrice GrimOS", font=("Arial", 16, "bold"), bg="#f0f0f0")  
    titre.pack(pady=10)  
  
    # Un champ de texte pour taper le calcul  
    entree = tk.Entry(window, font=("Arial", 14), width=20)  
    entree.pack(pady=10)  
  
    # Fonction qui sera appelée quand on clique sur le bouton  
    def calculer():  
        expression = entree.get()  
        try:  
            # Évaluation mathématique simple  
            resultat = eval(expression)
```

```

        messagebox.showinfo("Résultat", f"Le résultat est : {resultat}")
    except Exception as e:
        messagebox.showerror("Erreur", "Calcul invalide !")

# Le bouton d'action
btn = tk.Button(window, text="Calculer", font=("Arial", 12), command=calculer)
btn.pack(pady=10)

```

Explications :

1. Nous importons `tkinter` pour dessiner nos boutons et nos champs de texte.
2. Nous utilisons `window` comme "parent" pour tous nos composants (Label, Entry, Button). C'est ce qui permet à l'application d'être proprement contenue dans la fenêtre générée par GrimOS.
3. La fonction `eval()` est utilisée ici pour la simplicité de l'exemple afin de calculer mathématiquement le texte tapé par l'utilisateur.

3. Déclarer l'application au système

Si vous lancez GrimOS maintenant, votre application existe bien sur le disque dur, mais le système ne la connaît pas encore. Il faut l'inscrire dans le grand registre : le fichier

`config/applications.json`.

Ouvrez ce fichier et ajoutez un nouveau bloc à la fin de la liste (n'oubliez pas la virgule après le bloc précédent !) :

```

{
  "name": "Calculatrice",
  "module": "apps.calculatrice.app",
  "icon": "icons/terminal.png",
  "category": "Bureautique"
}

```

Note : Pour l'icône, nous utilisons ici temporairement l'icône du terminal (`icons/terminal.png`). Vous pourrez par la suite ajouter votre propre fichier `calculatrice.png` dans le dossier `icons/`.

4. Tester l'application

C'est l'heure de vérité. Relancez GrimOS (ou cliquez sur "Redémarrer GrimOS" dans le menu Démarrer si vous êtes déjà connecté).

1. Ouvrez le menu **Démarrer**.
2. Allez dans la catégorie **Bureautique**.
3. Cliquez sur **Calculatrice**.

Une magnifique fenêtre s'ouvre ! Vous pouvez la déplacer en cliquant sur sa barre de titre bleue, la redimensionner (si autorisé) et la fermer avec la croix rouge. Tout ce "simulacre de fenêtre" a été généré automatiquement par `core/window.py` sans que vous n'ayez eu à écrire une seule ligne de code pour le gérer.

Tapez `5 + 5 * 2` dans le champ et cliquez sur Calculer. Une petite fenêtre d'alerte s'ouvrira pour vous afficher le résultat : `15` .

Conclusion

Félicitations, vous venez de créer votre première application native pour GrimOS !

L'architecture est volontairement d'une simplicité enfantine. Que vous vouliez développer un simple lecteur de musique, un jeu de démineur ou un outil de surveillance réseau, le principe sera exactement le même : un dossier dans `apps/` , une fonction `start()` , et une déclaration JSON.

La seule limite est votre imagination et vos connaissances en Python !

Chapitre 5 : L'installation magique et le déploiement

Ce chapitre vous guide étape par étape pour transformer n'importe quel vieil ordinateur ou machine virtuelle en une station GrimOS complète, de la façon la plus simple et la plus fiable possible.



La Philosophie : Le Workflow "Deux Clés USB".

Comment transforme-t-on un PC vide en une machine GrimOS fully-fonctionnelle ?

1. Le Mythe de l'ISO vs La Méthode des Deux Clés

Historiquement, les systèmes d'exploitation (comme Windows ou Ubuntu) sont distribués sous forme de fichiers "ISO" amorçables. C'est un énorme fichier de plusieurs gigaoctets contenant l'intégralité du système pré-compilé.

Cependant, générer une image ISO personnalisée est un processus complexe, très lourd et difficile à maintenir (les outils de *live-build* évoluent sans cesse).

Dans la philosophie de simplicité de GrimOS, nous avons opté pour une approche beaucoup plus élégante et universelle : **La méthode des deux clés USB**.

1. **La clé Socle** : Elle contient l'installateur officiel de Debian "Minimal" (Netinst).
2. **La clé Magique** : Elle contient le dossier source de GrimOS et le script `install_grimos.sh`.

Cette séparation des pouvoirs garantit que GrimOS reposera toujours sur un socle Debian officiel, propre, certifié et à jour, sans avoir à gérer nous-mêmes la lourdeur de la compilation d'un noyau Linux.

2. Étape 1 : Le Socle Debian Minimal

La première étape de l'installation de GrimOS consiste à installer un "moteur" nu.

Lors de l'installation de Debian via la première clé, la règle d'or est de **tout désactiver** (GNOME, KDE, serveur web, environnement de bureau standard). La seule case qui doit rester cochée est "Utilitaires standards du système".

Pourquoi ? Parce que les environnements de bureaux classiques installent des centaines de "daemons" (services tournant en arrière-plan) pour gérer le réseau, le bluetooth, les imprimantes, ou la télémétrie. GrimOS est conçu pour faire tout cela lui-même, à sa manière, sans aucun bruit de fond.

Une fois installé, l'ordinateur démarre sur un écran noir effrayant avec un texte blanc : le fameux TTY. Le socle est prêt.

3. Étape 2 : Le Script d'Installation (`install_grimos.sh`)

C'est ici qu'intervient la deuxième clé USB. L'utilisateur insère la clé contenant GrimOS et exécute simplement le script `install_grimos.sh`. Que se passe-t-il sous le capot ?

A. Téléchargement des dépendances

Le script se connecte à internet et télécharge le minimum vital pour afficher des fenêtres :

- `xserver-xorg` et `xinit` : Les fondations de l'affichage graphique sur Linux.
- `python3-tk` : Le module Python permettant de dessiner l'interface.
- `iw` et `wpa_supplicant` : Les outils stricts pour le Wi-Fi.

B. Configuration de l'Autologin

Un système moderne se doit de démarrer directement sur le bureau. Le script modifie la configuration du gestionnaire de terminaux (`getty`) de Debian pour que l'utilisateur principal soit connecté automatiquement (Auto-login) sur le premier terminal (`tty1`) dès l'allumage de la machine, sans demander de mot de passe.

C. Le lancement automatique de X11 (`.profile`)

Une fois l'utilisateur connecté automatiquement en arrière-plan, le système lit son fichier de profil caché (`.profile`). Le script y injecte cette condition :

Terminal :

```
if [ -z "$DISPLAY" ] && [ "$(tty)" = "/dev/tty1" ]; then
    startx
fi
```

Traduction : "Si aucune interface graphique n'est affichée (`$DISPLAY` est vide), et que je suis sur l'écran principal (`tty1`), alors lance immédiatement l'interface graphique (`startx`)."

En combinant un socle extrêmement stable et un script de configuration totalement transparent, le déploiement de GrimOS prouve que l'installation d'un système d'exploitation n'a pas besoin d'être une "boîte noire" obscure, mais peut être une opération logique, compréhensible et parfaitement maîtrisée.



Guide d'Installation Officiel - GrimOS

Bienvenue dans le guide d'installation de GrimOS. Ce document est conçu pour vous accompagner pas-à-pas dans l'installation de votre système d'exploitation, même si vous n'êtes pas un expert en informatique.

L'installation se déroule en trois grandes étapes :

1. **Étape A** : L'installation du "moteur" de base (Debian minimal).
 2. **Étape B** : La préparation de votre clé USB contenant GrimOS.
 3. **Étape C** : Le déploiement automatique de GrimOS sur votre machine.
-



Étape A : Installation du système de base (Debian)

GrimOS repose sur un socle très solide appelé **Debian**. Nous allons d'abord installer une version "vide" de Debian, que notre script transformera ensuite en GrimOS.

1. Télécharger Debian

- Rendez-vous sur le site officiel de Debian et téléchargez l'image d'installation par le réseau (**netinst**) : [Télécharger Debian \(netinst\)](#).

2. Créer une clé USB d'installation (Clé n°1)

- Munissez-vous d'une première clé USB (8 Go minimum).
- Utilisez un logiciel comme [Rufus](#) (sur Windows) ou [BalenaEtcher](#) (sur Mac/Linux) pour "flasher" le fichier ISO Debian téléchargé sur votre clé USB.

3. Installer Debian sur le PC cible

- Branchez la clé USB sur le PC que vous souhaitez transformer en machine GrimOS et allumez-le.
- Démarrez sur la clé USB (souvent en tapotant **F12**, **F8**, ou **Suppr** au démarrage pour choisir le périphérique de démarrage).

- Choisissez l'installation normale ou graphique ("Graphical install") et suivez l'assistant :
- **Langue, Pays, Clavier** : Choisissez le français (ou votre langue).
- **Réseau** : Connectez-vous à internet (par câble ou Wi-Fi) quand cela vous est demandé.
- **Nom d'hôte** : Laissez par défaut (ex: `debian`).
- **Mot de passe administrateur (root)** : Laissez vide et faites "Continuer" (cela activera `sudo` pour notre utilisateur principal, ce qui est plus simple).
- **⚠ CRUCIAL - Création de l'utilisateur** : Lorsqu'on vous demande le nom complet du nouvel utilisateur et l'identifiant, tapez exactement `grimos` (tout en minuscules). Choisissez un mot de passe dont vous vous souviendrez.
- **Partitionnement des disques** : Choisissez "Assisté - utiliser un disque entier" et sélectionnez votre disque dur.
- **⚠ CRUCIAL - Sélection des logiciels** : À la fin de l'installation, un écran "Sélection des logiciels" apparaît. **C'est l'étape la plus importante.**
 - Décochez **absolument tout** (surtout "environnement de bureau Debian", "GNOME", etc.).
 - La seule case qui doit rester cochée est : **"Utilitaires standards du système"**.
- Une fois l'installation terminée, l'ordinateur va redémarrer. **Retirez la clé USB.**
- L'ordinateur démarrera sur un écran noir avec un texte vous demandant de vous connecter (`login`). C'est normal ! Éteignez le PC pour le moment ou laissez-le tel quel.

Étape B : Préparation de la clé USB "GrimOS" (Clé n°2)

Maintenant que le socle est prêt, nous allons préparer la clé USB qui contient tout l'univers visuel et les applications de GrimOS.

1. Sur votre ordinateur personnel, prenez une **deuxième clé USB** (formatée en FAT32 ou exFAT).
2. Ouvrez le dossier `grimos_build` (le dossier où se trouve ce guide).
3. **Copiez l'intégralité du contenu** de ce dossier à la racine de votre clé USB.

- Assurez-vous que le fichier `install_grimos.sh` et le fichier `main.py` se trouvent bien à la racine de cette clé USB, et non cachés dans un sous-dossier.

Étape C : Déploiement de GrimOS

C'est la dernière ligne droite ! Le script va tout faire à votre place.

1. Se connecter

- Allumez votre PC (celui avec l'écran noir installé à l'Étape A).
- Au message `debian login:`, tapez `grimos` et appuyez sur Entrée.
- Tapez votre mot de passe (les caractères ne s'affichent pas à l'écran, c'est normal) et appuyez sur Entrée.

2. Brancher et trouver la clé USB

- Branchez votre clé USB "GrimOS" (Clé n°2).
- Tapez la commande suivante pour lister les disques :

 **Terminal :**

```
lsblk
```

- Vous verrez une liste. Cherchez votre clé USB (souvent nommée `sdb1`, `sdc1`, et reconnaissable par sa taille). Notez ce nom. Nous utiliserons `sdb1` pour l'exemple.

3. Monter (connecter) la clé USB

- Tapez la commande suivante pour créer un point d'attache :

 **Terminal :**

```
sudo mkdir -p /mnt/usb
```

- Tapez votre mot de passe (celui de l'utilisateur `grimos`) s'il vous est demandé.

- "Branchez" logiciellement la clé USB :

Terminal :

```
sudo mount /dev/sdb1 /mnt/usb
```

(Remplacez `sdb1` par le nom trouvé à l'étape précédente si différent).

4. Lancer l'installation magique

- Allez dans le dossier de la clé USB :

Terminal :

```
cd /mnt/usb
```

- Lancez le script d'installation :

Terminal :

```
sudo bash install_grimos.sh
```

5. Laissez la magie opérer !

Le script va télécharger toutes les interfaces graphiques, configurer le démarrage automatique et installer vos applications. Cela peut prendre quelques minutes selon votre connexion internet.

6. Redémarrage final

Une fois que le script affiche que tout est terminé avec succès :

- Tapez :


Terminal :

```
cd /  
sudo umount /mnt/usb
```

- Débranchez votre clé USB.
- Redémarrez la machine :

Terminal :

```
sudo reboot
```

 **Félicitations !** Au redémarrage, la machine s'allumera toute seule, sans demander de mot de passe, et affichera directement le bureau magnifique de GrimOS !

Chapitre 6 : L'Automagie sous le capot (Réseau et USB)



L'un des plus grands défis de GrimOS a été de remplacer les "démons" Linux complexes (comme *NetworkManager* pour le réseau ou *Udisks2* pour les clés USB) par du pur code Python, lisible et modifiable par l'utilisateur.

Un système d'exploitation moderne (Linux, Windows ou macOS) est peuplé de "démons" (*Daemons* en anglais, ou *Services*). Ce sont de petits programmes invisibles qui tournent en boucle en arrière-plan dès l'allumage de la machine pour gérer diverses tâches : détecter une clé USB, maintenir la connexion réseau, vérifier les mises à jour, etc.

Le problème ? L'accumulation de ces dizaines de démons consomme énormément de mémoire vive (RAM) et de temps de calcul, même lorsque vous ne faites rien.

Dans ce chapitre, nous allons voir comment GrimOS parvient à s'en passer totalement pour la gestion du Réseau (Wi-Fi) et des clés USB, en utilisant une approche "à la main", 100% Python.

1. La gestion du Réseau (Wi-Fi) "à la main"

Sur un Linux grand public (comme Ubuntu), le réseau est géré par **NetworkManager**, un très gros démon qui fournit une interface complexe.

GrimOS a supprimé NetworkManager. À la place, il utilise l'outil fondamental et historique de Linux : `wpa_supplicant`. C'est le programme le plus léger possible capable de parler aux antennes Wi-Fi.

Le fichier `core/wifi.py` joue le rôle de traducteur entre l'interface de GrimOS et cet outil. Quand vous cliquez sur l'icône Wi-Fi dans la barre des tâches :

1. Python lance une simple ligne de commande invisible : `sudo wpa_cli scan`
2. Python lit la réponse brute en texte du système d'exploitation.
3. Python "parse" (découpe) ce texte pour extraire le nom des réseaux (SSID) et la puissance du signal.
4. L'interface affiche la liste déroulante.

Le gain : Aucun processus lourd ne surveille le réseau en permanence. Le scan n'a lieu que lorsque l'utilisateur le demande explicitement.

2. L'Auto-montage USB "Maison"

Insérer une clé USB sur un ordinateur moderne provoque une série d'actions complexes gérées par des démons comme `Udisks2` ou `gvfs`, qui montent la clé, envoient des notifications au bureau, indexent les fichiers pour la recherche, etc.

Dans GrimOS, la philosophie est brutale mais d'une efficacité redoutable. Le bureau (`core/desktop.py`) possède une petite boucle très légère qui s'exécute silencieusement toutes

les quelques secondes pour vérifier l'état de la batterie et... les disques durs.

Comment ça marche ?

1. À intervalle régulier, Python exécute la commande native Linux : `lsblk -J` (List Block Devices).
2. Cette commande très rapide renvoie la liste complète de tout ce qui est branché à la machine au format JSON.
3. Python analyse le JSON. S'il détecte un disque de type `RM` (Removable / Amovible) qui n'a pas de `mountpoint` (point de montage), cela signifie qu'une clé vient d'être insérée !
4. Automatiquement, le script lance la commande `sudo mount /dev/sdX /media/usb...` pour monter la clé de manière transparente.
5. Une icône "🔌 USB" apparaît alors dans la barre des tâches.

Lorsque l'utilisateur a terminé, il clique sur cette icône. Python exécute simplement `sudo umount` et la clé peut être retirée en toute sécurité.

3. La Frugalité comme philosophie

En déléguant ces opérations critiques non pas à d'énormes services en arrière-plan, mais à de minuscules commandes textuelles Unix (comme `mount`, `lsblk`, `wpa_cli`) orchestrées par quelques dizaines de lignes de code Python, GrimOS accomplit un miracle de légèreté.

C'est ainsi que la version finale de GrimOS, avec son interface graphique, son explorateur, sa détection USB et son réseau Wi-Fi actifs, réussit à consommer **moins de 100 Mégaoctets de RAM** au repos, là où d'autres systèmes saturent des ordinateurs équipés de 4 Gigaoctets.

GrimOS ne cache rien à son utilisateur : l'OS est un chef d'orchestre, et le terminal Unix est son instrument.

Chapitre 7 : L'Art du Trompe-l'œil : Le Moteur de Thèmes

L'un des défis majeurs lorsqu'on développe une interface graphique en Python brut (avec la bibliothèque standard Tkinter), c'est l'esthétique. Par défaut, Tkinter propose un rendu qui rappelle fortement les interfaces des années 90, et de nombreux développeurs l'abandonnent rapidement, le jugeant trop "moche" ou archaïque.

Pourtant, GrimOS réussit un tour de force : il est capable de simuler l'élégance de *macOS Classic*, les couleurs vibrantes d'un *Ubuntu* ou l'austérité rassurante d'un *Windows 98*.

Ce miracle repose sur **le moteur de thèmes**, une architecture technique ingénieuse qui sépare la logique des applications de leur apparence.

1. Séparer le fond de la forme

Dans une application GrimOS, vous ne verrez presque jamais un développeur écrire `tk.Button(bg="red", fg="white")`. S'il faisait cela, l'application ne s'adapterait pas lorsque l'utilisateur changerait de thème depuis les paramètres.

Au lieu de cela, l'application se contente de déclarer ses composants (Boutons, Textes) de manière neutre. C'est le moteur de thèmes qui, tel un peintre en bâtiment, passe derrière pour colorer chaque élément.

2. L'Injection globale (`core/theme.py`)

Si vous ouvrez le fichier `core/theme.py`, vous découvrirez le secret de GrimOS : la commande magique `option_add`.

```
def apply_theme(root, theme_name):
    theme = THEMES.get(theme_name, THEMES["GrimOS"])

    # Configuration globale pour tous les futurs widgets
    root.option_add("*background", theme["bg"])
    root.option_add("*foreground", theme["fg"])
    root.option_add("*font", theme["font"])

    # Styles spécifiques
```

```
root.option_add("*Button.background", theme["btn_bg"])
root.option_add("*Button.relief", theme["relief"])
```

Cette fonction intercepte le cœur même de Tkinter (l'objet `root`). Grâce à l'étoile `*`, elle applique une règle absolue : *"Désormais, tout objet créé dans le système aura ces couleurs et cette police par défaut"*. Ainsi, l'explorateur de fichiers, la calculatrice ou l'éditeur de texte se retrouvent instantanément rhabillés sans même s'en rendre compte.

3. Le dictionnaire magique

Qu'est-ce qu'un thème dans GrimOS ? Ce n'est pas un fichier CSS complexe. C'est simplement un petit bloc de dictionnaire (une liste de variables).

Si l'on prend l'exemple du célèbre thème **Win 98**, voici sa définition exacte dans le code :

```
"Win 98": {
    "bg": "#C0C0C0",           # Le fameux gris de Windows 98
    "fg": "black",             # Texte noir
    "font": ("MS Sans Serif", 9), # La police d'époque
    "btn_bg": "#C0C0C0",       # L'effet 3D "bouton surélevé"
    "relief": "raised",         # Le fond d'écran bleu/vert caractéristique
    "desktop_bg": "#008080",
    "desktop_fg": "white",
    "accent": "#000080",
    "title_bg": "#000080",     # La barre de titre bleu foncé
    "title_fg": "white",
    ...
}
```

Ce simple changement de 10 variables suffit à métamorphoser intégralement le système !

4. Créez votre propre thème

Le système est tellement ouvert que vous pouvez créer votre propre thème en 30 secondes.

1. Ouvrez `core/theme.py` avec l'Éditeur de Code.
2. Ajoutez un nouveau bloc dans le dictionnaire `THEMES`, par exemple :

```
"Cyberpunk": {
    "bg": "black",
    "fg": "#00FF00",
    "font": ("Courier New", 11, "bold"),
    "btn_bg": "#222222",
    "relief": "flat",
    "desktop_bg": "black",
    "desktop_fg": "#00FF00",
    "accent": "#FF00FF",
```

```
"title_bg": "#FF00FF",  
"title_fg": "black",  
"taskbar_bg": "black",  
"taskbar_fg": "#00FF00",  
"start_btn_bg": "#FF00FF",  
"start_btn_fg": "black",  
"panel_btn_bg": "#222222",  
"window_border": "#FF00FF",  
"icon_theme": "grimos"  
}
```

1. Sauvegardez et redémarrez GrimOS.
2. Allez dans l'application **Paramètres**, et sélectionnez "Cyberpunk" dans le menu déroulant.
3. Cliquez sur "Appliquer".

Chapitre 8 : Repenser la sécurité :

L'approche Mono-Utilisateur

La quasi-totalité des systèmes d'exploitation modernes, et plus particulièrement ceux basés sur Unix ou Linux, ont été conçus dès leur origine pour répondre aux besoins des entreprises et des universités. L'architecture de base suppose qu'un même ordinateur sera partagé par des dizaines d'utilisateurs simultanés, souvent connectés à distance, qu'il faut cloisonner pour éviter qu'ils n'espionnent ou ne détruisent le travail de leurs voisins.

C'est cette architecture qui explique pourquoi, sur un système Linux classique, on vous demande sans cesse votre mot de passe administrateur pour monter une simple clé USB, changer l'heure du système ou vous connecter à un réseau Wi-Fi.

Mais à la maison, sur un vieil ordinateur portable personnel, cette approche est-elle vraiment justifiée ? GrimOS a fait le choix radical de répondre par la négative.

1. Un ordinateur, Un humain

GrimOS assume pleinement son statut de système "Mono-Utilisateur". Il part du principe que l'humain assis physiquement devant le clavier est le seul et unique maître légitime de la machine.

Puisque l'utilisateur est le seul maître, il n'y a aucune raison de le soupçonner ou de l'enfermer dans une prison logicielle. GrimOS lève toutes les barrières ergonomiques liées à la sécurité multi-utilisateurs pour offrir une expérience fluide et sans friction.

2. Le secret : L'automatisation du Sudo

Sur Linux, la commande `sudo` ("Super User DO") permet à un utilisateur normal d'exécuter temporairement une tâche avec des privilèges d'administrateur, moyennant la saisie de son mot de passe.

Dans GrimOS, l'interface graphique a besoin d'utiliser `sudo` en permanence en arrière-plan : pour détecter le niveau de la batterie dans les fichiers matériels (`/sys/class`), pour monter un

disque dur (`mount`), ou pour analyser les ondes radio environnantes (`wpa_cli scan`).

Pour éviter d'inonder l'utilisateur de fenêtres de mot de passe, GrimOS utilise une méthode que les administrateurs système "puristes" qualifieraient d'hérésie : le mot de passe est enregistré en clair dans le fichier de configuration de l'utilisateur (`config/settings.json`).

```
{  
  "theme": "GrimOS",  
  "sudo_pwd": "mon_mot_de_passe_secret"  
}
```

Grâce à cela, lorsqu'une application Python a besoin d'agir sur le système, elle injecte simplement ce mot de passe de manière silencieuse (via `subprocess` avec un `input`). L'action est instantanée et totalement transparente pour l'utilisateur.

3. Les avantages ergonomiques

Cette approche débloque une ergonomie extraordinaire pour un système basé sur Linux :

- **Montage immédiat** : Une clé USB insérée apparaît instantanément dans l'explorateur, montée et prête à l'emploi.
- **Connexion transparente** : Le changement de réseau Wi-Fi s'effectue en un clic dans la barre des tâches.
- **Contrôle matériel** : Le réglage du volume sonore, l'extinction immédiate de la machine, ou l'accès aux sondes de température se font en temps réel, sans la lourdeur des outils de gestion de droits complexes comme `PolicyKit` .

4. Les compromis assumés (Les risques)

Une telle architecture vient évidemment avec un avertissement de sécurité fondamental.

Si un pirate parvient à exécuter un code malveillant sur votre machine, ou si quelqu'un vole votre ordinateur, il lui suffit de lire le fichier `settings.json` pour obtenir votre mot de passe et prendre le contrôle total du système.

GrimOS n'est donc pas conçu pour être utilisé comme un serveur exposé sur internet, ni dans un bureau ouvert où n'importe qui peut s'asseoir à votre clavier. C'est un

système profondément intime et personnel, un laboratoire privé.

Cependant, rien n'empêche un utilisateur averti d'améliorer cette sécurité. Dans l'esprit du "Faites-le vous-même" de GrimOS, un développeur pourrait tout à fait modifier le système pour remplacer ce stockage en clair par l'utilisation d'un *Keyring* (trousseau de clés chiffré), prouvant une fois de plus que dans GrimOS, le seul maître à bord, c'est vous.

Chapitre 9 : L'Avenir de GrimOS (Kiosques, Raspberry Pi et Conclusion)



Au fil de ces chapitres, nous avons exploré les entrailles de GrimOS pour comprendre comment il remplace les lourds mécanismes de l'informatique moderne par des concepts simples, lisibles et hautement optimisés.

Mais voir GrimOS uniquement comme un "ordinateur de bureau" pour surfer sur le web et écrire du texte serait une erreur. Sa légèreté inouïe en fait une arme redoutable pour d'autres domaines de l'informatique.

1. Au-delà du Bureau : Les Kiosques et Bornes d'arcade

Avez-vous déjà vu un panneau publicitaire numérique planter et afficher un menu Démarrer de Windows, ou un distributeur de billets bloqué sur une mise à jour système ? Ces erreurs surviennent parce que l'on utilise des systèmes d'exploitation généralistes et très lourds pour des tâches qui nécessitent d'afficher une seule application en plein écran.

L'architecture de GrimOS est parfaite pour le "Mode Kiosque" (Single-purpose machine). Puisque l'affichage graphique n'est qu'un simple script Python (`main.py`), un développeur peut, en quelques minutes :

- Supprimer la barre des tâches de `core/desktop.py` .
- Forcer le lancement de l'application "Visionneuse" ou "Navigateur Web" en plein écran, sans bouton pour la fermer.

Le résultat ? Vous obtenez un système d'exploitation impénétrable, ultra-stable, qui démarre en quelques secondes sur une machine d'une puissance dérisoire, idéal pour créer une borne d'arcade de rétro-gaming, un tableau de bord domotique mural, ou un écran d'affichage dynamique.

2. Le mariage parfait avec le Raspberry Pi

Le Raspberry Pi est le nano-ordinateur le plus célèbre au monde, prisé par les inventeurs pour ses "broches GPIO", ces petites tiges métalliques qui permettent à l'ordinateur de communiquer avec le monde réel (allumer des LED, lire des capteurs de température, contrôler des moteurs).

Généralement, pour contrôler ces broches, les utilisateurs écrivent des scripts... en Python ! Puisque GrimOS est **intégralement** écrit en Python, la frontière entre le matériel et l'interface graphique disparaît totalement.

Vous pourriez créer une application GrimOS avec un magnifique bouton vert dessiné par Tkinter, qui, lorsqu'il est cliqué, allume instantanément la lumière de votre salon, sans avoir besoin d'aucun serveur intermédiaire ni d'aucune API complexe. GrimOS est le système d'exploitation rêvé pour les hackers du quotidien et les passionnés d'électronique.

Conclusion générale

Ce livre touche à sa fin. En parcourant ces chapitres, vous n'avez pas seulement appris comment installer, utiliser ou développer sur GrimOS.

Vous avez découvert une philosophie technique à contre-courant de l'industrie : une approche où la complexité artificielle est remplacée par la lisibilité, où la puissance brute s'efface devant l'optimisation intelligente, et où l'utilisateur n'est plus un simple consommateur pris en otage, mais redevient le maître absolu et l'artisan de sa machine.

Que vous utilisiez GrimOS pour ressusciter un vieil ordinateur portable oublié, pour construire un tableau de bord domotique ou simplement comme un formidable laboratoire pour apprendre à programmer, une chose est sûre : l'informatique est de nouveau entre vos mains.

Bienvenue dans GrimOS.

Annexe : Comment nous avons développé GrimOS

Ce projet a été développé de manière innovante avec l'assistance d'un agent IA de codage nommé **Google Antigravity**. L'approche consistait à concevoir et générer le système sur une machine locale de développement puissante, puis à déployer, tester et itérer sur un vieil ordinateur via une connexion SSH.

Le principe fondateur de GrimOS est de s'exécuter sur du matériel ancien ou très modeste. Cependant, écrire du code, tester, et utiliser des outils modernes (comme des assistants IA ou de gros IDE) nécessite de la puissance. Alors comment développer confortablement pour GrimOS ?

La solution est de séparer l'environnement d'exécution de l'environnement de développement.



Le flux de développement idéal : on code sur un ordinateur moderne (à gauche) et on envoie instantanément le résultat sur la machine cible (à droite).

1. La philosophie du développement à distance

Plutôt que de brancher un clavier et un écran sur le vieux ordinateur (la "machine cible") et de s'épuiser sur un petit éditeur de texte lent, on utilise un ordinateur moderne (la "machine de développement").

Cela offre des avantages considérables :

- Vous pouvez utiliser votre éditeur de code favori (VSCode, PyCharm, etc.).
- Vous pouvez vous faire assister par des intelligences artificielles.

- Le vieil ordinateur est laissé 100% libre pour faire tourner GrimOS sans être ralenti par l'environnement de programmation.

2. Préparation des machines

A. Sur la machine cible (L'antique ordinateur)

Sur cette machine, l'installation doit être la plus pure possible.

1. **Système de base** : Installez une version de **Debian** sans aucun environnement de bureau (désélectionnez GNOME/XFCE à l'installation).
2. **Outils requis** : Une fois installé, vous aurez besoin d'un serveur d'affichage (X11), de Python, et surtout d'un serveur SSH pour accepter les connexions à distance :

Terminal :

```
sudo apt-get update  
sudo apt-get install xorg python3 python3-tk openssh-server
```

B. Sur la machine de développement (L'ordinateur moderne)

Sur votre ordinateur habituel (Windows, macOS, ou un Linux moderne), vous n'avez besoin que de deux choses :

1. Votre éditeur de code.
2. Un client SSH (inclus nativement dans le terminal de tous les systèmes modernes).

3. Sécuriser et faciliter la connexion (Clé SSH)

Pour éviter de devoir taper le mot de passe de la machine cible à chaque fois que vous y envoyez du code, nous allons créer une "clé numérique" (une clé SSH).

Étape 1 : Générer la clé sur l'ordinateur moderne Ouvrez votre terminal et tapez :

Terminal :


```
ssh-keygen -t ed25519 -C "mon_ordinateur_dev"
```

Appuyez sur `Entrée` à toutes les questions pour accepter les choix par défaut.

Étape 2 : Transférer la clé sur la machine cible Nous allons dire à l'antique ordinateur de faire confiance à cette clé. Remplacez `utilisateur` par votre nom d'utilisateur sur Debian, et `192.168.1.xxx` par l'adresse IP locale de la machine cible :

 **Terminal :**

```
ssh-copy-id utilisateur@192.168.1.xxx
```

(Il vous demandera le mot de passe une toute dernière fois pour enregistrer la clé).

Désormais, votre ordinateur moderne a un accès administrateur direct et transparent à la machine cible !

4. Le Flux de travail quotidien (Workflow)

Une fois les ordinateurs reliés, voici comment se déroule la création d'une application pour GrimOS.

1. Coder (En local)

Vous ouvrez le dossier source de GrimOS sur votre ordinateur moderne. Vous y ajoutez un fichier, par exemple `apps/mon_app/app.py`.

2. Transférer (SCP)

Dès que vous sauvegardez votre code, vous pouvez envoyer l'intégralité du dossier `grimos/` vers la machine cible grâce à la commande de copie sécurisée (`scp`) :

 **Terminal :**

```
scp -r ./grimos_build/* utilisateur@192.168.1.xxx:/home/utilisateur/grimos/
```

Grâce à la clé SSH configurée plus tôt, le transfert est instantané et invisible.

3. Exécuter et Déboguer (SSH)

Pour tester votre code, pas besoin de changer de clavier ! Depuis le terminal de votre ordinateur moderne, vous lancez la commande d'exécution à distance :

 **Terminal :**

```
ssh utilisateur@192.168.1.xxx "cd /home/utilisateur/grimos && DISPLAY=:0 python3 main.py"
```

L'interface graphique de GrimOS s'affichera immédiatement sur l'écran branché au vieil ordinateur, et si votre code contient des erreurs (bugs), **les messages d'erreurs s'afficheront directement dans le terminal de votre ordinateur moderne.**

C'est ce qui rend le développement de GrimOS si agréable : une boucle de rétroaction instantanée alliée au confort du matériel moderne !

Annexe : Noyau - app_manager.py

Rôle et utilité

Le fichier `core/app_manager.py` est le "Chef d'Orchestre" des logiciels de GrimOS. Puisqu'il n'y a pas de gestionnaire de fenêtres Unix classique, c'est ce script qui est responsable de lancer les applications, de les confiner dans de fausses fenêtres, et de les fermer. Il lit le registre (`config/applications.json`) pour savoir quelles applications existent.

Implémentation technique

- **Import dynamique** : Plutôt que d'importer toutes les applications au démarrage (ce qui saturerait la RAM), le gestionnaire utilise la fonction Python `importlib.import_module()`. L'application n'est chargée en mémoire que lorsque l'utilisateur clique sur son icône.
- **Le conteneur `Window`** : Lorsqu'une application est lancée, l'`AppManager` demande d'abord à `window.py` de créer un cadre vide (la fausse fenêtre). Ensuite, il appelle la fonction obligatoire `start(fenetre_interne)` du module de l'application en lui passant ce cadre.
- **Isolation des plantages** : L'exécution se fait dans un bloc `try/except`. Si le code d'une application (par exemple l'Éditeur) contient une erreur fatale, le crash est intercepté par l'`AppManager`. Au lieu de faire planter tout GrimOS, le système affiche simplement une petite boîte de dialogue d'erreur et détruit la fenêtre fautive.

Pistes de modification

- **Mode "Plein écran automatique"** : Si l'on souhaitait créer un système type "Tablette", on pourrait modifier `AppManager` pour qu'il force les fenêtres à prendre la taille maximale de l'écran dès leur instanciation.
- **Limitation des instances** : Actuellement, on peut ouvrir 10 fois l'Explorateur. Un développeur pourrait ajouter un dictionnaire des processus actifs dans ce fichier pour empêcher l'ouverture multiple d'une même application.

Code Source

```

import importlib
import traceback
import json
import os
import tkinter as tk
from tkinter import messagebox
from core.window import Window

class AppManager:
    def __init__(self, desktop):
        self.desktop = desktop
        self.active_windows = []
        self.session_file = os.path.join(os.path.dirname(__file__), '..', 'data', 'session.json')

    def launch_app(self, app_config, geometry=None, is_maximized=False, **kwargs):
        module_name = app_config.get("module")
        app_name = app_config.get("name", "Application")

        try:
            module = importlib.import_module(module_name)
            importlib.reload(module)

            if geometry:
                win = Window(self.desktop.desktop_frame, desktop=self.desktop, app_config=app_config,
                             title=app_name,
                             x=geometry["x"], y=geometry["y"],
                             width=geometry["width"], height=geometry["height"],
                             is_maximized=is_maximized, filepath=kwargs.get("filepath"))
            else:
                w = app_config.get("width", 400)
                h = app_config.get("height", 300)
                x = app_config.get("x", 50)
                y = app_config.get("y", 50)
                win = Window(self.desktop.desktop_frame, desktop=self.desktop, app_config=app_config,
                             title=app_name, width=w, height=h, x=x, y=y, filepath=kwargs.get("filepath"))

            self.active_windows.append(win)

            if hasattr(module, 'start'):
                module.start(win.content, app_manager=self, **kwargs)
                win.after(100, win.lift)
            else:
                raise AttributeError(f"Le module {module_name} n'a pas de fonction 'start(window, ...)'.")

        except Exception as e:
            if 'win' in locals() and win.wininfo_exists():
                win.destroy()
            err_msg = traceback.format_exc()
            self.show_error(app_name, err_msg)

    def show_error(self, app_name, error_traceback):
        messagebox.showerror(f"Erreur application: {app_name}", f"Une erreur est survenue : \n\n{error_traceback}")

    def save_session(self):
        session_data = []
        for win in self.active_windows:
            win.update_saved_geometry()
            session_data.append({
                "app_config": win.app_config,
                "geometry": win.saved_geometry,
                "is_maximized": win.is_maximized
            })

        try:
            os.makedirs(os.path.dirname(self.session_file), exist_ok=True)
            with open(self.session_file, 'w', encoding='utf-8') as f:
                json.dump(session_data, f)
        except Exception as e:
            print(f"Erreur sauvegarde session: {e}")

```

```
def restore_session(self):
    if not os.path.exists(self.session_file):
        return

    try:
        with open(self.session_file, 'r', encoding='utf-8') as f:
            session_data = json.load(f)

        for data in session_data:
            geom = data.get("geometry")
            if geom and (geom.get("width", 0) <= 10 or geom.get("height", 0) <= 10):
                geom = None
            self.launch_app(data["app_config"], geom, data.get("is_maximized", False))
    except Exception as e:
        print(f"Erreur restauration session: {e}")
```

Annexe : Noyau - audio.py

Rôle et utilité

Le script `core/audio.py` agit comme un traducteur entre l'interface Python de GrimOS et le serveur de son natif de Linux (généralement ALSA via les commandes `amixer`). Il permet à la barre des tâches de lire le volume actuel et de le modifier.

Implémentation technique

- **Utilisation de `subprocess`** : Le fichier ne s'appuie sur aucune bibliothèque externe complexe (comme *PyAudio* ou *PulseAudio-libs*). Il exécute directement des commandes shell avec `subprocess.run()`.
- **Lecture du volume** : Pour connaître le volume actuel, il lance `amixer sget Master`. Il découpe ensuite la réponse texte (parsing) pour trouver une valeur en pourcentage (ex: `[50%]`) et l'état de la sourdine (`[on]` ou `[off]`).
- **Modification du volume** : Lorsqu'on ajuste la jauge, il convertit la valeur en commande `amixer sset Master X%`.
- **Mode muet (Mute)** : Il propose une fonction de bascule (`toggle_mute`) qui utilise la commande `amixer sset Master toggle`.

Pistes de modification

- **Support de PulseAudio/PipeWire** : Si le système sous-jacent évoluait vers un serveur de son plus moderne, c'est ce fichier qu'il faudrait modifier pour utiliser `pactl` ou `wpctl` au lieu d'`amixer`.
- **Sélecteur de sortie** : Un développeur pourrait enrichir ce module pour lister les différentes cartes sons branchées (Haut-parleurs, Casque Bluetooth) via la commande `aplay -l` et permettre à l'utilisateur de choisir sa sortie audio depuis la barre des tâches.

Code Source

```

import subprocess
import re
import threading

class AudioManager:
    def __init__(self, desktop):
        self.desktop = desktop
        self.control_name = self._find_master_control()

    def _find_master_control(self):
        """Tente de trouver le contrôle principal ALSA (Master, PCM, ou autre)."""
        try:
            # Chercher le contrôle 'Master' en premier
            res = subprocess.run(['amixer', 'sget', 'Master'], capture_output=True, text=True)
            if res.returncode == 0:
                return 'Master'

            # Sinon, lister tous les contrôles scontrols
            res = subprocess.run(['amixer', 'scontrols'], capture_output=True, text=True)
            if "PCM" in res.stdout:
                return "PCM"

            # Prendre le premier dispo
            match = re.search(r"Simple mixer control '([^']+)'", res.stdout)
            if match:
                return match.group(1)
        except Exception:
            pass
        return 'Master' # Valeur par défaut de secours

    def get_volume(self):
        try:
            res = subprocess.run(['amixer', 'sget', self.control_name], capture_output=True, text=True)
            if res.returncode == 0:
                # Chercher par ex: [50%]
                match = re.search(r"[(\d+)\%]", res.stdout)
                if match:
                    return int(match.group(1))
        except Exception:
            pass
        return 0

    def set_volume(self, level):
        try:
            subprocess.run(['amixer', 'sset', self.control_name, f'{level}%', 'unmute'],
capture_output=True)
            # Démuter spécifiquement les haut-parleurs au cas où
            subprocess.run(['amixer', 'sset', 'Speaker', 'unmute'], capture_output=True)
        except Exception:
            pass

    def test_audio(self):
        # speaker-test est bloquant, on le lance dans un thread
        def run_test():
            try:
                # Test avec un son wav natif (court et reconnaissable : "Front Center")
                # Si aplay ne trouve pas le fichier, speaker-test fera un bruit rose ou un bip.
                res = subprocess.run(['aplay', '/usr/share/sounds/alsa/Front_Center.wav'],
capture_output=True)
                if res.returncode != 0:
                    # Fallback sur speaker-test
                    subprocess.run(['speaker-test', '-t', 'sine', '-f', '440', '-c', '1', '-l', '1'],
timeout=2, capture_output=True)
            except Exception:
                pass

        threading.Thread(target=run_test, daemon=True).start()

```

Annexe : Noyau - config.py

Rôle et utilité

Dans un système d'exploitation classique, les préférences utilisateurs sont souvent éparpillées dans de nombreux fichiers obscurs ou dans une "base de registre" complexe. Dans GrimOS, tout est centralisé. Le script `core/config.py` est l'unique responsable de la lecture et de la sauvegarde des paramètres globaux de l'ordinateur.

Implémentation technique

- **Le fichier JSON** : Ce script agit comme une passerelle d'accès vers le fichier `config/settings.json`. Ce format a été choisi car il est naturellement supporté par Python, facile à lire pour un humain, et rapide à traiter.
- **Valeurs par défaut** : Si le fichier `settings.json` est supprimé ou corrompu, `config.py` est suffisamment robuste pour le détecter. Il possède un dictionnaire statique en mémoire avec des valeurs par défaut (thème de base, fond d'écran uni) qu'il utilisera pour assurer le démarrage de la machine, puis recréera le fichier manquant.
- **Fonctions d'accès** : Il expose des fonctions simples comme `get_setting(key)` et `set_setting(key, value)`. Chaque appel à `set_setting` écrit de façon synchrone sur le disque pour garantir que les préférences ne sont pas perdues en cas de coupure de courant.

Pistes de modification

- **Chiffrement des données sensibles** : Actuellement, `settings.json` stocke le mot de passe administrateur en clair pour la fonction d'Autologin. Une modification majeure de sécurité consisterait à implémenter un module de chiffrement (comme `cryptography.fernet`) dans `config.py` pour chiffrer ce mot de passe.
- **Profils utilisateurs** : Si GrimOS devenait multi-utilisateurs, il faudrait modifier ce fichier pour qu'il charge non pas `config/settings.json`, mais `~/.config/grimos_settings.json` spécifique à chaque compte Linux.

Code Source

```
import json
import os

def load_settings():
    path = os.path.join(os.path.dirname(__file__), '..', 'config', 'settings.json')
    try:
        with open(path, 'r', encoding='utf-8') as f:
            return json.load(f)
    except Exception as e:
        print(f"Erreur chargement settings: {e}")
        return {
            "resolution": "1024x768",
            "background": "gray",
            "fullscreen": False
        }


def load_applications():
    path = os.path.join(os.path.dirname(__file__), '..', 'config', 'applications.json')
    try:
        with open(path, 'r', encoding='utf-8') as f:
            return json.load(f)
    except Exception as e:
        print(f"Erreur chargement applications: {e}")
        return []
```

Annexe : Noyau - desktop.py

Rôle et utilité

Si `main.py` allume la lumière, `desktop.py` est l'âme du système. Ce fichier est le composant le plus volumineux et le plus critique de GrimOS. Il dessine l'environnement visuel complet (Bureau, Barre des tâches, Menu Démarrer) et gère les processus vitaux en arrière-plan.

Implémentation technique

- **Structure Tkinter** : La classe `Desktop` instancie de multiples conteneurs (`tk.Frame`). Le fond de l'écran est un grand `Canvas` (pour potentiellement y dessiner des icônes), tandis que le bas de l'écran est réservé à la barre des tâches.
- **Le Menu Démarrer** : Il est généré dynamiquement. Le script lit la liste des applications, regroupe celles ayant la même `category`, et construit des sous-menus Tkinter à la volée.
- **La boucle asynchrone (The heartbeat)** : La fonction la plus importante est `check_power()` OU `update_status()`. Elle utilise la méthode `window.after(1000, fonction)` de Tkinter pour s'exécuter toutes les secondes. C'est cette boucle qui lit les fichiers `/sys/class/power_supply` pour mettre à jour la batterie, lit `/proc/stat` pour calculer la charge CPU, et appelle `lsblk` pour détecter l'insertion d'une clé USB.
- **Le Montage USB automatique** : Lorsqu'un disque de type "Amovible" sans point de montage est détecté par `lsblk -J`, `desktop.py` déclenche la commande `sudo mount` silencieusement et fait apparaître le bouton d'éjection (.

Pistes de modification

- **Glisser-déposer (Drag & Drop)** : Le bureau actuel est un `Canvas` inerte. Pour le rendre pleinement interactif, on pourrait y ajouter la création de raccourcis (`tk.Label` avec images) dotés de la fonctionnalité "cliquer-glisser" en écoutant les événements `<Button-1>` et `<B1-Motion>`.
- **Notifications système** : On pourrait ajouter une file d'attente de messages dans cette boucle asynchrone, qui déclencherait l'apparition de "Toast notifications" (petits

rectangles de couleur) en bas à droite de l'écran en cas de perte Wi-Fi ou de batterie faible.

Code Source

```
import tkinter as tk
from tkinter import simpledialog
import subprocess
from time import strftime
import sys
import os

from core.app_manager import AppManager
from core.config import load_applications

class ToolTip:
    def __init__(self, widget, text):
        self.widget = widget
        self.text = text
        self.tooltip_window = None
        self.widget.bind("<Enter>", self.show_tooltip)
        self.widget.bind("<Leave>", self.hide_tooltip)

    def show_tooltip(self, event=None):
        if self.tooltip_window or not self.text:
            return
        x = self.widget.winfo_rootx()
        y = self.widget.winfo_rooty() - 25
        self.tooltip_window = tw = tk.Toplevel(self.widget)
        tw.wm_overrideredirect(True)
        tw.wm_geometry(f"+{x}+{y}")
        label = tk.Label(tw, text=self.text, justify='left',
                        background="#ffffe0", relief='solid', borderwidth=1,
                        font=("Arial", "9", "normal"))
        label.pack(ipadx=2, ipady=1)

    def hide_tooltip(self, event=None):
        tw = self.tooltip_window
        self.tooltip_window = None
        if tw:
            tw.destroy()

class Desktop(tk.Frame):
    def __init__(self, parent, settings):
        super().__init__(parent)
        self.parent = parent
        self.settings = settings
        self.pack(fill="both", expand=True)

        self.app_manager = AppManager(self)
        self.apps = load_applications()

        self.toasts = []

        # Desktop area
        import sys, os
        sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
        from core.theme import THEMES
        theme_name = self.settings.get("theme", "GrimOS")
        self.theme_data = THEMES.get(theme_name, THEMES["GrimOS"])
        bg_color = self.theme_data.get("desktop_bg", "gray")

        # Taskbar
        self.taskbar = tk.Frame(self, height=40, bg=self.theme_data.get("taskbar_bg", "#333"))
        self.taskbar.pack(side="bottom", fill="x")
```

```

self.taskbar.pack_propagate(False)

self.desktop_frame = tk.Frame(self, bg=bg_color)
self.desktop_frame.pack(fill="both", expand=True)

self.load_desktop_icons()

# Load icons
self.icons = {}
icon_dir = os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))), "icons")
icon_theme = self.theme_data.get("icon_theme", "grimos")
icon_theme_dir = os.path.join(icon_dir, "themes", icon_theme)

for iname in ['menu_poweroff', 'menu_reboot', 'menu_logoff', 'btn_apply', 'btn_refresh',
'btn_kill', 'btn_start', 'btn_wifi', 'btn_usb', 'btn_help', 'menu_grimoire', 'btn_audio', 'btn_trash']:
    ipath = os.path.join(icon_theme_dir, iname + ".png")
    if not os.path.exists(ipath):
        ipath = os.path.join(icon_dir, iname + ".png")
    if os.path.exists(ipath):
        self.icons[iname] = tk.PhotoImage(file=ipath)

# Load app icons
self.app_icons = {}
for app in self.apps:
    icon_path = app.get("icon")
    if icon_path:
        full_path = os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))),
icon_path)
        if os.path.exists(full_path):
            self.app_icons[app.get("module")] = tk.PhotoImage(file=full_path)

# Start button
self.start_btn = tk.Button(self.taskbar, image=self.icons.get("btn_start"),
                            bg=self.theme_data.get("start_btn_bg", "blue"),
                            fg=self.theme_data.get("start_btn_fg", "white"),
                            command=self.toggle_start_menu)
self.start_btn.pack(side="left", padx=5, pady=5)
ToolTip(self.start_btn, "Démarrer")

# Taskbar window buttons container
self.taskbar_windows = tk.Frame(self.taskbar, bg=self.theme_data.get("taskbar_bg", "#333"))
self.taskbar_windows.pack(side="left", fill="both", expand=True, padx=5)
self.taskbar_btns = {}

# Clock
self.clock_lbl = tk.Label(self.taskbar, bg=self.theme_data.get("taskbar_bg", "#333"),
fg=self.theme_data.get("taskbar_fg", "white"), font=("Arial", 10))
self.clock_lbl.pack(side="right", padx=10)
self.update_clock()

# System Monitor
self.sysmon_canvas = tk.Canvas(self.taskbar, height=24, bg=self.theme_data.get("taskbar_bg",
"#333"), highlightthickness=0)
self.sysmon_canvas.pack(side="right", padx=10, pady=3)

self.last_cpu_total = 0
self.last_cpu_idle = 0
self.update_sysmon()

# Keyboard Layout
self.kb_layout = "fr"
self.kb_btn = tk.Button(self.taskbar, text="FR", bg=self.theme_data.get("panel_btn_bg", "#555"),
fg=self.theme_data.get("taskbar_fg", "white"), font=("Arial", 10, "bold"), command=self.toggle_keyboard)
self.kb_btn.pack(side="right", padx=5, pady=5)
self.apply_keyboard_layout()

# Wi-Fi Module
from core.wifi import WifiManager
self.wifi_manager = WifiManager(self)
self.sudo_pwd = None

```

```

        self.wifi_btn = tk.Button(self.taskbar, image=self.icons.get("btn_wifi"),
bg=self.theme_data.get("panel_btn_bg", "#555"), fg=self.theme_data.get("taskbar_fg", "white"),
command=self.show_wifi_menu)
        self.wifi_btn.pack(side="right", padx=5, pady=5)
        Tooltip(self.wifi_btn, "Wi-Fi")

    # USB Module
    self.usb_btn = tk.Button(self.taskbar, image=self.icons.get("btn_usb"),
bg=self.theme_data.get("panel_btn_bg", "#555"), fg=self.theme_data.get("taskbar_fg", "white"),
command=self.show_usb_menu)
    self.usb_btn.pack(side="right", padx=5, pady=5)
    Tooltip(self.usb_btn, "Périphériques USB")
    self.known_unmounted_usbs = set()
    self.after(3000, self.update_usb)

    # Audio Module
    from core.audio import AudioManager
    self.audio_manager = AudioManager(self)
    self.audio_btn = tk.Button(self.taskbar, image=self.icons.get("btn_audio"),
bg=self.theme_data.get("panel_btn_bg", "#555"), fg=self.theme_data.get("taskbar_fg", "white"),
command=self.show_audio_menu)
    self.audio_btn.pack(side="right", padx=5, pady=5)
    Tooltip(self.audio_btn, "Contrôle du volume")

    # Start Menu (hidden by default)
    self.start_menu = tk.Frame(self.desktop_frame, bg="white", relief="raised", bd=2)
    self.start_menu_visible = False
    self.build_start_menu()

    # Hide menu when clicking desktop
    self.desktop_frame.bind("<ButtonPress-1>", self.hide_start_menu)

    # Context menu (Right click)
    self.context_menu = tk.Menu(self, tearoff=0)
    self.context_menu.add_command(label="Paramètres", command=lambda:
self.app_manager.launch_app({"name": "Paramètres", "module": "apps.parametres.app"}))
    self.context_menu.add_separator()
    self.context_menu.add_command(label="Redémarrer GrimOS", command=self.shutdown_grimos)
    self.context_menu.add_separator()
    self.context_menu.add_command(label="Annuler", command=lambda: None)

    def show_context_menu(event):
        try:
            self.context_menu.tk_popup(event.x_root, event.y_root)
        finally:
            self.context_menu.grab_release()

    self.desktop_frame.bind("<Button-3>", show_context_menu)

    # Restore previous session if any
    self.app_manager.restore_session()

    # Start power management
    self.after(5000, self.check_power)

    # Start clipboard sync
    self.last_prim = ""
    self.after(1000, self.sync_clipboards)

    # Start IPC notifications watcher
    self.after(2000, self.check_notifications)

    def check_notifications(self):
        notify_file = "/tmp/grimos_notify"
        if os.path.exists(notify_file):
            try:
                with open(notify_file, "r", encoding="utf-8") as f:
                    msg = f.read().strip()
                os.remove(notify_file)
                if msg:
                    if msg.startswith("POPUP:"):
                        from tkinter import messagebox

```

```

        messagebox.showinfo("Message Système", msg[6:])
    else:
        self.show_toast(msg)
    except Exception:
        pass
    self.after(1000, self.check_notifications)

def sync_clipboards(self):
    try:
        prim = self.selection_get(selection="PRIMARY")
        if prim and prim != self.last_prim:
            self.clipboard_clear()
            self.clipboard_append(prim)
            self.last_prim = prim
    except:
        pass
    self.after(500, self.sync_clipboards)

def check_power(self):
    try:
        is_ac = True
        if os.path.exists('/sys/class/power_supply/AC/online'):
            with open('/sys/class/power_supply/AC/online', 'r') as f:
                val = f.read().strip()
                if val == '0':
                    is_ac = False
        elif os.path.exists('/sys/class/power_supply/ACAD/online'):
            with open('/sys/class/power_supply/ACAD/online', 'r') as f:
                val = f.read().strip()
                if val == '0':
                    is_ac = False

        if is_ac:
            subprocess.run(['xset', 's', 'off'], capture_output=True)
            subprocess.run(['xset', '-dpms'], capture_output=True)
        else:
            res = subprocess.run(['xprintidle'], capture_output=True, text=True)
            if res.returncode == 0:
                idle_ms = int(res.stdout.strip())
                if idle_ms > 180000: # 3 minutes
                    pwd = self.settings.get("sudo_pwd", "")
                    subprocess.run(['sudo', '-S', 'poweroff'], input=pwd+'\n', text=True,
capture_output=True)
            except Exception:
                pass
        self.after(10000, self.check_power)

def update_clock(self):
    time_string = strftime('%H:%M')
    self.clock_lbl.config(text=time_string)
    self.after(1000, self.update_clock)

def update_sysmon(self):
    try:
        with open('/proc/meminfo', 'r') as f:
            lines = f.readlines()
            mem = {}
            for line in lines:
                parts = line.split()
                if len(parts) >= 2:
                    mem[parts[0].strip(':')] = int(parts[1])
            ram_pct = int(100 * (1 - mem.get('MemAvailable', 0) / mem.get('MemTotal',
1)))

            swap_tot = mem.get('SwapTotal', 0)
            swap_pct = int(100 * (1 - mem.get('SwapFree', 0) / swap_tot)) if swap_tot > 0 else 0

            with open('/proc/stat', 'r') as f:
                cpu_line = f.readline().split()
                cpu_idle = int(cpu_line[4]) + int(cpu_line[5])
                cpu_total = sum(int(x) for x in cpu_line[1:8])
                cpu_diff = cpu_total - self.last_cpu_total
                idle_diff = cpu_idle - self.last_cpu_idle

```

```

cpu_pct = int(100 * (1 - idle_diff / cpu_diff)) if cpu_diff > 0 else 0
self.last_cpu_total = cpu_total
self.last_cpu_idle = cpu_idle

graph_width = self.settings.get("cpu_graph_width", 10)
if not hasattr(self, 'cpu_history'):
    self.cpu_history = [0] * graph_width
    self.ram_history = [0] * graph_width
    self.swap_history = [0] * graph_width

while len(self.cpu_history) > graph_width:
    self.cpu_history.pop(0)
    self.ram_history.pop(0)
    self.swap_history.pop(0)
while len(self.cpu_history) < graph_width:
    self.cpu_history.insert(0, 0)
    self.ram_history.insert(0, 0)
    self.swap_history.insert(0, 0)

self.cpu_history.pop(0)
self.cpu_history.append(cpu_pct)
self.ram_history.pop(0)
self.ram_history.append(ram_pct)
self.swap_history.pop(0)
self.swap_history.append(swap_pct)

temp = 0
if os.path.exists('/sys/class/thermal/thermal_zone0/temp'):
    with open('/sys/class/thermal/thermal_zone0/temp', 'r') as f:
        temp = int(int(f.read().strip()) / 1000)

cap = None
status = ""
try:
    for bat_dir in os.listdir('/sys/class/power_supply'):
        if bat_dir.startswith('BAT') or bat_dir.startswith('macsmc-battery'):
            with open(f'/sys/class/power_supply/{bat_dir}/capacity', 'r') as f:
                cap = int(f.read().strip())

            with open(f'/sys/class/power_supply/{bat_dir}/status', 'r') as f:
                stat_txt = f.read().strip()
                if stat_txt == "Charging": status = ">"
                elif stat_txt == "Discharging": status = "🔋"
                elif stat_txt == "Full": status = "🔋"
            break
except Exception:
    pass

self.sysmon_canvas.delete("all")
bar_w = 3
graph_px_width = graph_width * bar_w
graph_h = 20
pad_y = 2
current_x = 5
fg_color = self.theme_data.get("taskbar_fg", "white")

self.sysmon_canvas.create_rectangle(current_x, pad_y, current_x + graph_px_width, pad_y +
graph_h, outline="black")
for i, val in enumerate(self.cpu_history):
    h = int((val / 100) * graph_h)
    if h > 0:
        self.sysmon_canvas.create_rectangle(current_x + i * bar_w, pad_y + graph_h - h,
current_x + (i + 1) * bar_w, pad_y + graph_h, fill="green", outline="")

    current_x += graph_px_width + 5
self.sysmon_canvas.create_text(current_x, pad_y + graph_h//2, text=f"{temp:02d}°C",
fill=fg_color, anchor="w", font=("Arial", 9))
current_x += 35

self.sysmon_canvas.create_rectangle(current_x, pad_y, current_x + graph_px_width, pad_y +
graph_h, outline="black")
for i, val in enumerate(self.ram_history):

```

```

        h = int((val / 100) * graph_h)
        if h > 0:
            self.sysmon_canvas.create_rectangle(current_x + i * bar_w, pad_y + graph_h - h,
current_x + (i + 1) * bar_w, pad_y + graph_h, fill="blue", outline="")

        current_x += graph_px_width + 5

        self.sysmon_canvas.create_rectangle(current_x, pad_y, current_x + graph_px_width, pad_y +
graph_h, outline="black")
        for i, val in enumerate(self.swap_history):
            h = int((val / 100) * graph_h)
            if h > 0:
                self.sysmon_canvas.create_rectangle(current_x + i * bar_w, pad_y + graph_h - h,
current_x + (i + 1) * bar_w, pad_y + graph_h, fill="red", outline="")

        current_x += graph_px_width + 10

        if cap is not None:
            bat_w = 20
            bat_h = 10
            bat_y = pad_y + (graph_h - bat_h) // 2
            self.sysmon_canvas.create_rectangle(current_x, bat_y, current_x + bat_w, bat_y + bat_h,
outline=fg_color)
            self.sysmon_canvas.create_rectangle(current_x + bat_w, bat_y + 3, current_x + bat_w + 2,
bat_y + bat_h - 3, fill=fg_color, outline="")

            fill_w = int((cap / 100) * (bat_w - 2))
            if fill_w > 0:
                self.sysmon_canvas.create_rectangle(current_x + 1, bat_y + 1, current_x + 1 + fill_w,
bat_y + bat_h - 1, fill="orange", outline="")

            current_x += bat_w + 5
            self.sysmon_canvas.create_text(current_x, pad_y + graph_h//2, text=f"{status} {cap}%",
fill=fg_color, anchor="w", font=("Arial", 9))
            current_x += 45

        self.sysmon_canvas.config(width=current_x + 5)
    except Exception:
        pass
    self.after(1000, self.update_sysmon)

    def update_usb(self):
        pwd = self.settings.get("sudo_pwd")
        if pwd:
            try:
                res = subprocess.run(['lsblk', '-J', '-o', 'NAME,RM,MOUNTPOINT'], text=True,
capture_output=True)
                import json
                data = json.loads(res.stdout)

                current_unmounted = set()

                for bd in data.get('blockdevices', []):
                    if bd.get('rm') in [True, '1', 1]:
                        children = bd.get('children', [])
                        if children:
                            for child in children:
                                if not child.get('mountpoint'):
                                    current_unmounted.add(child.get('name'))
                        else:
                            if not bd.get('mountpoint'):
                                current_unmounted.add(bd.get('name'))

                # Find new unmounted drives
                new_drives = current_unmounted - self.known_unmounted_usbs
                for d in new_drives:
                    mnt_dir = f"/tmp/usb_{d}"
                    os.makedirs(mnt_dir, exist_ok=True)
                    subprocess.run(['sudo', '-S', 'mount', f'/dev/{d}', mnt_dir], input=pwd+'\n',
text=True, capture_output=True)

                self.known_unmounted_usbs = current_unmounted

```

```

        except:
            pass
        self.after(3000, self.update_usb)

    def show_usb_menu(self):
        pwd = self.settings.get("sudo_pwd")
        if not pwd:
            from tkinter import messagebox
            messagebox.showerror("Erreur", "Mot de passe Sudo non configuré dans Paramètres.")
            return

        try:
            res = subprocess.run(['lsblk', '-J', '-o', 'NAME,RM,MOUNTPOINT,SIZE'], text=True,
capture_output=True)
            import json
            data = json.loads(res.stdout)

            mounted_devices = []
            for bd in data.get('blockdevices', []):
                if bd.get('rm') in [True, '1', 1]:
                    children = bd.get('children', [])
                    if children:
                        for child in children:
                            if child.get('mountpoint'):
                                mounted_devices.append(child)
                    else:
                        if bd.get('mountpoint'):
                            mounted_devices.append(bd)

            if not mounted_devices:
                from tkinter import messagebox
                messagebox.showinfo("USB", "Aucun périphérique USB monté actuellement.")
                return

            if hasattr(self, 'usb_popup') and self.usb_popup.winfo_exists():
                self.usb_popup.destroy()
                return

            self.usb_popup = tk.Toplevel(self)
            self.usb_popup.title("Éjecter USB")
            self.usb_popup.geometry("300x200")

            self.usb_popup.update_idletasks()
            x = self.winfo_width() - 310
            y = self.winfo_height() - 250
            self.usb_popup.geometry(f"+{x}+{y}")
            self.usb_popup.transient(self.winfo_toplevel())

            tk.Label(self.usb_popup, text="Cliquez pour éjecter :", font=("Arial", 10,
"bold")).pack(pady=5)

            for dev in mounted_devices:
                name = dev.get('name')
                mountpoint = dev.get('mountpoint')
                size = dev.get('size')

                def unmount(n=name, m=mountpoint):
                    r = subprocess.run(['sudo', '-S', 'umount', f'/dev/{n}'], input=pwd+'\n', text=True,
capture_output=True)
                    from tkinter import messagebox
                    if r.returncode == 0:
                        self.known_unmounted_usbs.add(n)
                        subprocess.run(['sudo', '-S', 'rmdir', m], input=pwd+'\n', text=True)
                        messagebox.showinfo("USB", f"Le périphérique {n} a été éjecté en toute
sécurité.")
                        if self.usb_popup.winfo_exists():
                            self.usb_popup.destroy()
                    else:
                        messagebox.showerror("Erreur", r.stderr)

            tk.Button(self.usb_popup, text=f"🗑 /dev/{name} ({size})\n{mountpoint}",
command=unmount).pack(fill="x", padx=5, pady=2)

```

```

except Exception as e:
    from tkinter import messagebox
    messagebox.showerror("Erreur", str(e))

def show_audio_menu(self):
    if hasattr(self, 'audio_popup') and self.audio_popup.winfo_exists():
        self.audio_popup.destroy()
        return

    self.audio_popup = tk.Toplevel(self)
    self.audio_popup.overrideredirect(True)
    self.audio_popup.configure(bg="#222", bd=1, relief="solid")

    # Positionnement au-dessus du bouton
    x = self.audio_btn.winfo_rootx()
    y = self.audio_btn.winfo_rooty() - 230
    self.audio_popup.geometry(f"120x220+{x-30}+{y}")

    # Titre
    tk.Label(self.audio_popup, text="Volume", font=("Arial", 10, "bold"), bg="#222",
fg="white").pack(pady=5)

    # Slider
    current_vol = self.audio_manager.get_volume()

    def on_volume_change(val):
        self.audio_manager.set_volume(int(val))

    vol_scale = tk.Scale(self.audio_popup, from_=100, to=0, orient="vertical", bg="#222", fg="white",
highlightthickness=0, command=on_volume_change)
    vol_scale.set(current_vol)
    vol_scale.pack(fill="y", expand=True)

    # Bouton de test
    btn_test = tk.Button(self.audio_popup, text="Tester le son", bg="#4CAF50", fg="white",
relief="flat", command=self.audio_manager.test_audio)
    btn_test.pack(pady=5, padx=10, fill="x")

    # Binding pour fermer quand on clique ailleurs
    def check_focus(e):
        if str(e.widget) not in str(self.audio_popup):
            self.audio_popup.destroy()

    # self.audio_popup.bind("<FocusOut>", lambda e: self.audio_popup.destroy())
    self.audio_popup.focus_set()

def show_wifi_menu(self):
    if not self.wifi_manager.iface:
        from tkinter import messagebox
        messagebox.showerror("Wi-Fi", "Aucune interface Wi-Fi détectée sur ce système.")
        return

    pwd = self.settings.get("sudo_pwd")
    if not pwd:
        from tkinter import messagebox
        messagebox.showerror("Wi-Fi", "Veuillez configurer votre mot de passe système (Sudo) dans
l'application Paramètres.")
        return

    if hasattr(self, 'wifi_popup') and self.wifi_popup.winfo_exists():
        self.wifi_popup.destroy()
        return

    self.wifi_popup = tk.Toplevel(self)
    self.wifi_popup.title("Réseaux Wi-Fi")
    self.wifi_popup.geometry("300x450")

    self.wifi_popup.update_idletasks()
    x = self.winfo_width() - 310
    y = self.winfo_height() - 500
    self.wifi_popup.geometry(f"+{x}+{y}")

```



```

self.wifi_popup.transient(self.winfo_toplevel())

ssid_actuel, ip_actuelle = self.wifi_manager.get_current_status(pwd)
if ssid_actuel:
    info_frame = tk.Frame(self.wifi_popup, bg="#e0f7fa", bd=1, relief="solid")
    info_frame.pack(fill="x", padx=5, pady=5)
    tk.Label(info_frame, text=f"Connecté : {ssid_actuel}", font=("Arial", 10, "bold"),
bg="#e0f7fa", fg="#006064").pack()
    if ip_actuelle:
        tk.Label(info_frame, text=f"IP : {ip_actuelle}", font=("Arial", 9), bg="#e0f7fa",
fg="#00838f").pack()

    lbl = tk.Label(self.wifi_popup, text="Recherche de réseaux en cours...", font=("Arial", 10,
"italic"))
    lbl.pack(pady=10)

    listbox = tk.Listbox(self.wifi_popup, font=("Arial", 10))
    listbox.pack(fill="both", expand=True, padx=5, pady=5)

    btn_connect = tk.Button(self.wifi_popup, text="Se connecter", state="disabled")
    btn_connect.pack(fill="x", padx=5, pady=5)

    networks_data = []

    def on_select(event):
        if listbox.curselection():
            btn_connect.config(state="normal")

    listbox.bind('<<ListboxSelect>>', on_select)

    def do_connect():
        sel = listbox.curselection()
        if not sel: return
        ssid = networks_data[sel[0]]['ssid']
        psk = simpledialog.askstring("Wi-Fi", f"Clé de sécurité pour '{ssid}' (vide si réseau ouvert)
:")

        if psk is not None:
            lbl.config(text="Connexion en cours...")
            def connect_thread():
                success, msg = self.wifi_manager.connect_network(pwd, ssid, psk)
                def callback():
                    from tkinter import messagebox
                    if success:
                        messagebox.showinfo("Wi-Fi", f"Connecté à {ssid}.")
                        self.wifi_popup.destroy()
                    else:
                        messagebox.showerror("Erreur", msg)
                        lbl.config(text="Erreur.")
                self.after(0, callback)
            import threading
            threading.Thread(target=connect_thread, daemon=True).start()

    btn_connect.config(command=do_connect)

    def scan_thread():
        nets = self.wifi_manager.scan_networks(pwd)
        def callback():
            if not self.wifi_popup.winfo_exists(): return
            lbl.config(text=f"{len(nets)} réseaux détectés.")
            listbox.delete(0, tk.END)
            networks_data.clear()
            for net in nets:
                networks_data.append(net)
                listbox.insert(tk.END, f"{net['ssid']} ({net['signal']} dBm)")
            self.after(0, callback)

    import threading
    threading.Thread(target=scan_thread, daemon=True).start()

    def toggle_keyboard(self):
        if self.kb_layout == "fr":
            self.kb_layout = "us"

```

```

        self.kb_btn.config(text="US")
    else:
        self.kb_layout = "fr"
        self.kb_btn.config(text="FR")
    self.apply_keyboard_layout()

def apply_keyboard_layout(self):
    try:
        subprocess.run(["setxkbmap", self.kb_layout])
    except Exception as e:
        print(f"Erreur clavier: {e}")

def build_start_menu(self):
    for widget in self.start_menu.winfo_children():
        widget.destroy()

    tk.Label(self.start_menu, text="GrimOS", bg="lightgray", font=("Arial", 10,
"bold")).pack(fill="x", pady=2)

    self.start_menu_container = tk.Frame(self.start_menu, bg="white")
    self.start_menu_container.pack(fill="both", expand=True)

    self.start_menu_canvas = tk.Canvas(self.start_menu_container, bg="white", highlightthickness=0)
    self.start_menu_scrollbar = tk.Scrollbar(self.start_menu_container, orient="vertical",
command=self.start_menu_canvas.yview)

    self.start_menu_content = tk.Frame(self.start_menu_canvas, bg="white")

    self.start_menu_content.bind(
        "<Configure>",
        lambda e: self.start_menu_canvas.configure(scrollregion=self.start_menu_canvas.bbox("all"))
    )

    self.start_menu_canvas.create_window((0, 0), window=self.start_menu_content, anchor="nw",
width=230)
    self.start_menu_canvas.configure(yscrollcommand=self.start_menu_scrollbar.set)

    def _on_mousewheel(event):
        if self.start_menu_visible:
            self.start_menu_canvas.yview_scroll(int(-1*(event.delta/120)), "units")
        self.start_menu.bind_all("<MouseWheel>", _on_mousewheel)
        self.start_menu.bind_all("<Button-4>", lambda e: self.start_menu_canvas.yview_scroll(-1, "units"))
    if self.start_menu_visible else None)
    self.start_menu.bind_all("<Button-5>", lambda e: self.start_menu_canvas.yview_scroll(1, "units"))
    if self.start_menu_visible else None)

    categories = {}
    for app in self.apps:
        cat = app.get("category", "Autres")
        if cat not in categories:
            categories[cat] = []
        categories[cat].append(app)

    for cat in sorted(categories.keys()):
        tk.Label(self.start_menu_content, text=f"— {cat} —", bg="white", fg="gray", font=("Arial",
9, "italic")).pack(fill="x", pady=(5,0))
        for app in categories[cat]:
            icon_img = self.app_icons.get(app.get("module"))
            btn = tk.Button(self.start_menu_content, text=" " + app.get("name"), image=icon_img,
compound="left", relief="flat", anchor="w",
command=lambda a=app: self.launch_from_menu(a))
            btn.pack(fill="x", padx=10, pady=1)

    tk.Frame(self.start_menu_content, height=2, bg="black").pack(fill="x", pady=2)

def generate_grimoire():
    self.hide_start_menu()
    import os
    from tkinter import messagebox
    grimoire_path = os.path.expanduser("~/Grimoire_GrimOS.md")

    content = "# Le Grimoire GrimOS\n\n"

```

```

        content += "Bienvenue dans GrimOS (Graphical Runtime Integrated Minimal Operating System).\n"
        content += "Ce document est généré automatiquement et compile la documentation du
système.\n\n"

        content += "## 1. Créer et intégrer une application\n"
        content += "Pour créer une application compatible GrimOS, créez un fichier `.py` contenant
:\n"
        content += "`python\nimport tkinter as tk\n"
        content += "def start(window, app_manager=None, **kwargs):\n"
        content += "    # 'window' est un tk.Frame pré-crée par GrimOS.\n"
        content += "    tk.Label(window, text='Mon Appli').pack()\n```\n"
        content += "***Note de compatibilité : ** Ne créez JAMAIS de `tk.Tk()` dans la fonction
`start()`, cela casserait l'affichage sans gestionnaire de fenêtres.\n\n"
        content += "Ensuite, pour ajouter l'application au Menu Démarrer, éditez le fichier
`config/applications.json` dans le dossier de GrimOS :\n"
        content += "```\njson\n{\n  \"name\": \"Mon App\",\n  \"module\": \"chemin.vers.le.module\",\n  \"category\": \"Autres\"\n}\n```\n"
        content += "Relancez la session, et votre application apparaîtra !\n\n"

        content += "## 2. Aides des Applications Intégrées\n"
        content += "### Éditeur de Code\n"
        content += "Conçu pour le développement natif. Utilisez le bouton 'Nouveau GUI' pour générer
un code complet (boilerplate) pour démarrer une application GrimOS, testable même en standalone via son
propre bloc `__main__`.\n\n"

        content += "### Explorateur\n"
        content += "Double-cliquez pour naviguer. Clic-droit pour renommer ou supprimer. La case
'Fichiers cachés' permet d'afficher les éléments systèmes. Le bouton `>_ Terminal` permet d'ouvrir une
invite de commande dans le dossier visité.\n\n"

        content += "### Terminal\n"
        content += "Environnement émulé. Commandes courantes supportées avec historique. La commande
`sudo` est volontairement bloquée par sécurité (utilisez l'application Xterm pour l'administration
système pure).\n"

        try:
            with open(grimoire_path, "w", encoding="utf-8") as f:
                f.write(content)
            self.app_manager.launch_app({"name": "Éditeur de Code", "module": "apps.editeur.app"},
filepath=grimoire_path)
        except Exception as e:
            messagebox.showerror("Erreur", f"Impossible de générer le Grimoire : {e}")

        btn_help = tk.Button(self.start_menu_content, text=" Générer le Grimoire",
image=self.icons.get('menu_grimoire'), compound="left", relief="flat", anchor="w", fg="blue",
command=generate_grimoire)
        btn_help.pack(fill="x", padx=5, pady=2)

        tk.Frame(self.start_menu_content, height=1, bg="lightgray").pack(fill="x", pady=2)

        btn_logoff = tk.Button(self.start_menu_content, text=" Fermer la session",
image=self.icons.get('menu_logoff'), compound="left", relief="flat", anchor="w", fg="black",
command=self.shutdown_grimos)
        btn_logoff.pack(fill="x", padx=5, pady=2)

        btn_reboot = tk.Button(self.start_menu_content, text=" Redémarrer",
image=self.icons.get('menu_reboot'), compound="left", relief="flat", anchor="w", fg="orange",
command=self.reboot_system)
        btn_reboot.pack(fill="x", padx=5, pady=2)

        btn_poweroff = tk.Button(self.start_menu_content, text=" Arrêter",
image=self.icons.get('menu_poweroff'), compound="left", relief="flat", anchor="w", fg="red",
command=self.poweroff_system)
        btn_poweroff.pack(fill="x", padx=5, pady=2)

        def toggle_start_menu(self):
            if self.start_menu_visible:
                self.hide_start_menu()
            else:
                self.start_menu_content.update_idletasks()
                req_h = self.start_menu_content.winfo_reqheight() + 30 # Label de titre et bordures
                max_h = self.desktop_frame.winfo_height() - 20

```

```

        if req_h > max_h:
            self.start_menu_scrollbar.pack(side="right", fill="y")
            self.start_menu_canvas.pack(side="left", fill="both", expand=True)
            final_h = max_h
        else:
            self.start_menu_scrollbar.pack_forget()
            self.start_menu_canvas.pack(side="left", fill="both", expand=True)
            final_h = req_h

        self.start_menu.place(x=5, y=self.desktop_frame.winfo_height() - final_h - 5, height=final_h,
width=250)
        self.start_menu.lift()
        self.start_menu_visible = True

    def hide_start_menu(self, event=None):
        if self.start_menu_visible:
            self.start_menu.place_forget()
            self.start_menu_visible = False

    def launch_from_menu(self, app):
        self.hide_start_menu()
        self.app_manager.launch_app(app)

    def register_window(self, win):
        if win not in self.taskbar_btns:
            icon_img = self.app_icons.get(win.app_config.get("module"))
            tooltip_text = win.filepath if hasattr(win, "filepath") and win.filepath else win.title
            btn = tk.Button(self.taskbar_windows, image=icon_img, bg="#444", fg="white",
                           command=lambda: self.toggle_window(win))
            btn.pack(side="left", padx=2, pady=5)
            self.taskbar_btns[win] = btn
            Tooltip(btn, tooltip_text)

    def update_taskbar_btn(self, win):
        if win in self.taskbar_btns:
            if not win.winfo_ismapped():
                # Fenêtre minimisée -> fond plus sombre ou différent pour l'indiquer
                self.taskbar_btns[win].config(bg="#888", relief="sunken")
            else:
                self.taskbar_btns[win].config(bg="#444", relief="raised")

    def toggle_window(self, win):
        if win.winfo_ismapped():
            win.minimize()
        else:
            win.restore()

    def remove_window(self, win):
        if win in self.taskbar_btns:
            self.taskbar_btns[win].destroy()
            del self.taskbar_btns[win]
        if win in self.app_manager.active_windows:
            self.app_manager.active_windows.remove(win)

    def save_and_quit(self):
        self.app_manager.save_session()
        self.parent.destroy()

    def shutdown_grimos(self):
        self.save_and_quit()

    def reboot_system(self):
        from tkinter import messagebox
        pwd = self.settings.get("sudo_pwd")
        if not pwd:
            messagebox.showerror("Erreur", "Veuillez configurer votre mot de passe système (Sudo) dans les Paramètres.")
            return
        if pwd is not None:
            self.app_manager.save_session()
            try:

```

```

        res = subprocess.run(['sudo', '-S', 'systemctl', 'reboot'], input=pwd + '\n', text=True,
capture_output=True)
        if res.returncode != 0:
            messagebox.showerror("Erreur", f"Échec de la commande:\n{res.stderr}")
        except Exception as e:
            print(f"Erreur reboot: {e}")

    def poweroff_system(self):
        from tkinter import messagebox
        pwd = self.settings.get("sudo_pwd")
        if not pwd:
            messagebox.showerror("Erreur", "Veuillez configurer votre mot de passe système (Sudo) dans
les Paramètres.")
            return
        if pwd is not None:
            self.app_manager.save_session()
            try:
                res = subprocess.run(['sudo', '-S', 'systemctl', 'poweroff'], input=pwd + '\n',
text=True, capture_output=True)
                if res.returncode != 0:
                    messagebox.showerror("Erreur", f"Échec de la commande:\n{res.stderr}")
            except Exception as e:
                print(f"Erreur poweroff: {e}")

    def show_toast(self, message):
        toast = tk.Label(self.desktop_frame, text=message, bg="#333", fg="white", font=("Arial", 10,
"bold"), padx=15, pady=10, relief="solid", bd=1)

        y_pos = -10
        for t in self.toasts:
            self.desktop_frame.update_idletasks()
            y_pos -= t.winfo_reqheight() + 5

        toast.place(relx=1.0, rely=1.0, x=-10, y=y_pos, anchor="se")
        self.toasts.append(toast)

    def remove_toast():
        if toast in self.toasts:
            self.toasts.remove(toast)
            toast.destroy()

        current_y = -10
        for t in self.toasts:
            self.desktop_frame.update_idletasks()
            t.place(relx=1.0, rely=1.0, x=-10, y=current_y, anchor="se")
            current_y -= t.winfo_reqheight() + 5

        self.after(4000, remove_toast)

    def load_desktop_icons(self):
        import json
        desktop_config_path = os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))),
"config", "desktop.json")
        icon_dir = os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))), "icons")
        icon_theme = self.theme_data.get("icon_theme", "grimos")
        icon_theme_dir = os.path.join(icon_dir, "themes", icon_theme)

        if not os.path.exists(desktop_config_path):
            try:
                with open(desktop_config_path, "w") as f:
                    json.dump([], f)
            except:
                pass

        for w in self.desktop_frame.winfo_children():
            if getattr(w, "_is_desktop_icon", False):
                w.destroy()

        try:
            with open(desktop_config_path, "r") as f:
                icons = json.load(f)

```

```

for icon_data in icons:
    name = icon_data.get("name", "Fichier")
    path = icon_data.get("path", "")
    x = icon_data.get("x", 50)
    y = icon_data.get("y", 50)

    img_name = "blocnotes.png"
    if os.path.isdir(path):
        img_name = "explorateur.png"
    elif path.endswith(".py"):
        img_name = "terminal.png"

    img_path = os.path.join(icon_theme_dir, img_name)
    if not os.path.exists(img_path):
        img_path = os.path.join(icon_dir, img_name)

    try:
        photo = tk.PhotoImage(file=img_path)
    except:
        photo = None

    # Truncate long names
    disp_name = name if len(name) < 15 else name[:12]+"..."

    # Use theme colors for icon label
    fg_color = self.theme_data.get("desktop_fg", "white")

    btn = tk.Button(self.desktop_frame, text=disp_name, image=photo, compound="top",
relief="flat", bg=self.desktop_frame.cget("bg"), fg=fg_color, justify="center", width=80, command=lambda
p=path: self.launch_file(p))
    btn.image = photo # Keep reference
    btn._is_desktop_icon = True
    btn.place(x=x, y=y)

def make_draggable_and_menu(widget, data, icon_list):
    def start_drag(event):
        widget._drag_start_x = event.x
        widget._drag_start_y = event.y
    def do_drag(event):
        nx = widget.winfo_x() - widget._drag_start_x + event.x
        ny = widget.winfo_y() - widget._drag_start_y + event.y
        widget.place(x=nx, y=ny)
    def stop_drag(event):
        # Don't register drag if the widget was just clicked
        if getattr(widget, "_is_menu_open", False): return
        data["x"] = widget.winfo_x()
        data["y"] = widget.winfo_y()
        self.save_desktop_icons(icon_list)

    def show_menu(event):
        menu = tk.Menu(self.desktop_frame, tearoff=0)

        def delete_shortcut():
            if data in icon_list:
                icon_list.remove(data)
                self.save_desktop_icons(icon_list)
                widget.destroy()
                self.show_toast(f"Raccourci supprimé")

        menu.add_command(label=" Supprimer le raccourci",
image=self.icons.get('btn_trash'), compound="left", command=delete_shortcut)

        widget._is_menu_open = True
        try:
            menu.tk_popup(event.x_root, event.y_root)
        finally:
            menu.grab_release()
            widget._is_menu_open = False

    widget.bind("<Button-1>", start_drag)
    widget.bind("<B1-Motion>", do_drag)
    widget.bind("<ButtonRelease-1>", stop_drag)

```

```
        widget.bind("<Button-3>", show_menu)

        make_draggable_and_menu(btn, icon_data, icons)
    except Exception as e:
        print("Erreur desktop icons:", e)

    def save_desktop_icons(self, icons):
        import json
        desktop_config_path = os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))),
"config", "desktop.json")
        try:
            with open(desktop_config_path, "w") as f:
                json.dump(icons, f, indent=4)
        except:
            pass

    def launch_file(self, path):
        if not os.path.exists(path):
            self.show_toast(f"Fichier introuvable:\n{path}")
            return

        if os.path.isdir(path):
            self.app_manager.launch_app({"name": "Explorateur", "module": "apps.explorateur.app"},
filepath=path)
        elif path.endswith(".py"):
            self.app_manager.launch_app({"name": "Éditeur de Code", "module": "apps.editeur.app"},
filepath=path)
        elif path.endswith((".png", ".jpg", ".gif")):
            self.app_manager.launch_app({"name": "Visionneuse", "module": "apps.visionneuse.app"},
filepath=path)
        else:
            self.app_manager.launch_app({"name": "Bloc-notes", "module": "apps.blocnotes.app"},
filepath=path)
```

Annexe : Noyau - main.py

Rôle et utilité

Le fichier `main.py` est situé à la racine du projet (`grimos_build/`). C'est le tout premier fichier exécuté par Python lorsque le système d'exploitation démarre (généralement appelé par le script `.xinitrc` de Linux via la commande `startx`). C'est l'étincelle qui allume le moteur de GrimOS.

Implémentation technique

- **Initialisation de Tkinter** : C'est ici qu'est instancié l'objet `root = tk.Tk()`, qui est la fenêtre mère absolue de tout le système. Elle est configurée pour prendre tout l'écran (`-fullscreen`).
- **Gestionnaire global de clic (`on_global_click`)** : Ce script contient un "hack" essentiel. Il intercepte tous les clics gauches (`<ButtonPress-1>`) sur l'écran. Si l'utilisateur clique à l'intérieur d'une fausse fenêtre (`Window`), ce gestionnaire ordonne à la fenêtre de passer au premier plan (`w.lift()`), simulant le comportement d'un vrai bureau.
- **Boucle Principale** : L'exécution s'arrête sur `root.mainloop()`. C'est cette boucle infinie qui maintient l'ordinateur éveillé et gère l'affichage à 60 images par seconde.
- **Redémarrage à chaud** : Si la session est fermée avec une demande de redémarrage (flag `restart_requested`), le script utilise `os.execv` pour tuer le processus Python actuel et le relancer immédiatement à partir de zéro, permettant de recharger tous les fichiers du système sans redémarrer le serveur X11.

Pistes de modification

- **Gestion des écrans multiples** : Actuellement, GrimOS est pensé pour un seul écran. Un développeur pourrait modifier `root.geometry()` ici pour étendre le `Canvas` sur plusieurs moniteurs.

Code Source


```

import sys
import os
import tkinter as tk

# Add project root to path
sys.path.append(os.path.dirname(os.path.abspath(__file__)))

from core.config import load_settings
from core.desktop import Desktop

def main():
    settings = load_settings()

    root = tk.Tk()
    root.config(cursor="left_ptr")
    root.title("GrimOS")

    def on_global_click(event):
        try:
            if hasattr(event.widget, 'focus_target'):
                event.widget.focus_target.focus_set()
            else:
                event.widget.focus_set()
        except Exception:
            pass

        w = event.widget
        while hasattr(w, 'master') and w.master:
            if w.__class__.__name__ == 'Window':
                w.lift()
                break
            w = w.master

    root.bind_all("<ButtonPress-1>", on_global_click, add="+")

    if settings.get("fullscreen", False):
        w = root.winfo_screenwidth()
        h = root.winfo_screenheight()
        root.geometry(f"{w}x{h}+0+0")
        root.attributes('-fullscreen', True)
    else:
        res = settings.get("resolution", "1024x768")
        root.geometry(res)

    theme_name = settings.get("theme", "GrimOS")
    from core.theme import apply_theme
    apply_theme(root, theme_name)

    desktop = Desktop(root, settings)
    desktop.restart_requested = False

    root.mainloop()

    if getattr(desktop, "restart_requested", False):
        script = os.path.abspath(sys.argv[0])
        os.execl(sys.executable, sys.executable, script)

if __name__ == "__main__":
    main()

```

Annexe : Noyau - theme.py

Rôle et utilité

`core/theme.py` est le moteur esthétique de GrimOS. Il contient la base de données des "styles visuels" et la logique nécessaire pour injecter ces styles dans l'ensemble des composants Tkinter, permettant à GrimOS de changer radicalement d'apparence en temps réel.

Implémentation technique

- **Le Dictionnaire** `THEMES` : C'est le cœur du fichier. Il liste des profils comme "Win 98", "MacOS Classic", ou "Ubuntu". Chaque profil est un dictionnaire de variables (couleurs hexadécimales, noms de polices, styles de bordures).
- **L'injection globale** (`option_add`) : La fonction `apply_theme(root, theme_name)` utilise le mécanisme natif (mais peu connu) de Tkinter : la base de données d'options X11. En utilisant `root.option_add("*background", couleur)`, elle force tous les widgets qui seront instanciés par la suite à hériter de cette couleur, sans qu'il soit nécessaire de modifier le code des applications.
- **Les exceptions logiques** : Bien que l'injection globale soit puissante, le script inclut des règles spécifiques. Par exemple, les zones de saisie de texte (`Entry` ou `Text`) doivent souvent rester blanches avec du texte noir pour être lisibles, à moins que le système ne soit réglé sur un thème sombre pur (comme le thème "Ubuntu").

Pistes de modification

- **Création de nouveaux thèmes** : La modification la plus courante consiste simplement à ajouter un nouveau bloc au dictionnaire `THEMES` avec ses propres couleurs, pour créer un environnement "Hacker", "Rose Poudré", ou un mode "Lecture de Nuit" (couleurs chaudes ambrées).
- **Thématisation des icônes** : Actuellement, le moteur change surtout les couleurs. On pourrait enrichir ce script pour qu'il modifie aussi un chemin d'icônes global (ex: passer des icônes pixel-art aux icônes vectorielles lisses selon le thème choisi).

Code Source

```
import tkinter as tk

THEMES = {
    "GrimOS": {
        "bg": "lightgray",
        "fg": "black",
        "font": ("Arial", 10),
        "btn_bg": "#e0e0e0",
        "relief": "flat",
        "desktop_bg": "gray",
        "desktop_fg": "white",
        "accent": "blue",
        "title_bg": "darkblue",
        "title_fg": "white",
        "taskbar_bg": "#333333",
        "taskbar_fg": "white",
        "start_btn_bg": "blue",
        "start_btn_fg": "white",
        "panel_btn_bg": "#555555",
        "window_border": "black",
        "icon_theme": "grimos"
    },
    "Win 3.1": {
        "bg": "white",
        "fg": "black",
        "font": ("System", 10, "bold"),
        "btn_bg": "white",
        "relief": "solid",
        "desktop_bg": "#008080",
        "desktop_fg": "black",
        "accent": "black",
        "title_bg": "white",
        "title_fg": "black",
        "taskbar_bg": "white",
        "taskbar_fg": "black",
        "start_btn_bg": "white",
        "start_btn_fg": "black",
        "panel_btn_bg": "white",
        "window_border": "black",
        "icon_theme": "win31"
    },
    "Win 98": {
        "bg": "#C0C0C0",
        "fg": "black",
        "font": ("MS Sans Serif", 9),
        "btn_bg": "#C0C0C0",
        "relief": "raised",
        "desktop_bg": "#008080",
        "desktop_fg": "white",
        "accent": "#000080",
        "title_bg": "#000080",
        "title_fg": "white",
        "taskbar_bg": "#C0C0C0",
        "taskbar_fg": "black",
        "start_btn_bg": "#C0C0C0",
        "start_btn_fg": "black",
        "panel_btn_bg": "#C0C0C0",
        "window_border": "#C0C0C0",
        "icon_theme": "win98"
    },
    "Win XP": {
        "bg": "#ECE9D8",
        "fg": "black",
        "font": ("Tahoma", 8),
        "btn_bg": "#ECE9D8",
        "relief": "raised",
    }
}
```

```

        "desktop_bg": "#004E98",
        "desktop_fg": "white",
        "accent": "#003399",
        "title_bg": "#003399",
        "title_fg": "white",
        "taskbar_bg": "#245EDC",
        "taskbar_fg": "white",
        "start_btn_bg": "#3C8120",
        "start_btn_fg": "white",
        "panel_btn_bg": "#245EDC",
        "window_border": "#003399",
        "icon_theme": "wintp"
    },
    "MacOS Classic": {
        "bg": "#E8E8E8",
        "fg": "black",
        "font": ("Geneva", 9),
        "btn_bg": "#E8E8E8",
        "relief": "raised",
        "desktop_bg": "#4A7698",
        "desktop_fg": "black",
        "accent": "#4A7698",
        "title_bg": "white",
        "title_fg": "black",
        "taskbar_bg": "white",
        "taskbar_fg": "black",
        "start_btn_bg": "white",
        "start_btn_fg": "black",
        "panel_btn_bg": "white",
        "window_border": "black",
        "icon_theme": "mac"
    },
    "Ubuntu": {
        "bg": "#3c3b37",
        "fg": "#dfdc8",
        "font": ("Ubuntu", 10),
        "btn_bg": "#4e4d49",
        "relief": "flat",
        "desktop_bg": "#2c001e",
        "desktop_fg": "white",
        "accent": "#f07746",
        "title_bg": "#333333",
        "title_fg": "#dfdc8",
        "taskbar_bg": "#111111",
        "taskbar_fg": "white",
        "start_btn_bg": "#f07746",
        "start_btn_fg": "white",
        "panel_btn_bg": "#333333",
        "window_border": "#333333",
        "icon_theme": "ubuntu"
    },
    "Lubuntu": {
        "bg": "#f0f0f0",
        "fg": "#333333",
        "font": ("Ubuntu", 10),
        "btn_bg": "#00557f",
        "relief": "flat",
        "desktop_bg": "#1c3144",
        "desktop_fg": "white",
        "accent": "#00557f",
        "title_bg": "#1c3144",
        "title_fg": "white",
        "taskbar_bg": "#1c3144",
        "taskbar_fg": "white",
        "start_btn_bg": "#00557f",
        "start_btn_fg": "white",
        "panel_btn_bg": "#00557f",
        "window_border": "#1c3144",
        "icon_theme": "lubuntu"
    }
}

```

```
def apply_theme(root, theme_name):
    theme = THEMES.get(theme_name, THEMES["GrimOS"])

    # Configuration globale pour tous les futurs widgets
    root.option_add("*background", theme["bg"])
    root.option_add("*foreground", theme["fg"])
    root.option_add("*font", theme["font"])

    # Styles spécifiques
    root.option_add("*Button.background", theme["btn_bg"])
    root.option_add("*Button.relief", theme["relief"])
    root.option_add("*Button.borderwidth", 2 if theme["relief"] == "raised" else 1)

    # Pour les entrées de texte, on veut souvent du blanc sauf si c'est très sombre
    if theme_name in ["Ubuntu"]:
        root.option_add("*Entry.background", "#2b2a27")
        root.option_add("*Entry.foreground", "white")
        root.option_add("*Text.background", "#2b2a27")
        root.option_add("*Text.foreground", "white")
    else:
        root.option_add("*Entry.background", "white")
        root.option_add("*Entry.foreground", "black")
        root.option_add("*Text.background", "white")
        root.option_add("*Text.foreground", "black")

    return theme
```

Annexe : Noyau - wifi.py

Rôle et utilité

Le fichier `core/wifi.py` est responsable de la gestion des réseaux sans fil. En l'absence d'outils lourds comme *NetworkManager*, il s'interface directement avec l'outil système bas niveau `wpa_supplicant` pour scanner les réseaux, s'y connecter et vérifier le statut de la connexion.

Implémentation technique

- **Détection matérielle** : La méthode `_detect_iface()` lit directement le dossier Linux `/sys/class/net` pour chercher une interface réseau dont le nom commence par `wl` (ex: `wlan0` ou `wlp2s0`), s'assurant que le Wi-Fi fonctionnera indépendamment du nom donné par le noyau Linux.
- **Le Scanner (Parsing de lignes)** : La méthode `scan_networks()` exécute la commande `wpa_cli scan` puis `wpa_cli scan_results`. La vraie difficulté réside dans le fait de découper la chaîne de caractères renvoyée (séparée par des tabulations) pour extraire le nom du réseau (SSID) et sa puissance de signal (en dBm), tout en filtrant les réseaux cachés ou invalides.
- **L'injection de configuration** : La méthode `connect_network()` construit de toutes pièces une configuration temporaire en exécutant une suite de commandes `add_network`, `set_network`, puis `enable_network` via `wpa_cli`. Elle supporte aussi bien les réseaux ouverts (`key_mgmt=NONE`) que les réseaux protégés par mot de passe.

Pistes de modification

- **Support de l'EAP-PEAP (Réseaux d'entreprise)** : Actuellement, le script gère les mots de passes simples (WPA/WPA2-PSK). Si GrimOS devait être utilisé dans une université ou une entreprise, il faudrait modifier la méthode `connect_network` pour injecter des paramètres `identity` et `password` via la norme 802.1x.
- **Mémorisation des réseaux multiples** : Le script pourrait lire le fichier `/etc/wpa_supplicant/wpa_supplicant.conf` en python pour lister tous les réseaux déjà enregistrés

et proposer à l'utilisateur de les "Oublier" via un menu dédié dans l'interface.

Code Source

```
import subprocess
import os

class WifiManager:
    def __init__(self, desktop):
        self.desktop = desktop
        self.iface = self._detect_iface()

    def _detect_iface(self):
        try:
            net_path = "/sys/class/net"
            if os.path.exists(net_path):
                for iface in os.listdir(net_path):
                    if iface.startswith("wl"):
                        return iface
        except:
            pass
        return None

    def scan_networks(self, sudo_pwd):
        if not self.iface:
            return []

        try:
            # Déclenchement du scan via wpa_cli (car wpa_supplicant tourne déjà)
            subprocess.run(['sudo', '-S', '/sbin/wpa_cli', '-i', self.iface, 'scan'], input=sudo_pwd +
                '\n', text=True, capture_output=True)

            import time
            time.sleep(5) # Laisser le temps au scan complet de s'effectuer (souvent 3 à 5 secondes)

            # Récupération des résultats
            res = subprocess.run(['sudo', '-S', '/sbin/wpa_cli', '-i', self.iface, 'scan_results'],
                input=sudo_pwd + '\n', text=True, capture_output=True)

            # DEBUG : Sauvegarder la sortie brute pour analyse
            try:
                with open('/tmp/wifi_debug.txt', 'w') as f:
                    f.write(res.stdout)
            except:
                pass

            networks = {}
            lines = res.stdout.strip().split('\n')
            # Format attendu : bssid / frequency / signal level / flags / ssid (séparés par des
            tabulations)
            for line in lines[1:]:
                parts = line.split('\t')
                if len(parts) >= 5:
                    signal = parts[2]
                    ssid = parts[4]

                    if ssid and not ssid.startswith('\x00'):
                        try:
                            # Ne garder que le meilleur signal par SSID
                            sig_val = float(signal)
                            if ssid not in networks or networks[ssid]['signal'] < sig_val:
                                networks[ssid] = {'ssid': ssid, 'signal': sig_val}
                        except:
                            pass

            valid_nets = sorted(networks.values(), key=lambda x: x['signal'], reverse=True)
```

```

        return valid_nets
    except Exception as e:
        print("Scan error:", e)
        return []

def connect_network(self, sudo_pwd, ssid, psk):
    if not self.iface:
        return False, "Aucune interface Wi-Fi détectée"

    try:
        def run_wpa_cmd(args):
            cmd = ['sudo', '-S', '/sbin/wpa_cli', '-i', self.iface] + args
            res = subprocess.run(cmd, input=sudo_pwd + '\n', text=True, capture_output=True)
            return res.stdout.strip()

        out = run_wpa_cmd(['add_network'])
        # Output of add_network should be the network ID (integer as string)
        # but sometimes it has leading/trailing debug info. Let's just grab the last line.
        lines = out.split('\n')
        net_id = lines[-1].strip()

        if not net_id.isdigit():
            return False, f"Erreur de création de réseau ({out})"

        run_wpa_cmd(['set_network', net_id, 'ssid', f'"{ssid}"'])
        if psk:
            run_wpa_cmd(['set_network', net_id, 'psk', f'"{psk}"'])
        else:
            run_wpa_cmd(['set_network', net_id, 'key_mgmt', 'NONE'])

        run_wpa_cmd(['enable_network', net_id])
        run_wpa_cmd(['save_config'])
        run_wpa_cmd(['select_network', net_id])

        return True, "Connexion initiée."
    except Exception as e:
        return False, str(e)

def get_current_status(self, sudo_pwd):
    if not self.iface:
        return None, None

    try:
        res = subprocess.run(['sudo', '-S', '/sbin/wpa_cli', '-i', self.iface, 'status'],
            input=sudo_pwd + '\n', text=True, capture_output=True)

        ssid = None
        ip = None
        wpa_state = None
        for line in res.stdout.split('\n'):
            if line.startswith('ssid='):
                ssid = line.split('=', 1)[1]
            elif line.startswith('ip_address='):
                ip = line.split('=', 1)[1]
            elif line.startswith('wpa_state='):
                wpa_state = line.split('=', 1)[1]

        if wpa_state == 'COMPLETED':
            return ssid, ip
    except:
        pass
    return None, None

```


Annexe : Noyau - window.py

Rôle et utilité

Puisque GrimOS démarre directement sur un serveur graphique nu (sans GNOME ni XFCE), il n'existe **aucune notion de fenêtre** au niveau du système d'exploitation. C'est le fichier `core/window.py` qui invente cette notion. Il crée une fausse fenêtre (un rectangle) contenant une barre de titre, des boutons de contrôle, et la capacité d'être déplacée à la souris.

Implémentation technique

- **Composition Tkinter** : La classe `Window` hérite d'un `tk.Frame`. Elle est composée de deux parties : un `title_bar` (le bandeau supérieur coloré contenant le titre et la croix rouge) et un `content_frame` (la zone vide où l'application aura le droit de dessiner).
- **Mathématiques du Déplacement (Drag & Drop)** : C'est la mécanique la plus complexe du fichier. Le script "écoute" le clic gauche de la souris sur la barre de titre (`<ButtonPress-1>`) pour mémoriser les coordonnées initiales `(x, y)`. Ensuite, lorsqu'il détecte le mouvement (`<B1-Motion>`), il calcule la différence (le delta) et met à jour instantanément la position du composant global sur l'écran (`place(x=..., y=...)`), créant l'illusion parfaite d'une fenêtre qui glisse sous la souris.
- **Superposition (Z-Index)** : Lorsque l'utilisateur clique sur une fenêtre, le gestionnaire appelle `self.tkraise()`. C'est l'instruction qui ordonne au moteur graphique de placer ce rectangle par-dessus tous les autres, simulant la "mise au premier plan".

Pistes de modification

- **Redimensionnement (Resize)** : Actuellement, les fenêtres ont une taille fixe ou calculée. Ajouter des bords réactifs (des minuscules cadres invisibles de 2 pixels sur les bords) écoutant le déplacement de la souris permettrait de coder une fonctionnalité de redimensionnement manuel.
- **Animations d'ouverture** : Au lieu d'apparaître brutalement, on pourrait utiliser la fonction `after` pour créer une mini-animation de 0.2 secondes, faisant passer la fenêtre d'une

taille nulle à sa taille réelle, renforçant l'impression de fluidité du système.

Code Source

```
import tkinter as tk

class Window(tk.Frame):
    def __init__(self, parent, desktop, app_config, title="Fenêtre", width=400, height=300, x=50, y=50,
is_maximized=False, filepath=None):
        super().__init__(parent, highlightbackground="black", highlightthickness=1)
        self.parent = parent
        self.desktop = desktop
        self.app_config = app_config
        self.filepath = filepath

        if filepath:
            import os
            self.title = os.path.basename(filepath)
        else:
            self.title = title

        # Bounding box constraints
        parent_w = parent.winfo_width()
        if parent_w <= 1:
            parent_w = parent.winfo_screenwidth()

        parent_h = parent.winfo_height()
        if parent_h <= 1:
            parent_h = parent.winfo_screenheight()

        if width > parent_w: width = parent_w
        if height > parent_h - 40: height = parent_h - 40
        if x < 0: x = 0
        if y < 0: y = 0
        if x + width > parent_w: x = parent_w - width
        if y + 30 > parent_h: y = parent_h - 60

        self.is_maximized = False
        self.saved_geometry = {"x": x, "y": y, "width": width, "height": height}

        # Récupération du thème
        import sys, os
        sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
        from core.theme import THEMES
        theme_name = self.desktop.settings.get("theme", "GrimOS")
        self.theme_data = THEMES.get(theme_name, THEMES["GrimOS"])

        # Application de la bordure
        self.config(highlightbackground=self.theme_data.get("window_border", "black"))

        self.place(x=x, y=y, width=width, height=height)

        # Title bar
        self.title_bar = tk.Frame(self, bg=self.theme_data.get("title_bg", "darkblue"), relief="raised",
bd=1)
        self.title_bar.pack(fill="x")

        self.title_label = tk.Label(self.title_bar, text=self.title, bg=self.theme_data.get("title_bg",
"darkblue"), fg=self.theme_data.get("title_fg", "white"), font=("Arial", 10, "bold"))
        self.title_label.pack(side="left", padx=5)

        self.close_btn = tk.Button(self.title_bar, text="X", bg="red", fg="white", command=self.close,
bd=0, padx=5)
        self.close_btn.pack(side="right")

        self.max_btn = tk.Button(self.title_bar, text="[ ]", bg="gray", fg="white",
```

```

command=self.toggle_maximize, bd=0, padx=5)
    self.max_btn.pack(side="right", padx=2)

    self.min_btn = tk.Button(self.title_bar, text="_", bg="gray", fg="white", command=self.minimize,
bd=0, padx=5)
    self.min_btn.pack(side="right", padx=2)

    # Content area
    self.content = tk.Frame(self, bg="white")
    self.content.pack(fill="both", expand=True)

    # Resize grip
    self.sizegrip = tk.Label(self, text="▀", fg="gray", bg="lightgray", cursor="bottom_right_corner")
    self.sizegrip.place(relx=1.0, rely=1.0, anchor="se")

    def on_destroy(event):
        if str(event.widget) == str(self):
            self.desktop.remove_window(self)
        self.bind("<Destroy>", on_destroy, add="+")

    # Bindings
    self.title_bar.bind("<ButtonPress-1>", self.start_drag)
    self.title_label.bind("<ButtonPress-1>", self.start_drag)
    self.title_bar.bind("<B1-Motion>", self.do_drag)
    self.title_label.bind("<B1-Motion>", self.do_drag)

    self.sizegrip.bind("<ButtonPress-1>", self.start_resize)
    self.sizegrip.bind("<B1-Motion>", self.do_resize)

    self._drag_data = {"x": 0, "y": 0}
    self._resize_data = {"width": 0, "height": 0, "x": 0, "y": 0}

    if is_maximized:
        self.toggle_maximize()

    self.desktop.register_window(self)
    self.lift()

def start_drag(self, event):
    if self.is_maximized:
        return
    self.lift()
    self._drag_data["x"] = event.x
    self._drag_data["y"] = event.y

def do_drag(self, event):
    if self.is_maximized:
        return
    x = self.winfo_x() - self._drag_data["x"] + event.x
    y = self.winfo_y() - self._drag_data["y"] + event.y
    self.place(x=x, y=y)
    self.update_saved_geometry()

def start_resize(self, event):
    if self.is_maximized:
        return
    self.lift()
    self._resize_data["x"] = event.x_root
    self._resize_data["y"] = event.y_root
    self._resize_data["width"] = self.winfo_width()
    self._resize_data["height"] = self.winfo_height()

def do_resize(self, event):
    if self.is_maximized:
        return
    dx = event.x_root - self._resize_data["x"]
    dy = event.y_root - self._resize_data["y"]
    new_width = max(150, self._resize_data["width"] + dx)
    new_height = max(100, self._resize_data["height"] + dy)
    self.place(width=new_width, height=new_height)
    self.update_saved_geometry()

```

```

def toggle_maximize(self):
    if not self.is_maximized:
        self.update_saved_geometry()
        self.config(highlightthickness=0)
        self.place(x=0, y=0, width=self.parent.winfo_width(), height=self.parent.winfo_height())
        self.is_maximized = True
        self.max_btn.config(text="[R]")
    else:
        self.config(highlightthickness=1)
        self.place(x=self.saved_geometry["x"], y=self.saved_geometry["y"],
                    width=self.saved_geometry["width"], height=self.saved_geometry["height"])
        self.is_maximized = False
        self.max_btn.config(text="[ ]")

def update_saved_geometry(self):
    # Prevent layout artifacts from propagating empty dimensions during creation
    if self.winfo_width() > 10:
        self.saved_geometry = {
            "x": self.winfo_x(),
            "y": self.winfo_y(),
            "width": self.winfo_width(),
            "height": self.winfo_height()
        }

def minimize(self):
    self.update_saved_geometry()
    self.place_forget()
    self.desktop.update_taskbar_btn(self)

def restore(self):
    if self.is_maximized:
        self.config(highlightthickness=0)
        self.place(x=0, y=0, width=self.parent.winfo_width(), height=self.parent.winfo_height())
    else:
        self.config(highlightthickness=1)
        self.place(x=self.saved_geometry["x"], y=self.saved_geometry["y"],
                    width=self.saved_geometry["width"], height=self.saved_geometry["height"])
    self.lift()
    self.desktop.update_taskbar_btn(self)

def close(self):
    if hasattr(self, 'on_close_callback') and callable(self.on_close_callback):
        if self.on_close_callback() is False:
            return

    self.update_saved_geometry()
    app_module = self.app_config.get("module")
    if app_module:
        try:
            import json
            import os
            config_path = os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))),
'config', 'applications.json')
            with open(config_path, 'r', encoding='utf-8') as f:
                apps = json.load(f)

            updated = False
            for app in apps:
                if app.get("module") == app_module:
                    app["width"] = self.saved_geometry["width"]
                    app["height"] = self.saved_geometry["height"]
                    app["x"] = self.saved_geometry["x"]
                    app["y"] = self.saved_geometry["y"]
                    updated = True
                    break

            if updated:
                with open(config_path, 'w', encoding='utf-8') as f:
                    json.dump(apps, f, indent=4)

            for app in self.desktop.apps:
                if app.get("module") == app_module:

```

```
        app["width"] = self.saved_geometry["width"]
        app["height"] = self.saved_geometry["height"]
        app["x"] = self.saved_geometry["x"]
        app["y"] = self.saved_geometry["y"]
        break
    except Exception as e:
        print(f"Erreur de sauvegarde de géométrie: {e}")

self.desktop.remove_window(self)
self.destroy()
```

Annexe : Application - Bloc-notes

Rôle et utilité

L'application `apps/blocnotes/app.py` est l'équivalent du célèbre "Notepad" sous Windows. C'est un éditeur de texte pur, minimaliste, permettant à l'utilisateur de prendre des notes rapides, de copier/coller du texte ou de lire de petits fichiers sans la lourdeur d'un traitement de texte complet.

Implémentation technique

- **Composant** `tk.Text` : Le cœur de l'application repose entièrement sur le composant multi-lignes standard de Tkinter. Il est configuré pour occuper tout l'espace disponible (`fill="both", expand=True`).
- **Barre de défilement (Scrollbar)** : L'application instancie une `tk.Scrollbar` qu'elle lie manuellement au widget de texte (`yscrollcommand=scrollbar.set`), démontrant la façon standard de créer une zone défilante en Tkinter.
- **Menu Fichier** : Contrairement aux applications modernes avec des barres de boutons massives, le bloc-notes utilise un `tk.Menu` classique (Fichier > Nouveau, Ouvrir, Enregistrer, Quitter), ce qui lui confère son aspect délicieusement rétro.
- **Dialogue système** : Pour ouvrir et sauvegarder, l'application s'appuie sur `tkinter.filedialog` , ce qui permet d'utiliser la fenêtre de navigation native sans avoir à recoder un explorateur de fichiers interne.

Pistes de modification

- **Sauvegarde automatique** : Un développeur pourrait ajouter un mécanisme (via `window.after`) qui écrit discrètement le contenu du texte dans un fichier temporaire `/tmp/autosave.txt` toutes les minutes pour éviter les pertes de données.
- **Statistiques en temps réel** : L'ajout d'une petite barre d'état (`tk.Label`) en bas de la fenêtre indiquant le nombre de mots et de caractères tapés transformerait cet outil basique en un outil plus orienté vers la rédaction.

Code Source

```

import tkinter as tk
from tkinter import filedialog, messagebox

import os

def start(window, app_manager=None, **kwargs):
    text_area = tk.Text(window, wrap="word", undo=True)
    text_area.pack(fill="both", expand=True)

    menu_frame = tk.Frame(window, bg="lightgray")
    menu_frame.pack(side="top", fill="x", before=text_area)

    current_filepath = kwargs.get("filepath")

    def update_title(modified=False):
        if hasattr(window, 'master') and window.master.__class__.__name__ == 'Window':
            title_text = current_filepath if current_filepath else "Sans titre"
            prefix = "*" if modified else ""
            window.master.title_label.config(text=f"Bloc-notes - {prefix}{title_text}")

    update_title()

    def on_modify(event=None):
        update_title(modified=text_area.edit_modified())

    text_area.bind("<<Modified>>", on_modify)

    def set_clean():
        text_area.edit_modified(False)

    if current_filepath and os.path.exists(current_filepath):
        try:
            with open(current_filepath, "r", encoding="utf-8") as f:
                text_area.insert(tk.END, f.read())
            set_clean()
        except Exception as e:
            messagebox.showerror("Erreur", f"Impossible d'ouvrir le fichier :\n{e}")

    def check_unsaved():
        if text_area.edit_modified():
            ans = messagebox.askyesnocancel("Attention", "Voulez-vous enregistrer les modifications ?")
            if ans is True:
                return save_file()
            elif ans is False:
                return True
            else:
                return False
        return True

    def new_file():
        if not check_unsaved(): return
        nonlocal current_filepath
        current_filepath = None
        text_area.delete(1.0, tk.END)
        set_clean()
        update_title()

    def open_file():
        if not check_unsaved(): return
        nonlocal current_filepath
        init_dir = os.path.dirname(current_filepath) if current_filepath else os.path.expanduser("~")
        filepath = filedialog.askopenfilename(initialdir=init_dir, defaultextension=".txt", filetypes=
[("All Files", "*.*"), ("Text Files", "*.txt")])
        if not filepath:
            return
        try:

```

```

        with open(filepath, "r", encoding="utf-8") as f:
            content = f.read()
            text_area.delete(1.0, tk.END)
            text_area.insert(tk.END, content)
            current_filepath = filepath
            set_clean()
            update_title()
    except Exception as e:
        messagebox.showerror("Erreur", f"Impossible d'ouvrir :\n{e}")

def save_file():
    nonlocal current_filepath
    if not current_filepath:
        return save_file_as()
    try:
        with open(current_filepath, "w", encoding="utf-8") as f:
            f.write(text_area.get(1.0, "end-1c"))
        set_clean()
        update_title()
        return True
    except Exception as e:
        messagebox.showerror("Erreur", f"Impossible de sauvegarder :\n{e}")
        return False

def save_file_as():
    nonlocal current_filepath
    init_dir = os.path.dirname(current_filepath) if current_filepath else os.path.expanduser("~")
    filepath = filedialog.asksaveasfilename(initialdir=init_dir, defaultextension=".txt", filetypes=
[("All Files", "*.*"), ("Text Files", "*.txt")])
    if not filepath:
        return False
    current_filepath = filepath
    return save_file()

icon_dir = os.path.join(os.path.dirname(os.path.dirname(os.path.dirname(os.path.abspath(__file__)))),
"icons")
window.icons = {}
for iname in ['menu_new_file', 'menu_open', 'menu_save', 'menu_save_as']:
    ipath = os.path.join(icon_dir, iname + ".png")
    if os.path.exists(ipath):
        window.icons[iname] = tk.PhotoImage(file=ipath)

btn_new = tk.Button(menu_frame, text=" Nouveau", image=window.icons.get('menu_new_file'),
compound="left", command=new_file, relief="flat")
btn_new.pack(side="left", padx=2, pady=2)

btn_open = tk.Button(menu_frame, text=" Ouvrir", image=window.icons.get('menu_open'),
compound="left", command=open_file, relief="flat")
btn_open.pack(side="left", padx=2, pady=2)

btn_save = tk.Button(menu_frame, text=" Enregistrer", image=window.icons.get('menu_save'),
compound="left", command=save_file, relief="flat")
btn_save.pack(side="left", padx=2, pady=2)

def show_help():
    help_text = (
        "Aide du Bloc-notes\n\n"
        "Raccourcis clavier disponibles :\n"
        "• Ctrl + N : Nouveau fichier\n"
        "• Ctrl + O : Ouvrir un fichier\n"
        "• Ctrl + S : Enregistrer\n"
        "• Ctrl + Shift + S : Enregistrer sous...\n"
        "• Ctrl + C : Copier la sélection\n"
        "• Ctrl + X : Couper la sélection\n"
        "• Ctrl + V : Coller\n"
        "• Ctrl + Z : Annuler la modification\n"
        "• Ctrl + Y : Refaire la modification\n"
        "Indicateur visuel :\n"
        "L'étoile (*) dans le titre signifie que le fichier contient des modifications non
enregistrées."
    )
    messagebox.showinfo("Aide", help_text)

```



```
    btn_save_as = tk.Button(menu_frame, text=" Enregistrer sous...",
image=window.icons.get('menu_save_as'), compound="left", command=save_file_as, relief="flat")
    btn_save_as.pack(side="left", padx=2, pady=2)

    btn_help = tk.Button(menu_frame, text=" ? Aide ", command=show_help, relief="flat", bg="lightblue")
    btn_help.pack(side="right", padx=5, pady=2)

    text_area.focus_set()

    text_area.bind("<Control-n>", lambda e: new_file() or "break")
    text_area.bind("<Control-o>", lambda e: open_file() or "break")
    text_area.bind("<Control-s>", lambda e: save_file() or "break")
    text_area.bind("<Control-S>", lambda e: save_file_as() or "break")

# Lier le gestionnaire de fermeture si possible
    if hasattr(window, 'master') and window.master.__class__.__name__ == 'Window':
        window.master.on_close_callback = check_unsaved
```

Annexe : Application - Caméra

Rôle et utilité

`apps/camera/app.py` est une application utilitaire permettant de visualiser le flux vidéo d'une webcam branchée à l'ordinateur, et d'enregistrer des photos. C'est une démonstration brillante de la capacité de GrimOS à s'interfacer avec du matériel externe.

Implémentation technique

- **La bibliothèque `cv2` (OpenCV)** : Cette application est l'une des rares à nécessiter une dépendance lourde externe. Elle utilise `cv2.VideoCapture(0)` pour capturer le flux matériel de la première caméra détectée par le système Linux (`/dev/video0`).
- **Conversion d'image en temps réel** : Tkinter ne sait pas afficher un flux vidéo brut. Le script doit donc, dans une boucle très rapide (`window.after(15)`), capturer l'image de la webcam, convertir ses couleurs de BGR (le format d'OpenCV) vers RGB, puis transformer cette matrice en une image compatible avec l'interface via `ImageTk.PhotoImage` .
- **Nettoyage mémoire** : Le flux vidéo consommant beaucoup de ressources, l'application veille à libérer l'accès à la caméra (`cap.release()`) lorsque l'utilisateur ferme la fenêtre, évitant ainsi un crash matériel persistant.

Pistes de modification

- **Mode Enregistrement Vidéo** : L'application actuelle ne prend que des photos. En exploitant la classe `cv2.VideoWriter` , on pourrait ajouter un bouton rouge "REC" permettant d'enregistrer des vidéos au format MP4 directement sur le disque.
- **Filtres en direct** : Toujours grâce à OpenCV, on pourrait ajouter une barre d'outils offrant des effets amusants en temps réel (Noir et Blanc, effet miroir, détection de contours) avant l'affichage dans le cadre Tkinter.

Code Source

```

import tkinter as tk
from tkinter import messagebox
from PIL import Image, ImageTk
import os
import datetime

def start(window, app_manager=None, **kwargs):
    # Essayons d'importer cv2. S'il n'est pas là, on affiche un message d'erreur clair.
    try:
        import cv2
    except ImportError:
        tk.Label(window, text="La librairie OpenCV n'est pas installée.\n\nVeuillez exécuter :\nsudo apt-get install python3-opencv\n\npuis relancez l'application.", fg="red", font=("Arial", 12)).pack(expand=True)
        return

    top_frame = tk.Frame(window, bg="lightgray", height=40)
    top_frame.pack(side="top", fill="x")

    icon_dir = os.path.join(os.path.dirname(os.path.dirname(os.path.dirname(os.path.abspath(__file__)))), "icons")
    window.icons = getattr(window, "icons", {})
    if 'btn_camera' not in window.icons:
        ipath = os.path.join(icon_dir, "btn_camera.png")
        if os.path.exists(ipath):
            window.icons['btn_camera'] = tk.PhotoImage(file=ipath)

    video_container = tk.Frame(window, bg="black")
    video_container.pack(fill="both", expand=True)

    video_lbl = tk.Label(video_container, bg="black")
    video_lbl.pack(expand=True)

    try:
        cap = cv2.VideoCapture(0)
    except Exception as e:
        messagebox.showerror("Erreur", f"Impossible d'accéder à la webcam : {e}")
        return

    if not cap.isOpened():
        messagebox.showerror("Erreur", "Aucune webcam détectée sur /dev/video0.")
        return

    def update_frame():
        if not window.wininfo_exists():
            cap.release()
            return

        ret, frame = cap.read()
        if ret:
            # OpenCV utilise BGR, ImageTk nécessite RGB
            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            img = Image.fromarray(frame_rgb)
            imgtk = ImageTk.PhotoImage(image=img)

            video_lbl.imgtk = imgtk # garder la référence
            video_lbl.configure(image=imgtk)

        window.after(30, update_frame)

    def take_photo():
        ret, frame = cap.read()
        if ret:
            img_dir = os.path.expanduser("~/Images")
            os.makedirs(img_dir, exist_ok=True)
            timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
            filename = f"photo_{timestamp}.jpg"
            filepath = os.path.join(img_dir, filename)

            cv2.imwrite(filepath, frame)

```

```
        if app_manager and hasattr(app_manager, 'desktop'):
            app_manager.desktop.show_toast(f"Photo sauvegardée :\n{filepath}")

    btn_photo = tk.Button(top_frame, text=" Prendre une photo", image=window.icons.get('btn_camera'),
                           compound="left", font=("Arial", 11, "bold"), bg="#4CAF50", fg="white", relief="flat", command=take_photo)
    btn_photo.pack(side="left", padx=10, pady=5)

    def on_destroy(event):
        if str(event.widget) == str(window):
            if cap.isOpened():
                cap.release()

    window.bind("<Destroy>", on_destroy)
    update_frame()
```

Annexe : Application - Éditeur de Code

Rôle et utilité

`apps/editeur/app.py` est une application capitale de l'écosystème GrimOS. Plus avancé que le simple Bloc-notes, c'est l'outil de développement officiel du système. Puisque GrimOS encourage l'utilisateur à modifier son propre code source, il était impératif de fournir un outil capable de lire du Python confortablement.

Implémentation technique

- **Coloration syntaxique "Fait-maison"** : Contrairement aux IDE modernes qui utilisent d'énormes bibliothèques, cet éditeur utilise le système natif de "Tags" (balises) du composant `tk.Text`. Une fonction Python analyse le texte avec des expressions régulières (Regex) pour repérer les mots-clés (`def`, `class`, `import`), les chaînes de caractères (entre guillemets) ou les commentaires (débutant par `#`), et leur applique instantanément une couleur spécifique.
- **Numérotation des lignes** : Un deuxième `tk.Text` étroit est placé à gauche de la zone de texte principale. Un événement (`<KeyRelease>` ou `<MouseWheel>`) synchronise le défilement des deux zones pour maintenir les numéros de lignes alignés avec le code.
- **Auto-indentation** : Lorsqu'on appuie sur `Entrée`, l'éditeur vérifie le niveau d'indentation de la ligne précédente (le nombre d'espaces) et l'applique automatiquement à la nouvelle ligne, un confort indispensable en Python.

Pistes de modification

- **Lancement de script intégré** : L'éditeur pourrait intégrer un bouton "Play" (Exécuter). En utilisant `subprocess.run()`, l'éditeur pourrait lancer le fichier Python actuellement ouvert dans une petite fenêtre terminal embarquée en bas de l'écran pour afficher le résultat de l'exécution en direct.
- **Support d'autres langages** : Actuellement codé en dur pour Python et JSON, le dictionnaire d'expressions régulières de coloration pourrait être extrait dans un fichier de

configuration séparé, permettant à l'utilisateur d'ajouter les mots-clés du HTML, du C ou du Bash.

Code Source

```
import tkinter as tk
from tkinter import filedialog, messagebox
import re
import os

class LineNumbers(tk.Canvas):
    def __init__(self, *args, **kwargs):
        tk.Canvas.__init__(self, *args, **kwargs)
        self.textwidget = None

    def attach(self, text_widget):
        self.textwidget = text_widget

    def redraw(self, *args):
        self.delete("all")
        i = self.textwidget.index("@0,0")
        while True :
            dline= self.textwidget.dlineinfo(i)
            if dline is None: break
            y = dline[1]
            linenum = str(i).split(".")[0]
            self.create_text(2, y, anchor="nw", text=linenum, font=("Consolas", 10))
            i = self.textwidget.index("%s+1line" % i)

def start(window, app_manager=None, filepath=None, **kwargs):
    frame = tk.Frame(window)
    frame.pack(fill="both", expand=True)

    toolbar = tk.Frame(frame, bg="lightgray", bd=1, relief="raised")
    toolbar.pack(side="top", fill="x")

    current_file = [filepath] if filepath else [None]

    text_frame = tk.Frame(frame)
    text_frame.pack(fill="both", expand=True)

    linenumbers = LineNumbers(text_frame, width=35, bg="lightgrey")
    linenumbers.pack(side="left", fill="y")

    scrollbar = tk.Scrollbar(text_frame)
    scrollbar.pack(side="right", fill="y")

    text_area = tk.Text(text_frame, font=("Consolas", 11), yscrollcommand=scrollbar.set, undo=True)
    text_area.pack(side="left", fill="both", expand=True)
    scrollbar.config(command=text_area.yview)

    linenumbers.attach(text_area)

    text_area.tag_configure("keyword", foreground="blue", font=("Consolas", 11, "bold"))
    text_area.tag_configure("string", foreground="green")
    text_area.tag_configure("comment", foreground="gray", font=("Consolas", 11, "italic"))

    KEYWORDS = ["def", "class", "import", "from", "return", "if", "elif", "else", "for", "while", "in",
    "and", "or", "not", "True", "False", "None", "try", "except", "pass", "break", "continue", "with", "as"]

    def highlight_syntax(event=None):
        text_area.tag_remove("keyword", "1.0", tk.END)
        text_area.tag_remove("string", "1.0", tk.END)
        text_area.tag_remove("comment", "1.0", tk.END)
```

```

content = text_area.get("1.0", tk.END)

# Strings
for match in re.finditer(r'".*?"|\'.*?\'', content):
    start_pos = text_area.index(f"1.0 + {match.start()} chars")
    end_pos = text_area.index(f"1.0 + {match.end()} chars")
    text_area.tag_add("string", start_pos, end_pos)

# Comments
for match in re.finditer(r'#.+', content):
    start_pos = text_area.index(f"1.0 + {match.start()} chars")
    end_pos = text_area.index(f"1.0 + {match.end()} chars")
    text_area.tag_add("comment", start_pos, end_pos)

# Keywords
for kw in KEYWORDS:
    for match in re.finditer(rf'\b{kw}\b', content):
        start_pos = text_area.index(f"1.0 + {match.start()} chars")
        end_pos = text_area.index(f"1.0 + {match.end()} chars")
        tags = text_area.tag_names(start_pos)
        if "string" not in tags and "comment" not in tags:
            text_area.tag_add("keyword", start_pos, end_pos)

def on_text_change(event=None):
    if text_area.edit_modified() or event is not None:
        linenumbers.redraw()
        if hasattr(window, 'highlight_after'):
            window.after_cancel(window.highlight_after)
            window.highlight_after = window.after(500, highlight_syntax)
        text_area.edit_modified(False)

text_area.bind("<<Modified>>", lambda event: on_text_change())
text_area.bind("<KeyRelease>", on_text_change)
text_area.bind("<MouseWheel>", lambda e: window.after(10, linenumbers.redraw))
text_area.bind("<Button-1>", lambda e: window.after(10, linenumbers.redraw))

def auto_indent(event):
    line = text_area.get("insert linestart", "insert")
    match = re.match(r'^([ \t]+)', line)
    if match:
        text_area.insert("insert", "\n" + match.group(1))
        return "break"

text_area.bind("<Return>", auto_indent)

def update_title(title_text):
    if hasattr(window, 'master') and window.master.__class__.__name__ == 'Window':
        window.master.title_label.config(text=title_text)

if current_file[0] and os.path.exists(current_file[0]):
    try:
        with open(current_file[0], 'r', encoding='utf-8') as f:
            text_area.insert("1.0", f.read())
            update_title(f"Éditeur de Code - {os.path.basename(current_file[0])}")
            highlight_syntax()
            linenumbers.redraw()
    except Exception as e:
        messagebox.showerror("Erreur", str(e))
else:
    update_title("Éditeur de Code - Nouveau Fichier")

def open_file():
    path = filedialog.askopenfilename()
    if path:
        current_file[0] = path
        try:
            with open(path, 'r', encoding='utf-8') as f:
                text_area.delete("1.0", tk.END)
                text_area.insert("1.0", f.read())
            update_title(f"Éditeur de Code - {os.path.basename(path)}")
            highlight_syntax()
            linenumbers.redraw()

```

```

        except Exception as e:
            messagebox.showerror("Erreur", str(e))

def save_file():
    if not current_file[0]:
        save_as_file()
    else:
        try:
            with open(current_file[0], 'w', encoding='utf-8') as f:
                f.write(text_area.get("1.0", tk.END))
            messagebox.showinfo("Sauvegarde", "Fichier sauvegardé avec succès.")
        except Exception as e:
            messagebox.showerror("Erreur", str(e))

def save_as_file():
    path = filedialog.asksaveasfilename(defaulttextextension=".py")
    if path:
        current_file[0] = path
        save_file()
        update_title(f"Éditeur de Code - {os.path.basename(path)}")

def new_file():
    current_file[0] = None
    text_area.delete("1.0", tk.END)
    update_title("Éditeur de Code - Nouveau Fichier")
    highlight_syntax()
    linenumbers.redraw()

def new_gui_file():
    new_file()
    boilerplate = '''import tkinter as tk
from tkinter import messagebox

def start(window, app_manager=None, filepath=None, **kwargs):
    # Changement du titre de la fenêtre GrimOS (si exécuté dans GrimOS)
    if hasattr(window, "master") and hasattr(window.master, "title_label"):
        window.master.title_label.config(text="Mon App GrimOS")

    # Conteneur principal
    frame = tk.Frame(window, bg="white")
    frame.pack(fill="both", expand=True, padx=10, pady=10)

    # Instructions
    tk.Label(frame, text="Saisissez un texte :", font=("Arial", 12), bg="white").pack(pady=(10, 0))

    # Zone de saisie (1 ligne)
    entry_var = tk.StringVar()
    entry = tk.Entry(frame, textvariable=entry_var, font=("Arial", 12), width=30)
    entry.pack(pady=10)

    def on_click():
        texte = entry_var.get()
        messagebox.showinfo("Popup", f"Texte saisi : {texte}")

    # Bouton d'action
    btn = tk.Button(frame, text="Afficher le Popup", command=on_click, font=("Arial", 11))
    btn.pack(pady=10)

# Point d'entrée pour tester l'application indépendamment de GrimOS
if __name__ == "__main__":
    root = tk.Tk()
    root.geometry("400x300")

    # En environnement GrimOS (bare X11 sans gestionnaire de fenêtres), tk.Tk n'a pas de bordures.
    # Nous ajoutons une fausse barre de titre pour pouvoir fermer et tester l'application sereinement.
    title_bar = tk.Frame(root, bg="darkblue", relief="raised", bd=1)
    title_bar.pack(fill="x")
    tk.Label(title_bar, text="Test Mode - Mon App", bg="darkblue", fg="white", font=("Arial", 10,
"bold")).pack(side="left", padx=5)
    tk.Button(title_bar, text="X", bg="red", fg="white", command=root.destroy, bd=0,
padx=5).pack(side="right")
'''
    exec(boilerplate)
    start(root)

```



```

# Zone de contenu
content = tk.Frame(root)
content.pack(fill="both", expand=True)

start(content)
root.mainloop()
...
    text_area.insert("1.0", boilerplate)
    highlight_syntax()
    linenumbers.redraw()

def show_help():
    aide = "Éditeur de Code GrimOS\n\n"
    aide += "COMPATIBILITÉ GRIMOS (Règles d'Or) :\n"
    aide += "1. Votre code DOIT avoir une fonction `start(window, app_manager=None, **kwargs)`.\n"
    aide += "2. L'argument `window` fourni est un Frame Tkinter pré-crée par le système. "
    aide += "Attachez TOUS vos éléments visuels (Boutons, Labels) à `window`.\n"
    aide += "3. Ne créez JAMAIS de `tk.Tk()` dans la fonction `start()`, cela casserait le
système.\n\n"
    aide += "CONSEILS :\n"
    aide += "- Utilisez le bouton 'Nouveau GUI' pour générer une structure parfaite.\n"
    aide += "- Le bloc `if __name__ == '__main__':` sert uniquement à tester l'application en
standalone."
    messagebox.showinfo("Aide & Documentation", aide)

    btn_new = tk.Button(toolbar, text=" Nouveau", image=(app_manager.desktop.icons.get("menu_new_file")
if app_manager else None), compound="left", command=new_file)
    btn_new.pack(side="left", padx=2, pady=2)

    btn_new_gui = tk.Button(toolbar, text="Nouveau GUI", command=new_gui_file)
    btn_new_gui.pack(side="left", padx=2, pady=2)

    btn_open = tk.Button(toolbar, text=" Ouvrir", image=(app_manager.desktop.icons.get("menu_open") if
app_manager else None), compound="left", command=open_file)
    btn_open.pack(side="left", padx=2, pady=2)

    btn_save = tk.Button(toolbar, text=" Enregistrer", image=(app_manager.desktop.icons.get("menu_save")
if app_manager else None), compound="left", command=save_file)
    btn_save.pack(side="left", padx=2, pady=2)

    btn_saveas = tk.Button(toolbar, text=" Enregistrer sous...", image=
(app_manager.desktop.icons.get("menu_save_as") if app_manager else None), compound="left",
command=save_as_file)
    btn_saveas.pack(side="left", padx=2, pady=2)

    btn_help = tk.Button(toolbar, text=" Aide", image=(app_manager.desktop.icons.get("btn_help") if
app_manager else None), compound="left", command=show_help)
    btn_help.pack(side="right", padx=2, pady=2)

    window.after(100, linenumbers.redraw)

```

Annexe : Application - Explorateur

Rôle et utilité

`apps/explorateur/app.py` est l'application la plus complexe et la plus utilisée de GrimOS. C'est l'interface graphique qui permet de naviguer dans l'arborescence des fichiers du système Linux, d'ouvrir des documents avec les bonnes applications, et de se connecter aux réseaux.

Implémentation technique

- **Le Treeview** : Le cœur de l'affichage repose sur `ttk.Treeview`, un tableau avancé permettant d'afficher des colonnes (Nom, Taille, Date) et de gérer des listes déroulantes de dossiers.
- **Routage intelligent** : L'explorateur possède un dictionnaire associant les extensions de fichiers à l'application adéquate (ex: `.png` lance `visionneuse`, `.py` lance `editeur`). Lorsqu'on double-clique sur un fichier, le script utilise la référence globale de l'`AppManager` pour déclencher dynamiquement le lancement de la nouvelle fenêtre.
- **Support Samba (SMB)** : L'application inclut une fonctionnalité avancée permettant de se connecter à des partages réseaux Windows/Linux. Elle utilise la commande `sudo mount -t cifs` en arrière-plan pour accrocher le dossier distant dans un sous-dossier de l'utilisateur (généralement `~/Réseau`), rendant les fichiers distants accessibles comme s'ils étaient sur le disque dur local.

Pistes de modification

- **Copier-Coller de fichiers** : Actuellement, l'explorateur est très orienté vers la navigation. L'ajout d'une barre de menu ou d'un menu contextuel (clic droit) avec les opérations fondamentales (Copier, Couper, Coller, Renommer) en appelant les fonctions natives Python (`shutil.copy`, `os.rename`) le rendrait pleinement mature.
- **Génération de miniatures** : Au lieu d'afficher une icône générique pour les images (`.jpg`), le script pourrait utiliser `PIL` (Pillow) en arrière-plan pour générer des miniatures de 32x32 pixels et les afficher dans le Treeview, à la manière d'un explorateur moderne.

Code Source

```

import os
import shutil
import tkinter as tk
from tkinter import messagebox, simpledialog
from tkinter import ttk
import subprocess

def start(window, app_manager=None, **kwargs):
    top_frame = tk.Frame(window, bg="lightgray")
    top_frame.pack(side="top", fill="x")

    icon_dir = os.path.join(os.path.dirname(os.path.dirname(os.path.dirname(os.path.abspath(__file__)))),
"icons")
    window.icons = {}
    for iname in ['menu_new_folder', 'menu_new_file', 'menu_rename', 'menu_delete', 'icon_folder',
'icon_file_txt', 'icon_file_py', 'icon_file_img', 'icon_file_generic', 'btn_search', 'btn_network',
'btn_disconnect', 'btn_paste', 'btn_print', 'btn_refresh']:
        ipath = os.path.join(icon_dir, iname + ".png")
        if os.path.exists(ipath):
            window.icons[iname] = tk.PhotoImage(file=ipath)

    # Initialize Clipboard on Desktop
    if app_manager and not hasattr(app_manager.desktop, "file_clipboard"):
        app_manager.desktop.file_clipboard = {"action": None, "path": None}

    def go_up():
        current = path_var.get()
        parent = os.path.dirname(current)
        if parent != current:
            path_var.set(parent)
            refresh_list()

    path_var = tk.StringVar(value=os.path.expanduser("~/"))
    path_entry = tk.Entry(top_frame, textvariable=path_var)
    path_entry.pack(side="left", fill="x", expand=True, padx=2, pady=2)

    def connect_network():
        pwd = app_manager.desktop.settings.get("sudo_pwd") if app_manager else None
        if not pwd:
            messagebox.showerror("Erreur", "Veuillez configurer votre mot de passe système dans
Paramètres.")
            return

        dialog = tk.Toplevel(window)
        dialog.title("Connecter Réseau (SMB)")
        dialog.geometry("600x480")
        dialog.transient(window)
        dialog.grab_set()

        frame_top = tk.Frame(dialog)
        frame_top.pack(fill="both", expand=True, padx=10, pady=5)

        tk.Label(frame_top, text="Adresses découvertes sur le réseau :").pack(anchor="w")

        tree_frame = tk.Frame(frame_top)
        tree_frame.pack(fill="both", expand=True, pady=5)

        scroll = tk.Scrollbar(tree_frame)
        scroll.pack(side="right", fill="y")

        tree_shares = ttk.Treeview(tree_frame, show="tree", yscrollcommand=scroll.set, height=8)
        tree_shares.pack(side="left", fill="both", expand=True)
        scroll.config(command=tree_shares.yview)

    def scan_network():
        import socket

```

```

import threading

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
try:
    s.connect(('10.255.255.255', 1))
    local_ip = s.getsockname()[0]
except Exception:
    local_ip = "192.168.1.1"
finally:
    s.close()

subnet = ".".join(local_ip.split(".")[0:3]) + "."
found_ips = []

def check_ip(ip):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.settimeout(0.3)
    try:
        if sock.connect_ex((ip, 445)) == 0:
            found_ips.append(ip)
    except Exception: pass
    finally: sock.close()

threads = []
for i in range(1, 255):
    t = threading.Thread(target=check_ip, args=(subnet + str(i),))
    threads.append(t)
    t.start()

for t in threads:
    t.join()

# Now extract shares and hostnames using smbclient if installed
detailed_shares = {}
for ip in found_ips:
    hostname = ""
    try:
        res_nmb = subprocess.run(['nmblookup', '-A', ip], capture_output=True, text=True,
timeout=1)
        if res_nmb.returncode == 0:
            for line in res_nmb.stdout.split('\n'):
                if '<00>' in line and 'GROUP' not in line:
                    hostname = line.split('<00>')[0].strip()
                    break
    except: pass

    display_name = f"{hostname} ({ip})" if hostname else f"[{ip}]"
    detailed_shares[display_name] = []

    try:
        res_smb = subprocess.run(['smbclient', '-L', ip, '-N', '-g'], capture_output=True,
text=True, timeout=2)
        if res_smb.returncode == 0:
            has_shares = False
            for line in res_smb.stdout.split('\n'):
                if line.startswith('Disk|'):
                    parts = line.split('|')
                    if len(parts) >= 2:
                        share_name = parts[1]
                        if share_name not in ('print$', 'IPC$'):
                            detailed_shares[display_name].append(f"//{ip}/{share_name}")
                            has_shares = True
            if not has_shares:
                detailed_shares[display_name].append(f"//{ip}/Partage")
        else:
            detailed_shares[display_name].append(f"//{ip}/Partage")
    except:
        detailed_shares[display_name].append(f"//{ip}/Partage")

    return detailed_shares

def on_scan(auto=False):

```

```

        btn_scan.config(state="disabled", text="Recherche...")
        dialog.update()
        shares = scan_network()
        btn_scan.config(state="normal", text=" Actualiser", image=window.icons.get('btn_search'),
compound="left")
        tree_shares.delete(*tree_shares.get_children())
        if shares:
            icon_folder = window.icons.get('icon_folder')
            for machine, share_list in shares.items():
                machine_id = tree_shares.insert("", "end", text=f" {machine}", open=False)
                for share in share_list:
                    share_name = share.split('/')[1]

                    # Check if already mounted
                    parts = share.strip("/").split("/")
                    if len(parts) >= 2:
                        ip_addr = parts[0]
                        mount_p = os.path.expanduser(f"~/Réseau/{ip_addr}/{share_name}")
                        disp_text = f" {share_name} (Connecté ☒)" if os.path.ismount(mount_p) else
f" {share_name}"
                    else:
                        disp_text = f" {share_name}"

                    if icon_folder:
                        tree_shares.insert(machine_id, "end", text=disp_text, image=icon_folder,
values=(share,))
                    else:
                        tree_shares.insert(machine_id, "end", text=disp_text, values=(share,))
            first = tree_shares.get_children()[0]
            tree_shares.selection_set(first)
            on_tree_select(None)
            if not auto:
                msg = f"{len(shares)} partage(s) trouvé(s) sur le réseau !"
                if not shutil.which("smbclient"):
                    msg += "\n\n(Note: Installez 'smbclient' via le terminal système pour afficher
les noms exacts des dossiers)."
                messagebox.showinfo("Scanner", msg)
            else:
                if not auto:
                    messagebox.showinfo("Scanner", "Aucune machine trouvée.")

def on_tree_select(e):
    selected = tree_shares.selection()
    if selected:
        item = tree_shares.item(selected[0])
        if item.get('values'):
            share_path = item['values'][0]
            try:
                machine_name = tree_shares.item(tree_shares.parent(selected[0]))['text'].strip()
                entry_share_var.set(f"{machine_name} -> {share_path}")
            except:
                entry_share_var.set(share_path)

tree_shares.bind("<<TreeviewSelect>>", on_tree_select)

        btn_scan = tk.Button(frame_top, text=" Actualiser", image=window.icons.get('btn_search'),
compound="left", command=on_scan, bg="#ffeb3b")
        btn_scan.pack(anchor="e", pady=5)

        tk.Label(dialog, text="Adresse du partage sélectionné (ex: NAS ->
//192.168.1.10/Data):").pack(pady=(10,0))
        entry_share_var = tk.StringVar()
        entry_share = tk.Entry(dialog, textvariable=entry_share_var, width=50)
        entry_share.pack(pady=5)

        tk.Label(dialog, text="Nom d'utilisateur (laisser vide si invité :)").pack(pady=5)
        entry_user = tk.Entry(dialog, width=30)
        entry_user.pack(pady=5)

        tk.Label(dialog, text="Mot de passe :").pack(pady=5)
        entry_pass = tk.Entry(dialog, width=30, show="*")
        entry_pass.pack(pady=5)

```

```

def on_connect():
    share_input = entry_share_var.get().strip()
    user = entry_user.get()
    passw = entry_pass.get()
    if not share_input: return

    if " -> " in share_input:
        machine_str, share = share_input.split(" -> ", 1)
    else:
        machine_str, share = "", share_input

    if not share.startswith("//"):
        messagebox.showerror("Erreur", "Le format doit être //IP/Partage")
        return

    share_parts = share.strip("/").split("/")
    if len(share_parts) < 2:
        messagebox.showerror("Erreur", "Veuillez préciser le nom du dossier à la fin (ex:
//192.168.1.10/Documents)")
        return

    server_ip = share_parts[0]
    share_name = share_parts[1]

    machine_clean = machine_str.split(" ")[0]
    machine_clean = "".join(c for c in machine_clean if c.isalnum() or c in ('-',
',', '_', '.')).strip()
    machine_dir = f"{machine_clean}_{server_ip}" if machine_clean and machine_clean.lower() !=
"inconnu" else server_ip

    mount_point = os.path.expanduser(f"~/Réseau/{machine_dir}/{share_name}")
    os.makedirs(mount_point, exist_ok=True)

    if os.path.ismount(mount_point):
        messagebox.showinfo("Information", f"Ce partage est déjà connecté dans :\n{mount_point}")
        path_var.set(mount_point)
        refresh_list()
        dialog.destroy()
        return

    opts = f"username={user},password={passw}" if user else "guest"
    cmd = f"mount -t cifs '{share}' '{mount_point}' -o {opts},uid=${(id -u)},gid=${(id -g)}"

    res = subprocess.run(['sudo', '-S', 'bash', '-c', cmd], input=pwd+'\n', capture_output=True,
text=True)

    if res.returncode == 0:
        messagebox.showinfo("Succès", f"Partage monté dans :\n{mount_point}")
        path_var.set(mount_point)
        refresh_list()
        dialog.destroy()
    else:
        messagebox.showerror("Erreur", f"Échec du montage:\n{res.stderr}\n\nAvez-vous installé
cifs-utils ?")

    btn_frame = tk.Frame(dialog)
    btn_frame.pack(pady=15)

    tk.Button(btn_frame, text="Annuler", command=dialog.destroy).pack(side="left", padx=5)
    tk.Button(btn_frame, text="Connecter", bg="lightgreen", command=on_connect).pack(side="left",
padx=5)

    dialog.bind("<Escape>", lambda e: dialog.destroy())

    dialog.after(200, lambda: on_scan(auto=True))

    btn_network = tk.Button(top_frame, text=" Réseau", image=window.icons.get('btn_network'),
compound="left", relief="flat", bg="lightblue", command=connect_network)
    btn_network.pack(side="right", padx=5, pady=5)

def umount_all():
    pwd = app_manager.desktop.settings.get("sudo_pwd") if app_manager else None

```

```

if not pwd: return
reseau_dir = os.path.expanduser("~/Réseau")
if not os.path.exists(reseau_dir): return

has_unmounted = False
for root, dirs, files in os.walk(reseau_dir, topdown=False):
    for d in dirs:
        full_path = os.path.join(root, d)
        if os.path.ismount(full_path):
            subprocess.run(['sudo', '-S', 'umount', full_path], input=pwd+'\n', text=True)
            has_unmounted = True

subprocess.run(['sudo', '-S', 'find', reseau_dir, '-type', 'd', '-empty', '-delete'],
input=pwd+'\n', text=True)
if has_unmounted and app_manager:
    app_manager.desktop.show_toast("Tous les partages ont été déconnectés.")

def finish_umount():
    if path_var.get().startswith(os.path.expanduser("~/Réseau")):
        path_var.set(os.path.expanduser("~"))
        refresh_list()

window.after(500, finish_umount)

btn_umount = tk.Button(top_frame, text=" Tout démonter", image=window.icons.get('btn_disconnect'),
compound="left", relief="flat", bg="#ffcccc", command=umount_all)
btn_umount.pack(side="right", padx=5, pady=5)

btn_go = tk.Button(top_frame, text="Aller", relief="flat", command=lambda: refresh_list())
btn_go.pack(side="right", padx=5, pady=5)

show_hidden = tk.BooleanVar(value=False)

list_frame = tk.Frame(window)
list_frame.pack(fill="both", expand=True)

scrollbar = tk.Scrollbar(list_frame)
scrollbar.pack(side="right", fill="y")

style = ttk.Style()
style.configure("Explorateur.Treeview", rowheight=24, font=("Arial", 10))

tree = ttk.Treeview(list_frame, show="tree", yscrollcommand=scrollbar.set,
style="Explorateur.Treeview")
tree.pack(side="left", fill="both", expand=True)
scrollbar.config(command=tree.yview)

def refresh_list(event=None):
    current_path = path_var.get()
    if not os.path.exists(current_path) or not os.path.isdir(current_path):
        return

    tree.delete(*tree.get_children())
    tree.insert("", "end", iid="..", text=" .. (Dossier parent)",
image=window.icons.get('icon_folder'))

    try:
        items = os.listdir(current_path)
        items.sort(key=lambda x: (not os.path.isdir(os.path.join(current_path, x))), x.lower()))
        for item in items:
            if not show_hidden.get() and item.startswith('.'):
                continue
            is_dir = os.path.isdir(os.path.join(current_path, item))
            if is_dir:
                icon = window.icons.get('icon_folder')
            else:
                ext = os.path.splitext(item)[1].lower()
                if ext in ['.png', '.jpg', '.jpeg', '.gif', '.bmp']:
                    icon = window.icons.get('icon_file_img')
                elif ext in ['.txt', '.md', '.log', '.csv']:
                    icon = window.icons.get('icon_file_txt')
                elif ext in ['.py']:

```

```

        icon = window.icons.get('icon_file_py')
    else:
        icon = window.icons.get('icon_file_generic')
    tree.insert("", "end", iid=item, text="  " + item, image=icon)
except PermissionError:
    tree.insert("", "end", iid="ACCESS_DENIED", text="  <Accès refusé>",
image=window.icons.get('icon_file_generic'))

def get_selected_item_path(item_id):
    if not item_id or item_id == ".." or item_id == "ACCESS_DENIED":
        return None
    return os.path.join(path_var.get(), item_id)

def on_double_click(event):
    selection = tree.selection()
    if not selection:
        return

    item_id = selection[0]
    current_path = path_var.get()

    if item_id == "..":
        new_path = os.path.dirname(current_path)
        path_var.set(new_path)
        refresh_list()
    elif item_id == "ACCESS_DENIED":
        return
    else:
        new_path = os.path.join(current_path, item_id)
        if os.path.isdir(new_path):
            path_var.set(new_path)
            refresh_list()
        else:
            ext = os.path.splitext(item_id)[1].lower()
            if ext in ['.png', '.jpg', '.jpeg', '.gif', '.bmp']:
                if app_manager:
                    app_manager.launch_app({"name": "Visionneuse", "module": "apps.visionneuse.app"},
filepath=new_path)
            elif ext in ['.html', '.htm']:
                if app_manager:
                    app_manager.launch_app({"name": "Navigateur Web", "module":
"apps.navigateur.app"}, filepath=new_path)
            elif ext in ['.py', '.json', '.sh', '.c', '.cpp', '.h', '.js', '.css', '.md']:
                if app_manager:
                    app_manager.launch_app({"name": "Éditeur de Code", "module": "apps.editeur.app"},
filepath=new_path)
            else:
                if app_manager:
                    app_manager.launch_app({"name": "Bloc-notes", "module": "apps.blocnotes.app"},
filepath=new_path)

# --- Actions de gestion de fichiers ---
def ask_string_dialog(title, prompt, callback, initialvalue=""):
    dialog_frame = tk.Frame(window, bg="white", highlightbackground="black", highlightthickness=2)
    dialog_frame.place(relx=0.5, rely=0.5, anchor="center", width=300, height=130)

    tk.Label(dialog_frame, text=title, bg="lightgray", font=("Arial", 10, "bold")).pack(fill="x")
    tk.Label(dialog_frame, text=prompt, bg="white").pack(pady=5)

    entry_var = tk.StringVar(value=initialvalue)
    entry = tk.Entry(dialog_frame, textvariable=entry_var)
    entry.pack(padx=10, fill="x")

    def on_ok(event=None):
        val = entry_var.get()
        dialog_frame.destroy()
        if val:
            callback(val)

    def on_cancel(event=None):
        dialog_frame.destroy()

```



```

btn_frame = tk.Frame(dialog_frame, bg="white")
btn_frame.pack(pady=10)
tk.Button(btn_frame, text="OK", command=on_ok, width=8).pack(side="left", padx=5)
tk.Button(btn_frame, text="Annuler", command=on_cancel, width=8).pack(side="left", padx=5)

entry.bind("<Return>", on_ok)
entry.bind("<Escape>", on_cancel)

window.after(100, lambda: entry.focus_set())

def new_file():
    def on_name(name):
        try:
            new_path = os.path.join(path_var.get(), name)
            if not os.path.exists(new_path):
                open(new_path, 'a').close()
                refresh_list()
            else:
                messagebox.showerror("Erreur", "Un fichier avec ce nom existe déjà.")
        except Exception as e:
            messagebox.showerror("Erreur", f"Impossible de créer le fichier :\n{e}")
    ask_string_dialog("Nouveau Fichier", "Nom du nouveau fichier texte :", on_name)

def new_folder():
    def on_name(name):
        try:
            new_path = os.path.join(path_var.get(), name)
            os.makedirs(new_path, exist_ok=False)
            refresh_list()
        except Exception as e:
            messagebox.showerror("Erreur", f"Impossible de créer le dossier :\n{e}")
    ask_string_dialog("Nouveau Dossier", "Nom du nouveau dossier :", on_name)

def rename_item(item_id):
    old_path = get_selected_item_path(item_id)
    if not old_path: return
    old_name = os.path.basename(old_path)

    def on_name(new_name):
        if new_name != old_name:
            try:
                new_path = os.path.join(os.path.dirname(old_path), new_name)
                os.rename(old_path, new_path)
                refresh_list()
            except Exception as e:
                messagebox.showerror("Erreur", f"Impossible de renommer :\n{e}")
    ask_string_dialog("Renommer", "Nouveau nom :", on_name, initialValue=old_name)

def delete_item(item_id):
    target_path = get_selected_item_path(item_id)
    if not target_path: return

    if target_path.startswith(os.path.expanduser("~/Réseau")):
        messagebox.showerror("Interdit", "La suppression est interdite sur les partages réseau montés par sécurité.")
        return

    target_name = os.path.basename(target_path)
    is_dir = os.path.isdir(target_path)

    msg = f"Voulez-vous vraiment supprimer définitivement le {'dossier' if is_dir else 'fichier'} '{target_name}' ?"
    if is_dir:
        msg += "\n\nATTENTION: Tout le contenu du dossier sera supprimé !"

    if messagebox.askyesno("Confirmer la suppression", msg, icon='warning'):
        try:
            if is_dir:
                shutil.rmtree(target_path)
            else:
                os.remove(target_path)

```

```

        refresh_list()
    except Exception as e:
        messagebox.showerror("Erreur", f"Impossible de supprimer :\n{e}")

def create_shortcut(item_id):
    target_path = get_selected_item_path(item_id)
    if not target_path: return

    target_name = os.path.basename(target_path)

    import json
    desktop_config_path =
os.path.join(os.path.dirname(os.path.dirname(os.path.dirname(os.path.abspath(__file__)))), "config",
"desktop.json")
    try:
        icons = []
        if os.path.exists(desktop_config_path):
            with open(desktop_config_path, "r") as f:
                icons = json.load(f)

        x = 20 + (len(icons) % 10) * 80
        y = 20 + (len(icons) // 10) * 80

        icons.append({"name": target_name, "path": target_path, "x": x, "y": y})

        with open(desktop_config_path, "w") as f:
            json.dump(icons, f, indent=4)

        if app_manager and hasattr(app_manager, "desktop"):
            app_manager.desktop.load_desktop_icons()
            app_manager.desktop.show_toast(f"Raccourci '{target_name}' créé.")
    except Exception as e:
        messagebox.showerror("Erreur", f"Impossible de créer le raccourci :\n{e}")

def print_item(item_id):
    target_path = get_selected_item_path(item_id)
    if not target_path or os.path.isdir(target_path): return

    try:
        res = subprocess.run(['lp', target_path], capture_output=True, text=True)
        if res.returncode == 0:
            if app_manager and hasattr(app_manager, "desktop"):
                app_manager.desktop.show_toast(f"Impression lancée pour
'{os.path.basename(target_path)}'")
            else:
                messagebox.showinfo("Impression", f"Fichier envoyé à l'imprimante
:\n{os.path.basename(target_path)}")
            else:
                messagebox.showerror("Erreur Impression", f"Impossible d'imprimer:\n{res.stderr}\n\nAvez-
vous configuré une imprimante par défaut ?")
        except FileNotFoundError:
            messagebox.showerror("Erreur", "La commande 'lp' est introuvable. Installez 'cups' pour
imprimer.")

def copy_item(item_id):
    if not app_manager: return
    target_path = get_selected_item_path(item_id)
    if target_path:
        app_manager.desktop.file_clipboard = {"action": "copy", "path": target_path}
        app_manager.desktop.show_toast(f"Copié : {os.path.basename(target_path)}")

def cut_item(item_id):
    if not app_manager: return
    target_path = get_selected_item_path(item_id)
    if target_path:
        app_manager.desktop.file_clipboard = {"action": "cut", "path": target_path}
        app_manager.desktop.show_toast(f"Coupé : {os.path.basename(target_path)}")

def paste_item():
    if not app_manager or not hasattr(app_manager.desktop, "file_clipboard"): return
    clip = app_manager.desktop.file_clipboard
    if not clip.get("path"): return

```

```

src = clip["path"]
action = clip["action"]
dst_dir = path_var.get()
dst = os.path.join(dst_dir, os.path.basename(src))

if not os.path.exists(src):
    messagebox.showerror("Erreur", "Le fichier source n'existe plus.")
    return

if os.path.exists(dst):
    messagebox.showerror("Erreur", "Un fichier ou dossier de ce nom existe déjà ici.")
    return

try:
    if action == "copy":
        if os.path.isdir(src):
            shutil.copytree(src, dst)
        else:
            shutil.copy2(src, dst)
            app_manager.desktop.show_toast(f"Collé : {os.path.basename(src)}")
    elif action == "cut":
        shutil.move(src, dst)
        app_manager.desktop.file_clipboard = {"action": None, "path": None}
        app_manager.desktop.show_toast(f"Déplacé : {os.path.basename(src)}")
        refresh_list()
except Exception as e:
    messagebox.showerror("Erreur", f"Erreur : \n{e}")

def disconnect_network(m_path):
    pwd = app_manager.desktop.settings.get("sudo_pwd") if app_manager else None
    if not pwd: return

    has_unmounted = False
    if os.path.ismount(m_path):
        subprocess.run(['sudo', '-S', 'umount', m_path], input=pwd+'\n', text=True)
        has_unmounted = True
    else:
        for root, dirs, files in os.walk(m_path, topdown=False):
            for d in dirs:
                full_p = os.path.join(root, d)
                if os.path.ismount(full_p):
                    subprocess.run(['sudo', '-S', 'umount', full_p], input=pwd+'\n', text=True)
                    has_unmounted = True

    if has_unmounted:
        subprocess.run(['sudo', '-S', 'find', m_path, '-type', 'd', '-empty', '-delete'],
            input=pwd+'\n', text=True)
        if not os.path.exists(m_path):
            # Clean up parent if it became empty
            parent = os.path.dirname(m_path)
            subprocess.run(['sudo', '-S', 'find', parent, '-type', 'd', '-empty', '-delete'],
                input=pwd+'\n', text=True)

        if app_manager: app_manager.desktop.show_toast(f"Déconnecté : {os.path.basename(m_path)}")

    def finish_disconnect():
        if not os.path.exists(path_var.get()):
            path_var.set(os.path.expanduser("~/Réseau"))
            refresh_list()

    window.after(500, finish_disconnect)
else:
    messagebox.showerror("Erreur", "Aucun point de montage actif trouvé ici.")

# --- Menu Contextuel ---
context_menu = tk.Menu(window, tearoff=0)

def show_context_menu(event):
    item_id = tree.identify_row(event.y)

    tree.selection_remove(tree.selection())

```

```

context_menu.delete(0, tk.END)

if item_id:
    tree.selection_set(item_id)
    target_path = get_selected_item_path(item_id)

    if item_id not in ("..", "ACCESS_DENIED"):
        context_menu.add_command(label=" Copier", command=lambda: window.after(50, copy_item,
item_id))

        if not target_path.startswith(os.path.expanduser("~/Réseau")):
            context_menu.add_command(label=" Couper", command=lambda: window.after(50, cut_item,
item_id))

            context_menu.add_separator()
            context_menu.add_command(label=" Renommer", image=window.icons.get('menu_rename'),
compound="left", command=lambda: window.after(50, rename_item, item_id))
            context_menu.add_command(label=" Supprimer", image=window.icons.get('menu_delete'),
compound="left", command=lambda: window.after(50, delete_item, item_id))
        else:
            context_menu.add_separator()

        context_menu.add_command(label=" Raccourci Bureau", command=lambda: window.after(50,
create_shortcut, item_id))

        if target_path and os.path.isfile(target_path):
            ext = os.path.splitext(target_path)[1].lower()
            if ext in ['.txt', '.py', '.md', '.json', '.log', '.csv', '.png', '.jpg', '.jpeg']:
                context_menu.add_command(label=" Imprimer", image=window.icons.get('btn_print'),
compound="left", command=lambda: window.after(50, print_item, item_id))

        if target_path and target_path.startswith(os.path.expanduser("~/Réseau")) and target_path
!= os.path.expanduser("~/Réseau"):
            rel_path = os.path.relpath(target_path, os.path.expanduser("~/Réseau"))
            if len(rel_path.split(os.sep)) <= 2:
                context_menu.add_command(label=" Déconnecter",
image=window.icons.get('menu_delete'), compound="left", command=lambda p=target_path: window.after(50,
disconnect_network, p))

            context_menu.add_separator()

        if app_manager and hasattr(app_manager.desktop, "file_clipboard") and
app_manager.desktop.file_clipboard.get("path"):
            context_menu.add_command(label=" Coller", image=window.icons.get('btn_paste'),
compound="left", command=lambda: window.after(50, paste_item))
            context_menu.add_separator()

        context_menu.add_command(label=" Nouveau dossier", image=window.icons.get('menu_new_folder'),
compound="left", command=lambda: window.after(50, new_folder))
        context_menu.add_command(label=" Nouveau fichier texte", image=window.icons.get('menu_new_file'),
compound="left", command=lambda: window.after(50, new_file))

    try:
        context_menu.tk_popup(event.x_root, event.y_root)
    finally:
        context_menu.grab_release()

# Bindings
tree.bind("<Double-Button-1>", on_double_click)
tree.bind("<Button-3>", show_context_menu) # Clic droit

# Fermer le menu si on clique ailleurs
window.bind("<Button-1>", lambda e: context_menu.unpost(), add="+")
tree.bind("<Button-1>", lambda e: context_menu.unpost(), add="+")

path_entry.bind("<Return>", refresh_list)

# Boutons d'accès rapide
btn_go = tk.Button(top_frame, text="Aller", command=refresh_list, relief="flat")
btn_go.pack(side="right", padx=2, pady=2)

btn_term = tk.Button(top_frame, text=">_ Terminal", command=lambda: app_manager.launch_app({"name":
"Terminal", "module": "apps.terminal.app"}, filepath=path_var.get()) if app_manager else None,

```

```
relief="flat", bg="#e0e0e0")
    btn_term.pack(side="right", padx=5, pady=2)

    chk_hidden = tk.Checkbutton(top_frame, text="Fichiers cachés", variable=show_hidden,
command=refresh_list, bg="lightgray")
    chk_hidden.pack(side="right", padx=5)

def show_help():
    msg = (
        "Aide de l'Explorateur\n\n"
        "Navigation :\n"
        "• Double-cliquez sur un dossier pour y entrer.\n"
        "• Double-cliquez sur '.. (Dossier parent)' pour remonter.\n"
        "• Double-cliquez sur un fichier pour l'ouvrir.\n\n"
        "Actions :\n"
        "• Clic-droit sur un élément pour le renommer ou le supprimer.\n"
        "• Le menu Clic-droit permet aussi de créer des fichiers/dossiers."
    )
    messagebox.showinfo("Aide Explorateur", msg)

    btn_help = tk.Button(top_frame, text=" Aide", image=(app_manager.desktop.icons.get("btn_help") if
app_manager else None), compound="left", command=show_help, relief="flat", bg="lightblue")
    btn_help.pack(side="right", padx=5, pady=2)

refresh_list()
```

Annexe : Application - Imprimante

Rôle et utilité

`apps/imprimante/app.py` est une interface minimaliste pour interagir avec le gestionnaire d'impression standard de Linux (CUPS - *Common UNIX Printing System*). Elle permet de lister les imprimantes disponibles et d'envoyer rapidement une page de test pour vérifier la connexion matérielle.

Implémentation technique

- **Interaction avec CUPS via `lpstat`** : Au lieu de coder des drivers complexes, l'application exécute la commande `lpstat -p` en arrière-plan. Elle "parse" (découpe) la sortie texte pour en extraire le nom de chaque imprimante configurée et son état actuel (prête, en veille, ou erreur).
- **Lancement de l'impression** : Lorsqu'on clique sur le bouton de test, le script ne génère pas de document graphique. Il lance la commande système `lpr -P [nom_imprimante] /usr/share/cups/data/testprint`, déclenchant l'impression de la page de test officielle de Linux de façon parfaitement transparente.
- **Statut asynchrone** : Une fonction actualise périodiquement l'état des imprimantes à l'écran, permettant à l'utilisateur de savoir si le câble est bien branché ou si le périphérique est reconnu, sans jamais ouvrir le redouté "terminal".

Pistes de modification

- **Gestion des files d'attente (Spooler)** : On pourrait ajouter un second tableau en dessous affichant le résultat de la commande `lpq`, qui liste les documents actuellement en attente d'impression. Un bouton permettrait alors d'annuler une impression (`cancel [job_id]`).
- **Interface d'impression PDF** : Cette application pourrait être modifiée pour accepter un argument (ex: le chemin d'un fichier PDF). L'explorateur pourrait ainsi faire un "clic-droit >

Imprimer" qui ouvrirait cette fenêtre pour choisir l'imprimante avant d'envoyer le document à CUPS.

Code Source

```
import tkinter as tk
from tkinter import messagebox, simpledialog
import subprocess
import os

def start(window, app_manager=None, **kwargs):
    frame = tk.Frame(window, bg="white")
    frame.pack(fill="both", expand=True)

    icon_dir = os.path.join(os.path.dirname(os.path.dirname(os.path.dirname(os.path.abspath(__file__)))),
"icons")
    window.icons = getattr(window, "icons", {})
    for iname in ['btn_search', 'btn_print', 'btn_refresh']:
        if iname not in window.icons:
            ipath = os.path.join(icon_dir, iname + ".png")
            if os.path.exists(ipath):
                window.icons[iname] = tk.PhotoImage(file=ipath)

    header = tk.Label(frame, text="Gestionnaire d'Impression (CUPS)", font=("Arial", 12, "bold"),
bg="white", fg="blue")
    header.pack(pady=10)

    list_frame = tk.Frame(frame, bg="white")
    list_frame.pack(fill="both", expand=True, padx=10, pady=5)

    listbox = tk.Listbox(list_frame, font=("Arial", 10), height=8)
    listbox.pack(side="left", fill="both", expand=True)

    scrollbar = tk.Scrollbar(list_frame, orient="vertical", command=listbox.yview)
    scrollbar.pack(side="right", fill="y")
    listbox.config(yscrollcommand=scrollbar.set)

    # Actions Frame
    btn_frame = tk.Frame(frame, bg="white")
    btn_frame.pack(fill="x", padx=10, pady=10)

    def refresh_printers():
        listbox.delete(0, tk.END)
        try:
            # Récupérer l'imprimante par défaut
            res_def = subprocess.run(['lpstat', '-d'], capture_output=True, text=True)
            default_printer = ""
            if res_def.returncode == 0 and "destination système par défaut :" in res_def.stdout or
"system default destination:" in res_def.stdout:
                default_printer = res_def.stdout.split(":")[1].strip()

            res = subprocess.run(['lpstat', '-p'], capture_output=True, text=True)
            if res.returncode == 0 and res.stdout.strip():
                for line in res.stdout.split('\n'):
                    if line.startswith("printer"):
                        parts = line.split()
                        if len(parts) >= 2:
                            pname = parts[1]
                            state = " ".join(parts[2:])
                            display_text = f"{pname} - {state}"
                            if pname == default_printer:
                                display_text = f"★ [DÉFAUT] {display_text}"
                            listbox.insert(tk.END, display_text)
            else:
                listbox.insert(tk.END, "Aucune imprimante trouvée ou CUPS non installé.")
```

```

except FileNotFoundError:
    listbox.insert(tk.END, "Erreur: 'lpstat' introuvable. Avez-vous installé 'cups' ?")

def set_default():
    pwd = app_manager.desktop.settings.get("sudo_pwd") if app_manager else None
    if not pwd:
        messagebox.showerror("Erreur", "Veuillez configurer votre mot de passe système dans les Paramètres.")
        return

    selection = listbox.curselection()
    if not selection:
        messagebox.showwarning("Attention", "Veuillez sélectionner une imprimante.")
        return

    item = listbox.get(selection[0])
    printer_name = item.replace("★ [DÉFAUT] ", "").split()[0]

    if printer_name == "Aucune" or printer_name == "Erreur:": return

    try:
        cmd = f"lpadmin -d {printer_name}"
        res = subprocess.run(['sudo', '-S', 'bash', '-c', cmd], input=pwd+'\n', text=True,
capture_output=True)
        if res.returncode == 0:
            messagebox.showinfo("Succès", f"{printer_name} est maintenant l'imprimante par défaut !")
            refresh_printers()
        else:
            messagebox.showerror("Erreur", f"Impossible de définir l'imprimante par défaut
:\n{res.stderr}")
    except Exception as e:
        messagebox.showerror("Exception", str(e))

def print_test():
    selection = listbox.curselection()
    if not selection:
        messagebox.showwarning("Attention", "Veuillez sélectionner une imprimante.")
        return

    item = listbox.get(selection[0])
    printer_name = item.replace("★ [DÉFAUT] ", "").split()[0]

    if printer_name == "Aucune" or printer_name == "Erreur:":
        return

    try:
        # lp -d printer_name /usr/share/cups/data/testprint
        res = subprocess.run(['lp', '-d', printer_name, '/usr/share/cups/data/testprint'],
capture_output=True, text=True)
        if res.returncode == 0:
            messagebox.showinfo("Succès", f"Page de test envoyée à {printer_name}.")
        else:
            messagebox.showerror("Erreur", f"Échec de l'impression:\n{res.stderr}")
    except FileNotFoundError:
        messagebox.showerror("Erreur", "La commande 'lp' est introuvable.")

def add_network_printer():
    pwd = app_manager.desktop.settings.get("sudo_pwd") if app_manager else None
    if not pwd:
        messagebox.showerror("Erreur", "Veuillez configurer votre mot de passe système (Sudo) dans les Paramètres.")
        return

    ip = simpledialog.askstring("Ajouter Imprimante", "Entrez l'adresse IP de l'imprimante réseau
:\n(ex: 192.168.1.50)", parent=window)
    if not ip:
        return

    name = simpledialog.askstring("Nom de l'imprimante", "Donnez un nom court à l'imprimante (sans
espace) :\n(ex: Bureau)", parent=window)
    if not name or " " in name:
        messagebox.showerror("Erreur", "Nom invalide (ne doit pas contenir d'espaces).")

```



```

        return

    # lpadmin -p <name> -v ipp://<ip>/ipp/print -E -m everywhere
    cmd = f"lpadmin -p {name} -v ipp://{ip}/ipp/print -E -m everywhere"
    try:
        res = subprocess.run(['sudo', '-S', 'bash', '-c', cmd], input=pwd+'\n', text=True,
            capture_output=True)
        if res.returncode == 0:
            # Set as default
            subprocess.run(['sudo', '-S', 'bash', '-c', f"lpadmin -d {name}"], input=pwd+'\n',
                text=True, capture_output=True)
            messagebox.showinfo("Succès", f"L'imprimante {name} a été ajoutée avec succès et définie
            par défaut.")
            refresh_printers()
        else:
            # Fallback to simple socket if IPP Everywhere fails (some older printers)
            cmd_fallback = f"lpadmin -p {name} -v socket://{ip} -E"
            res_fb = subprocess.run(['sudo', '-S', 'bash', '-c', cmd_fallback], input=pwd+'\n',
                text=True, capture_output=True)
            if res_fb.returncode == 0:
                messagebox.showinfo("Succès", f"L'imprimante {name} a été ajoutée (Mode basique sans
                driver IPP).")
                refresh_printers()
            else:
                messagebox.showerror("Erreur", f"L'ajout a échoué.\n{res.stderr}\n{res_fb.stderr}")
    except Exception as e:
        messagebox.showerror("Exception", str(e))

def auto_search():
    pwd = app_manager.desktop.settings.get("sudo_pwd") if app_manager else None
    if not pwd:
        messagebox.showerror("Erreur", "Veuillez configurer votre mot de passe système (Sudo) dans
        les Paramètres.")
        return

    messagebox.showinfo("Recherche", "La détection automatique va démarrer (cela peut prendre
    quelques secondes)...")
    try:
        res = subprocess.run(['ippfind'], capture_output=True, text=True)
        if res.returncode == 0 and res.stdout.strip():
            uris = res.stdout.strip().split('\n')
            uri = uris[0] # Prendre la première imprimante trouvée

            name = simpledialog.askstring("Imprimante trouvée", f"Une imprimante a été détectée
            !\nURI: {uri}\n\nDonnez-lui un nom court (sans espace) :", parent=window)
            if not name or " " in name:
                return

            cmd = f"lpadmin -p {name} -v {uri} -E -m everywhere"
            res_add = subprocess.run(['sudo', '-S', 'bash', '-c', cmd], input=pwd+'\n', text=True,
                capture_output=True)
            if res_add.returncode == 0:
                subprocess.run(['sudo', '-S', 'bash', '-c', f"lpadmin -d {name}"], input=pwd+'\n',
                    text=True, capture_output=True)
                messagebox.showinfo("Succès", f"L'imprimante {name} a été configurée et définie par
                défaut !")
                refresh_printers()
            else:
                messagebox.showerror("Erreur", f"L'ajout a échoué.\n{res_add.stderr}")
        else:
            messagebox.showinfo("Résultat", "Aucune imprimante IPP détectée sur le réseau local.")
    except FileNotFoundError:
        messagebox.showerror("Erreur", "L'outil de recherche (ippfind) est introuvable. Installez
        cups-client.")

    btn_add = tk.Button(btn_frame, text="✚ Ajouter (IP)", command=add_network_printer, bg="lightblue")
    btn_add.pack(side="left", padx=5)

    btn_search = tk.Button(btn_frame, text=" Détection Auto", image=window.icons.get('btn_search'),
        compound="left", command=auto_search, bg="#ffeb3b")
    btn_search.pack(side="left", padx=5)

```

```
btn_test = tk.Button(btn_frame, text=" Page de Test", image=window.icons.get('btn_print'),
compound="left", command=print_test, bg="lightgreen")
btn_test.pack(side="left", padx=5)

btn_default = tk.Button(btn_frame, text="★ Définir Défaut", command=set_default, bg="orange")
btn_default.pack(side="left", padx=5)

btn_ref = tk.Button(btn_frame, text=" Actualiser", image=window.icons.get('btn_refresh'),
compound="left", command=refresh_printers)
btn_ref.pack(side="right", padx=5)

# Note d'installation
note = tk.Label(frame, text="Note: L'impression nécessite d'installer les utilitaires système:\nsudo
apt-get install cups cups-client printer-driver-all",
font=("Arial", 8, "italic"), bg="white", fg="gray")
note.pack(pady=5)

refresh_printers()
```

Annexe : Application - Navigateur Web

Rôle et utilité

L'application `apps/navigateur/app.py` permet à l'utilisateur de surfer sur Internet. Cependant, un moteur de rendu HTML/CSS/JS complet étant bien trop lourd pour être codé entièrement en Python dans le cadre de ce projet, cette application agit plutôt comme une "enveloppe" (wrapper) autour d'un navigateur extrêmement léger existant sur Linux.

Implémentation technique

- **Encapsulation via X11 (Reparenting)** : C'est la technique la plus spectaculaire de cette application. Le script lance le navigateur `surf` (un navigateur minimaliste basé sur WebKit) via `subprocess.Popen`. Mais plutôt que de le laisser s'afficher dans sa propre fenêtre flottante gérée par le serveur X, le script utilise des commandes X11 natives (comme `xdotool`) pour capturer la fenêtre de `surf` et l'incruster à l'intérieur de la zone Tkinter générée par GrimOS.
- **Barre d'adresse (UI native)** : La zone supérieure (barre d'adresse, boutons Précédent/Suivant) est codée en pur Tkinter. Lorsqu'on tape une URL, le script envoie un signal logiciel à l'enveloppe `surf` pour qu'il charge la page, donnant l'illusion d'une application unifiée.
- **Presse-papiers asynchrone** : Pour permettre les "Copier-Coller" entre la zone web (gérée par `surf` /X11) et l'environnement GrimOS (géré par Tkinter), un daemon silencieux en arrière-plan synchronise la mémoire X-Selection avec le presse-papiers interne de Python.

Pistes de modification

- **Changement de moteur** : `surf` étant très basique, un développeur pourrait modifier ce fichier pour utiliser un moteur Python natif via `PyQtWebEngine` ou `tkinterweb`. Cela alourdirait l'application, mais offrirait un bien meilleur support du Javascript moderne et éviterait les complexités de l'encapsulation X11.

- **Gestion des favoris (Bookmarks)** : On pourrait facilement ajouter un fichier JSON local `~/.config/grimos_bookmarks.json` et des boutons d'étoiles pour permettre à l'utilisateur de sauvegarder ses sites préférés.

Code Source

```
import tkinter as tk
import subprocess
import os
import json
from urllib.parse import urlparse

def load_bookmarks():
    path = os.path.join(os.path.dirname(__file__), '..', '..', 'config', 'bookmarks.json')
    try:
        with open(path, 'r', encoding='utf-8') as f:
            return json.load(f)
    except:
        return []

def save_bookmarks(bookmarks):
    path = os.path.join(os.path.dirname(__file__), '..', '..', 'config', 'bookmarks.json')
    try:
        os.makedirs(os.path.dirname(path), exist_ok=True)
        with open(path, 'w', encoding='utf-8') as f:
            json.dump(bookmarks, f, indent=4)
    except Exception as e:
        print(f"Erreur favoris: {e}")

def start(window, app_manager=None, filepath=None, **kwargs):
    main_frame = tk.Frame(window, bg="#f0f0f0")
    main_frame.pack(fill="both", expand=True)

    icon_dir = os.path.join(os.path.dirname(os.path.dirname(os.path.dirname(os.path.abspath(__file__)))),
                              "icons")
    window.icons = getattr(window, "icons", {})
    for iname in ['btn_arrow_left', 'btn_home', 'btn_rocket', 'btn_trash']:
        if iname not in window.icons:
            ipath = os.path.join(icon_dir, iname + ".png")
            if os.path.exists(ipath):
                window.icons[iname] = tk.PhotoImage(file=ipath)

    top_bar = tk.Frame(main_frame, bg="#e0e0e0", height=40)
    top_bar.pack(side="top", fill="x", padx=5, pady=5)

    browser_proc = [None]

    bookmarks_btn = tk.Menubutton(top_bar, text="Favoris ▼", font=("Arial", 11, "bold"), relief="flat",
                                   bg="#e0e0e0", activebackground="#ccc", cursor="hand2")
    bookmarks_btn.pack(side="left", padx=5)

    def send_cmd(cmd_str):
        if browser_proc[0] is not None and browser_proc[0].poll() is None:
            try:
                browser_proc[0].stdin.write(cmd_str.encode('utf-8'))
                browser_proc[0].stdin.flush()
            except:
                pass

    btn_back = tk.Button(top_bar, image=window.icons.get('btn_arrow_left'), relief="flat", bg="#e0e0e0",
                          activebackground="#ccc", cursor="hand2", command=lambda: send_cmd("BACK\n"))
    btn_back.pack(side="left", padx=(5, 0))

    btn_home = tk.Button(top_bar, image=window.icons.get('btn_home'), relief="flat", bg="#e0e0e0",
```

```

activebackground="#ccc", cursor="hand2", command=lambda: load_bookmark("https://duckduckgo.com"))
btn_home.pack(side="left", padx=5)

bookmarks_menu = tk.Menu(bookmarks_btn, tearoff=0)
bookmarks_btn.config(menu=bookmarks_menu)

url_var = tk.StringVar(value="https://duckduckgo.com")

def refresh_bookmarks_menu():
    bookmarks_menu.delete(0, tk.END)
    bookmarks_menu.add_command(label="✚ Ajouter la page actuelle", command=add_current_bookmark)

    bookmarks = load_bookmarks()
    if bookmarks:
        del_menu = tk.Menu(bookmarks_menu, tearoff=0)
        for b in bookmarks:
            del_menu.add_command(label=b['name'], command=lambda b_url=b['url']:
delete_bookmark(b_url))
            bookmarks_menu.add_cascade(label=" Supprimer un favori...",
image=window.icons.get('btn_trash'), compound="left", menu=del_menu)
            bookmarks_menu.add_separator()

        for b in bookmarks:
            bookmarks_menu.add_command(label=b['name'], command=lambda b_url=b['url']:
load_bookmark(b_url))

def add_current_bookmark():
    url = url_var.get()
    if not url: return
    try:
        parsed = urlparse(url)
        name = parsed.netloc if parsed.netloc else url
        name = name.replace("www.", "")
    except:
        name = url

    bookmarks = load_bookmarks()
    if not any(b['url'] == url for b in bookmarks):
        bookmarks.append({"name": name, "url": url})
        save_bookmarks(bookmarks)
        refresh_bookmarks_menu()

def delete_bookmark(url):
    bookmarks = load_bookmarks()
    bookmarks = [b for b in bookmarks if b['url'] != url]
    save_bookmarks(bookmarks)
    refresh_bookmarks_menu()

def load_bookmark(url):
    url_var.set(url)
    launch_browser()

refresh_bookmarks_menu()

url_entry = tk.Entry(top_bar, textvariable=url_var, font=("Arial", 12))
url_entry.pack(side="left", fill="x", expand=True, padx=(0, 5))

def show_help():
    from tkinter import messagebox
    msg = (
        "Navigateur GrimOS :\n\n"
        "• Saisissez une URL (ex: duckduckgo.com) ou un chemin local (file:///...) et faites
Entrée.\n"
        "• Utilisez les boutons Retour et Accueil pour naviguer.\n"
        "• Cliquez sur le bouton 'Favoris ▼' pour ajouter ou retrouver vos sites préférés.\n\n"
        "Technique : Ce navigateur utilise WebKitGTK intégré de façon transparente dans l'interface
GrimOS (via Overlay).")
    messagebox.showinfo("Aide du Navigateur", msg)

    btn_help = tk.Button(top_bar, text=" ? Aide ", font=("Arial", 10), bg="#ccc", relief="flat",
cursor="hand2", command=show_help)

```

```

btn_help.pack(side="right", padx=5)

# The container that the GTK window will overlay
# pady=(0, 15) leaves a 15px gap at the bottom so the GrimOS resize grip is visible
web_container = tk.Frame(main_frame, bg="#e0e0e0")
web_container.pack(side="bottom", fill="both", expand=True, pady=(0, 15))

lbl_inactive = tk.Label(web_container, text="Cliquez pour activer le navigateur", bg="#e0e0e0",
fg="#888", font=("Arial", 12))
lbl_inactive.pack(expand=True)

def lift_window(event):
    window.master.lift()

web_container.bind("<Button-1>", lift_window)
lbl_inactive.bind("<Button-1>", lift_window)

def track_browser_position():
    if browser_proc[0] is not None and browser_proc[0].poll() is None:
        try:
            # Check Z-order to hide GTK when behind other windows
            windows = window.master.master.winfo_children()
            is_topmost = False
            if windows and windows[-1] == window.master:
                is_topmost = True

            if is_topmost:
                root_x = web_container.winfo_rootx()
                root_y = web_container.winfo_rooty()
                w = web_container.winfo_width()
                h = web_container.winfo_height()

                if w > 10 and h > 10:
                    send_cmd(f"MOVE {root_x} {root_y} {w} {h}\n")
                else:
                    # Move off-screen when not active
                    send_cmd(f"MOVE -9999 -9999 10 10\n")
        except Exception:
            pass
    window.after(30, track_browser_position)

def launch_browser(event=None):
    url = url_var.get()
    if not url.startswith("http") and not url.startswith("file://"):
        url = "https://" + url
    url_var.set(url)

    if browser_proc[0] is not None and browser_proc[0].poll() is None:
        send_cmd(f"LOAD {url}\n")
        window.master.lift()
        return

    engine_script = os.path.join(os.path.dirname(__file__), "browser_engine.py")

    cmd = [
        'python3', engine_script,
        '--url', url
    ]

    browser_proc[0] = subprocess.Popen(cmd, stdin=subprocess.PIPE, stdout=subprocess.PIPE,
stderr=open("/tmp/webkit_browser.log", "w"))

import threading
def read_stdout():
    try:
        for line in iter(browser_proc[0].stdout.readline, b''):
            line_str = line.decode('utf-8').strip()
            if line_str.startswith("URL "):
                new_url = line_str[4:]
                window.after(0, lambda: url_var.set(new_url))
            elif line_str.startswith("DOWNLOAD_START "):
                path = line_str[15:]

```

```

        if app_manager and hasattr(app_manager, 'desktop'):
            window.after(0, lambda p=path:
app_manager.desktop.show_toast(f"Téléchargement démarré :\n{p}"))
        elif line_str.startswith("DOWNLOAD_FINISH"):
            if app_manager and hasattr(app_manager, 'desktop'):
                window.after(0, lambda: app_manager.desktop.show_toast("Téléchargement
terminé. "))
            except:
                pass

        threading.Thread(target=read_stdout, daemon=True).start()

        # Start tracking and overlaying the GTK window
        window.master.lift()
        track_browser_position()

        url_entry.bind("<Return>", launch_browser)

        btn_launch = tk.Button(top_bar, text="Aller", image=window.icons.get('btn_rocket'), compound="left",
command=launch_browser, font=("Arial", 10, "bold"), bg="#2196F3", fg="white", relief="flat")
        btn_launch.pack(side="right")

    def on_destroy(event):
        if str(event.widget) == str(window) and browser_proc[0] is not None:
            try:
                browser_proc[0].stdin.close() # This tells GTK to quit gracefully
                browser_proc[0].kill()
            except:
                pass

    window.bind("<Destroy>", on_destroy)

    if filepath:
        if filepath.startswith("http") or filepath.startswith("file://"):
            url_var.set(filepath)
        else:
            url_var.set(f"file://{os.path.abspath(filepath)}")

    window.after(200, launch_browser)

```

Annexe : Application - Paramètres

Rôle et utilité

`apps/parametres/app.py` est le panneau de contrôle de GrimOS. C'est l'interface graphique qui permet à l'utilisateur de modifier le fond d'écran ou de changer le thème global du système (couleurs et bordures) sans avoir à ouvrir de fichiers de configuration manuellement.

Implémentation technique

- **Lecture du registre de Thèmes** : Le script importe directement la variable `THEMES` depuis `core.theme` pour récupérer la liste de tous les thèmes visuels disponibles (Win 98, Ubuntu, etc.) et les injecte dans une liste déroulante Tkinter (`ttk.Combobox`).
- **Interaction avec `config.py`** : Lorsqu'une modification est validée, l'application utilise la fonction `set_setting("theme", nouveau_theme)` du module système `core.config`, garantissant ainsi que la modification sera sauvegardée de façon persistante sur le disque dur.
- **Rechargement à chaud (Hot-Reload)** : Après avoir sauvegardé, l'application exécute un appel direct à la fonction de mise à jour du bureau (`app_manager.desktop.update_desktop_bg()`) et au moteur de thème pour que les couleurs de l'écran changent instantanément, sans qu'il soit nécessaire de redémarrer la session.

Pistes de modification

- **Gestion de l'heure et de la date** : Ajouter un onglet permettant d'utiliser la commande système `date -s` (avec `sudo` via le mot de passe stocké) pour régler l'horloge de l'ordinateur, ou pour configurer un serveur NTP.
- **Options d'alimentation** : On pourrait y ajouter des curseurs permettant de configurer la luminosité de l'écran (en modifiant `/sys/class/backlight`) ou de gérer le délai avant la mise en veille de la machine (via `xset`).

Code Source


```

import tkinter as tk
from tkinter import messagebox
from tkinter import colorchooser
import json
import os

def start(window, app_manager=None, **kwargs):
    frame = tk.Frame(window, bg="white")
    frame.pack(fill="both", expand=True, padx=10, pady=10)

    tk.Label(frame, text="Paramètres du Système", font=("Arial", 14, "bold"), bg="white").pack(pady=10)
    desktop = app_manager.desktop

    # --- THÈMES ---
    tk.Label(frame, text="Thème de l'interface :", font=("Arial", 12, "bold"), bg="white").pack(pady=5)

    import sys, os
    sys.path.append(os.path.dirname(os.path.dirname(os.path.dirname(os.path.abspath(__file__)))))
    from core.theme import THEMES

    theme_var = tk.StringVar(value=desktop.settings.get("theme", "GrimOS"))
    theme_frame = tk.Frame(frame, bg="white")
    theme_frame.pack(fill="x", pady=5)

    theme_menu = tk.OptionMenu(theme_frame, theme_var, *THEMES.keys())
    theme_menu.pack(side="left", padx=5)

    def apply_theme_setting():
        desktop.settings["theme"] = theme_var.get()
        try:
            settings_path = os.path.join(
                os.path.dirname(os.path.dirname(os.path.dirname(os.path.abspath(__file__)))),
                "config", "settings.json"
            )
            with open(settings_path, "w", encoding="utf-8") as f:
                json.dump(desktop.settings, f, indent=4)

            desktop.restart_requested = True
            desktop.save_and_quit()
        except Exception as e:
            import traceback
            with open("/tmp/grimos_theme_error.txt", "w") as f:
                f.write(traceback.format_exc())
            messagebox.showerror("Erreur", f"Erreur détaillée sauvegardée dans /tmp/grimos_theme_error.txt\n{str(e)}")

    tk.Button(theme_frame, text=" Appliquer", image=(app_manager.desktop.icons.get("btn_apply") if
    app_manager else None), compound="left", command=apply_theme_setting).pack(side="left", padx=5)

    tk.Frame(frame, height=2, bg="black").pack(fill="x", pady=10)
    tk.Label(frame, text="Sécurité / Système :", font=("Arial", 12, "bold"), bg="white").pack(pady=5)

    sudo_frame = tk.Frame(frame, bg="white")
    sudo_frame.pack(fill="x", pady=2)
    tk.Label(sudo_frame, text="Mot de passe Sudo :", bg="white").pack(side="left", padx=5)

    sudo_entry = tk.Entry(sudo_frame, width=15)
    sudo_entry.pack(side="left", padx=5)
    sudo_entry.insert(0, desktop.settings.get("sudo_pwd", ""))

    def save_sudo():
        desktop.settings["sudo_pwd"] = sudo_entry.get()
        try:
            settings_path = os.path.join(
                os.path.dirname(os.path.dirname(os.path.dirname(os.path.abspath(__file__)))),
                "config", "settings.json"
            )
            with open(settings_path, "w", encoding="utf-8") as f:
                json.dump(desktop.settings, f, indent=4)
            messagebox.showinfo("Succès", "Mot de passe système (Sudo) enregistré avec succès.")
        except Exception as e:

```

```

        messagebox.showerror("Erreur", str(e))

    tk.Button(sudo_frame, text=" Enregistrer", image=(app_manager.desktop.icons.get("menu_save") if
app_manager else None), compound="left", command=save_sudo).pack(side="left", padx=5)

    cpu_frame = tk.Frame(frame, bg="white")
    cpu_frame.pack(fill="x", pady=2)
    tk.Label(cpu_frame, text="Largeur Graphique CPU :", bg="white").pack(side="left", padx=5)

    cpu_entry = tk.Entry(cpu_frame, width=5)
    cpu_entry.pack(side="left", padx=5)
    cpu_entry.insert(0, str(desktop.settings.get("cpu_graph_width", 10)))

def save_cpu_width():
    try:
        val = int(cpu_entry.get())
        if val < 1 or val > 50:
            raise ValueError("La valeur doit être entre 1 et 50.")
        desktop.settings["cpu_graph_width"] = val
        settings_path = os.path.join(
            os.path.dirname(os.path.dirname(os.path.abspath(__file__))),
            "config", "settings.json"
        )
        with open(settings_path, "w", encoding="utf-8") as f:
            json.dump(desktop.settings, f, indent=4)
        messagebox.showinfo("Succès", "Largeur du graphique CPU enregistrée.\nL'effet sera immédiat
sur la barre des tâches.")
    except Exception as e:
        messagebox.showerror("Erreur", str(e))

    tk.Button(cpu_frame, text=" Enregistrer", image=(app_manager.desktop.icons.get("menu_save") if
app_manager else None), compound="left", command=save_cpu_width).pack(side="left", padx=5)

```

Annexe : Application - Gestionnaire des tâches (TaskMgr)

Rôle et utilité

L'application `apps/taskmgr/app.py` permet à l'utilisateur de surveiller les processus en cours d'exécution sur la machine et de "tuer" (fermer de force) les applications qui ne répondent plus. C'est l'équivalent du classique "Ctrl+Alt+Suppr" sous Windows.

Implémentation technique

- **Analyse du système (Lecture de `/proc`)** : Plutôt que d'utiliser des commandes externes lourdes comme `ps` ou `top`, le script lit directement les données brutes dans le répertoire `/proc/` de Linux. Chaque dossier numéroté dans `/proc/` correspond à un processus. Le script extrait le nom et la consommation mémoire de chaque processus à la vitesse de l'éclair.
- **Boucle de rafraîchissement** : L'interface graphique contient un tableau (Treeview) qui est vidé et repeuplé toutes les 2 secondes grâce à un appel asynchrone (`window.after(2000, refresh)`).
- **Terminaison de processus** : Lorsqu'un utilisateur sélectionne une ligne et clique sur "Terminer", l'application extrait le PID (Identifiant de Processus) et exécute la commande native `kill -9 [PID]` pour détruire immédiatement l'application bloquée.

Pistes de modification

- **Graphiques dynamiques** : Le gestionnaire de tâches actuel affiche une simple liste textuelle. On pourrait y intégrer le composant de dessin (`Canvas`) utilisé dans le `desktop.py` pour afficher l'historique de consommation CPU/RAM de l'ordinateur sur les 60 dernières secondes, sous forme de courbes.
- **Filtrage intelligent** : Actuellement, le gestionnaire affiche tous les processus du système Linux, y compris les démons internes qui peuvent intimider l'utilisateur. Ajouter une case

à cocher "Ne montrer que mes applications" pour masquer tout ce qui n'appartient pas au processus de l'utilisateur principal serait un vrai plus ergonomique.

Code Source

```
import tkinter as tk
from tkinter import ttk, messagebox
import os
import signal

def start(window, app_manager=None, **kwargs):
    top_frame = tk.Frame(window, bg="lightgray")
    top_frame.pack(side="top", fill="x")

    btn_kill = tk.Button(top_frame, text=" Fin de tâche", image=
(app_manager.desktop.icons.get("btn_kill") if app_manager else None), compound="left", bg="red",
fg="white", font=("Arial", 9, "bold"))
    btn_kill.pack(side="left", padx=5, pady=5)

    btn_refresh = tk.Button(top_frame, text=" Actualiser", image=
(app_manager.desktop.icons.get("btn_refresh") if app_manager else None), compound="left", font=("Arial",
9))
    btn_refresh.pack(side="left", padx=5, pady=5)

    def show_help():
        msg = (
            "Aide du Gestionnaire des Tâches\n\n"
            "• Cette liste affiche les processus en cours sur le système.\n"
            "• Le bouton rouge 'Fin de tâche' envoie un signal d'arrêt immédiat (kill) au programme.\n\n"
            "⚠ Attention : Tuer un processus vital (comme systemd, xorg, ou dbus) causera "
            "le plantage immédiat de l'interface ou du système."
        )
        messagebox.showinfo("Aide Gestionnaire", msg)

    btn_help = tk.Button(top_frame, text=" Aide", image=(app_manager.desktop.icons.get("btn_help") if
app_manager else None), compound="left", command=show_help, relief="flat", bg="lightblue")
    btn_help.pack(side="right", padx=5, pady=2)

    tree_frame = tk.Frame(window)
    tree_frame.pack(fill="both", expand=True)

    scrollbar = tk.Scrollbar(tree_frame)
    scrollbar.pack(side="right", fill="y")

    tree = ttk.Treeview(tree_frame, columns=("PID", "Name", "State"), show="headings",
yscrollcommand=scrollbar.set)
    tree.heading("PID", text="PID")
    tree.heading("Name", text="Nom du processus")
    tree.heading("State", text="État")
    tree.column("PID", width=60, anchor="center")
    tree.column("Name", width=250)
    tree.column("State", width=60, anchor="center")
    tree.pack(side="left", fill="both", expand=True)
    scrollbar.config(command=tree.yview)

    def refresh():
        for i in tree.get_children():
            tree.delete(i)

        try:
            pids = [pid for pid in os.listdir('/proc') if pid.isdigit()]
            pids.sort(key=int)
            for pid_str in pids:
                try:
                    with open(f"/proc/{pid_str}/stat", "r") as f:
```

```

        stat = f.read().split()
        # stat[1] is the process name, usually in parentheses like (systemd)
        name = stat[1].strip("(")")
        state = stat[2]
        tree.insert("", "end", values=(pid_str, name, state))
    except Exception:
        continue
except Exception as e:
    messagebox.showerror("Erreur", f"Impossible de lire /proc:\n{e}")

def kill_proc():
    selected = tree.selection()
    if not selected:
        return
    item = tree.item(selected[0])
    pid = int(item['values'][0])
    name = item['values'][1]

    # Prevent killing vital system processes lightly
    if pid in (1, 2) or "xorg" in str(name).lower() or "python" in str(name).lower():
        if not messagebox.askyesno("Attention Système", f"Tuer '{name}' (PID: {pid}) pourrait crasher
le système ! Continuer ?"):
            return

    if messagebox.askyesno("Confirmer", f"Voulez-vous forcer l'arrêt de '{name}' (PID: {pid}) ?"):
        try:
            os.kill(pid, signal.SIGKILL)
            window.after(500, refresh)
        except PermissionError:
            messagebox.showerror("Accès refusé", f"Vous n'avez pas les droits pour tuer ce processus
(PID: {pid}).")
        except Exception as e:
            messagebox.showerror("Erreur", f"Impossible de terminer le processus:\n{e}")

btn_refresh.config(command=refresh)
btn_kill.config(command=kill_proc)

refresh()

```

Annexe : Application - Terminal

Rôle et utilité

L'application `apps/terminal/app.py` offre un accès visuel à la ligne de commande à l'intérieur même de GrimOS. Idéal pour les développeurs, il permet d'exécuter des commandes Git, des pings, ou d'installer des paquets sans avoir à quitter l'environnement graphique via le bouton "Fermer la session".

Implémentation technique

- **Boucle d'Écoute Entrée/Sortie (I/O)** : Créer un terminal graphique en Python est notoirement complexe. Plutôt que de coder un véritable émulateur pseudo-terminal (PTY) complet de zéro, l'application lance un processus `bash` en arrière-plan (`subprocess.Popen` avec `stdin=PIPE, stdout=PIPE`).
- **Redirection d'affichage** : Un `Thread` Python (processus léger parallèle) est lancé en arrière-plan pour "écouter" en continu les réponses du bash caché. Dès que bash répond (ex: à un `ls`), le Thread capture ce texte et l'injecte dans le composant d'affichage `tk.Text` de la fenêtre.
- **Interception clavier** : La saisie est gérée minutieusement. Si l'utilisateur appuie sur `Entrée` , le texte qu'il vient de taper est extrait et poussé dans le "tuyau" (pipe) d'entrée du bash caché, puis effacé de l'écran pour laisser place au résultat de la commande.

Pistes de modification

- **Support des programmes interactifs** : L'implémentation actuelle via `subprocess` classique gère mal les programmes qui nécessitent des interactions continues au clavier (comme `nano` , `htop` ou la saisie d'un mot de passe SSH caché). Le remplacement du backend par une véritable implémentation pseudo-terminale (`pty.fork()` ou `os.openpty()`) rendrait ce terminal compatible à 100% avec les standards UNIX.
- **Historique de commandes** : Mémoriser les 50 dernières commandes tapées dans une liste et permettre de naviguer dedans avec les Flèches Haut/Bas, comme dans un vrai

terminal Bash.

Code Source

```
import tkinter as tk
import subprocess
import threading
import os
import signal

from tkinter import messagebox

def start(window, app_manager=None, filepath=None, **kwargs):
    top_frame = tk.Frame(window, bg="lightgray")
    top_frame.pack(side="top", fill="x")

    def show_help():
        msg = (
            "Aide du Terminal GrimOS\n\n"
            "• Ce terminal est un environnement émulé en Python.\n"
            "• Flèche Haut / Bas : Naviguer dans l'historique.\n"
            "• Le copier-coller standard (Ctrl+C / Ctrl+V) est supporté.\n"
            "• La commande 'sudo' est désactivée par sécurité. "
            "Utilisez 'Xterm' pour l'administration."
        )
        messagebox.showinfo("Aide Terminal", msg)

    btn_help = tk.Button(top_frame, text=" ? Aide ", command=show_help, relief="flat", bg="lightblue")
    btn_help.pack(side="right", padx=5, pady=2)

    text_area = tk.Text(window, bg="black", fg="white", font=("Consolas", 10), insertbackground="white")
    text_area.pack(fill="both", expand=True)

    current_process = None
    command_history = []
    history_index = 0

    if filepath and os.path.isdir(filepath):
        current_cwd = filepath
    else:
        current_cwd = os.path.expanduser("~")

    def print_prompt():
        home = os.path.expanduser("~")
        display_cwd = current_cwd
        if display_cwd.startswith(home):
            display_cwd = "~" + display_cwd[len(home):]

        text_area.insert(tk.END, f"geo@grimos:{display_cwd}$ ")
        text_area.mark_set("input_start", "insert")
        text_area.mark_gravity("input_start", "left")
        text_area.see(tk.END)

    print_prompt()
    text_area.focus_set()

    def on_key_press(event):
        nonlocal current_process, history_index

        # Ctrl+C
        if event.state & 4 and event.keysym.lower() == "c":
            if current_process is not None and current_process.poll() is None:
                try:
                    os.killpg(os.getpgid(current_process.pid), signal.SIGTERM)
                except Exception:
                    pass
```

```

        text_area.config(state=tk.NORMAL)
        text_area.insert(tk.END, "^C\n")
        text_area.see(tk.END)
        return "break"
    return

# Allow copy/paste via keyboard
if event.state & 4:
    return

if event.keysym == "Return":
    command = text_area.get("input_start", "end-1c")
    text_area.mark_set("insert", tk.END)
    text_area.insert(tk.END, "\n")

    if command.strip():
        if not command_history or command_history[-1] != command:
            command_history.append(command)
            history_index = len(command_history)

            text_area.config(state=tk.DISABLED)
            threading.Thread(target=execute_in_background, args=(command,), daemon=True).start()
        else:
            print_prompt()
            return "break"

elif event.keysym in ("BackSpace", "Left"):
    if text_area.compare("insert", "<=", "input_start"):
        return "break"

elif event.keysym == "Up":
    if command_history and history_index > 0:
        history_index -= 1
        text_area.delete("input_start", "end")
        text_area.insert("input_start", command_history[history_index])
        return "break"

elif event.keysym == "Down":
    if command_history and history_index < len(command_history):
        history_index += 1
        text_area.delete("input_start", "end")
        if history_index < len(command_history):
            text_area.insert("input_start", command_history[history_index])
        return "break"

elif event.keysym in ("Prior", "Next"):
    return "break"

def on_any_key(event):
    if event.char and event.keysym not in ("BackSpace", "Return", "Left", "Right", "Up", "Down"):
        if text_area.compare("insert", "<", "input_start"):
            text_area.mark_set("insert", tk.END)

def execute_in_background(command):
    nonlocal current_process, current_cwd

    cmd_stripped = command.strip()
    cmd_parts = cmd_stripped.split()

    # Interception de cd
    if cmd_parts and cmd_parts[0] == "cd" and "&&" not in cmd_stripped and ";" not in cmd_stripped:
        target_dir = cmd_parts[1] if len(cmd_parts) > 1 else os.path.expanduser("~")
        new_cwd = os.path.abspath(os.path.join(current_cwd, target_dir))
        if os.path.isdir(new_cwd):
            current_cwd = new_cwd
            window.after(0, append_output, "", "")
        else:
            window.after(0, append_output, "", f"cd: {target_dir}: Aucun fichier ou dossier de ce
type\n")
        return

# Interdire sudo explicitement

```



```

    if cmd_stripped.startswith("sudo ") or cmd_stripped == "sudo":
        msg = "Erreur : L'usage de 'sudo' est désactivé dans ce terminal pour des raisons de sécurité  
et de stabilité.\n"
        msg += "Si vous avez besoin des droits administrateur, veuillez 'Fermer la session' depuis le  
Menu Démarrer pour retourner au vrai terminal Linux.\n"
        msg += "Une fois vos tâches terminées, retapez simplement 'startx'.\n"
        window.after(0, append_output, "", msg)
        return

    try:
        current_process = subprocess.Popen(
            command,
            shell=True,
            cwd=current_cwd,
            stdin=subprocess.DEVNULL,
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
            text=True,
            start_new_session=True # Detach from TTY so sudo fails immediately instead of hanging
        )
        stdout, stderr = current_process.communicate()
        window.after(0, append_output, stdout, stderr)
    except Exception as e:
        window.after(0, append_output, "", str(e))
    finally:
        current_process = None

def append_output(stdout, stderr):
    text_area.config(state=tk.NORMAL)
    if stdout:
        text_area.insert(tk.END, stdout)
    if stderr:
        text_area.insert(tk.END, stderr)

    if (stdout and not stdout.endswith('\n')) or (stderr and not stderr.endswith('\n')):
        text_area.insert(tk.END, "\n")

    print_prompt()

text_area.bind("<KeyPress>", on_key_press)
text_area.bind("<Key>", on_any_key, add="+")

```

Annexe : Application - Visionneuse d'Images

Rôle et utilité

L'application `apps/visionneuse/app.py` est appelée par l'Explorateur lorsque l'utilisateur double-clique sur un fichier image (`.png` , `.jpg` , `.gif`). Elle permet d'afficher l'image de façon proportionnelle tout en autorisant le zoom et le déplacement panoramique (cliquer-glisser).

Implémentation technique

- **Pillow (PIL) au cœur du rendu** : Tkinter natif ne sait lire que des formats d'images basiques (comme le GIF ou des vieux formats PPM). C'est pourquoi la visionneuse utilise la puissante librairie Python `Pillow` (`PIL.Image` , `PIL.ImageTk`) capable de décoder n'importe quel format (JPEG, WebP, etc.).
- **Le Canvas comme surface de dessin** : Plutôt que de mettre l'image dans une simple étiquette statique (`Label`), l'image est peinte (`create_image`) sur un `tk.Canvas` . Cela permet d'avoir un contrôle total sur ses coordonnées X et Y à l'intérieur de la fenêtre.
- **Zoom Mathématique** : Lorsque l'utilisateur utilise la molette de la souris (`<MouseWheel>` ou `<Button-4>/<Button-5>`), le script multiplie les dimensions actuelles de l'image par un ratio (ex: `1.1` pour agrandir, `0.9` pour rétrécir). Il redemande ensuite à Pillow de recalculer l'image (`Image.ANTIALIAS`) pour éviter que les pixels ne bavent trop, puis remplace l'ancienne image du Canvas par la nouvelle.

Pistes de modification

- **Boutons de rotation** : Ajouter une petite barre d'outils en bas de l'écran avec deux boutons permettant de faire tourner l'image de 90° vers la droite ou vers la gauche (`image.rotate()`), et un bouton pour la sauvegarder dans son nouveau sens.
- **Diaporama (Slideshow)** : Si l'utilisateur clique sur un bouton "Diaporama", le script pourrait analyser le dossier actuel, lister toutes les images, et utiliser un appel

asynchrone (`window.after(3000)`) pour charger et afficher automatiquement la photo suivante toutes les 3 secondes.

Code Source

```
import tkinter as tk
from tkinter import messagebox
from PIL import Image, ImageTk
import os

def start(window, app_manager=None, **kwargs):
    frame = tk.Frame(window, bg="black")
    frame.pack(fill="both", expand=True)

    def show_help():
        msg = (
            "Aide de la Visionneuse\n\n"
            "• Cette application affiche les images redimensionnées pour s'adapter à la fenêtre.\n"
            "• Redimensionnez la fenêtre pour agrandir l'image.\n"
            "• Fermez avec la croix rouge pour quitter."
        )
        messagebox.showinfo("Aide Visionneuse", msg)

    btn_help = tk.Button(frame, text=" ? Aide ", command=show_help, relief="flat", bg="lightblue")
    btn_help.place(x=5, y=5)

    lbl_img = tk.Label(frame, bg="black")
    lbl_img.pack(fill="both", expand=True)

    filepath = kwargs.get("filepath")

    def update_title():
        if hasattr(window, 'master') and window.master.__class__.__name__ == 'Window':
            title_text = filepath if filepath else "Aucune image"
            window.master.title_label.config(text=f"Visionneuse - {title_text}")

    update_title()

    if not filepath or not os.path.exists(filepath):
        lbl_img.config(text="Fichier introuvable ou aucune image sélectionnée", fg="white")
        return

    try:
        # Initial window sizing based on image
        img = Image.open(filepath)
        img_w, img_h = img.size

        if hasattr(window, 'master') and window.master.__class__.__name__ == 'Window':
            win_obj = window.master
            parent_w = win_obj.parent.winfo_width() or win_obj.parent.winfo_screenwidth()
            parent_h = win_obj.parent.winfo_height() or win_obj.parent.winfo_screenheight()

            max_w = int(parent_w * 0.8)
            max_h = int(parent_h * 0.8)

            target_w = min(img_w, max_w)
            target_h = min(img_h, max_h)

            target_w = max(400, target_w)
            target_h = max(300, target_h)

            win_obj.place(width=target_w, height=target_h + 30)
    except Exception as e:
        print(f"Erreur initialisation image: {e}")
```

```
def load_image(event=None):
    try:
        # Assure the frame layout is calculated
        frame.update_idletasks()
        w = frame.winfo_width()
        h = frame.winfo_height()

        # Use reasonable default if too small
        if w <= 10 or h <= 10:
            w, h = 400, 300

        img = Image.open(filepath)
        img.thumbnail((w, h), Image.Resampling.LANCZOS)
        photo = ImageTk.PhotoImage(img)
        lbl_img.config(image=photo)
        lbl_img.image = photo # Prevent garbage collection
    except Exception as e:
        lbl_img.config(text=f"Erreur de chargement:\n{e}", fg="red")

# Load on resize, but debounce or just once on start
frame.bind("<Configure>", lambda e: window.after(50, load_image))
window.after(50, load_image)
```