Procédure de build Windows — Garage v4.4.5

Document de référence pour produire un exécutable Windows propre, reproductible et distribuable.

---

1 Pré-requis (une seule fois)

- Windows 10 ou 11
- Python installé (**3.10+ recommandé**)
- Tkinter inclus (standard avec Python Windows)
- PowerShell

Vérification rapide :

```
python --version
```

---

2 Arborescence attendue du projet

À la racine du dépôt Git :

```
garage/
├─ garage.py
├─ assets/
│   ├─ logo.png          ← logo de l'application
│   ├─ logo.ico          ← icône Windows
│   └─ AIDE.md           ← aide embarquée (obligatoire)
├─ data/
│   └─ garage_empty.db   ← base vierge distribuée
├─ .venv/                ← environnement virtuel (local)
├─ requirements.txt
├─ README.md
└─ .gitignore
```

Ne jamais inclure dans le repo

- `garage.db` réel
- photos véhicules personnelles
- données utilisateur

---

3 Environnement virtuel (par machine)

Dans le dossier du projet :

```
python -m venv .venv
.venv\Scripts\Activate.ps1
```

Installer les dépendances nécessaires au build :

```
pip install --upgrade pip
```

```
pip install pillow matplotlib pyinstaller
```

4□Test AVANT build (obligatoire)

Toujours avec le venv activé :

```
python garage.py
```

Vérifier impérativement :

- l'application démarre
- aucune erreur en console
- onglet **Entretiens** fonctionnel
- l'aide s'affiche correctement

✕Si ça bug ici → on ne build pas.

5□Nettoyage avant build (recommandé)

Depuis la racine du projet :

```
Remove-Item -Recurse -Force .\build, .\dist, .\Garage.spec -ErrorAction
SilentlyContinue
```

6□Build PyInstaller (Windows)

Commande officielle :

```
pyinstaller `
  --name Garage `
  --onefile `
  --windowed `
  --icon assets\logo.ico `
  --add-data "assets;assets" `
  --add-data "data;data" `
  garage.py
```

7□Résultat attendu

Exécutable généré :

```
dist\Garage.exe
```

Au **premier lancement**, l'application :

- crée `%APPDATA%\Garage\`
- copie `garage_empty.db` → `garage.db`

- copie `assets\AIDE.md` → dossier utilisateur

8□Vérifications finales

Lancer :

```
.\dist\Garage.exe
```

Contrôler :

- icône visible (explorateur + barre des tâches)
- `%APPDATA%\Garage\garage.db` présent
- `%APPDATA%\Garage\AIDE.md` présent
- aide fonctionnelle dans l'application

9□Distribution

- placer `Garage.exe` dans un dossier `Apps/` (hors repo)
- zipper l'exécutable
- publier la release (GitHub Releases recommandé)
- aucune installation requise côté utilisateur
- données utilisateur isolées par Windows

→ Bonnes pratiques (rappel)

- build **uniquement** depuis un venv
- code figé avant build
- `assets/` et `data/` toujours embarqués
- jamais de données personnelles dans le repo
- si un build échoue → lire **les dernières lignes PyInstaller**

→ Nettoyage propre du repo Garage

1□Éléments à ne jamais versionner

✕À exclure absolument :

```
.venv/
__pycache__/
build/
```

```
dist/
*.spec
garage.db
```

2️⃣ `.gitignore` minimum requis

```
# Python
__pycache__/
*.pyc

# Virtual env
.venv/

# PyInstaller
build/
dist/
*.spec

# OS
.DS_Store
Thumbs.db
desktop.ini

# Local databases
garage.db
```

3️⃣ Vérifier que Git est clean

```
git status
```

Résultat attendu :

```
nothing to commit, working tree clean
```

Si des artefacts apparaissent malgré tout :

```
git rm -r --cached .venv build dist __pycache__
git rm --cached *.spec
git commit -m "chore: clean repo (remove build artifacts, venv, caches)"
```

4️⃣ Nettoyage disque (hors Git)

```
Remove-Item -Recurse -Force .\build, .\dist, .\__pycache__ -ErrorAction
SilentlyContinue
Remove-Item -Force .\Garage.spec -ErrorAction SilentlyContinue
```

Optionnel (reset total) :

```
Remove-Item -Recurse -Force .\.venv
```

5□Règle d’or pour les releases

Un build = **artefact**, pas du code.

Avant chaque release :

1. `git status`
2. build
3. zip
4. GitHub Release

→ Notes Windows (venv)

Windows **ne garde pas le venv actif**,
il garde simplement le dossier `.venv`.

Illusion fréquente due à :

- VS Code (activation automatique)
- profil PowerShell
- travail toujours dans le même dossier

Commandes utiles :

```
deactivate
python -c "import sys; print(sys.prefix)"
```

Bonus : Script ,psi « build-Windows »

```
<#  build-windows.ps1 — Garage (Windows) v4.4.5
    Script de build Windows (portable) pour un repo multi-OS.

    Usage (PowerShell, à la racine du repo) :
      .\build-windows.ps1
      .\build-windows.ps1 -Version 4.4.5
      .\build-windows.ps1 -AppsDir "$env:USERPROFILE\Apps"
      .\build-windows.ps1 -KeepBuildDirs

    Notes:
    - Ce script NE lance PAS garage.py (tu testes avant).
    - Il génère dist\Garage.exe puis un ZIP dans releases\
#>

[CmdletBinding()]
param(
  [string]$Version = "4.4.5",
  [switch]$KeepBuildDirs,
  [string]$AppsDir = ""
)
```

```
Set-StrictMode -Version Latest
$ErrorActionPreference = "Stop"

function Info($msg) { Write-Host "ℹ️  $msg" }
function Ok($msg)   { Write-Host "✅ $msg" }
function Warn($msg) { Write-Host "⚠️  $msg" }
function Fail($msg) { Write-Host "❌ $msg"; exit 1 }

# --- 0) Contexte / chemins ---
$Root      = (Get-Location).Path
$GaragePy  = Join-Path $Root "garage.py"
$AssetsDir = Join-Path $Root "assets"
$DataDir   = Join-Path $Root "data"

$IconIco = Join-Path $AssetsDir "logo.ico"
$LogoPng = Join-Path $AssetsDir "logo.png"
$AideMd  = Join-Path $AssetsDir "AIDE.md"
$EmptyDb = Join-Path $DataDir   "garage_empty.db"

$VenvDir  = Join-Path $Root ".venv"
$VenvPy   = Join-Path $VenvDir "Scripts\python.exe"
$Activate = Join-Path $VenvDir "Scripts\Activate.ps1"

$DistExe  = Join-Path $Root "dist\Garage.exe"
$OutDir   = Join-Path $Root "releases"
$ZipName  = "Garage-v$Version-windows-portable.zip"
$ZipPath  = Join-Path $OutDir $ZipName
$HashPath = "$ZipPath.sha256"

Info "Build Windows — Garage v$Version"
Info "Repo: $Root"

# --- 1) Vérifs fichiers requis ---
Info "Vérification des fichiers requis…"
if (!(Test-Path $GaragePy))  { Fail "garage.py introuvable à la racine." }
if (!(Test-Path $AssetsDir)) { Fail "Dossier assets/ introuvable." }
if (!(Test-Path $DataDir))   { Fail "Dossier data/ introuvable." }

if (!(Test-Path $IconIco)) { Fail "Manquant: assets\logo.ico" }
if (!(Test-Path $LogoPng)) { Fail "Manquant: assets\logo.png" }
if (!(Test-Path $AideMd))  { Fail "Manquant: assets\AIDE.md" }
if (!(Test-Path $EmptyDb)) { Fail "Manquant: data\garage_empty.db" }
Ok "Structure OK."

# --- 2) Venv + deps build ---
if (!(Test-Path $VenvPy)) {
  Info "Création du venv .venv…"
  python -m venv $VenvDir
  if (!(Test-Path $VenvPy)) { Fail "Échec création venv (.venv\Scripts\python.exe introuvable)." }
  Ok "Venv créé."
} else {
```

```
  Info "Venv déjà présent."
}

Info "Activation du venv…"
. $Activate

Info "Installation / mise à jour des dépendances build…"
& $VenvPy -m pip install --upgrade pip | Out-Host
& $VenvPy -m pip install pillow matplotlib pyinstaller | Out-Host
Ok "Dépendances OK."

# --- 3) Nettoyage build/dist/spec ---
if (-not $KeepBuildDirs) {
  Info "Nettoyage build/dist/Garage.spec…"
  Remove-Item -Recurse -Force (Join-Path $Root "build") -ErrorAction SilentlyContinue
  Remove-Item -Recurse -Force (Join-Path $Root "dist")  -ErrorAction SilentlyContinue
  Remove-Item -Force (Join-Path $Root "Garage.spec")    -ErrorAction SilentlyContinue
  Ok "Nettoyage OK."
} else {
  Warn "KeepBuildDirs activé: on conserve build/dist/spec."
}

# --- 4) Build PyInstaller ---
Info "Build PyInstaller…"
& $VenvPy -m PyInstaller `
  --name Garage `
  --onefile `
  --windowed `
  --icon "$IconIco" `
  --add-data "assets;assets" `
  --add-data "data;data" `
  "$GaragePy" | Out-Host

if (!(Test-Path $DistExe)) { Fail "Build terminé mais dist\Garage.exe introuvable." }
Ok "EXE généré: $DistExe"

# --- 5) ZIP de release + SHA256 ---
if (!(Test-Path $OutDir)) { New-Item -ItemType Directory -Path $OutDir | Out-Null }

Remove-Item -Force $ZipPath  -ErrorAction SilentlyContinue
Remove-Item -Force $HashPath -ErrorAction SilentlyContinue

Info "Création ZIP: $ZipName"
Compress-Archive -Path $DistExe -DestinationPath $ZipPath -Force
if (!(Test-Path $ZipPath)) { Fail "ZIP non créé: $ZipPath" }
Ok "ZIP créé: $ZipPath"

Info "Calcul SHA256…"
$hash = (Get-FileHash -Algorithm SHA256 -Path $ZipPath).Hash.ToLower()
"$hash  $ZipName" | Set-Content -Encoding ASCII -Path $HashPath
Ok "SHA256 OK (fichier: $HashPath)"
```

```
# --- 6) Option: copie dans AppsDir (mise à jour “au même chemin” pour garder les raccourcis) ---
if ($AppsDir.Trim() -ne "") {
  if (!(Test-Path $AppsDir)) {
    Info "Création AppsDir: $AppsDir"
    New-Item -ItemType Directory -Path $AppsDir | Out-Null
  }
  $DestExe = Join-Path $AppsDir "Garage.exe"
  Copy-Item -Force $DistExe $DestExe
  Ok "EXE copié vers: $DestExe"
}

# --- 7) Résumé ---
Write-Host ""
Write-Host "==================== RÉSUMÉ ====================="
Write-Host "Version       : v$Version"
Write-Host "EXE           : $DistExe"
Write-Host "ZIP release   : $ZipPath"
Write-Host "SHA256 file   : $HashPath"
if ($AppsDir.Trim() -ne "") { Write-Host "AppsDir copy  : $AppsDir\Garage.exe" }
Write-Host "================================================="
Write-Host ""
Ok "Build Windows terminé."
```