Guide pas-à-pas: transformer ton projet en application Desktop avec Electron + Vite + React

Objectif

* Lancer l’UI React (Vite) dans une fenêtre Electron en dev.
* Charger le bundle Vite en production.
* Garder une architecture propre et sécurisée (preload, isolation, scripts).

1. Analyse du projet

* Structure détectée:
  + backend/ (API Python)
  + frontend/ (probablement mobile/web)
  + desktop/ (React + Vite pour desktop)
* L’UI desktop utilise React + Vite. On ajoute Electron autour pour en faire une app desktop.

1. Choix technique

* Electron sert de “shell” natif (fenêtre, menu, accès OS).
* Vite sert pour le dev (hot reload) et build des assets.
* Stratégie de chargement:
  + Dev: Electron charge [http://localhost:3000](http://localhost:3000/) (serveur Vite).
  + Prod: Electron charge le fichier dist/index.html généré par Vite.

1. Création du processus principal Electron

* Fichier ajouté: desktop/electron/main.js
  + Crée une BrowserWindow (1280x800).
  + webPreferences sécurisées: nodeIntegration=false, contextIsolation=true, sandbox=true, preload défini.
  + Dev: charge [http://localhost:3000](http://localhost:3000/)
  + Prod: charge file://.../dist/index.html
  + Empêche l’ouverture de fenêtres externes dans l’app (ouvre le navigateur par défaut).
  + Gère la single-instance (évite plusieurs instances).

1. Préload sécurisé

* Fichier ajouté: desktop/electron/preload.js
  + Expose un pont minimal via contextBridge (API sûre).
  + Point d’extension si tu veux de l’IPC plus tard (dialog, fs, etc.).

1. Mise à jour de desktop/package.json

* main: défini sur electron/main.js (point d’entrée Electron).
* Scripts ajoutés:
  + dev:desktop = lance Vite + attend le port 3000 + lance Electron
  + electron = lance Electron après que Vite soit prêt (wait-on)
  + start:desktop = lance Electron en pointant sur le bundle (utile après un build Vite)
* Outils ajoutés:
  + concurrently: exécuter plusieurs scripts (Vite + Electron)
  + wait-on: attendre l’ouverture du port 3000 avant de lancer Electron

1. Vite

* vite.config.js déjà OK (port 3000, plugin React).
* En prod, exécute vite build pour générer desktop/dist. Electron chargera ce dossier.

1. Lancer l’application en développement (Windows/PowerShell)

* Ouvre PowerShell dans c:\Users\achref\Desktop\scanapp\appscanstock
* Installe et lance côté desktop:
  + cd desktop; npm install; npm run dev:desktop
* Si tu utilises “&&” dans PowerShell, ça échoue. Utilise des “;” comme ci-dessus.
* Le script démarre Vite (port 3000), attend qu’il soit prêt, puis lance Electron.

1. Lancer le backend (API)

* Tu as un backend Python. Démarre-le avant d’utiliser l’UI (l’UI appelle <http://127.0.0.1:8000/api>).
* Depuis le dossier backend:
  + PowerShell: cd ..\backend; ./start\_server.ps1
* Vérifie que 127.0.0.1:8000/api répond. L’UI desktop consomme cette API via axios.

1. Construire pour la production

* Étapes minimales:
  + cd desktop; npm run build
  + Ensuite lance: npm run start:desktop (Electron chargera file://…/dist/index.html)
* Pour générer un installeur (packager):
  + Option simple: ajouter electron-builder
    - npm i -D electron-builder
    - Ajouter un script "build:desktop": "vite build && electron-builder"
    - Ajouter une config build (fichiers à inclure: dist/**, electron/**).
  + Résultat: un .exe/.msi pour Windows (ex. cible nsis).

1. Sécurité et bonnes pratiques

* contextIsolation=true + preload: évite d’exposer Node.js au renderer.
* nodeIntegration=false: réduit les risques XSS.
* shell.openExternal pour liens externes.
* Ne déplace pas la logique sensible dans le renderer; utilise IPC via preload si besoin.
* Pense à gérer CORS côté backend si nécessaire.

1. Points d’attention fréquents

* Fenêtre blanche en dev: Vite pas prêt, d’où wait-on. Vérifie le port 3000.
* Conflits de port: Modifie vite.config.js (server.port) et la cible devURL dans main.js si nécessaire.
* API en prod: 127.0.0.1:8000 doit tourner aussi en prod ou bien packager le backend avec l’app. Sinon, configure une URL distante.
* Antivirus/SmartScreen: signe l’app ou utilise un installeur standard (nsis) pour réduire les alertes.

1. Évolution possible

* Ajouter IPC pour fonctions natives (dialog, impression, fichiers).
* Icône, menu applicatif, deep links, protocoles personnalisés.
* Auto-update (electron-updater) si tu distribues publiquement.
* Déplacer electron en devDependencies pour optimiser le packaging (optionnel).
* Environnement: variables d’ENV pour BASE\_URL (dev/prod).

Résumé des fichiers clés modifiés/ajoutés

* desktop/electron/main.js: processus principal Electron
* desktop/electron/preload.js: pont sécurisé
* desktop/package.json: main, scripts dev:desktop/start:desktop, deps dev (concurrently, wait-on)
* desktop/index.html et vite.config.js: inchangés (fonctionnent déjà avec Vite)

Commandes essentielles (PowerShell)

* Dev UI + Electron: cd desktop; npm install; npm run dev:desktop
* Dev backend: cd ..\backend; ./start\_server.ps1
* Build UI pour prod: cd desktop; npm run build
* Lancer Electron sur le build: npm run start:desktop