**Guide: Creating an EXE from a Flask (Python) Backend**

This guide walks you through converting your Flask application into a standalone .exe executable using **PyInstaller**.

**📁 Prerequisites**

1. Python 3.x installed on your system
2. Flask application entry file (e.g., run.py)
3. All dependencies listed in requirements.txt
4. A virtual environment activated (venv)

**🧰 Step-by-Step Instructions**

**1. Install PyInstaller**

pip install pyinstaller

**2. Structure Your Flask App**

Ensure your app looks something like:

billing-stocks-management-system-be/

│

├── app/

├── run.py

├── .env

├── requirements.txt

**3. Run PyInstaller**

Navigate to the directory with your run.py file:

cd path/to/your/app

Run PyInstaller with the following command:

pyinstaller --onefile run.py

✅ This generates a folder named dist/ which will contain run.exe

**4. Include Extra Files (Optional)**

If you use .env, HTML templates, or static files, add them:

pyinstaller --onefile \

--add-data ".env;." \

--add-data "app/templates;app/templates" \

--add-data "app/static;app/static" \

run.py

**Syntax for --add-data:**

"source;destination"

**5. Run the EXE**

After the build completes, navigate to the dist folder and run:

cd dist

./run.exe

Your Flask server should now start from the .exe.

**🛠 Troubleshooting**

* **Missing modules?**: Make sure they are installed in your virtual environment.
* **Dynamic files not found?**: Use --add-data to explicitly include them.
* **Google APIs or other configs**: Ensure your Google credentials and any API keys are accessible.

**📦 Bonus: Clean Up Build Files**

pyinstaller --clean --onefile run.py

To delete build folders:

rm -rf build dist \_\_pycache\_\_ run.spec

**📝 Notes**

* You can rename run.exe using the --name flag:

pyinstaller --onefile --name billing\_app run.py

Let me know if you'd like to also generate an .exe for the frontend using Electron or Tauri!