Collagis Automation App — Setup and Packaging Guide

Prerequisites

Before setting up the automation, ensure the following requirements are met:

- Python 3.12. installed on your system.
- **PyInstaller** installed for packaging the app into an executable.

  ```
  pip3 install pyinstaller
  ```

- **Google Chrome browser** installed. If not, install it using Homebrew:

  ```
  brew install --cask google-chrome
  ```

- **ChromeDriver** compatible with your installed Chrome version. Install via Homebrew for the signed and trusted version:

  ```
  brew install chromedriver
  ```

  This ensures macOS recognises the ChromeDriver binary as safe and prevents signature warnings.

- **OpenAI API key** with access to the **GPT-5** model.
- **Active internet connection** for AI processing and ServiceNow or portal form submissions.

Installation and execution of the binary build

Step 1: Pull the Latest Code

Clone or pull the latest version of the repository:

```
git clone https://github.com/pathfindr-ai/collagis-community-buddy.git
cd collagis
git pull
```

Step 2: Environment Variables

Create a file named **.env** at the root of the project, where the built automation binary is located and add your environment variables.
You can copy them from .env.example or create one manually.

Update the `.env` file with **full absolute paths**, for variables:

```
TRANSCRIPTS_FOLDER=/Users/paulventura/ConversationConfig/ConversationLogs
HOSTS_CSV_PATH=/Users/paulventura/downloads/collagis/assets/hosts.csv
REPORTS_FILE=/Users/paulventura/downloads/collagis/assets/reports/flow_reports.
json
ERROR_FILE=/Users/paulventura/downloads/collagis/assets/reports/error_report.cs
v
```

In the environment file, set this value to **true** so that in the **maintenance request** case, the form is submitted automatically.

```
SMOKE_TEST=True
```

Step 3: Setting Up macOS Auto-Run (Cron Scheduling)

At the **root of the project inside scripts folder**, you'll find **three shell scripts**:

- `setup_autorun.sh` → creates the scheduled auto-run job (runs every 5 minutes)
- `start_autorun.sh` → manually starts the job
- `stop_autorun.sh` → stops the job immediately

The setup script uses **macOS Launchd** (not traditional cron) to automatically run the automation build (`AutomationAppV4`) at defined intervals — by default, **every 5 minutes**.

1: Changing the Paths in setup_autorun.sh

Open the setup_autorun.sh (this Bash script) in any text editor and **update the variables** in the configuration section after reviewing the details below.
Each variable controls where the autorun service looks for files and logs.

Below is the configuration section from the script:

```
# --- CONFIG ---
PLIST_PATH="$HOME/Library/LaunchAgents/com.collagis.autorun.plist"
APP_PATH="/Users/paulventura/downloads/collagis/dist/AutomationAppV4"
WORK_DIR="/Users/paulventura"
LOG_OUT="/tmp/collagis_autorun.log"
LOG_ERR="/tmp/collagis_autorun.err"
```

Explanation of each path: Always use absolute paths.

Variable	Description	Example
PLIST_PATH	Path where the macOS Launch Agent configuration file is created.	~/Library/LaunchAgents/com.collagis.autorun.plist
APP_PATH	Absolute path of the compiled automation app executable.	/Users/<username>/downloads/collagis/dist/AutomationAppV4

WORK_DIR	The working directory must contain:
	
	The **dist** folder with AutomationAppV4 and .env
	
	Note: env should be at the root of the working dir
	
	All folders and files referenced in the .env configuration
	
	TRANSCRIPTS_FOLDER=/Users/mini/assets
	HOSTS_CSV_PATH=/Users/mini/assets/hosts.csv
	REPORTS_FILE=/Users/mini/assets/reports/flow_reports.json
	ERROR_FILE=/Users/mini/assets/reports/error_report.csv
		/Users/<username>
		
		.env file path:
		/Users/<username>/.env

LOG_OUT / LOG_ERR	Paths for storing standard output and error logs.	/tmp/collagis_autorun.log, /tmp/collagis_autorun.err

2: Running the Setup Script

Provide Access to the Transcripts Folder

To ensure the automation app can read, write, and move transcript files, grant full permissions to the transcripts directory by running the following command in

```
chmod -R 777 /Users/paulventura/ConversationConfig/ConversationLogs
```

Give Execute Permissions to the Binary

Open Terminal at the project root and run:

```

# Navigate to the dist folder
cd dist

# Set the executable bit for AutomationAppV4
chmod +x AutomationAppV4

# (Optional) Verify permissions
ls -l "AutomationAppV4"
```

This ensures macOS can run the binary manually or via the autorun service

Make the Setup Script Executable

Navigate to the scripts folder:

```
cd ../scripts

# Set the executable bit for the setup script
chmod +x setup_autorun.sh
```

Run the Setup Script

Still in the `scripts` folder, run:

```
./setup_autorun.sh
```

This will:

- Create the `LaunchAgents` directory (if not present)
- Generate the `.plist` file with the correct paths
- Register it with macOS Launchd
- Start the scheduled run (every 5 minutes)
- Display setup verification and recent logs

3. Important — Security Prompt on First Run

When running the automation for the first time, macOS may display a warning such as:

> “AutomationAppV4” or “ChromeDriver” cannot be opened because it is from an unidentified developer.

If this occurs:

1. Click **Done** on the popup.
2. Open **System Settings** → **Privacy & Security**.
3. Scroll down to the **Security** section.
4. Click **Open Anyway** next to the warning for ChromeDriver or AutomationAppV4.
5. Enter your password if prompted, then click **Open**.

Once approved, ChromeDriver and AutomationAppV4 will be trusted and **future runs will no longer show this warning**.

Tip: It is recommended to run the app **once manually** (directly from the terminal as mentioned in the Development and Packaging -> Packaging the Application -> Step 3) before using the setup script this ensures all security permissions are granted and paths are verified.

4: Verify That the Schedule Was Created

Check if the Launch Agent is active:

```
launchctl list | grep collagis
```

- If you see `com.collagis.autorun` → it’s active.

- If nothing appears → it's stopped or failed to load.

To view logs:

```
cat /tmp/collagis_autorun.log     # standard logs
cat /tmp/collagis_autorun.err     # error logs
```

5. Control Scripts

Stop Script — `stop_autorun.sh`

Stop the automation job immediately.
Run with:

```
cd scripts
chmod +x stop_autorun.sh
./stop_autorun.sh
```

Start Script — `start_autorun.sh`

Restarts the automation job manually (useful after changes or testing).

Run with:

```
cd scripts
chmod +x start_autorun.sh
./start_autorun.sh
```

Summary of Steps

Step	Action	Command

1	Pull latest code	git clone … && cd collagis
2	Create .env file at root	Add variables manually or copy from .env.example
3	Update setup_autorun.sh	Change APP_PATH and WORK_DIR to absolute paths
4	Run setup	bash setup_autorun.sh
5	Start/Stop automation	bash start_autorun.sh / bash stop_autorun.sh

---

Development and Packaging section

Step 1: Pull the Latest Code

Clone or pull the latest version of the repository:

```
git clone https://github.com/pathfindr-ai/collagis-community-buddy.git
cd Collagis
git pull
```

Step 2: Create, Activate a Virtual Environment and Install Requirements

Note:
This step is **necessary if you plan to modify the code files** (for example, in the `v4/` folder).

Ensure you are using **Python 3.12.4** (recommended).

```
python3 -m venv venv
source venv/bin/activate
```

Install all dependencies from `requirements.txt`:

```
pip install -r requirements.txt
```

Step 3: Environment Variables

At the **root** of the project, create a file named `.env` and add your environment variables. You can copy them from `.env.example` or create one manually.

Important Variables for Testing

Variable	Example Value	Description
`SMOKE_TEST`	`False`	Set `SMOKE_TEST=False` when testing the flow (e.g., to verify the webform process without actually submitting the form). Set `SMOKE_TEST=True` in production or normal runs to enable real form submission.
`LEDGER_CLEANUP_DAYS`	`90`	Number of days to retain processed file records before automatic cleanup.
`REPORTS_FILE`	`./assets/reports/flow_reports.json`	Path to the reports file used as a lightweight database for tracking flow results.
`LOAD_ALL_HASHES`	`True`	If `True`, loads all processed file hashes into memory (recommended for small datasets). Set to `False` for large datasets to enable

		on-demand (lazy) hash checks.

Step 4: Verify Project Structure & Run the Main file

All the **version 4 code files** — including logic for the three cases (**webform**, **it_request**, and **visitor_check_in**) — are located in the `v4/` folder.

The **compiled application build** is located inside the `dist/` folder under the name `AutomationAppV4`.

Once installed, your project structure should look like this:

```
COLLAGIS/
│
├── assets/
├── build/
├── dist/
│      ├── AutomationAppV4
├── test/
├── v4/
│    ├── automation_script_v4.py
│    ├── extract_data.py
│    ├── fill_form_details.py
│    ├── mappings.py
│    └── ...
├── .env
├── .env.example
├── requirements.txt
├── README.md
```

After changes in the code you may run the main file from the root of the project:

```
python -m v4.automation_script_v4
```

Step 5. Packaging the Application

1: Update Environment File

Update the `.env` file with **full absolute paths**, especially for:

- `TRANSCRIPTS_FOLDER`
- `HOSTS_CSV_PATH` — csv for mapping host name to email for visitor check-in case
- `REPORTS_FILE`

It is recommended to keep the transcripts folder **inside the `collagis` folder** (from GitHub), as it will also be referenced in the **cron job** scheduling.

2: Build the Executable

Note: A pre-built executable is already available, so rebuilding is only required if you make code changes.

If you need to rebuild, follow these steps:

1. **Create a fresh virtual environment**
    - Delete any existing virtual environment (e.g. `buildenv`, `venv`, etc.).
    - Create a new one:

    ```
    python3 -m venv buildenv
    source buildenv/bin/activate
    ```

    - **Install dependencies**

    ```
    pip install -r requirements.txt
    ```

2. **Unset sensitive environment variables**
   Before building, clear any secrets from the environment to prevent them from being embedded in the binary:

    ```
    unset OPENAI_API_KEY
    unset SERVICENOW_PASSWORD
    unset SMTP_PASSWORD
    ```

3. **Build the executable**
   From the root directory of your project, run:

    ```
    pyinstaller --onefile --exclude .env --name
    ```

```
AutomationAppV4 v4/automation_script_v4.py
```

This ensures a clean, secure build with no sensitive keys or configuration files included.

3: Test the Build

Note: Skip this step if you don't want to test the existing build

To test the existing build, simply run:

```
cd dist
```

```
./AutomationAppV4
```

This will launch the packaged executable. You can use this step anytime to verify that the automation is functioning correctly after environment or path updates.

When first running:

1. You may see a macOS warning about ChromeDriver or AutomationAppV4.
2. Click **Done** on the popup.
3. Go to **System Settings > Privacy & Security**.
4. Scroll to the **Security** section, and click **Open Anyway**.
5. Enter your password if prompted, then click **Open**.
6. The app will now run normally. Future launches will no longer show this warning.

Viewing Detailed Logs and Processed Files

All logs and execution details for each processed transcript file are saved in the paths defined in your `.env` file:

```
REPORTS_FILE=
/Users/questmini/Downloads/collagis/assets/reports/flow_reports.json
ERROR_FILE=
/Users/questmini/Downloads/collagis/assets/reports/error_report.csv
```

Flow Report (flow_reports.json)

This file contains **detailed logs** for each transcript processed — including the flow type, success or failure status, timestamp, and reference details.

Each entry represents one processed file.

```
Sample:
[
  {
    "flow": "it_request",
    "success": true,
    "reference": "INC0010001",
    "error": null,
    "screenshot_path": null,
    "timestamp": "2025-10-22T11:53:30.014115",
    "file_name": "log-sample-it.txt"
  },
  {
    "flow": "extract_data_transcript_it_request_data",
    "success": false,
    "reference": null,
    "error": "IT extraction failed: Missing mandatory fields: caller_id",
    "screenshot_path": null,
    "timestamp": "2025-10-22T12:01:41.037657",
    "file_name": "test.txt"
  }
]
```

Fields:

- `flow` — Type of request (e.g. `maintenance_request`, `it_request`, `visitor_check_in`)
- `success` — `true` if processed successfully, otherwise `false`
- `reference` — Generated ServiceNow reference ID (if applicable)
- `error` — Error message if processing failed
- `timestamp` — When the transcript was processed
- `file_name` — The original transcript file name

Error Report (error_report.csv)

This CSV file lists all **failed transcript files** along with the **reason for failure** (e.g. missing data, connection issues, or invalid format).

You can open this file in Excel or any text editor for a quick overview of failed cases.

Processed and Failed File Directories

In the same folder where your reports are stored (`assets/reports/`), two additional directories are created automatically:

```
assets/
└── reports/
    ├── flow_reports.json
    ├── error_report.csv
├── processed/
└── failed/
```

- **`processed/`** → Contains all successfully processed transcript files
- **`failed/`** → Contains transcript files that encountered errors during processing

This structure helps you easily review which transcripts were handled successfully and which need manual review.

Configuration

Transcript Format

The system expects **JSON-formatted transcripts** with the following structure:

```
{
  "Author": "ChatGPT",
  "Text": "Please tell me your first name."
},
{
  "Author": "User",
  "Text": "John Smith"
}
```

Host CSV Format

The system expects a **CSV file** containing host information in the following format:

```
name,email
John Smith,john@gmail.com
Sarah Lee,sarah.lee@company.com
```

Column	Description
`name`	Full name of the host
`email`	Email address of the host

Common Build and Runtime Errors

**Error: `ModuleNotFoundError: No module named 'langchain_openai'`

If this error occur in either of the following cases:

- When running the binary for the **first time**, or
- When running the **same version binary directly from GitHub** without rebuilding locally

Fix:

If you encounter this error, follow the same build steps outlined above:

1. **Create a new virtual environment**

    ```
    python3 -m venv buildenv
    source buildenv/bin/activate
    ```

2. **Install dependencies**

```
pip install -r requirements.txt
```

3. **Unset any sensitive environment variables**

```
unset OPENAI_API_KEY
unset SERVICENOW_PASSWORD
unset SMTP_PASSWORD
```

4. **Rebuild the executable**

```
 pyinstaller --onefile --exclude .env
--name AutomationAppV4
v4/automation_script_v4.py
```

5. **Test the build again**

```
cd dist
chmod +x "AutomationAppV4"
./AutomationAppV4
```

6. **Run the Setup Script (for automated scheduling)**

   Note: Make sure the setup_autorun.sh script includes the absolute paths to both the working directory and the `AutomationAppV4` executable.

   After confirming the binary runs correctly, follow these steps to configure the scheduled automation:

   1. **Navigate to the scripts folder**

```
    cd ../scripts
```

   2. **Set the executable bit for the setup script**

```
    chmod +x
   setup_autorun.sh
```

   3. **Run the Setup Script**

```
./setup_autorun.sh
```

This will:

- Create the `LaunchAgents` directory (if not already present)
- Generate the `.plist` file with correct paths
- Register it with macOS **Launchd**
- Start the scheduled run (every 5 minutes)
- Display setup verification and recent logs