DUNWELL SOFTWARE ™ • OFFICIAL DOCUMENTATION HANDBOOK

Dunwell DevStudio — Complete User Guide

From First-Time Local AI Engine Setup to Building, Testing & Self-Repairing Your Own Apps 100% Offline

EDITION	SKILL LEVEL	PRIVACY & COST	TOTAL LENGTH
Live Release (v1.0)	Complete Beginners	100% Offline • £0 API	3 Parts • 22 Pages

TABLE OF CONTENTS

Start with Part 1 (Step 0) before launching DevStudio

PART 1 • STEP 0 Installing Your Free Local AI Engine (LM Studio or Ollama) — Pages 2 – 7

1.	Read This First — Why You Need This Step	Page 2
2.	LM Studio vs. Ollama — Which One Should You Choose?	Page 2
3.	Quick 20-Second PC Check (Picking the Right Brain Size)	Page 3
4.	Option A: Installing & Setting Up LM Studio (Recommended)	Page 4
5.	Option B: Installing & Setting Up Ollama (Quiet Background Engine)	Page 5
6.	The Final Green-Light Check Inside Dunwell DevStudio	Page 6
7.	Quick Jargon Buster — 6 Scary Words Made Simple	Page 6
8.	Plain-English Troubleshooting (First-Day Hiccups)	Page 7
9.	Official Dunwell Software Links	Page 7

PART 2 Complete Beginner's Foundations Guide — Pages 8 – 17

1.	Welcome — What Dunwell DevStudio Is & Your 3-Helper AI Team	Page 8
2.	Your First 5 Minutes — Quick Setup Check	Page 9
3.	Every Button & Panel Explained in Plain English	Page 9
4.	Plain-English Dictionary of Common File Types	Page 11
5.	How to Write Great Prompts (The 4-Part Formula)	Page 14
6.	Your First 10-Minute Practice Project	Page 15
7.	Golden Safety Habits & Quick Troubleshooting	Page 16
8.	Where to Go Next — Unlocking Advanced Mode	Page 17

PART 3 The Built-In Self-Repair Loop Guide — Pages 18 – 22

1.	How the Two-Stage Self-Repair Loop Works	Page 18
2.	Real-World Walkthrough: From Error to Working App	Page 18
3.	Controls & Safety Nets at a Glance	Page 21
4.	Everyday Tips for Getting the Best Results	Page 22

DUNWELL DEVSTUDIO™ · STEP 0 — MUST-DO BEFORE YOU START

Installing Your Free Local AI Engine (LM Studio or Ollama)

A Complete No-Jargon Setup Guide for Total Beginners — Required Before Using Dunwell DevStudio

PUBLISHER	REQUIRED STEP	EXTRA COST	SETUP TIME
Dunwell Software (www.dunwellsoftware.com)	Complete Before Launching DevStudio	£0 (Both Are 100% Free)	Roughly 10–15 Minutes

1. Read This First — Why You Need This Step

Think of **Dunwell DevStudio** like a smart car's dashboard, steering wheel, and sat-nav. It organises your files, coordinates your 3-Helper AI team, makes safety backups, and runs your finished apps—but it does **not** come with a giant AI brain pre-packed inside the download.

Why? Because offline AI brains are large files (often between 2 GB and 20 GB), and the best size depends on how much memory your specific computer has. To supply the "engine under the bonnet", you install one free helper program on your Windows PC—either **LM Studio** or **Ollama**. Once installed and given one downloaded AI brain, it runs **100% privately on your own PC** with zero cloud accounts and zero monthly fees, and Dunwell DevStudio connects to it automatically.

> **Golden Rule — Pick ONE, Not Both:** Without either **LM Studio** or **Ollama** running on your computer, Dunwell DevStudio has no AI brain to talk to and cannot generate code. You only need to install **one** of them. If you are a complete beginner who prefers clicking buttons instead of typing commands, choose **Option A: LM Studio**.

2. LM Studio vs. Ollama — Which One Should You Choose?

Both programs do the exact same job behind the scenes: they download a free AI brain onto your hard drive and open a private doorway on your own PC so Dunwell DevStudio can talk to it. The only real difference is how you control them.

Feature	Option A: LM Studio (Recommended for Beginners)	Option B: Ollama (Simple Background Engine)
How You Use It	A full visual Windows app with search boxes, download buttons, and clear on/off switches.	Runs quietly as a tiny llama icon next to your Windows clock; models are downloaded with one short typed command.

Feature	Option A: LM Studio (Recommended for Beginners)	Option B: Ollama (Simple Background Engine)
Choosing a Model	Shows friendly colour-coded fit indicators telling you whether an AI brain will fit on your graphics card before you download it.	You type the exact short name of the model you want (we give you ready-to-copy commands in Section 5).
Daily Routine	Open LM Studio, make sure your model is loaded and the **Local Server** switch is ON, then open DevStudio.	Starts automatically with Windows—as long as the llama icon is visible by your clock, just open DevStudio.
Local Doorway	`http://127.0.0.1:1234` (private to your PC only)	`http://127.0.0.1:11434` (private to your PC only)
Best For	Total beginners who want visual menus and easy model browsing without touching command windows.	People who want a “set it once and forget it” background engine with minimal screen clutter.

3. Quick 20-Second PC Check (Picking the Right Brain Size)

Before downloading an AI brain, take 20 seconds to check how much memory your PC has. Downloading an AI brain that is too large for your computer will not break anything, but it will make replies crawl at one word every few seconds.

1. Press **Ctrl + Shift + Esc** together on your keyboard to open Windows **Task Manager**.
2. Click the **Performance** tab (the little heartbeat-graph icon on the left side).
3. Click **Memory** and note the total number in the top-right corner (for example, `16.0 GB` or `32.0 GB`).
4. If you see **GPU 0** or **GPU 1** listed below Memory, click it (choose the NVIDIA or AMD graphics card if two appear) and look at **Dedicated GPU memory** at the bottom—for example, `8.0 GB`, `12.0 GB`, or `24.0 GB`. This graphics card memory is called **VRAM**, and it is what makes local AI super fast.

Use the simple table below to see which size AI brain suits your PC best. In AI model names, a number like `7B` means “7 billion parameters” (roughly how knowledgeable the brain is—bigger is smarter, smaller is faster), and `Q4_K_M` simply means “neatly compressed so it stays sharp while fitting easily in memory”.

Your PC Hardware	Recommended Brain Size	Approx. Download Size	Great Starter Coding Models to Pick
Everyday Laptop / Office PC (8–16 GB RAM, no gaming GPU or 4 GB VRAM)	**3B to 4B** (`Q4_K_M`)	2 GB – 3 GB	`Qwen 2.5 Coder 3B` or `Gemma 3 4B`

Your PC Hardware	Recommended Brain Size	Approx. Download Size	Great Starter Coding Models to Pick
Mid-Range Gaming PC (6–8 GB Dedicated GPU VRAM)	**7B to 8B** (`Q4_K_M`)	4.5 GB – 5.5 GB	`Qwen 2.5 Coder 7B` (brilliant all-round starter)
Strong Gaming / Creator PC (12–16 GB Dedicated GPU VRAM)	**12B to 14B** (`Q4_K_M`)	8 GB – 10 GB	`Qwen 2.5 Coder 14B` or `Gemma 12B`
Enthusiast PC (24 GB+ Dedicated GPU VRAM)	**24B to 32B** (`Q4_K_M`)	16 GB – 20 GB	`Qwen 2.5 Coder 32B` or `Gemma 27B-31B`

Free Hard Drive Space Check: Make sure the drive where you save your models has at least double the model's download size free (for example, keep at least 15–20 GB free when starting out with a 7B model).

4. Option A: Installing & Setting Up LM Studio (Recommended)

Follow these six simple steps if you chose **LM Studio**. You will not need to type a single technical command—just point and click.

1. **Download & Install LM Studio:** Open your web browser, search for **"LM Studio official download"** (at `lmstudio.ai`), and click **Download LM Studio for Windows**. Once downloaded, double-click the installer file and follow the on-screen prompts until installation finishes. *How to check it worked:* LM Studio opens cleanly on your screen or appears in your Windows Start menu.
2. **Finish the Welcome Screen:** When LM Studio opens for the first time, it may offer to download a starter model straight away. You can either follow its prompt or click **Skip** so you can pick the exact coding model from Section 3 yourself. Look along the thin left-hand sidebar—you will see icons for **Chat**, **Developer** (looks like `>_`), **My Models** (folder icon), and **Discover / Search** (magnifying glass).
3. **Search For & Download Your First AI Brain:** Click the **Discover / Search** magnifying glass on the left sidebar. Type your chosen starter model from the table in Section 3 (for example, `qwen2.5-coder-7b-instruct`) into the search bar. Click the result from a trusted publisher (look for **GGUF** in the format), check LM Studio's compatibility indicator on the right—aim for a `Q4_K_M` file showing a green **Full GPU offload possible** badge (or partial offload on laptops)—and click **Download**. *How to check it worked:* The download bar along the bottom reaches 100% and turns green.
4. **Load the Model Into Memory:** Downloading a brain puts it on your hard drive, like putting a book on a shelf; **loading** it opens the book so the AI can actually think! Click the **Select a model to load** bar at the very top centre of LM Studio and click the model you just downloaded. Wait a few seconds while the progress bar fills. *How to check it worked:* Your model's name stays displayed cleanly in the top bar with a green dot or Eject button next to it.

5. **Turn ON the Local Server (The Step Most Beginners Miss!):** Click the **Developer** icon (`>_`) on the left-hand sidebar. Near the top-left of that screen, find the toggle switch labelled **Status: Stopped** and click it so it switches to **Status: Running**. Leave the port at its default **`1234`** . *How to check it worked:* The status turns green showing **Running** on `http://127.0.0.1:1234` (if Windows Firewall asks for local network access, click **Allow**).
6. **Optional — Enable Automatic VRAM Cleanup for ComfyUI Later:** If you plan to use Dunwell DevStudio's ComfyUI image bridge later on, LM Studio includes a tiny built-in helper called the `lms` CLI (documented at lmstudio.ai/docs/cli/local-models/load) that lets DevStudio unload and reload VRAM hands-free. Beginners starting in Standard Mode can safely skip this for now.

Your Daily LM Studio Routine: Leave LM Studio open (or minimised to the taskbar) whenever you use Dunwell DevStudio. Before clicking Send in DevStudio, just double-check two things in LM Studio: (1) a model is **loaded** in the top bar, and (2) the Developer **Local Server** switch is **Running**.

5. Option B: Installing & Setting Up Ollama (Quiet Background Engine)

Skip this section if you already set up LM Studio above. Follow these four steps if you prefer **Ollama** running quietly in the background every time your PC starts.

1. **Download & Install Ollama:** Open your web browser, search for **"Ollama official download"** (at `ollama.com`), choose **Windows**, and run the installer by clicking **Install**. *How to check it worked:* Look at the bottom-right corner of your Windows taskbar next to the clock (you may need to click the tiny upward arrow `^`). A small white **llama icon** appears—that proves Ollama's local server is already awake on `http://127.0.0.1:11434` .
2. **Open Windows Command Prompt:** Press the **Windows key** on your keyboard, type `cmd` , and press **Enter**. A plain dark window opens—do not worry, you only need to paste one short line into it.
3. **Download Your First AI Brain:** Pick the line below that matches your PC from Section 3, type (or copy and paste) it into the dark window, and press **Enter**:

```
For everyday laptops (8–16 GB RAM, light/no gaming GPU):
ollama pull qwen2.5-coder:3b

For most gaming PCs (6–8 GB Dedicated GPU VRAM):
ollama pull qwen2.5-coder:7b

For strong gaming PCs (12–16 GB Dedicated GPU VRAM):
ollama pull qwen2.5-coder:14b
```

4. **Wait for "Success" & Verify:** Watch the download progress bar reach 100% until the window prints `success` . Next, type `ollama list` and press **Enter**. *How to check it worked:* Your downloaded model name appears in the list. You can now close the dark window completely—Ollama stays running silently by your clock and loads your model automatically whenever Dunwell DevStudio asks for it!

6. The Final Green-Light Check Inside Dunwell DevStudio

With either **LM Studio** (model loaded + Local Server Running) or **Ollama** (llama icon visible + one model pulled) ready on your PC, you are finally ready to launch **Dunwell DevStudio**.

1. **Install & Open Dunwell DevStudio:** Download Dunwell DevStudio from the Microsoft Store (apps.microsoft.com/detail/9PLF7CND11CW) and launch it from your Windows Start menu.
2. **Click Check Setup:** Click the **Check Setup** button in the top navigation bar of DevStudio. DevStudio automatically scans both `127.0.0.1:1234` (LM Studio) and `127.0.0.1:11434` (Ollama) to find your active local AI engine. *How to check it worked:* DevStudio displays a green connected status confirming it can talk to your loaded local AI model.
3. **Optional — Double-Check Your Hardware Tier:** Click **Hardware & Models** in the top bar. DevStudio inspects your PC automatically and confirms your Hardware Tier so you always know if your machine can comfortably handle an even smarter AI brain later on.
4. **Head Straight Into the Foundations Guide:** You are all set! Move on to the **Complete Beginner's Foundations Guide** to create your first practice project and watch your 3-Helper team build your first app.

7. Quick Jargon Buster — 6 Scary Words Made Simple

You may bump into these six technical words while downloading models in LM Studio or Ollama. Here is what they actually mean in plain English:

Term You Might See	What It Means in Plain English
Local LLM / Model	The downloadable "AI brain" file stored on your hard drive (for example, `Qwen 2.5 Coder 7B`).
GGUF	Simply the standard file format used for offline AI brains on home PCs—like `.mp3` is for music or `.jpg` is for photos.
VRAM	Your graphics card's dedicated high-speed memory. Fitting an AI brain inside VRAM makes it reply 5× to 10× faster than running on regular CPU memory.
Local Server & `127.0.0.1`	`127.0.0.1` is every computer's built-in nickname for "myself". A Local Server is just a private internal hallway letting LM Studio or Ollama pass messages directly to Dunwell DevStudio—nothing ever leaves your house or touches the internet.
Port (`1234` / `11434`)	Think of a port like a numbered front door on your PC: LM Studio answers door `1234`, and Ollama answers door `11434` so they never bump into each other.
Tokens per Second (`t/s`)	How fast your AI brain types. Anything around `15–30+ t/s` feels snappy and smooth; under `5 t/s` means the chosen model is slightly too big for your PC.

8. Plain-English Troubleshooting (First-Day Hiccups)

What Went Wrong	Most Likely Cause	Simple Plain-English Fix
DevStudio Check Setup says cannot connect	LM Studio's Local Server toggle is still switched OFF, or Ollama is not running in the taskbar.	In LM Studio, open the **Developer (`>_`)** tab and switch Status to **Running**. For Ollama, open Ollama from the Start menu so the llama icon appears by your clock.
LM Studio is Running, but DevStudio says no model loaded	You downloaded an AI brain, but did not load it into active memory yet.	Click **Select a model to load** in the top bar of LM Studio, pick your downloaded model, wait for the bar to fill, then click **Check Setup** in DevStudio again.
Replies are painfully slow or my PC stutters	The downloaded AI brain is larger than your graphics card's VRAM, forcing your PC to fall back to slower main memory.	Eject the large model and download one step smaller from Section 3 (for example, drop from `14B` down to `7B`, or `7B` down to `3B`, always picking `Q4_K_M`).
Windows Security / Firewall popped up a box	Windows checks before letting a local program open an internal port on your machine.	Tick **Private networks** and click **Allow** so Dunwell DevStudio can communicate locally with LM Studio or Ollama.
My C: drive is nearly full	By default, AI models save inside your Windows user folder on the `C:` drive.	In LM Studio, open **My Models** → **Models Directory** and point it to a second drive (such as `D:\AI_Models`). For Ollama, move models by setting the Windows environment variable `OLLAMA_MODELS` to your bigger drive.

9. Official Dunwell Software Links

- **Dunwell Software Portal & Complete Documentation:** https://www.dunwellsoftware.com
- **Dunwell DevStudio on the Microsoft Store:** https://apps.microsoft.com/detail/9PLF7CND11CW
- **LM Studio CLI Reference (Optional VRAM Handoff):** https://lmstudio.ai/docs/cli/local-models/load

DUNWELL DEVSTUDIO™ · CORE FOUNDATIONS SERIES · START HERE

Complete Beginner’s Guide to Dunwell DevStudio

Basic Functions, Every Button Explained in Plain English, Common File Types & Ready-to-Use Starter Prompts

PUBLISHER	AUDIENCE	PRIVACY MODEL	AI BACKENDS
Dunwell Software (www.dunwellsoftware.com)	Complete Beginners (Zero Coding Needed)	100% Offline Local PC	LM Studio or Ollama

1. Welcome — What Dunwell DevStudio Actually Is

If you have never written a single line of code in your life, you are in the exact right place. **Dunwell DevStudio** is a private software workshop that lives directly on your Windows PC. Instead of forcing you to memorise programming languages, it lets you describe what you want to build in plain everyday English—such as a kitchen timer, a personal budget tracker, a website, or a game script—and coordinates a built-in team of three AI helpers to plan, build, and test it for you.

Unlike cloud AI websites that charge monthly fees and send your files over the internet, Dunwell DevStudio runs **100% offline**. It connects to a free local AI engine installed on your own computer (either **LM Studio** or **Ollama**), and every project, setting, and automatic backup stays safely on your own hard drive inside your chosen project folder and Windows `%APPDATA%`.

Meet Your 3-Helper AI Team

When you type a request and click **Send & Start**, Dunwell DevStudio does not just spit out a messy wall of code and hope for the best. It hands your idea through three specialist helpers in order—just like a real small studio:

1. **1. The Planner (The Architect):** Reads your plain-English request and writes a short, clear step-by-step checklist of which files need creating or changing. You can watch the top bar light up on **1. Planner** while it thinks.
2. **2. The Builder (The Craftsperson):** Follows The Planner’s checklist, automatically saves a safety backup copy of any existing files first, and writes the actual code files directly into your project folder.
3. **3. The Inspector (The Quality Tester):** Checks The Builder’s work and runs your project’s test command to catch typos or crashes. If **Auto-fix bugs (3x)** is ticked and The Inspector spots a problem, it hands the error straight back to The Builder to fix automatically—up to three times—without you needing to understand the error message.

> **Standard Mode vs. Advanced Mode:** When you first start out, leave the **Advanced** toggle switch at the top of the screen turned **OFF** (Standard Mode). Standard Mode keeps the screen clean, fast, and clutter-free for everyday apps, scripts, and websites. Only switch **Advanced** to **ON** later when you are ready to connect external 3D/game tools (Roblox Studio, Blender, Godot, ComfyUI, Unreal Engine 5, or Unity).

2. Your First 5 Minutes — Quick Setup Check

Before asking your 3-Helper team to build your first app, run through these four quick checks. Once set up, DevStudio remembers your preferences every time you open it.

1. **Start Your Local AI Engine:** Open either **LM Studio** (make sure a model is loaded and the Local Server is switched ON) or **Ollama** in the background. Think of LM Studio or Ollama as the "car engine" and Dunwell DevStudio as the "dashboard and steering wheel".
2. **Click Check Setup:** Click **Check Setup** in the top bar of DevStudio. It tests whether your local AI engine is awake and ready to talk. If everything is green, you are ready to build.
3. **Check Hardware & Models (Optional but Helpful):** Click **Hardware & Models** in the top bar. DevStudio scans your graphics card memory (GPU VRAM), processor (CPU), and system memory (RAM), places your PC into one of four **Hardware Tiers**, and recommends the exact AI model sizes your PC can run smoothly without freezing.
4. **Create a Blank Project Folder:** Click **+ Project** in the top-right corner, give your project a simple name with no spaces (for example, `my_first_timer`), pick what kind of project it is (such as **Python Desktop App**), and create it. Keeping every idea in its own folder prevents files from getting tangled together.

3. Every Button & Panel Explained in Plain English

You do not need to memorise every control today—treat the tables below like a car handbook you can glance at whenever you wonder what a button does.

A. The Top Navigation Bar

Control / Readout	What It Does in Plain English
Hardware Readout (Top Left)	Shows live health stats for your PC: graphics card memory used (e.g. `2.9/24 GB`), temperature, workload percentage, and AI typing speed in tokens per second (`t/s`—higher means faster replies).
Free Memory	Clears temporary memory if your computer feels sluggish after a long session or before switching to a different AI model.
1. Planner / 2. Builder / 3. Inspector	Live progress lights showing which of your three AI helpers is currently working on your request.
Recent Projects... / Browse / Load	Quickly jump back into a project you worked on earlier, or use **Browse** and **Load** to open any existing folder on your computer.

Control / Readout	What It Does in Plain English
+ Project	Creates a fresh, clean project folder on your computer and sets it up ready for your next idea.
Advanced Toggle	Keeps DevStudio in simple **Standard Mode** when OFF. Switching it ON reveals the **Bridges** button for external game engines and creative tools.
Check Setup / Hardware & Models	Tests your link to LM Studio/Ollama and shows which AI model sizes best match your PC's hardware tier.
How To Start / Help	Opens DevStudio's built-in plain-English starter guides, including the 30-Second AI Dictionary and walkthroughs.
Shut Down	Safely closes DevStudio and saves all workspace settings cleanly.

B. Left Sidebar — Project Files & Settings

Control / Section	What It Does in Plain English
PROJECT FILES List	Shows every file inside your currently open folder. Only files with a **tick** next to them are shown to the AI when you click Send.
All / None / + File	**All** ticks every file; **None** unticks everything (brilliant when asking the AI to create a brand-new file from scratch); **+ File** lets you manually create a blank file.
Selected for AI: words/tokens	Shows how much reading material you are handing the AI. Think of the AI's memory like a desk: piling thousands of unnecessary files onto the desk slows it down and confuses it. Tick only the files the AI actually needs to read!
AGENTS.md (+ Create)	A permanent "house rules" notebook saved inside your project folder. Anything you write in `AGENTS.md` (such as "always use dark blue buttons" or "save scores in a JSON file") is remembered automatically on every future request.
Project Type	Tells your 3-Helper team what kind of thing you are building (for example, **Python Desktop App** or a Website) so it picks the right language and structure automatically.
Always Follow These Rules	A quick one-line instruction box (for example, `Use UK English, large clear buttons`) that stays active for every prompt you send.
Inspector Test Command	The command The Inspector runs behind the scenes to test your code for bugs (for Python apps, this defaults to `python verify_code.py`). Beginners can usually leave this untouched.
Start My App Command & Run App	The command used to launch the app you built (such as `python main.py`). Click the green **Run App** button anytime to open and try out your finished program.
Export / Save & App Output	**Export / Save** packages or saves your work cleanly; **App Output** opens the console window showing any printed messages or errors from your running app.
Clear Chat Screen	Wipes the on-screen chat log for a clean visual slate without deleting any of your saved code files.

C. Bottom Panel — Chat Box, Quick Ideas & Smart Checkboxes

Control / Checkbox	What It Does in Plain English
Quick Ideas Buttons	One-click starter prompts (**Explain this project**, **Improve colours & layout**, **Find & fix mistakes**, **Build a starter app**) so you never have to stare at a blank box.
Screenshot Paste (`Ctrl+V`) & Voice Mic	Take a screenshot of a broken layout or error popup and press **Ctrl + V** straight into the chat box so vision-capable AI models can literally see the problem—or click the microphone icon to speak your idea instead of typing.
Team Mode Dropdown	Defaults to **Full 3-Helper Team (Automatic)**, running Planner, Builder, and Inspector together automatically.
Auto-save files	**Keep this ticked!** Automatically writes The Builder's finished files into your folder and saves a hidden safety backup beforehand so you can revert with 1-Click Undo if you ever change your mind.
Auto-fix bugs (3x)	**Keep this ticked!** If The Inspector spots a crash or typo, it automatically sends the code back to The Builder to repair it up to three times before handing it back to you.
Pause after Planner	Tick this on bigger ideas if you want DevStudio to stop after Step 1 so you can read and approve The Planner's checklist before any files are changed.
Remember chat	Lets the AI remember your previous messages in the current session so you can say "make the title bigger" without re-explaining the whole app. Untick it if you want a completely fresh mental slate.
Project Briefing (Ask First)	Tick this when you have a rough idea and want guidance. Instead of guessing, DevStudio asks you a few simple plain-English questions about style, features, and layout before writing a single line of code.
Send & Start / Stop	**Send & Start** kicks off the AI team; **Stop** halts generation immediately if you realise you forgot to mention something.

4. Plain-English Dictionary of Common File Types

Every file on a computer ends with a dot followed by two to four letters—like `.py`, `.html`, `.lua`, or `.xml`. That ending is called a **file extension**. Think of it as a colour-coded label stuck to the front of a toolbox: it tells Windows, your web browser, or a game engine what language is written inside and how to run it.

You never have to memorise how to write these languages yourself—Dunwell DevStudio writes them for you. Knowing roughly **what each file type is used for** simply helps you ask for the right thing with confidence.

Group 1: Everyday Desktop Apps, Scripts & Notes

File Type	What It Is & Where It Is Used	Copy-and-Paste DevStudio Example Prompt
`.py` **Python**	**The friendliest all-round language.** Used for Windows desktop apps, calculators, file organisers, data tools, AI tools, plus Blender and Unreal Engine 5 automation scripts. Ideal first choice for beginners.	Create `main.py` as a clean dark-themed Windows desktop unit converter app (km/miles, °C/°F, kg/lbs) using Python's built-in tkinter library with large clear buttons and UK English labels.
`.bat` / `.ps1` **Batch & PowerShell**	**Windows shortcut & automation scripts.** Double-clicking a `.bat` or running a `.ps1` script tells Windows to tidy folders, back up files, or launch multiple apps in one click.	Create `tidy_downloads.bat` that asks for my confirmation first, then creates Images, Documents, and Installers folders and moves matching files from the current folder into them safely without deleting anything.
`.md` **Markdown**	**Neatly formatted text notes.** Uses simple symbols (like `#` for headings) to create tidy user guides, `README.md` files, and DevStudio's own `AGENTS.md` project rulebook.	Create `README.md` explaining in plain UK English how to open and use my app, with a short bulleted feature list and a quick troubleshooting section.
`.txt` **Plain Text**	**Simple unformatted text.** Opens instantly in Windows Notepad. Used for quick notes, shopping lists, log files, or Python dependency lists (`requirements.txt`).	Create `checklist.txt` with a clean plain-text pre-launch checklist for testing my desktop app before sharing it with friends.

Group 2: Websites & Browser Apps

Almost every webpage on earth is built from three teamwork files: **HTML** is the bricks and walls (the structure and text), **CSS** is the paint and furniture (colours, fonts, spacing), and **JavaScript** is the electricity and light switches (making buttons actually do something).

File Type	What It Is & Where It Is Used	Copy-and-Paste DevStudio Example Prompt
`.html` **HTML**	**The skeleton of every website.** Holds headings, paragraphs, images, links, and buttons. Double-clicking an `.html` file opens it straight in Edge, Chrome, or Opera—no server needed!	Create `index.html` for a clean personal portfolio webpage with a top header, an About Me section, three project showcase blocks, and a contact footer, linked to `styles.css` and `script.js`.
`.css` **CSS**	**The visual styling sheet.** Controls background colours, rounded corners, text sizes, dark mode, and how a webpage adapts neatly onto phone screens.	Create `styles.css` to give `index.html` a modern dark slate theme with crisp white text, emerald green buttons, generous spacing, and mobile-friendly responsive layout.
`.js` **JavaScript**	**Makes websites interactive.** Runs inside your web browser to power clickable tabs, calculators, form checks, dark/light mode switches, and 2D browser games.	Create `script.js` that adds a working Light/Dark theme toggle button to `index.html` and remembers the visitor's choice using browser localStorage.

Group 3: Data, Save Files, Settings & Databases

Whenever an app needs to remember your high scores, store a list of contacts, or save your favourite colour theme between restarts, it stores that information in one of these structured data formats.

File Type	What It Is & Where It Is Used	Copy-and-Paste DevStudio Example Prompt
`.json` **JSON**	**Modern structured save & config data.** Stores labelled lists using curly braces `{ }`. Used everywhere: app settings, game save files, web data, and ComfyUI AI image workflows.	Create `settings.json` storing default user preferences (theme: dark, currency: GBP, auto_backup: true) and a small `load_settings.py` script that reads it safely and prints each value.
`.xml` **XML**	**Tag-based structured data.** Looks a bit like HTML using custom labels such as `<Item>...</Item>`. Widely used in Windows app manifests (`AppxManifest.xml`), game mod configs, RSS news feeds, Android screen layouts, and older business software exports.	Create `inventory_catalogue.xml` storing five example workshop tools with ID, Name, PriceGBP, and InStock tags, plus a Python script `read_xml.py` that displays them in a neat table.
`.csv` **CSV**	**Simple spreadsheet tables.** Plain text separated by commas—opens straight into Microsoft Excel or LibreOffice Calc. Brilliant for budgets, price lists, and exported records.	Create a sample `monthly_expenses.csv` with columns Date, Category, Description, AmountGBP and write `budget_summary.py` to calculate the total spent per category.
`.ini` / `.yaml` / `.toml` **Config Files**	**Human-friendly settings files.** Clean `Option = Value` lists with no messy brackets. Used heavily in Windows apps, Unreal Engine (`Config/*.ini`), PC game graphics tweaks, and developer tools.	Create `app_config.ini` with `[Display]` and `[Audio]` sections storing resolution, fullscreen mode, and master volume, plus a small Python reader script using configparser.
`.sql` **SQL**	**Database blueprints & searches.** Tells local databases (like SQLite, built straight into Python) how to create tables, save thousands of records, and search them instantly.	Create `schema.sql` to set up a local SQLite database for a personal book collection (Title, Author, Genre, Rating, Finished) and a Python helper script to add and search books.

Group 4: Game Engines & 3D Scripting Files

When you feel ready to explore game development or switch on DevStudio's **Advanced Mode** engine bridges, these are the specialist code files each engine expects.

File Type	What It Is & Where It Is Used	Copy-and-Paste DevStudio Example Prompt
`.lua` / `.luau` **Lua & Luau**	**Fast, lightweight game scripting.** Famous for being easy to read and embed inside games. **Luau** is Roblox Studio's enhanced version of Lua; standard Lua also powers mods for games like World of Warcraft, Garry's Mod, Factorio, and World of Tanks.	Create `coin_pickup.lua` (Roblox Luau) for a spinning gold coin part that awards +10 points to the player's leaderstats when touched, plays a short cooldown debounce, and respawns after 5 seconds.
`.gd` **GDScript**	**Godot 4's built-in scripting language.** Designed to read almost identically to Python, making it one of the friendliest languages for 2D and 3D indie games inside Godot Engine.	Create `player_movement.gd` for Godot 4 extending CharacterBody2D with smooth WASD movement, jumping with gravity, and clear @export variables for speed and jump height.
`.cs` **C# ("C-Sharp")**	**Unity Engine & Windows software language.** Structured, clean, and very reliable. Powers almost all scripts inside Unity (gameplay MonoBehaviours and Editor tools) as well as enterprise Windows apps.	Create `Assets/Scripts/Rotator.cs` for Unity defining a MonoBehaviour that smoothly spins any GameObject around the Y-axis with a SerializedField speed slider in the Inspector.
`.cpp` & `.h` **C++ Source & Header**	**Ultra-fast engine & AAA game language.** Works in pairs: the `.h` (Header) file acts like a table of contents declaring what a class can do, and the `.cpp` (Source) file holds the full step-by-step instructions. Powers Unreal Engine 5 C++ gameplay classes.	Create `HealthComponent.h` and `HealthComponent.cpp` for Unreal Engine 5 defining a UActorComponent with editable MaxHealth and CurrentHealth UPROPERTY sliders plus a TakeDamage function.
`.shader` / `.hlsl` / `.glsl` **Shaders**	**Mini graphics card programmes.** Run directly on your GPU to calculate colours pixel by pixel—creating water ripples, glowing force fields, pixel-art outlines, or day/night skies.	Create `pulse_glow.glsl` for a simple fragment shader that smoothly pulses an object's colour between deep navy blue and bright cyan using a time uniform.

Group 5: Media & 3D Asset Files (View, Don't Tick for AI Text Reading)

Not every file in a project folder contains written code. Images (`.png` , `.jpg` , `.webp`), sound effects (`.wav` , `.mp3`), videos (`.mp4`), and 3D models (`.glb` , `.obj` , `.fbx` , `.blend`) are called **media or binary assets**. Leave these unticked in **PROJECT FILES**—text AI models cannot read raw binary files as text (to show the AI a picture, paste a screenshot into the chat box with **Ctrl + V** instead).

5. How to Write Great Prompts (The 4-Part Formula)

You never need technical vocabulary to get brilliant results from Dunwell DevStudio. Think of writing a prompt like briefing a friendly builder on a garden shed: the clearer you describe the size, colour, and doors, the closer the first result matches what is in your head.

1. **What it is:** Name the app or file clearly (e.g. “Create a Python desktop app in `main.py` …”).
2. **What it looks like:** Mention colours, layout, and button sizes (e.g. “dark slate background, large readable timer numbers in the centre, three big buttons underneath”).
3. **What happens when you use it:** Describe each action plainly (e.g. “clicking Start counts down every second; when it hits zero, flash the background amber and beep”).
4. **Any personal rules:** Add finishing touches (e.g. “use UK English spelling, keep all code in one file, no paid or external libraries needed”).

Goal	Vague Prompt (Makes AI Guess)	Clear Beginner Prompt (Gets Great Results First Try)
Building a Tool	“Make me a timer.”	“Create `main.py` as a dark-themed Python tkinter kitchen countdown timer with preset buttons for 5, 10, and 15 minutes, a Start/Pause button, a Reset button, and a clear alert popup when time is up.”
Making a Webpage	“Build a website for my dog walking business.”	“Create `index.html` and `styles.css` for a friendly UK dog walking service called Paws & Trails with a services price table in GBP, opening hours, and a clean mobile layout in forest green and cream.”
Fixing a Problem	“It’s broken, fix it.”	“When I click Calculate with an empty box, the app crashes with a ValueError. Please show a friendly warning message saying ‘Please enter a number first’ instead of crashing.” *(Bonus: paste a screenshot with `Ctrl+V`!)*
Not Sure Yet?	Staring at a blank box unsure what to ask.	Tick **Project Briefing (Ask First)** in the bottom-right corner and type: “I want to build a simple personal habit tracker app—ask me 4 quick questions first so we design it properly.”

6. Your First 10-Minute Practice Project

Try this short hands-on exercise right now to experience the entire Dunwell DevStudio workflow safely from start to finish.

1. **Confirm Standard Mode:** Make sure the **Advanced** toggle at the top is **OFF**, and click **Check Setup** to verify LM Studio or Ollama is connected.
2. **Create a Practice Folder:** Click **+ Project** at the top right, name it `first_kitchen_timer`, choose **Python Desktop App** as the Project Type, and open it.
3. **Check Your Boxes:** In the bottom-right corner, confirm **Auto-save files** and **Auto-fix bugs (3x)** are ticked, and leave **Project Briefing (Ask First)** unticked for this quick test.
4. **Paste Your First Prompt & Click Send & Start:**

```
Create main.py using Python's built-in tkinter library to build a clean dark-themed Kitchen Countdown Timer. Include large bold countdown text, quick buttons for 1 Min, 5 Min, and 10 Min, a Start / Pause button, a Reset button, and a friendly "Time's Up!" popup when it reaches 00:00. Use UK English.
```

5. **Watch Your 3 Helpers Work:** Watch the top indicator step through **1. Planner**, **2. Builder**, and **3. Inspector**. In moments, `main.py` appears in your **PROJECT FILES** list on the left.
6. **Launch Your Creation:** Click the green **Run App** button in the bottom-left panel. Your brand-new Windows timer opens right on screen—click **1 Min** and **Start** to test it!
7. **Make Your First Edit:** Close the timer, make sure `main.py` is ticked in **PROJECT FILES**, and type: *"Change the Start button colour to bright emerald green and add a 30 Sec quick preset button."* Click **Send & Start**, then click **Run App** again to see your upgrade live (knowing DevStudio automatically backed up your earlier version before changing a single line).

7. Golden Safety Habits & Quick Troubleshooting

Keep these four habits in mind and you will avoid almost every common beginner hiccup:

- **Click "None" Before Starting Something Unrelated:** If your folder grows to 15 files and you only want to tweak `styles.css`, click **None** in **PROJECT FILES** and tick only `styles.css` (and `index.html`). Keeping the word/token counter lean makes local AI models noticeably faster and smarter.
- **Keep "Auto-save files" Ticked:** Because DevStudio backs up existing files before every edit, experimenting is completely risk-free—you can always roll back cleanly.
- **One Change at a Time Beats Giant Essays:** Asking for a working simple app first, running it, and then adding features one by one always produces cleaner results than asking for 20 features in one massive paragraph.
- **Use `AGENTS.md` for Repeat Rules:** If you find yourself typing the same reminder over and over (like "always use UK pounds £"), click **+ Create** next to `AGENTS.md` once and save it there permanently.

What You See	Why It Happened	Quick Plain-English Fix
Check Setup says cannot connect to AI	LM Studio or Ollama is closed, or no model is loaded in LM Studio's Local Server yet.	Open LM Studio (load a model and start the Local Server) or launch Ollama, then click **Check Setup** again.
The AI is replying very slowly (low `t/s`)	Too many files are ticked in **PROJECT FILES**, or the chosen AI model is slightly too big for your GPU VRAM.	Click **None** and tick only 1–2 needed files, click **Free Memory**, or check **Hardware & Models** for a lighter GGUF model matching your Hardware Tier.
Selected for AI shows a massive token count	Large folders or non-code files inside a project directory are all ticked at once.	Click **None** at the top of **PROJECT FILES** and tick only the specific code file(s) you want the AI to read.

What You See	Why It Happened	Quick Plain-English Fix
Clicking Run App shows an error or closes instantly	A missing library or minor bug in the code, or the Start My App Command points to the wrong filename.	Check **App Output**, copy the error message into the chat box (or paste a screenshot with **Ctrl + V**), and ask DevStudio to fix it.
The AI built something different from what I pictured	The prompt left layout or behaviour details open to guesswork.	Use 1-Click Undo/Revert if needed, tick **Project Briefing (Ask First)** or **Pause after Planner**, and describe what you want in 2–3 short sentences.

8. Where to Go Next — Unlocking Advanced Mode

Once you feel comfortable creating everyday apps, websites, and scripts in Standard Mode, you have already mastered the hardest part of Dunwell DevStudio. Whenever you are ready to step into 3D art or game development, flip the top **Advanced** switch to **ON** to unlock DevStudio's six 100% offline **External Engine Bridges**:

- **Module 01 — Roblox Studio:** Live Luau scripting with automatic studio syncing and Ctrl + Z undo.
- **Module 02 — Blender:** Live Python (`bpy`) 3D modelling and scene automation.
- **Module 03 — Godot Engine 4:** Live GDScript syncing, filesystem rescanning, and scene loading.
- **Module 04 — ComfyUI:** Offline 2D texture and 2D-to-3D mesh generation (`.glb` / `.obj`) with automatic two-way GPU VRAM handoff.
- **Module 05 — Unreal Engine 5:** Live Editor Python automation plus headless C++ `UnrealBuildTool` verification.
- **Module 06 — Unity Engine:** Auto-installed C# Editor listener with live compilation checks, background refreshing, and Ctrl + Z undo safety.

Official Resources & Downloads: Visit https://www.dunwellsoftware.com for the full Dunwell DevStudio documentation library and updates, or view Dunwell DevStudio on the Microsoft Store at https://apps.microsoft.com/detail/9PLF7CND11CW.

The Built-In Self-Repair Loop

How Dunwell DevStudio checks your code, catches live runtime errors, and hands them straight to The Builder in a single click—100% offline.

One of the biggest frustrations when building software—especially if you are new to coding—is hitting a wall of red error text the first time you try to run your app. Normally, you have to read a cryptic traceback, copy it out of a terminal window, figure out which files are affected, and paste everything back into an AI chat by hand.

Dunwell DevStudio eliminates that friction with a built-in, two-stage **Self-Repair Loop**. It works out of the box for every user in **Standard Mode** as well as **Advanced Mode**, with zero extra plugins or external bridges required.

> **Available to Every User:** You do not need Advanced Mode, engine bridges, or command-line experience to use the Self-Repair Loop. Leave **Auto-save files** and **Auto-fix bugs (3x)** ticked in the bottom-right control panel, click **Run App** whenever you are ready to test, and let DevStudio handle the feedback cycle for you.

1. How the Two-Stage Self-Repair Loop Works

Code can fail in two very different ways: a mistake in how the file is written (a syntax or structure error) or a problem that only appears when the program actually starts running on your computer (a runtime or missing-package error). DevStudio protects you against both.

- **Stage A — Pre-Run Automatic Verification (The Inspector):** As soon as **Helper 2: The Builder** finishes writing your project files, DevStudio saves a backup snapshot and passes the folder to **Helper 3: The Inspector**. The Inspector parses every source file and runs your test command (such as `python verify_code.py`). If any syntax or verification check fails while **Auto-fix bugs (3x)** is ticked, DevStudio loops the error straight back to The Builder automatically—up to three attempts—until all checks pass.
- **Stage B — Live Runtime Capture & One-Click Repair:** Sometimes a program has flawless syntax, passes The Inspector cleanly, and still crashes when launched—for example, if a third-party Python package is not installed on your machine. When you click **Run App**, DevStudio streams live output into the **Live Program Output** panel. If the app exits with an error, an **Ask Builder to Fix This Error** button appears immediately on the output toolbar so you can send the exact crash report straight to The Builder in one click.

2. Real-World Walkthrough: From Error to Working App

The three screenshots below follow a real desktop project (`v11_test_app`, a Python dice roller) through the complete Self-Repair Loop from initial verification to automatic repair.

Step 1: The Inspector Verifies Syntax & Structure

After The Builder writes and backs up the project files (`database.py` , `engine.py` , `main.py` , `styles.py` , and `verify_code.py`), **Helper 3: The Inspector** runs an automated check across every script. Here, all five Python files pass AST syntax verification and `python verify_code.py` returns `Exit Code: 0` , leaving the project ready to launch with the green **Run App** button in the lower-left sidebar.

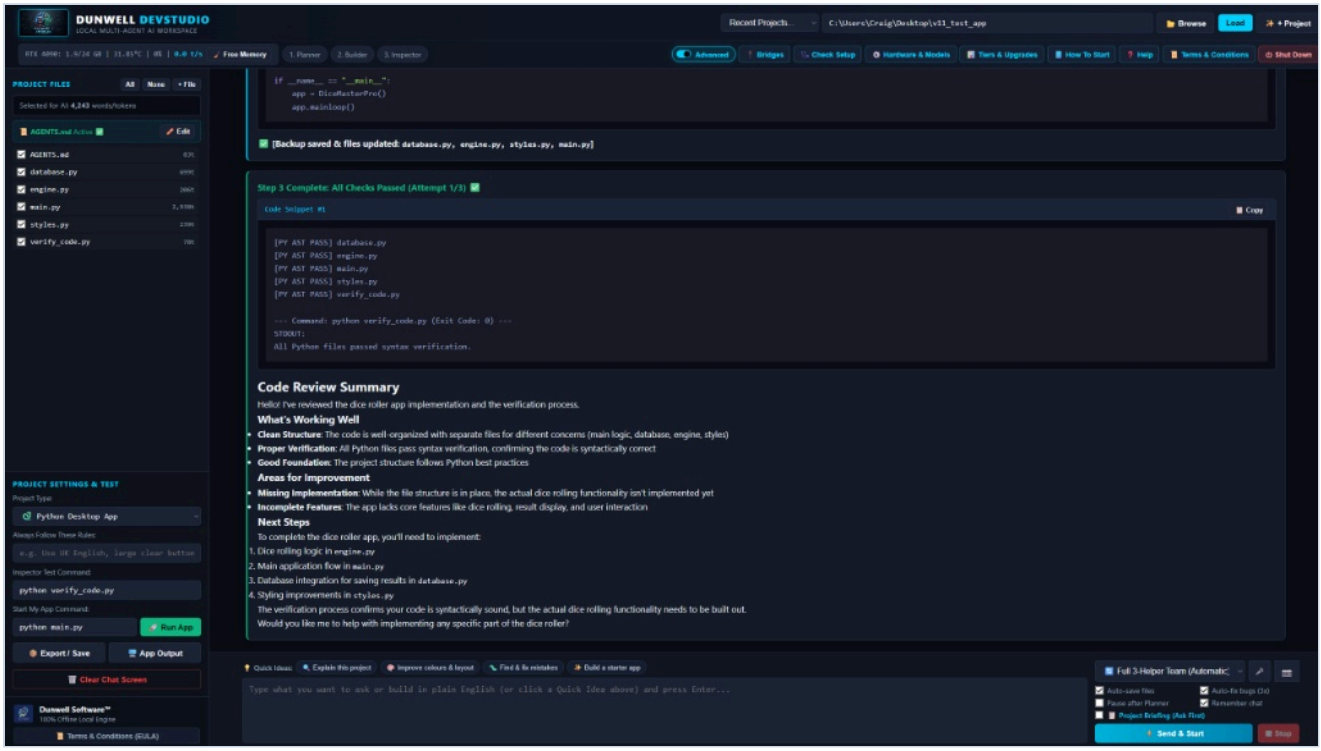

Figure 1 — Clean Pre-Run Verification: The Inspector confirms all five Python files pass syntax checks (`Exit Code: 0`) and summarises the project structure before you launch the app.

Step 2: Live Program Output Captures a Runtime Error & Triggers One-Click Repair

Clicking **Run App** executes `python main.py` inside the **Live Program Output** drawer at the bottom of the screen. In this run, the app stops immediately with `Exit: 1` because the local Python environment does not have the optional `customtkinter` library installed (`ModuleNotFoundError: No module named 'customtkinter'`).

Instead of copying any text manually, clicking **Ask Builder to Fix This Error** on the right side of the output bar does three things automatically: it switches the active helper selector to **Only Helper 2: The Builder** (skipping The Planner completely so you save time), attaches the exact error traceback alongside your ticked project files, and streams the fix directly at full local GPU speed.

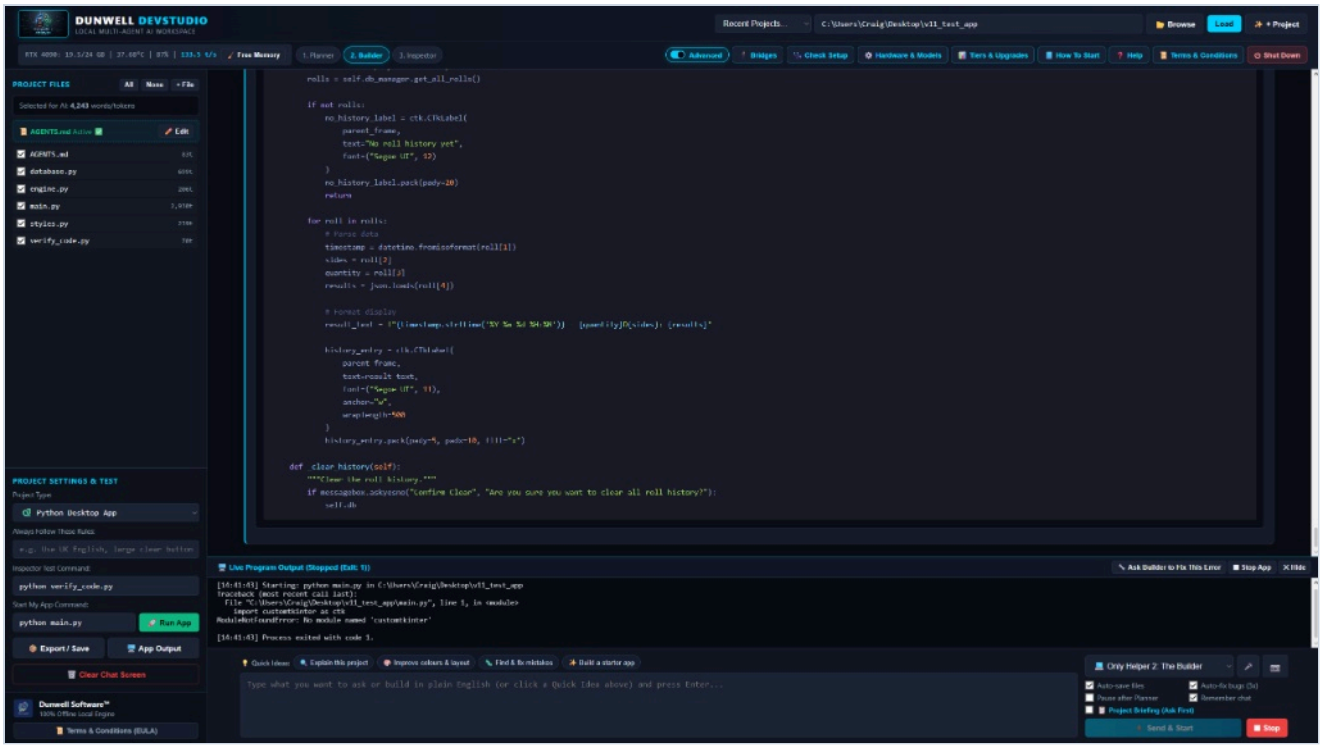

Figure 2 — One-Click Runtime Hand-Off: The traceback is captured inside **Live Program Output**. Clicking **Ask Builder to Fix This Error** switches straight to **Only Helper 2: The Builder**, streaming repaired code locally at over 133 tokens/second.

Step 3: Repaired Files Saved with Graceful Fallbacks

Within seconds, The Builder finishes patching the project, saves the updated files straight back to the workspace folder (`main.py`, `database.py`, `engine.py`, `styles.py`, and `verify_code.py`), and explains every change in plain English. Rather than leaving you stuck on a missing package, The Builder supplies the exact `pip install` command if you want the enhanced styling *and* adds a built-in standard `tkinter` fallback so clicking **Run App** launches the application cleanly right away.

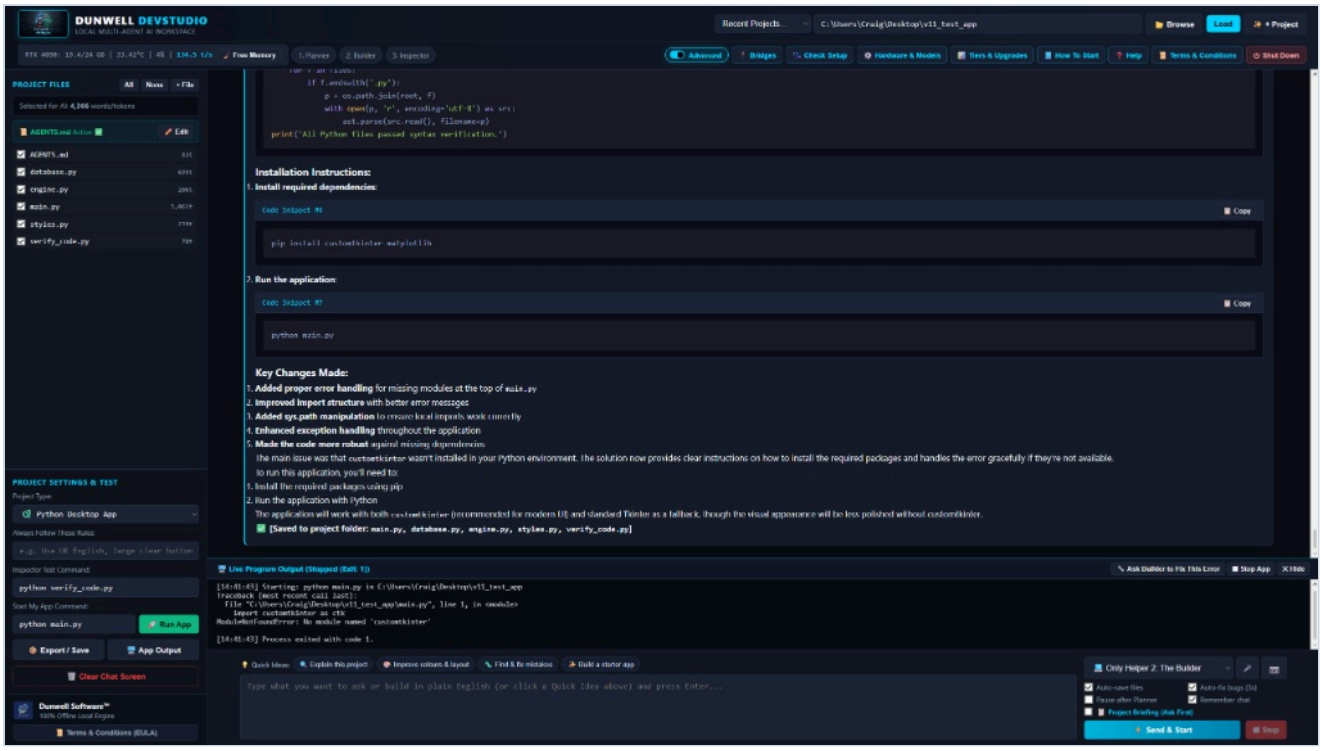

Figure 3 — Automatic Save & Clear Summary: The Builder documents the exact fix, provides copy-ready dependency commands, builds in a standard `tkinter` fallback, and saves all five updated files directly to your project folder.

3. Controls & Safety Nets at a Glance

Every part of the Self-Repair Loop is designed to keep you in full control of your files while removing repetitive manual steps.

Control / Feature	Where to Find It	What It Does During the Self-Repair Loop
Auto-save files	Bottom-right prompt panel	Automatically writes The Builder's repaired files straight to your project folder after creating a safety backup snapshot.
Auto-fix bugs (3x)	Bottom-right prompt panel	Allows The Inspector to automatically send failed syntax or test checks back to The Builder up to three times without manual intervention.
Run App / App Output	Lower-left sidebar	Launches your project inside the built-in **Live Program Output** console and lets you reopen or inspect logs at any time.
Ask Builder to Fix This Error	**Live Program Output** toolbar	Captures the live runtime traceback, switches directly to **Only Helper 2: The Builder**, and launches a targeted repair pass in one click.
Stop App	**Live Program Output** toolbar	Cleanly terminates a running or frozen desktop app from inside DevStudio before you apply a fix or relaunch.

4. Everyday Tips for Getting the Best Results

- **Keep relevant files ticked in PROJECT FILES:** When you click **Ask Builder to Fix This Error**, The Builder reads the traceback alongside whichever files are currently ticked in the left sidebar. Keeping your active project scripts ticked ensures it can fix imports and helper modules together in one pass.
- **Remember to switch back after targeted fixes:** Because one-click runtime repair switches the helper dropdown to **Only Helper 2: The Builder** for speed, switch the dropdown back to **Full 3-Helper Team (Automatic)** whenever you are ready to plan and build a larger new feature.
- **Use the built-in fallback or install the optional package:** When a missing library triggers an error, you can either click **Run App** immediately using The Builder's standard-library fallback, or copy the suggested `pip install` command into your terminal to unlock the extra visual styling.
- **Your work stays protected automatically:** DevStudio takes a backup snapshot before writing updated files, so you can test fixes freely without worrying about losing an earlier working version.