SOFTWARE & ARTIFICIAL INTELLIGENCE

Building Your First Real AI Project

Go from a scattered idea to a working, documented AI tool on GitHub — the same workflow behind every credible software portfolio piece.

SKILL LEVEL: BEGINNER–INTERMEDIATE **BUILD TIME:** 1–2 WEEKENDS **EST. COST:** $0–10

> Before you start: this blueprint assumes basic Python (variables, functions, loops). If that's new, spend a few hours on a free intro course first — you don't need to be fluent, just comfortable.

01 Objective

Most beginner "AI projects" are a single Jupyter cell that calls an API once and never gets touched again. By the end of this blueprint, you'll have something different: a small but complete AI-powered tool — with a real project structure, version control, error handling, a simple interface, and a README — that you can point someone to and say **"here's what I built."** The goal isn't a fancy model. It's proving you can take an idea from a blank folder to something a stranger could install and run.

02 Prerequisites

- Python 3.10+ installed, and comfort running scripts from a terminal
- A free GitHub account (github.com) — this is where your project will live
- Comfort installing packages with `pip` — if this is new, practice with one small package first

03 Recommended Toolkit

TOOL	PURPOSE	NOTES
Python 3.10+	Core language	Free — python.org
VS Code	Code editor	Free, with the Python extension installed
Git + GitHub	Version control & hosting	Free for public repositories
Virtual environment (venv)	Keeps project dependencies isolated	Built into Python, no install needed
An AI API key (OpenAI, Anthropic, or Hugging Face)	Powers the "AI" part of your project	Most offer a small free tier; budget ~$5–10 for testing
Streamlit	Turns a Python script into a simple web interface	Free, one `pip install`

04 Tools Required

- Computer with VS Code and Python installed
- A terminal (built into VS Code)

> **USING AI RESPONSIBLY**
> If you use an AI assistant (like Claude or ChatGPT) to help write parts of your code, that's normal practice — but you still need to understand every line well enough to explain it, and you should say so plainly in your README under an "AI Assistance" note. Never submit AI-written code for a competition or application without disclosing it if disclosure is required, and never let generated code touch real user data before you've read and understood what it does.

05 Project Architecture

Every AI project — no matter how small — has the same four moving parts. Keeping them in separate files from day one is what makes a project easy to debug, extend, and explain in an interview.

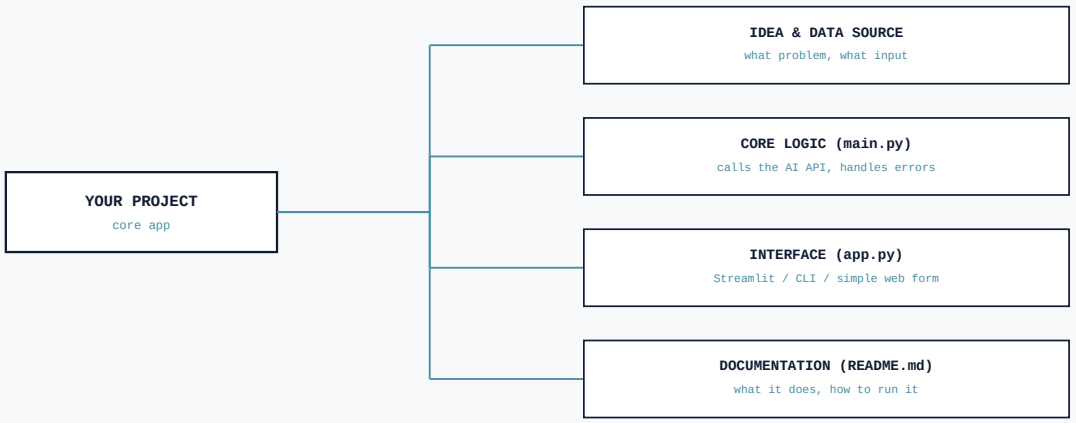

FIG. 1 — THE FOUR PARTS EVERY AI PROJECT NEEDS, KEPT IN SEPARATE FILES

COMPONENT	FILE	RESPONSIBILITY
Core logic	main.py / core.py	Talks to the AI API, processes the response, handles failures
Interface	app.py	Takes user input, displays output — Streamlit is the fastest beginner option
Configuration	.env (not committed)	Stores your API key outside the code
Documentation	README.md	Explains the project to anyone who finds it, including future you

06 Step-by-Step Build

1. **Choose an idea worth building.** Pick a problem you actually have — a study tool, a text summarizer for your own reading, a simple recommender. Avoid "build a chatbot" with no target user; specificity is what makes a project defensible later.
2. **Set up the project folder.** Create a folder, run `python -m venv venv`, activate it, then `git init`.
3. **Make your first commit before writing any AI code.** Commit a README stub and a `.gitignore` (include `venv/` and `.env`). This habit alone prevents the most common beginner mistake: leaking an API key.
4. **Write the core function first, without a UI.** Get one working call to your AI API printing a result to the terminal. Confirm it works before building anything on top of it.
5. **Add error handling.** Wrap the API call in a try/except block and handle at least: missing API key, network failure, and empty input. This is the single biggest gap between "demo that works once" and "tool that works."
6. **Build a minimal interface.** Wrap your core function in a Streamlit app — a text box, a button, and an output area is enough.
7. **Write the README properly.** What it does, how to install it, how to run it, and one line about how AI assistance was used, if any.
8. **Deploy it.** Streamlit Community Cloud, Render, or Hugging Face Spaces all offer free hosting — push your repo and connect it in a few clicks.

07 Code Walkthrough

This is a minimal, honest version of a core AI function — not a toy that hides its failure modes. Notice the three things a beginner script usually skips: reading the key from the environment (never hardcoded), a docstring explaining what the function does, and explicit error handling.

```
# core.py — the function everything else calls
import os
from anthropic import Anthropic

client = Anthropic(api_key=os.environ.get("ANTHROPIC_API_KEY"))

def summarize(text: str) -> str:
    """Return a 2-sentence summary of `text`, or raise a clear error."""
    if not text.strip():
        raise ValueError("Input text is empty.")
    if not os.environ.get("ANTHROPIC_API_KEY"):
        raise EnvironmentError("Missing ANTHROPIC_API_KEY — set it in your .env file.")
```

```
    try:
        response = client.messages.create(
            model="claude-sonnet-5",
            max_tokens=200,
            messages=[{"role": "user", "content": f"Summarize in 2 sentences: {text}"}]
        )
        return response.content[0].text
    except Exception as e:
        # Fail loudly and clearly instead of silently returning None
        raise RuntimeError(f"AI request failed: {e}")
```

Note: the API key is read from an environment variable, never written into the file itself.

08 Testing & Troubleshooting

SYMPTOM	LIKELY CAUSE	FIX
GitHub flags a secret / key stops working	API key was committed to the repo	Revoke the key immediately, generate a new one, and add `.env` to `.gitignore` before continuing
App crashes with no useful message	No error handling around the API call	Wrap the call in try/except and print a specific message for each failure type
Deployment fails but it runs locally	Missing `requirements.txt`	Run `pip freeze > requirements.txt` before deploying
Reviewer can't get it running	README missing setup steps	Add exact install and run commands — assume zero context
One 300-line file, hard to explain	No separation between logic and interface	Split into core.py and app.py as in Section 5

09 Where This Goes Next

FROM WORKING TOOL TO PORTFOLIO PIECE

A deployed project with a clean README is already more than most applicants show — but the step that makes it memorable is a short write-up of what problem it solves and what you'd improve next, pinned to your GitHub profile. Pair this blueprint with **SA-002 (Deploying and Scaling AI Applications)** to learn how to handle real users and rate limits once people other than you are using it — and see The University & Scholarship Playbook module for how to describe this project on applications.