Sprint V2.1 — E2E Round-Trip (OpenAI)

What I need from you (once, in the new thread)

1. **Repo pointer:** confirm the branch you want me to work from (e.g., `milestone/promptforge-20250914-2255` or `feature/v2-scaffold`).

   Response: milestone/promptforge-20250914-2255

   This was a brief branch created to test that secure items are being ignored and filtered on a push otherwise it contains everything to the last real push.

   So this can be a starting off point

   I need more education on how github works why do we have three active branches going on

   I thought when we did a push it closed branches but maybe there's some other action needed I could use some education on how this works

2. **API config:** a funded OpenAI key available to the app (don't paste it here).

   - Keep it local in `.env` (already ignored).

     Done: Placed in G:\My Drive\Code\Python\PromptForge\.env

   - Optional org: `OPENAI_ORG_ID=` if you use multiple orgs.

     NA

3. **Models to target:** names for Channel A (code/JSON) and Channel B (prose). If unsure, say "use sensible defaults."

I could use more education on this meaning of models. Here is the iinitial use case to start with.

troubleshooting debugging and fixing code as we are going through testing

AI creates the code I run the code and have to copy and paste the results of the errors out of the terminal output into a chat get your response copy your your corrections paste them into Visual Studio and test and I become a copy paste jockey what I would like to do is be able to troubleshoot this where the the application would take an error that I receive from VS terminal output structure it so that I can provide supporting information such as a screenshot, and source files that you may not have the most recent copy of. The prompt language would request a fix.

Your response would be the fix in a structured format that I could enter directly into Visual Studio code presuming it meets my supervisory approval

in addition to that the prose would be communicating the thinking how things are supposed to work, expected outcomes

The structured code back would be in one or more complete code files that could be added directly to Visual Studio so that I don't have to copy and paste and fat-finger and make mistakes.

It would remember basic rules like:

always deliver a complete

file diffs are prohibited

Provide syntax compliant with the environment, PowerShell 7, Python 3.12, etc.

and any other rules or directives that would be added and included in a prompt.

4. **Project Root path** you want to use for Apply/Undo tests (a test repo is fine).

We are eating our own dogfood. I will latest the application on itself. Douglas Hofstadter would be proud of this recursion

5. **Acceptance files:** 1–2 tiny sample files you'd like Channel A to modify (so we can Validate → Apply → Undo visibly).

   I envision providing sample errors. The response would be terminal commands or file corrections from errors received in the terminal output.

   Our first tests will be getting the Open AI API to work.

6. **Confirm Python** = 3.12, Windows 11 (yep), and that you're okay keeping the **mock provider** available as a fallback.

   Yes Python 3.12, Windows 11, ODBC 18. Should we have an environment section.

   I have set up the OPEN AI API key

What I'll deliver in Sprint V2.1

- **Provider switch** in Settings: `openai` / `mock` (env `PF_PROVIDER` honored).
- **OpenAI E2E**: Call Model (A) returns a valid `{files:[...]}` payload ready for **Validate** → **Apply**; Call Model (B) returns prose.

  It occurs to me that In every case, Model A and B are expected in return. A would be code changes, B would be narrative about what changed, expected outcomes, etc. I don't see it as different models, rather partitioning the code and commands that need to be in compliance, from the prose that is for human consumption.

- **Graceful API errors**: friendly banner for quota/invalid key with "Retry" + "Copy error" (still logged to `pfv2.log`).
  Yes

- **Show Effective Rules** button in Review (renders the exact SYSTEM rules used for the last Build).
  Yes, and include them if needed in the prompt as a reminder and to set expectations for the prompt
- **Tooltips (JSON-driven)** on major controls; easy to edit.
  Yes
- **Dry-Run**: paste a Channel-A JSON payload and run **Validate** → **Apply** without any API call.
  Yes
- **One-click venv repair** from Settings (reinstalls deps and rewrites shims if needed).

  I think One-click venv repair is a perfect example of a repeatable use case. Others are:

  Create a handoff archive for a new sprint

  Provide commonly used command for any given project

  Pytest

  Git commits, push

  Database backups

  Ideally any of these would be similar to a customizable terminal task tool that is tailored to project.

  The user should be able to click a button and the repeatable post to VS would be based on the need.

  That could be build based on the response, or recommendation of the AI, similar to how Colntinue or Copilot offers it in VS. You have the advantage of more contest, and stronger AI, and acting as an agent.

Acceptance checks (what we'll verify together)

- Build Prompt shows rules in **Prompt Preview** → **SYSTEM**; Review tab can "Show Effective Rules."
  Yes

- **Call Model (A)** → **Validate Reply** → **Apply Files…** modifies the target files; **Undo Last Apply** restores them.
  Yes. And perhaps use AI to fix broken invalid replies.

- **Call Model (B)** returns prose; both channels work with OpenAI and with the mock provider.
  Yes. As mentioned before, I don't see how A and B wouldn't always be needed in every round trip. I will need structured code to apply, and prose to explain.
- On an invalid/empty key, UI shows a clear banner (no raw stack traces); error is copyable and logged.
  Yes
- Tooltips appear on hover for Scenario, Build Prompt, Call A/B, Validate, Apply, Undo, Settings.
  Yes. Will ask AI (you) to flesh out, improve and update as we make changes to the application, so that's an expected deliverable on any major change in functionality.

What to carry into the new thread

Skip the whole transcript. Post this minimal handoff:

Handoff to new thread (paste this):

- Repo: `https://github.com/coreyprator/PromptForge`
- Branch: `promptforge-20250914-2255`
- Python: 3.12 (venv at `C:\venvs\promptforge-v2`) Yes. PowerShell 7 (not bash)
- Provider: **OpenAI**, funded key in `.env` (not shared) Yes. Done.
- Project Root for tests: G:\My Drive\Code\Python\PromptForge
- Target models: `<A model>`, `<B model>` (or "use defaults") Both

- Sample files to modify: `Will provide troubleshooting problems as described.`

- Sprint: **“Sprint V2.1 — E2E Round-Trip (OpenAI)”**

- Goal: Call A→Validate→Apply→Undo + Call B + graceful errors + tooltips + mock fallback
  Yes

Optional: create a clean handoff ZIP (if you want)

You already have this, but here’s the one-liner again:

```
pwsh .\scripts\publish_all.ps1 `
  -RepoRoot "G:\My Drive\Code\Python\PromptForge" `
  -Milestone "Pre V2.1 handoff" `
  -NewBranch -AutoTag
```

It exports seeds, zips a handoff, commits, tags, and pushes.

```
PS G:\My Drive\Code\Python\PromptForge> pwsh .\scripts\publish_all.ps1 `
>>   -RepoRoot "G:\My Drive\Code\Python\PromptForge" `
>>   -Milestone "Pre V2.1 handoff" `
>>   -NewBranch -AutoTag
git: C:\Program Files\Git\cmd\git.exe
python: G:\My Drive\Code\Python\cubist_art\.venv\Scripts\python.exe

== Ensure Git repo
== Configure 'origin' remote

> git remote set-url origin https://github.com/coreyprator/PromptForge.git


== Select branch

> git checkout -B milestone/promptforge-20250915-2110

Switched to a new branch 'milestone/promptforge-20250915-2110'


== Ensure ruff/pytest

G:\My Drive\Code\Python\cubist_art\.venv\Scripts\python.exe: No module named ruff


== Export seeds from DB

Exported JSON seeds to seeds


== Ruff format + fix

> python -m ruff format .

G:\My Drive\Code\Python\cubist_art\.venv\Scripts\python.exe: No module named ruff

> python -m ruff check --fix .

G:\My Drive\Code\Python\cubist_art\.venv\Scripts\python.exe: No module named ruff

No tests folder; skipping pytest.


== Archive source

Staging with ROBOCOPY...
```

```
Created: .\handoff\promptforge_handoff-20250915-211049.zip

== Commit + (optional) tag + push
> git add -A
> git commit -m "Pre V2.1 handoff" --allow-empty
[milestone/promptforge-20250915-2110 59aa9f2] Pre V2.1 handoff              ]
 1 file changed, 0 insertions(+), 0 deletions(-)
> git tag -a milestone-20250915-2110 -m "Pre V2.1 handoff"
> git push -u origin milestone/promptforge-20250915-2110
Enumerating objects: 6, done.es of 33.6 KB (0.0 MB/s)                        ]
Counting objects: 100% (6/6), done.
Delta compression using up to 32 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 214.27 KiB | 4.29 MiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
remote:
remote: Create a pull request for 'milestone/promptforge-20250915-2110' on GitHub by visiting:
remote: https://github.com/coreyprator/PromptForge/pull/new/milestone/promptforge-20250915-2110
remote:
```

```
To https://github.com/coreyprator/PromptForge.git

 * [new branch]      milestone/promptforge-20250915-2110 ->
milestone/promptforge-20250915-2110

branch 'milestone/promptforge-20250915-2110' set up to track 'origin/milestone/pr> git
push origin --tags

Enumerating objects: 1, done.es of 33.6 KB (0.0 MB/s)                      ]

Counting objects: 100% (1/1), done.

Writing objects: 100% (1/1), 199 bytes | 5.00 KiB/s, done.

Total 1 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)

To https://github.com/coreyprator/PromptForge.git


✅ Published: Pre V2.1 handoff

  Branch: milestone/promptforge-20250915-2110

  Tag:    milestone-20250915-2110

PS G:\My Drive\Code\Python\PromptForge>
```

Cost notes (since you're OK with API usage)

- Keep **Channel A** responses lean: set a **max output tokens** limit and avoid returning entire file bodies unless needed (we'll enforce schema size + chunking rules). OK
- We'll add simple **rate + cost guards** (e.g., backoff, token ceilings) you can tweak in Settings or `.env`.

  OK

Quick launch on a new machine (recap)

```
pwsh .\v2\scripts\setup.ps1
.\v2\scripts\pf.ps1 gui
# (Or direct) C:\venvs\promptforge-v2\Scripts\python.exe -m
promptforge_cli gui
```

If you paste the “Handoff to new thread” block into your new chat and answer the few <> fields, I’ll start Sprint V2.1 right away.

This is it.