# ChatGPT - Approaches-Reasoning-Journal

PROMPT:

|  |
| --- |
| // Context: I’m going to be working with VS Code and GitHub Copilot on creating a feature, from the idea and docs, specs, to the implementation in C#.  // Objective: I also want to record/save somewhere, in some kind of documentation, the whole process I’m doing, including what I think and I’m going to do and also every prompt I create and use with GHCP, so later I can showcase the process and reasoning I did. What would be the best way to save/write my whole process and reasoning including my own prompts?  // Requirements: Initially open to multiple approaches  // Format: Initially open to multiple approaches  // Constraints: I don't want to over-engineer, I need to be agile. |

Here are battle-tested ways to capture the full arc—from ideation → specs → code → Copilot prompts—without breaking your flow in VS Code.

**1) “Everything-in-repo” workflow (my top pick)**

Keep all artifacts versioned alongside the code so your process is reproducible and showcase-ready.

**Repo layout**

feature-foo/

├─ docs/

│ ├─ 00-vision.md

│ ├─ rfcs/001-feature-foo.md

│ └─ adr/ADR-0001-decisions.md

├─ journal/

│ ├─ 2025-09-12.md # running log for the day

│ └─ prompts/2025-09-12.md # raw Copilot prompts + responses

├─ tours/

│ └─ feature-foo.tour # CodeTour walkthrough

├─ notebooks/

│ └─ exploration.dib # .NET (Polyglot) Notebook: C# + prose

└─ src/…

**What goes where**

* docs/00-vision.md — problem statement, scope, success metrics.
* docs/rfcs/ — design/spec discussion (treat like a “lightweight PRD”).
* docs/adr/ — one ADR per irreversible decision (trade-offs + rationale).
* journal/YYYY-MM-DD.md — stream-of-consciousness engineering log (your reasoning, TODOs, dead ends, links to commits).
* journal/prompts/DATE.md — **every** Copilot prompt with context + outcome (see template below).
* tours/\*.tour — **CodeTour** files that replay the codebase/story as a guided walkthrough inside VS Code.
* notebooks/\*.dib — **.NET Interactive (Polyglot) Notebooks** mixing C# cells with markdown for “literate implementation” and runnable snippets.

**Copying Copilot Chat**  
In VS Code, open Copilot Chat → select messages → right-click → **Copy** (or “Open in Editor” if available) → paste into journal/prompts/DATE.md. Do this after each meaningful exchange (or at natural milestones).

**Prompt log template**

## [HH:MM] Goal

Implement X (link to RFC section/issue).

\*\*Prompt\*\*

> “Generate a C# interface IFoo with …”

\*\*Context Supplied\*\*

- Files: `src/Foo.cs`, `docs/rfcs/001-feature-foo.md#api`

- Selection: lines 20–60 of Foo.cs

\*\*Copilot Output (trimmed)\*\*

```csharp

public interface IFoo { … }

**Result**

* Accepted after edits (commit abc123).
* Follow-up prompt planned: validation rules for Bar.

**Notes**  
Trade-off: kept interface minimal to unblock API client.

\*\*Daily journal template\*\*

```md

# 2025-09-12 — Feature Foo Dev Log

## Intent

Ship CRUD for Foo with validation + telemetry.

## Plan

- [ ] Data model

- [ ] API surface (RFC §2)

- [ ] Unit tests

- [ ] Observability

## Notes & Reasoning

- Considered GUID vs ULID → chose GUID (ADR-0002).

- Perf risk in LINQ projection; benchmark later.

## Links

- RFC: docs/rfcs/001-feature-foo.md

- ADRs: docs/adr/ADR-0001-decisions.md

- PR: #42

- Commits: abc123, def456

**2) Use GitHub Issues/PRs as your living log**

If you want timestamps, permalinks, and team visibility:

* Open a single **tracking issue** (“Feature Foo Dev Log”).
* Post each Copilot prompt as a comment (paste prompt + key response + link to commit).
* Keep the **PR description** as an executive summary; link back to your RFC + ADRs.
* Pros: everything is time-stamped and reviewable; easy to share.  
  Cons: prompts may be noisy for watchers; still good to mirror into journal/prompts.

**3) “Literate implementation” with .NET Notebooks**

* Create a .dib notebook in VS Code (Polyglot Notebooks extension).
* Interleave markdown (reasoning, design) with **C#** cells (runnable prototypes/tests).
* Great for documenting tricky algorithms and perf experiments right next to code.

**4) Record the narrative with CodeTour**

* Install **CodeTour** (by Microsoft).
* Add a tour that jumps file-to-file explaining “why” decisions.
* Store \*.tour in tours/ so others can replay the story inside VS Code.

**5) Light automation that helps you keep notes**

**Quick-append task (macOS/Linux)**  
Add this VS Code task to send the clipboard (e.g., a Copilot chat chunk) into today’s prompt file.

// .vscode/tasks.json

{

"version": "2.0.0",

"tasks": [

{

"label": "Append clipboard to today's prompt log",

"type": "shell",

"command": "mkdir -p journal/prompts && echo \"\\n\\n## [$(date +'%H:%M')]\" >> journal/prompts/$(date +'%Y-%m-%d').md && pbpaste >> journal/prompts/$(date +'%Y-%m-%d').md"

}

]

}

Workflow: copy from Copilot Chat → run task (⌘⇧B) → logged.

**Commit message discipline**  
Use Conventional Commits and reference the journal:

feat(foo): add IFoo + DI registration (#42)

Docs: journal/2025-09-12.md – reasoning & trade-offs

**6) Keep it clean (privacy & IP)**

* **Never** paste secrets; add a pre-commit secret scan (gitleaks/trufflehog) and .gitignore any scratch files.
* If you include Copilot output, add a short header in prompt logs clarifying it may include AI-generated code.
* For public sharing, scrub internal URLs and credentials and run a quick grep for sensitive terms before pushing.

**7) Minimal starter kit (drop in)**

Create these files and you’re rolling:

* docs/00-vision.md (1-pager problem & success metrics)
* docs/rfcs/001-feature-foo.md (spec/design)
* docs/adr/ADR-0001-decisions.md (first decision)
* journal/2025-09-12.md and journal/prompts/2025-09-12.md
* .vscode/tasks.json (clipboard-to-log task)
* Optional: tours/feature-foo.tour, notebooks/exploration.dib