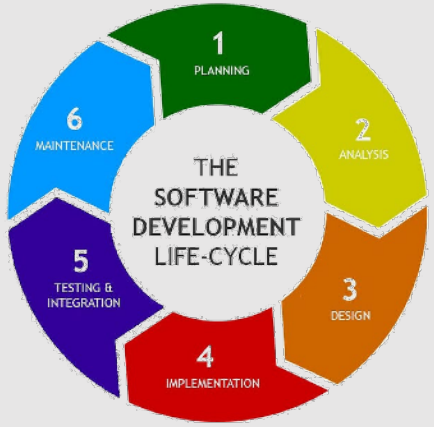
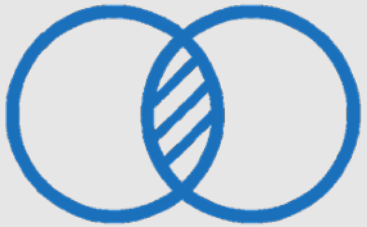
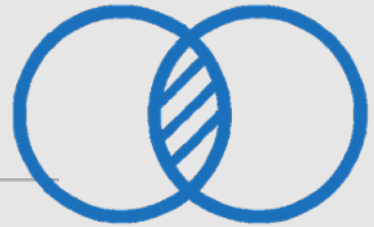
Incorporating AI into your SDLC

Leveraging AI tooling across the phases of your software development lifecycle







Presented by
Brent Laster
Tech Skills Transformations LLC

AI in Production

AI in Production

About me

- Founder, Tech Skills Transformations LLC
- https://getskillsnow.com
- info@getskillsnow.com

- Long career in corporate as dev, manager, and director in DevOps and other areas

- Author
  - O'Reilly "reports"
  - Books
    - Professional Git
    - Jenkins 2 – Up and Running
    - Learning GitHub Actions
    - AI-Enabled SDLC
    - Learning GitHub Copilot

- Speaker

- Social media
- ❑ **LinkedIn: brentlaster**
- ❑ **X: @BrentCLaster**
- ❑ **Bluesky: brentclaster.bsky.social**
- ❑ **GitHub: brentlaster**

LISTED BELOW ARE A FEW OF THE TECHNOLOGIES FOR WHICH WE OFFER HANDS-ON TRAINING. THESE CAN BE CUSTOMIZED FOR ANY SIZE TEAM FROM 1-100 AND FOR ANY LEVEL FROM BEGINNER TO ADVANCED.

- ArgoCD
- Containers
- Docker
- Gerrit
- Git
- GitHub Actions
- GitHub Codespaces
- GitHub Copilot
- GitHub Foundations
- GitHub Security
- GitLab
- GitOps
- Gradle
- Grafana
- Helm
- Jenkins
- Kubernetes
- Kustomize
- LLMs
- Prometheus
- Tekton
- VS Code

Background

Rise of AI Coding Assistants (Software Dev Asst's = SDA)

- LLMs and AI apps have exploded since OpenAI added chat to its models in November 2022
- Code creation and related are excellent fits for Gen AI due to structure, syntax, and wealth of examples
- At core, provide inline code suggestions as well as NL chat interfaces
- But also provide additional powerful functions
  - Repository indexing (RAG)
  - Summarizing (PRs, Issues, etc.)
  - Agent workflows (From big ideas to complete implementation)
- Users can leverage them in three modes:
  - Acceleration
  - Exploration
  - Directive

tabnine

Amazon Code Whisperer

GitHub Copilot

codeium

Ask Codi

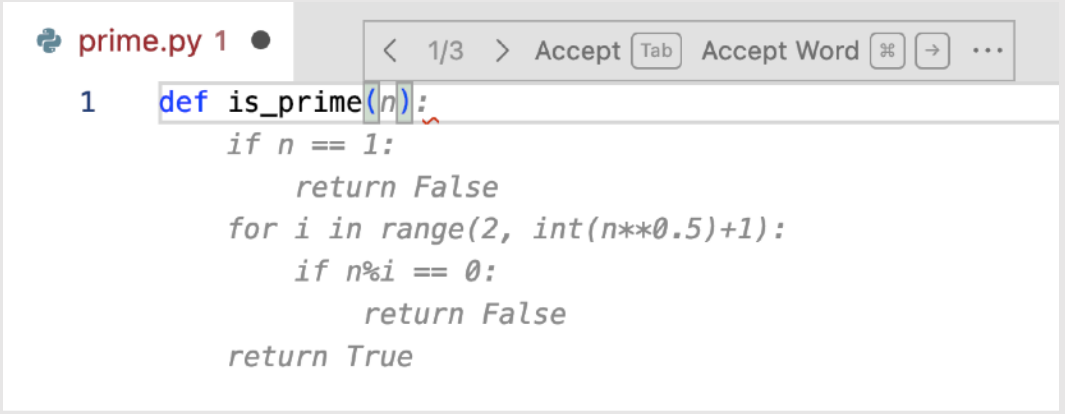
replit

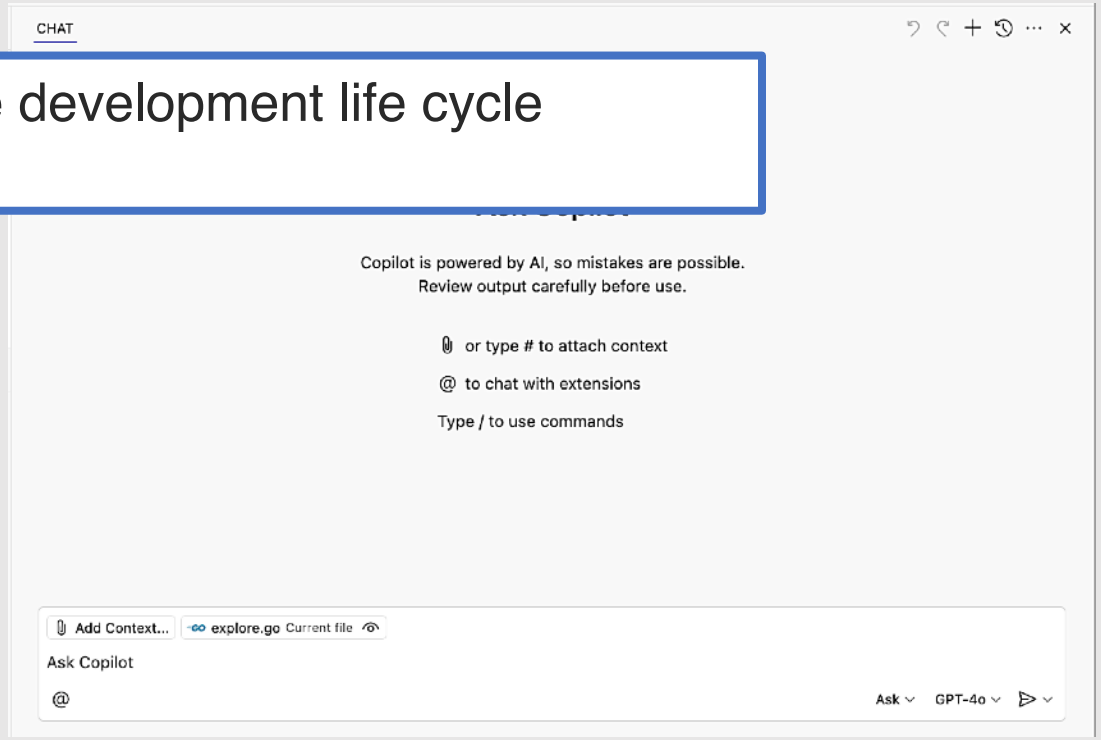
ChatGPT

Uses and Roles

- ***Accelerators/Acceleration*** - code completion - working in the flow and AI provides completion suggestions (usually in the editor)
- ***Explorers/Exploration*** - using the chat interface to ask how to code something? What is the API?
- ***Directors/Di*** specific instructions to create something via the chat interface or in the editor (via comments)
- All of these *can* lead to increased developer productivity

We can apply these across the software development life cycle (SDLC)!

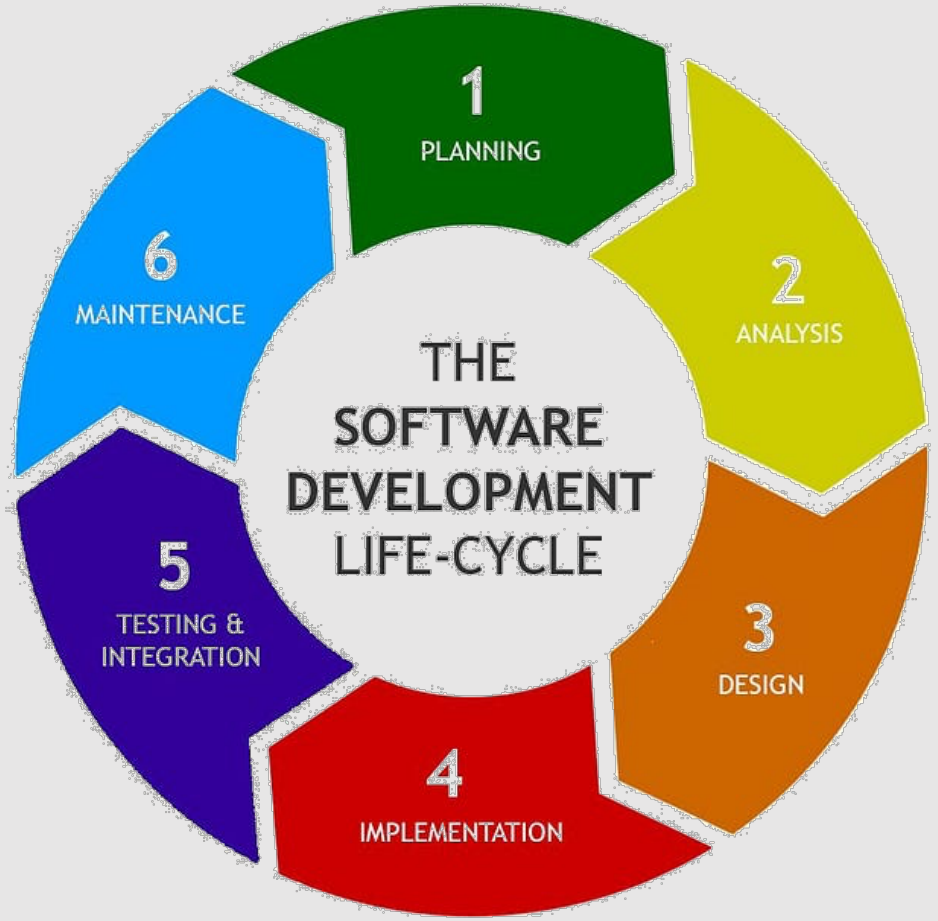




source: "https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/unleashing-developer-productivity-with-generative-ai

The Software Development Lifecycle (SDLC)

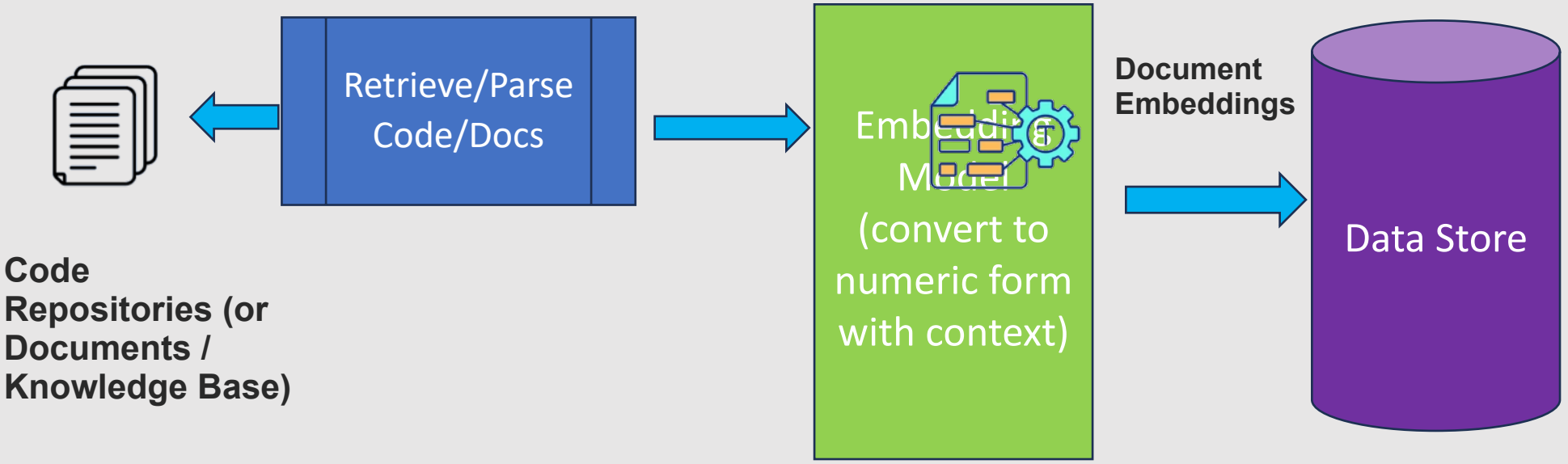
- Structured process used to create high-quality software efficiently and consistently
- Outlines and defines the steps to plan, develop, test, and deploy
- Typical phases include
  - Planning
  - Analysis
  - Design
  - Implementation
  - Testing and integration
  - Maintenance
- Defined phases may vary across organizations/projects



Planning

Background : RAG

- Goal: Search your local data (code, docs, etc.) to give additional context for LLM to work with and give you more targeted, useful responses (considers your data along with its training)

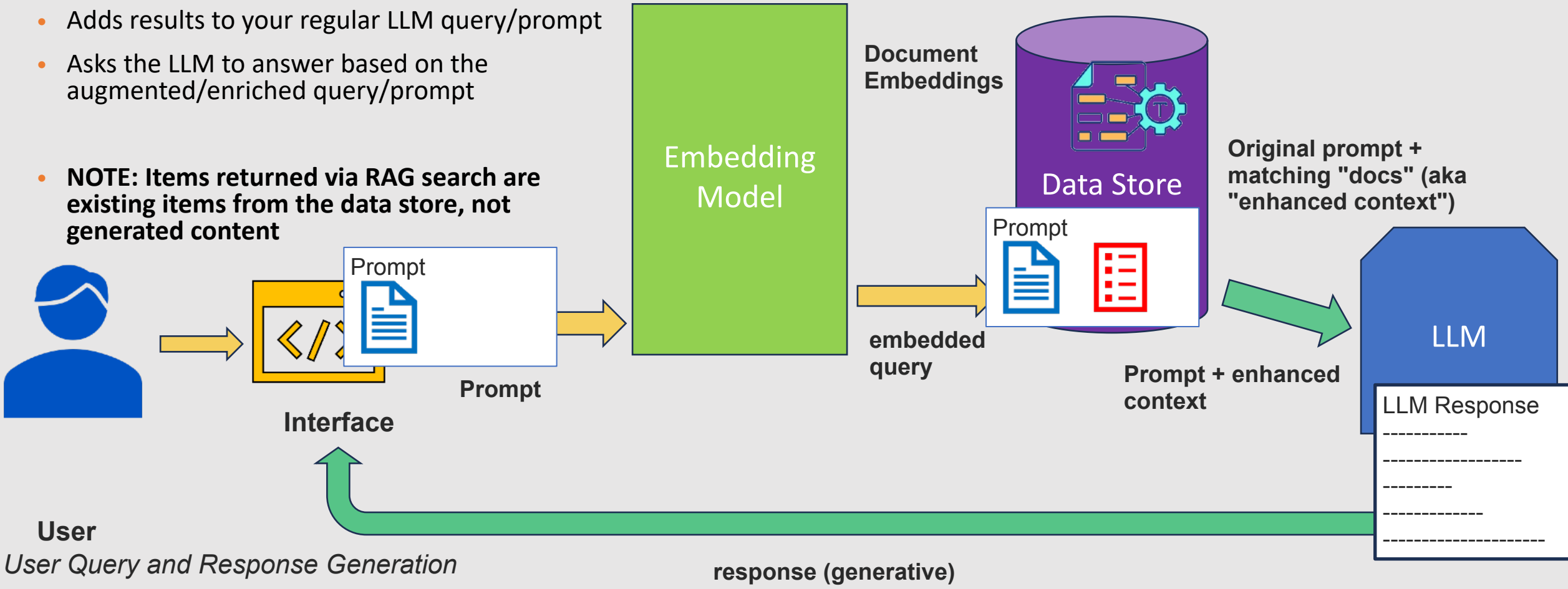


- Your data is parsed and stored with information about other data it's related to in a data store
  - parsing = tokens, storing as numeric data = embeddings, information about other data = vectors (set of numbers that represent how much a term is related to others in the data store)

Background: RAG

- For queries/prompts, application gathers results (most relevant ones) from the vector database with your data
- Adds results to your regular LLM query/prompt
- Asks the LLM to answer based on the augmented/enriched query/prompt

- **NOTE: Items returned via RAG search are existing items from the data store, not generated content**



Planning with AI (approaches)

- Index your code (targeted RAG) – usually automatic now
- Make code repositories accessible for AI (via RAG/indexes)
  - Easily find and identify reusable code
- Ask questions of your code to simplify estimates, avoid rework, ground knowledge

- How did we implement function X?
- Does our code use dependency Y that has this security vulnerability? If so, where?
- We want to migrate from one platform/cloud/language / framework to another - where do we need to focus our efforts?
- The open-source library we use has changed licensing terms - how are we using that?
- Do we have code that does Z?

Planning: Prototyping

- For brainstorming/considering requirements
- Can quickly draft pseudo code in comments
- AI can generate "draft" code
- Doesn't have to be perfect, just enough for estimation/planning

```
# Given a list of app features and their dependencies as (feature, depends_on) pairs
# Determine a valid order in which the user can unlock and access features
# Raise an error if a cycle exists (invalid dependency loop)
# Example input:
#     features = [
#         ("upload", "signup"),
#         ("share", "upload"),
#         ("analytics", "share"),
#         ("signup", None)
#     ]
# Expected output:
#     ["signup", "upload", "share", "analytics"]

# Define a function to accept the feature dependency list
# Build a directed graph using adjacency lists
# Use topological sort to determine feature unlock order
# Validate and raise an error if dependencies form a cycle
# Return an ordered list of unlockable features
```

```
from collections import defaultdict, deque

def determine_feature_unlock_order(dependencies):
    # Build graph and in-degree count
    graph = defaultdict(list)
    in_degree = defaultdict(int)
    features = set()

    for feature, depends_on in dependencies:
        features.add(feature)
        if depends_on:
            graph[depends_on].append(feature)
            in_degree[feature] += 1
            features.add(depends_on)
        else:
            in_degree[feature] += 0  # Ensure entry in in-degree map

    # Start with features that have no dependencies
    queue = deque([f for f in features if in_degree[f] == 0])
    unlock_order = []

    while queue:
        current = queue.popleft()
        unlock_order.append(current)

        for dependent in graph[current]:
            in_degree[dependent] -= 1
            if in_degree[dependent] == 0:
                queue.append(dependent)

    if len(unlock_order) != len(features):
        raise ValueError("Circular feature dependency detected")

    return unlock_order


# Example usage
if __name__ == "__main__":
    features = [
        ("upload", "signup"),
        ("share", "upload"),
        ("analytics", "share"),
        ("signup", None)
    ]
    order = determine_feature_unlock_order(features)
    print("Feature unlock order:", order)
```


Development

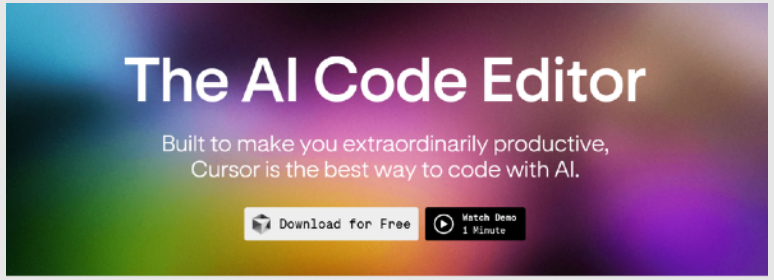
Types of AI Assistance for Coding

- Extensions that add AI code completions and chat

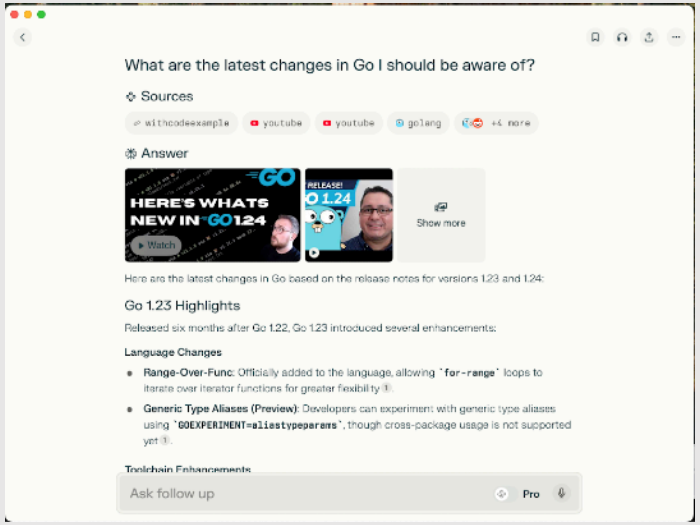
GitHub Copilot

tabnine

- AI Coding IDEs
  - Based on VS Code
  - AI-first design
  - Tend to focus more fully on project-wide insights and application building
  - Think “collaborate like Copilot and handle complex tasks like an agent”



- Research-focused AI assistants
  - Offer answers based in current content
  - Synthesize info from multiple sources (cited) into concise, natural language answers
  - Uses LLM on back end
  - Great at real-time info retrieval
  - Can also do most things that ChatGPT can do (but with up-to-date info and sources)



How coding assistants can help and what to remember

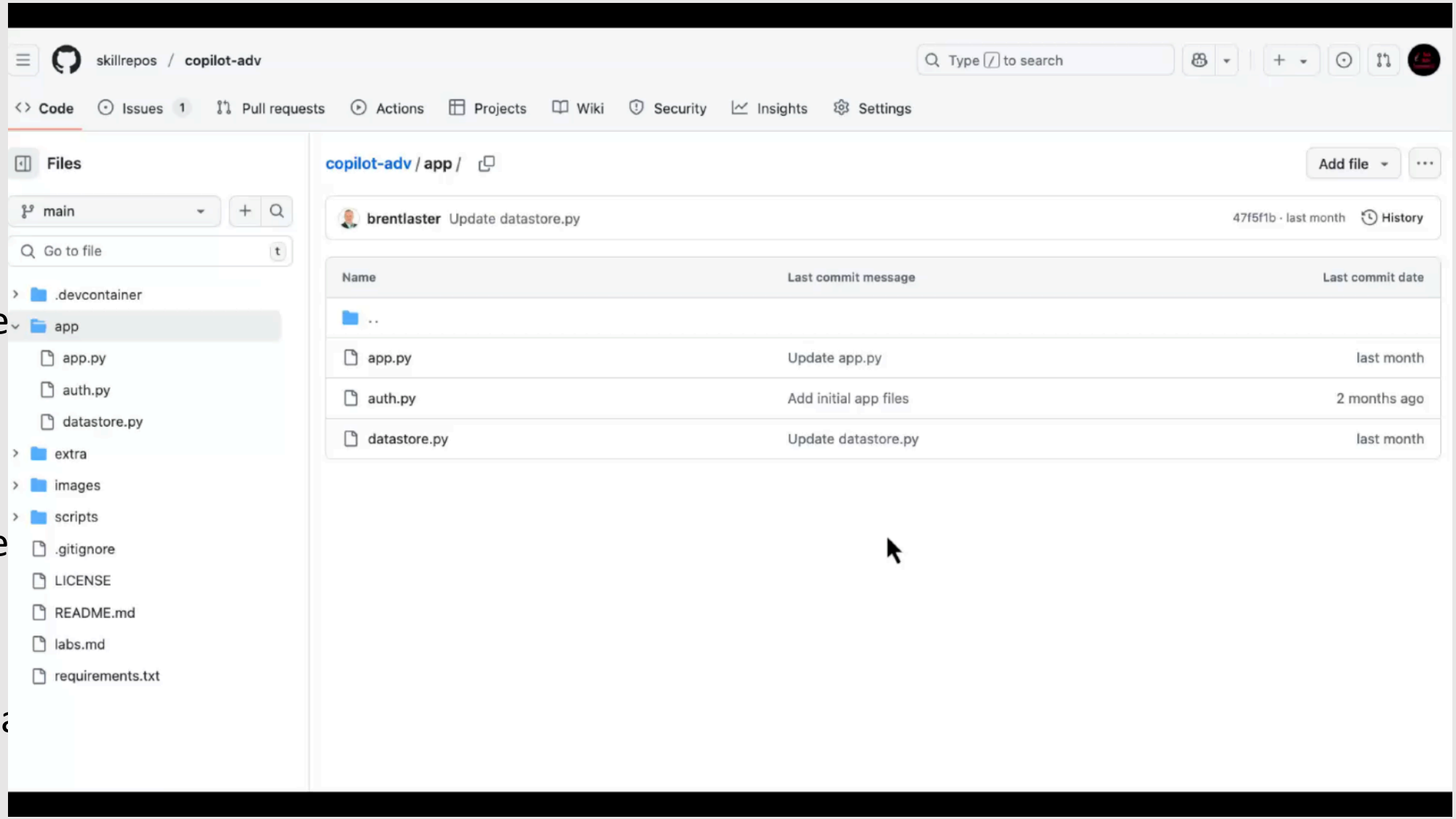
- Code suggestions for nearly any current language
- Comment driven code creation
- Automatic test generation
- SQL generation (for queries, tables, indices, stored procs)
- Regular expression generation
- Pattern/mappings generation
- Document code
- Explain code
- Fixing code
- Reviewing code
- Translating code
- Generate specs/configs
- What is the API to do...

- Reduce cognitive load
- Let programmers focus on the parts that need their expertise

- AI needs good context
- AI needs good guidance or queries (prompts)
- AI can hallucinate
- AI can be out-of-date
- AI doesn't do real-time validation of generated data
- AI doesn't care about tech debt
- BOTTOM LINE: AI-generated code intended for production must ALWAYS be subject to same oversight and review

Leveraging Agents

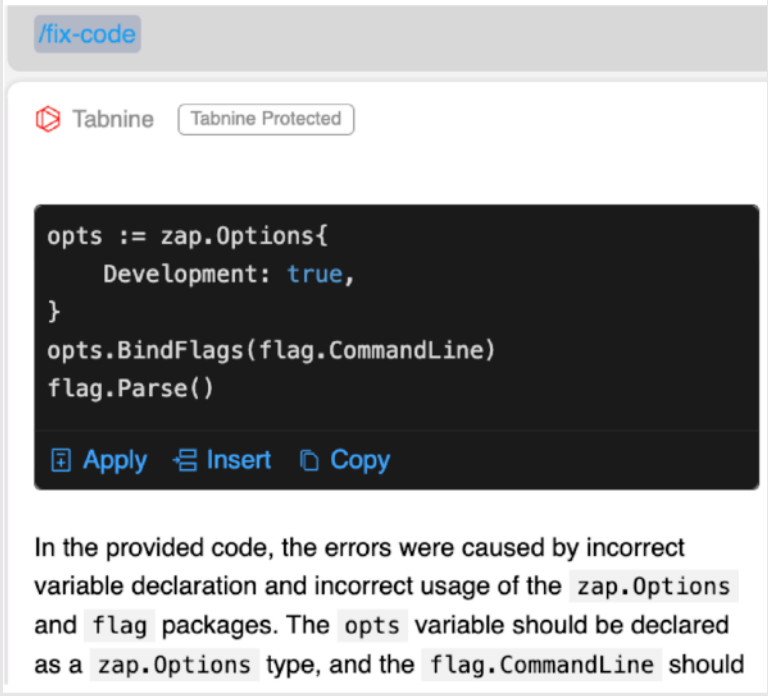
- Most AI assistants have agentic functions built in
- Agent = AI w/ agency
  - ability to gathe input
  - act
  - make changes
- End-to-end change
- Autonomous with oversight
- Goes beyond typica AI assist functions

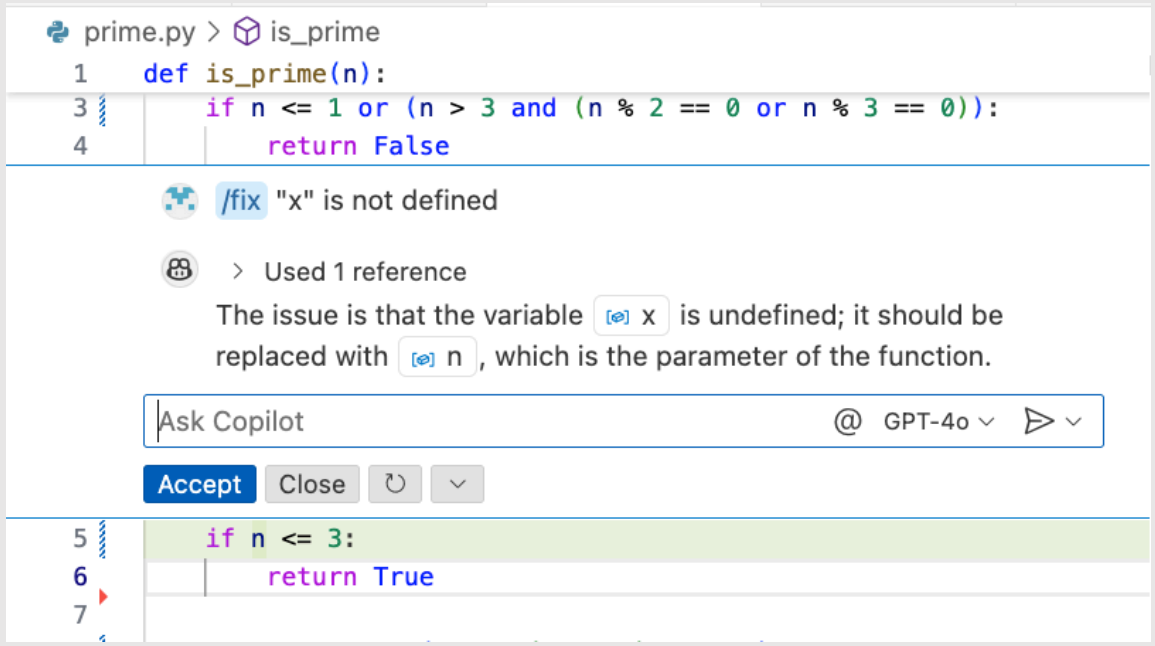


Resolving Issues

Resolving issues with AI

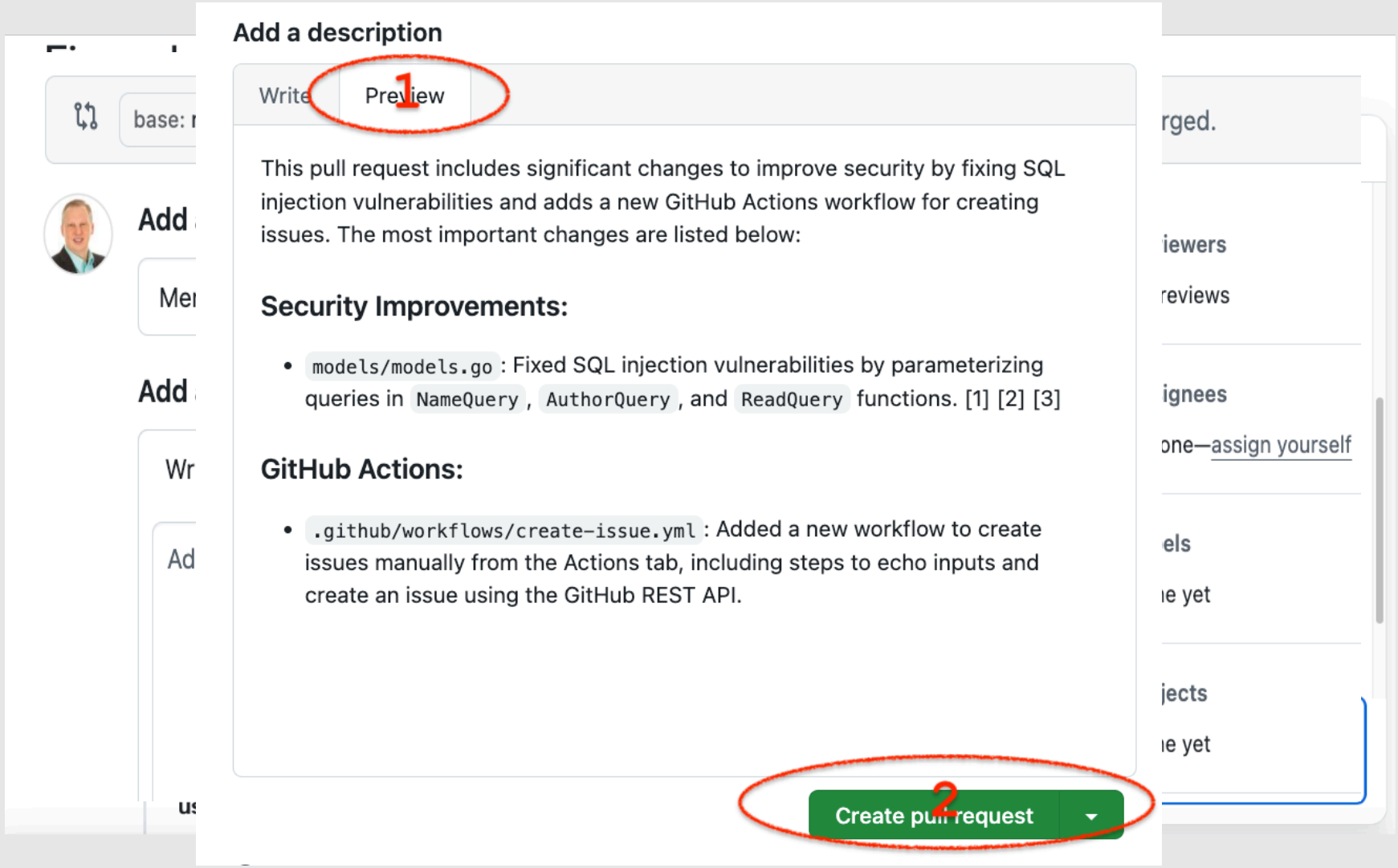
- Shortcut fix commands available
- Prompts to help find and fix issues (or potential issues)
  - What are some places in this repository where logic/performance/security errors can occur?
  - What are the most recent significant changes to the codebase?
  - What are the code changes that affected this code in branch x from date y to date z?
  - How can I make this code be more efficient with system resources?
  - Where else might this problem occur?
  - What other parts of the codebase use this routine?





Working with tracking mechanisms (Copilot examples)

- Given an issue, can prompt Copilot for next steps
- Prompt to provide summary of issue
  - Includes overview of issue based on what Copilot can gather from the system using tools
- Generate pull request summaries



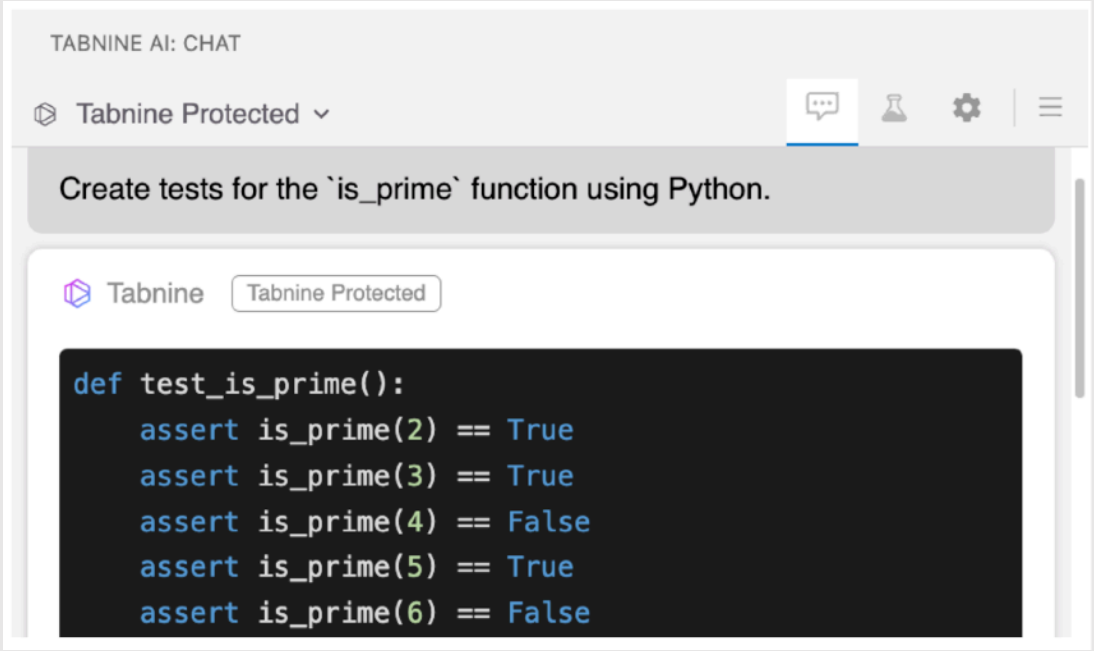
AI review – Having Copilot review code in VS Code

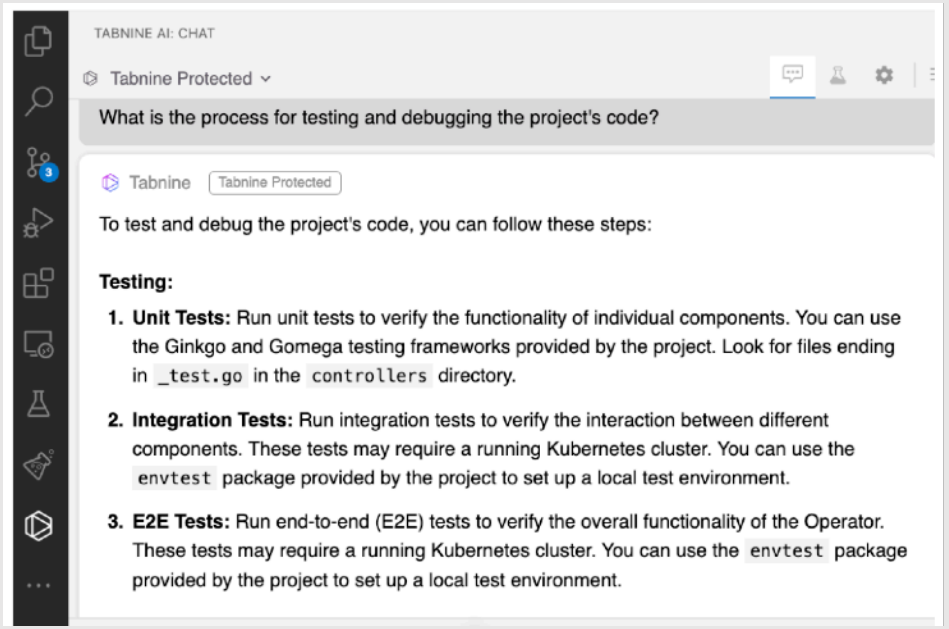
- AI-powered feedback on code as you write it
  - Review selection:
    - » select code in VS Code and ask for initial review
    - » Available now for all Copilot subscribers
- Review changes: ask Copilot for deeper review of all changes before pushing from “Source Control” tab
  - » Can also be done in pull requests on GitHub.com
  - » Currently must be on waitlist

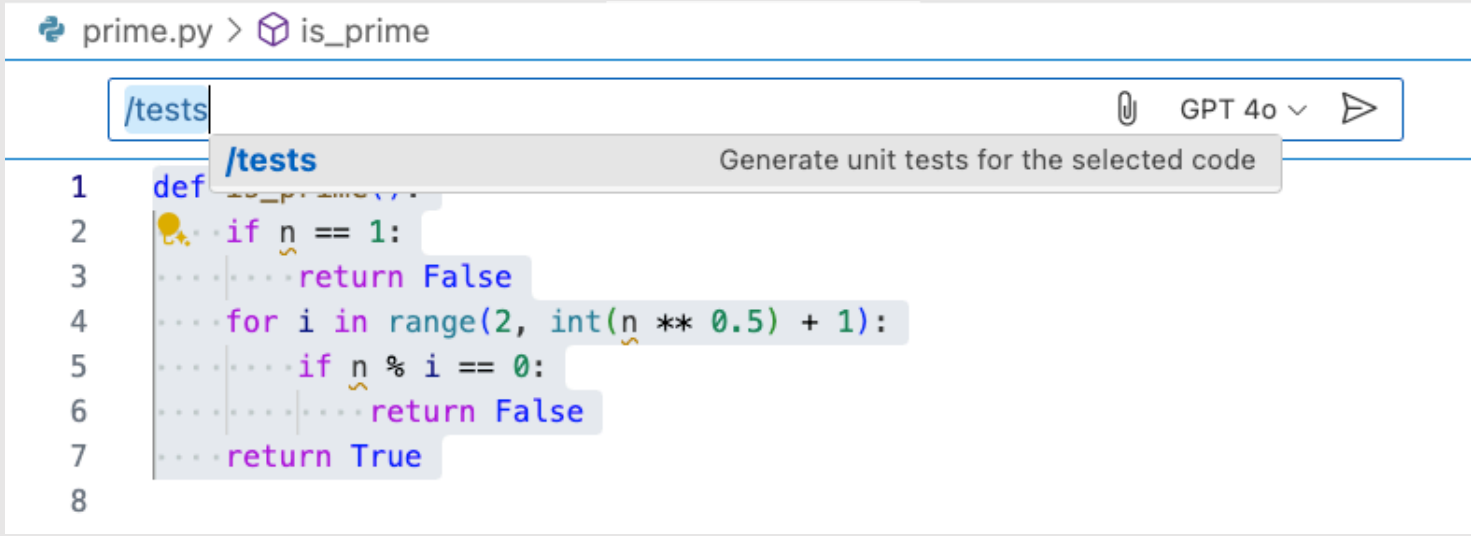
Testing

Testing with AI

- Create tests for …
- What is the process for testing and debugging the project's code?
- Generate unit tests / integration tests for …

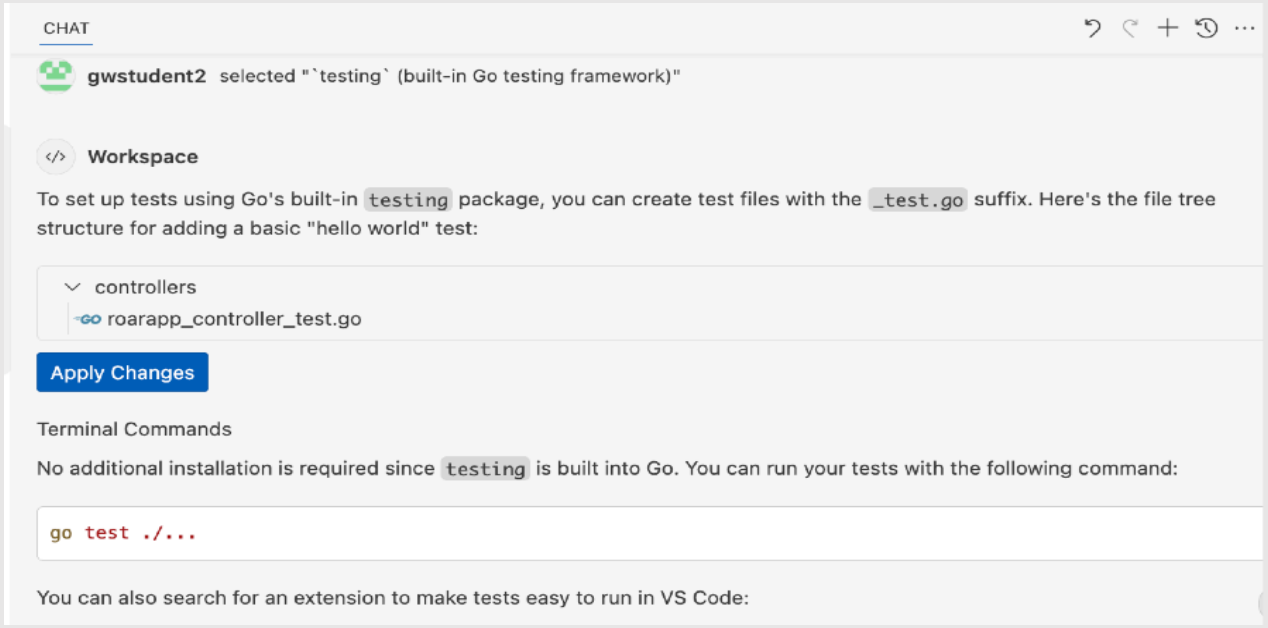


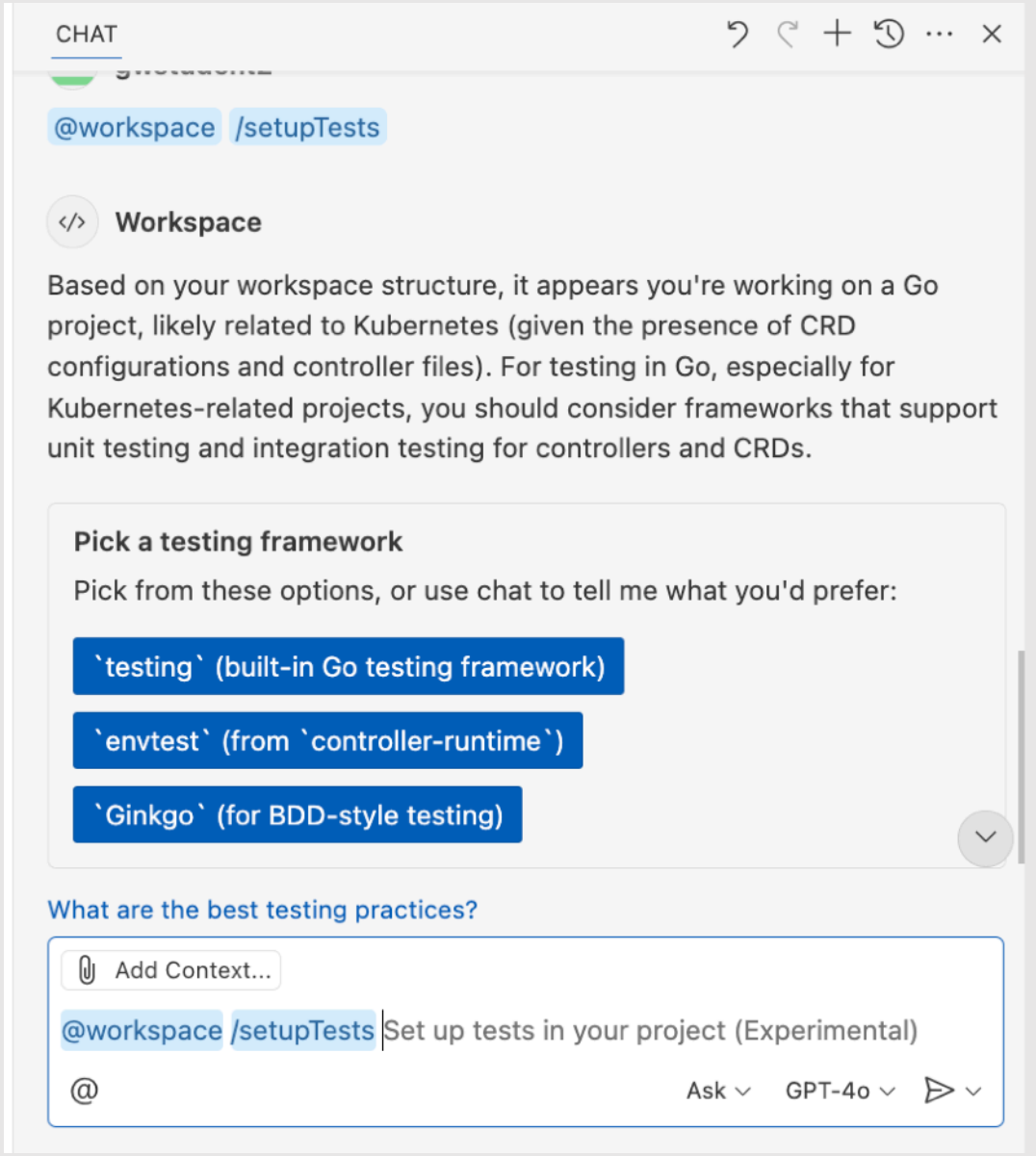




AI Testing – from the start

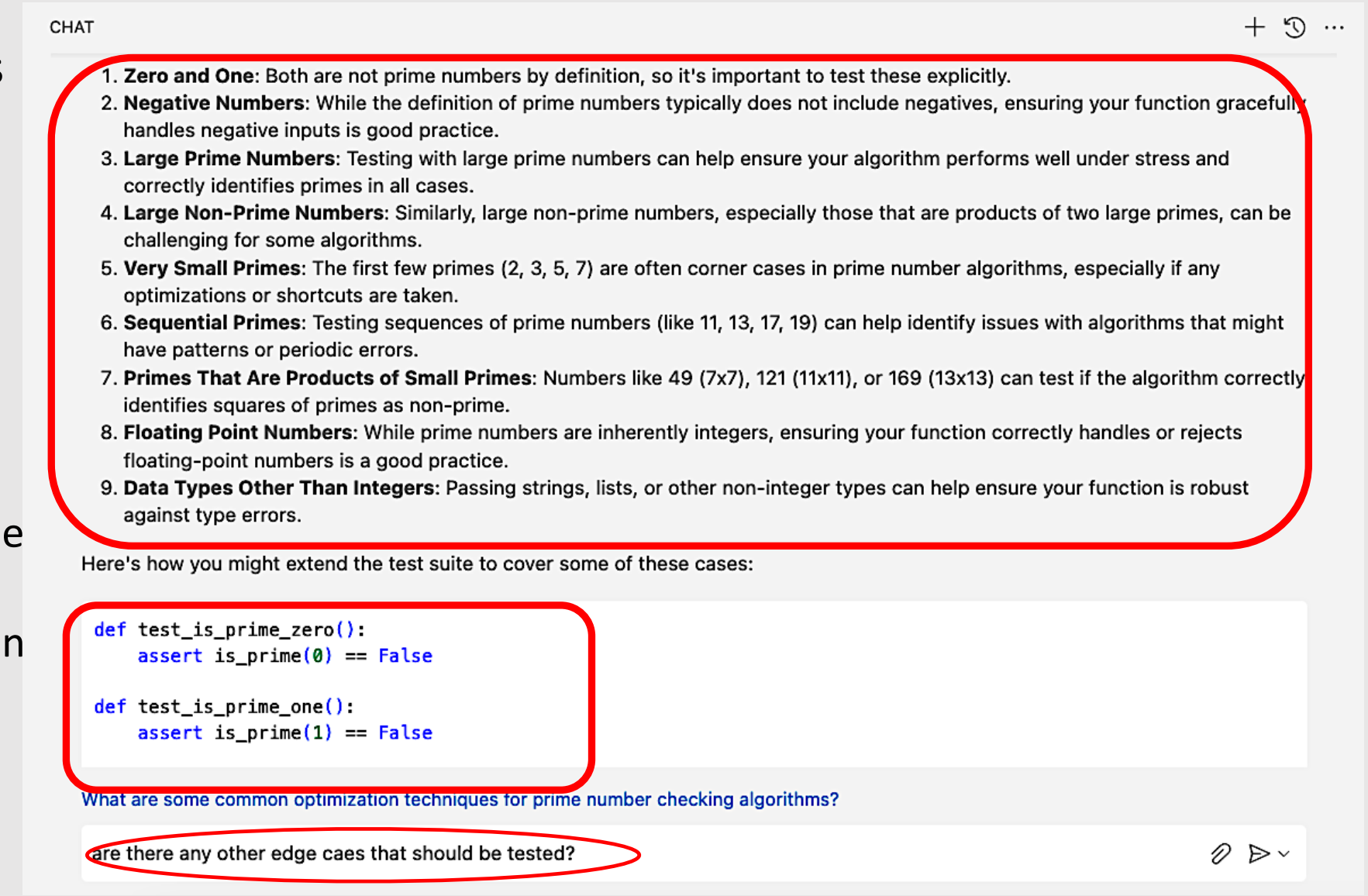
- Can help set up a testing framework for your code
- Recommend appropriate frameworks
- Provide step-by-step setup and configuration instructions
- Suggest relevant testing extensions for your dev env
- Can have custom test generation instructions





Testing – beyond the basics

- What other edge cases should I be testing?
- How do I test X for performance?
- How do I test Y for security?
- Test Driven Development
  - Create dependency file
  - Create test class (fail)
  - Create implementation class (succeed)



Creating Tests Via Prompts

- **Prompt for Unit Tests:**
  - "*Develop a comprehensive suite of unit tests for the <method/function> in <language/framework>. Write multiple test methods that cover a wide range of scenarios, including edge cases, exception handling, and data validation.*"
- **Prompt for Integration Tests:**
  - "*Write integration tests for the <method/function> in the <class>. Use mocks to simulate the <external service> and verify that it works correctly when <condition>.*"

Source: https://docs.github.com/en/copilot/using-github-copilot/example-use-cases/writing-tests-with-github-copilot#writing-unit-tests-with-copilot-chat

Refactoring/Updating

Refactoring with AI

- Consider a code example

```
def is_prime(n):
    return n > 1 and all(n % i for i in range(2, int(n**0.5) + 1))
```

- Can refactor via prompt

```
The is_prime function should have more meaningful variable names
 (max of 3 characters) and comments for better readability.
```

- Resulting code

```
# Check if the input is a positive integer greater than 1
    if not isinstance(num, int) or num <= 1:
        raise ValueError("Input must be a positive integer greater
than 1")

    # Check if the number is greater than 1 and not divisible by
any number up to its square root
    if num > 1 and all(num % div for div in range(2, int(num**0.5)
+ 1)):
        return True
    else:
        return False
```

Refactoring for better testability

- Can also ask Copilot to refactor code to be more suitable for testing
- For example: " Refactor the code in #file:<filename> to make it more easily testable"

```python
def is_prime(n):
    if not isinstance(n, int):
        raise ValueError("Input must be an integer")
    if n < 0:
        raise ValueError("Input must be a non-negative integer")

    if n < 2:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

GitHub Copilot

Used 1 reference

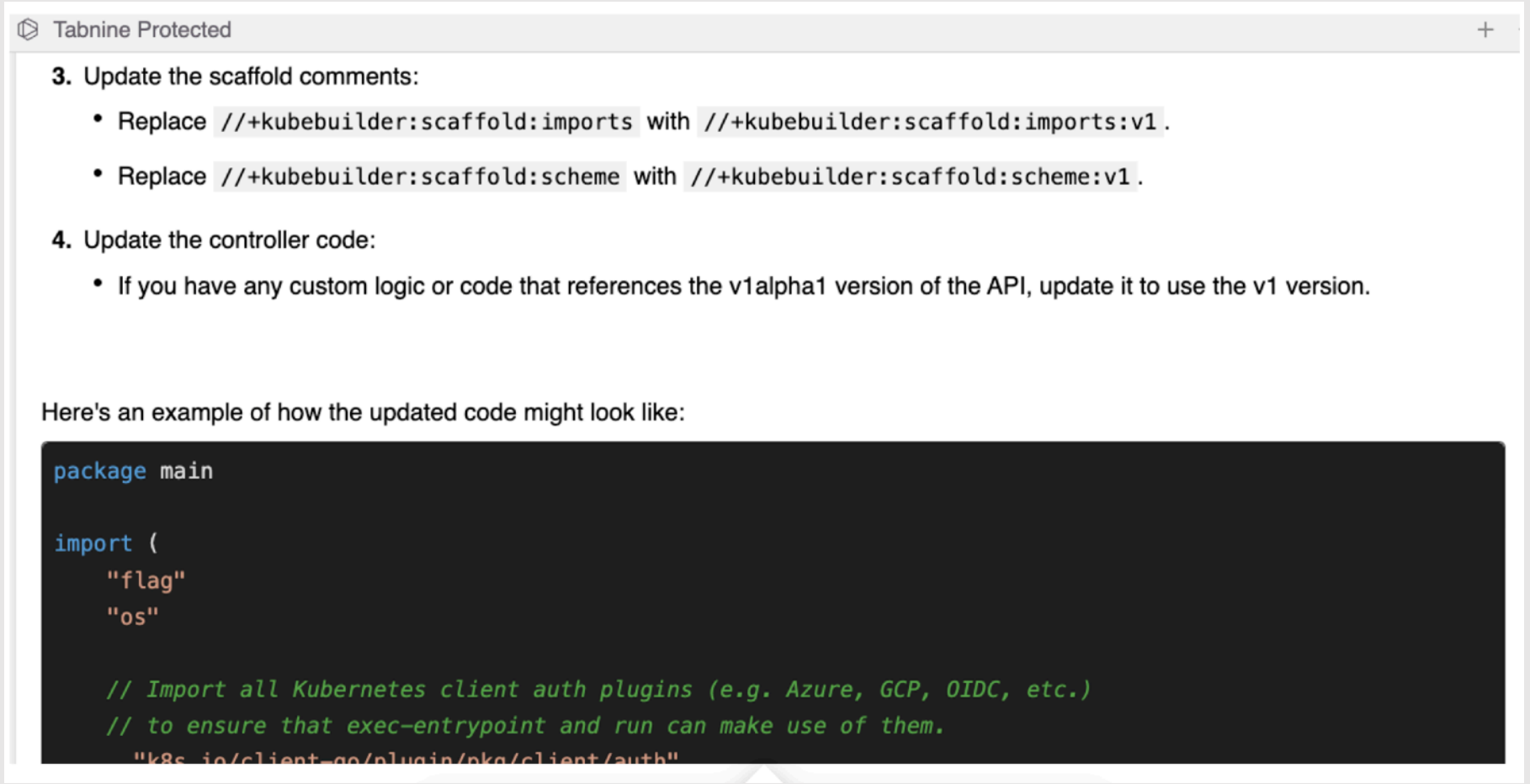
To make the code more testable, we can refactor it by separating the validation logic from the core functionality. This way, we can test the validation and the prime-checking logic independently. Here is the refactored code:

```python
def validate_input(n):
    if not isinstance(n, int):
        raise TypeError("Input must be an integer")
    if n <= 1:
        raise ValueError("Input must be greater than 1")

def is_prime(n):
    validate_input(n)

    if n == 1:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True
```

Now, we can write separate unit tests for `validate_input` and `is_prime` functions. This separation of concerns makes the code more modular and easier to test.

Exploratory Refactoring and Updating with AI

- Can probe with questions to decide what needs to be updated?
  - Where do we use API v1alpha1?
  - What are the changes between v1alpha1 and v1 API versions?
- Can prompt to do updates such as “Change v1alpha1 API to v1”
- AI-recommended change areas
  - import paths
  - references to API versions
  - scaffold comments
  - Controller code



Augmenting Code with AI

- Augmenting = adding additional functionaly to existing code
- Prompt “The is_prime function should log whether the in put is prime or not”
- AI suggested additional code that
  - pulls in a logging framework, initializes a logger instance
  - adds logging calls for each possible output

`logging` module in Python. Here's an example of how you can do it:

```
import logging

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def is_prime(n):
    if not isinstance(n, int) or n <= 1:
        raise ValueError("Input must be a positive integer greater than 1")
    if n > 1 and all(n % i for i in range(2, int(n**0.5) + 1)):
        logger.info(f"{n} is a prime number")
        return True
    else:
        logger.info(f"{n} is not a prime number")
        return False
```

Insert Copy New file

In this updated version, the `logging.basicConfig(level=logging.INFO)` line sets up the

Changing Implementations with AI

- Can select code and prompt for alternative implementations
- Consider example code
- Prompt “Explain this function in detail and suggest alternative implementations”
- Also - porting
- Prompt “convert to Go”

```
package main

import "fmt"

func fibonacci(n int) []int {
    sequence := []int{0, 1}
    for sequence[len(sequence)-1] + sequence[len(sequence)-2] <= n {
        sequence = append(sequence, sequence[len(sequence)-1] +
sequence[len(sequence)-2])
    }
    return sequence
}

func main() {
    n := 100
    fmt.Println(fibonacci(n))
}
```

Documentation

Using AI for general documentation via comments

- Basic methods may just do “header” comments
- By being more specific in prompt, can get more extensive comments

```
1   def is_prime(n):
    /doc
    /doc                                  Add document
    Gene
2       """
3       Check if a number is prime.
4
5       A prime number is a natural number
6       This function uses a trial divisio
7
8       Parameters:
9           n (int): The number to check f
10
11      Returns:
12          bool: True if the number is pr
13
14      Examples:
15          >>> is_prime(2)
16          True
17          >>> is_prime(4)
18          False
19          >>> is_prime(17)
20          True
21      """
22      """Check if a number is prime."""
23      if n <= 1:
24          return False
```

```
# Initialize the variable i to 5
# We will use this variable to check for factors of n
i = 5

# Loop to check for factors of n
# The loop continues until i squared is greater than n
while i * i <= n:
    # Check if n is divisible by i or i + 2
    # If it is, then it is not a prime number
    if n % i == 0 or n % (i + 2) == 0:
```

Can you explain the logic used?

Verbosely comment this code so that anyone can easily understand what each line is doing

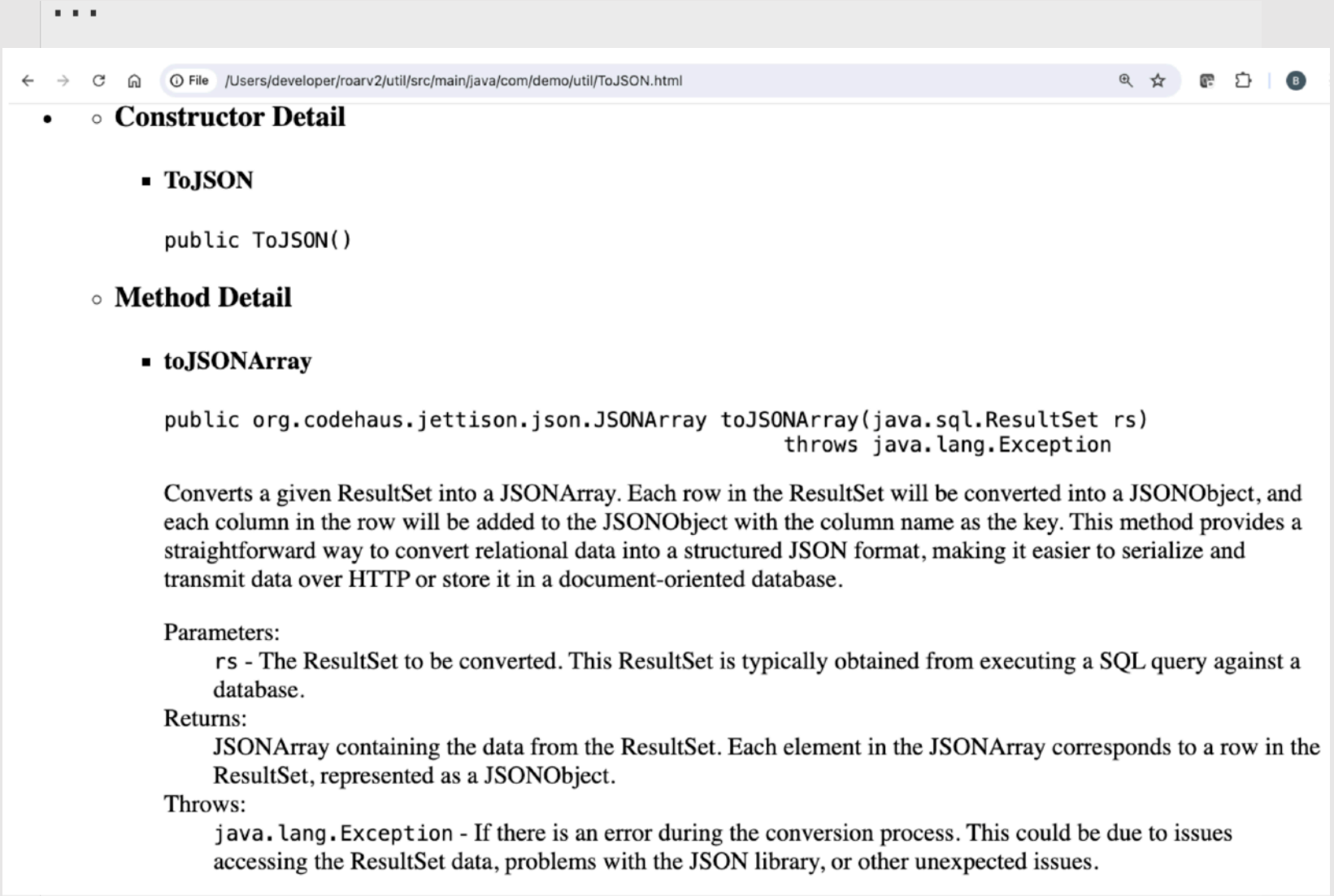
prime.py Current file

@

GPT 4o

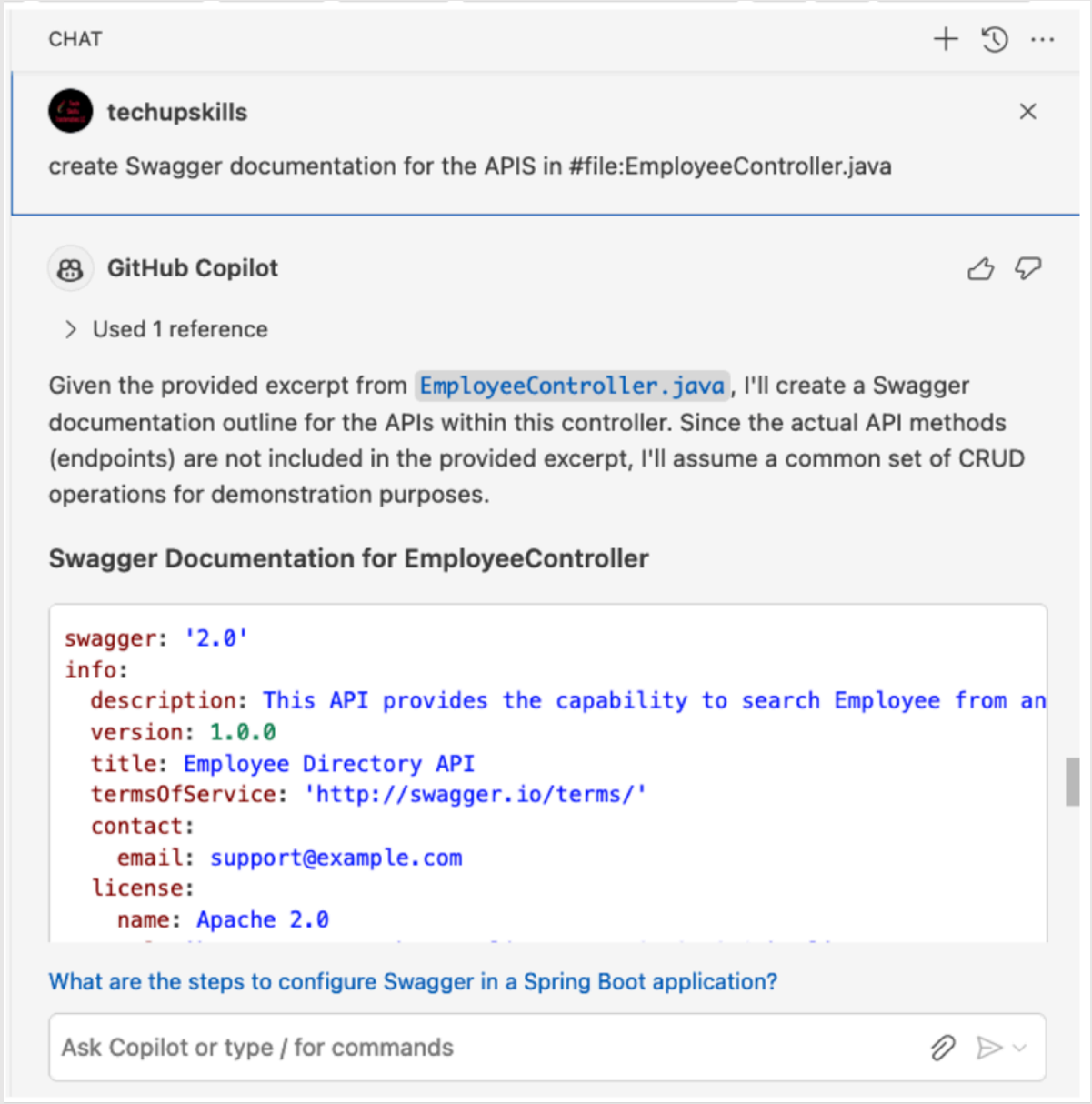
Generating Framework-compatible documentation

- If the SDA recognizes that the code is in a language that has a standard documentation tool associated with it, it will default to creating comments ready for that tool
- Given example code…
- Have Copilot document it
- Can then generate framework doc from it



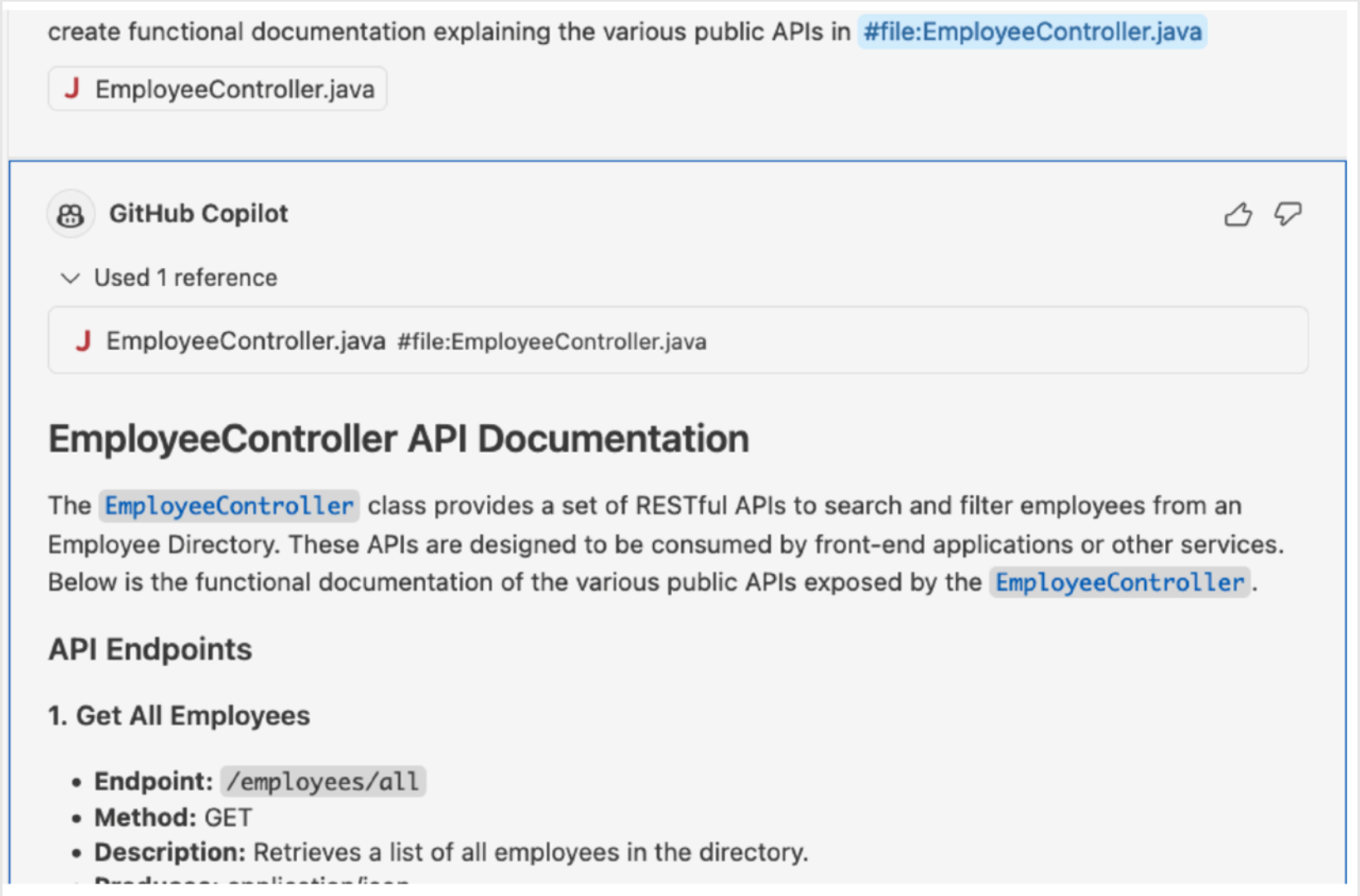
Generating Documentation for APIs

- If your codebase is setup for an API documentation framework like Swagger, then you can have the SDA generate the corresponding Swagger documentation for your APIs



Generating Functional Documentation

- SDAs are also capable of creating functional documentation that is targeted towards external users
- May also produce a set of related info including *Identifying Public APIs*, *Method Signatures*, *Endpoint Mapping*, etc.



Extracting Summary Documentation

- Copilot can also be used to extract a high-level summary of the key documentation from certain types of projects
- This can be extremely useful for quickly understanding the significant details of the code base for a project

The key documentation extracted from the provided code and workspace structure highlights the setup and configuration of a Spring Boot application with Swagger for API documentation. Here's a summary:

Project Structure and Configuration

- **Project Base**: Spring Boot application designed for demonstrating Swagger integration.
- **Group ID**: `com.techupskills.springboot.swagger`
- **Artifact ID**: `springboot-swagger`
- **Version**: `0.0.1-SNAPSHOT`
- **Java Version**: 17
- **Swagger Version**: 3.0.0
- **Packaging**: JAR
- **Parent**: Spring Boot Starter Parent with version 3.1.0
- **Source Encoding**: UTF-8

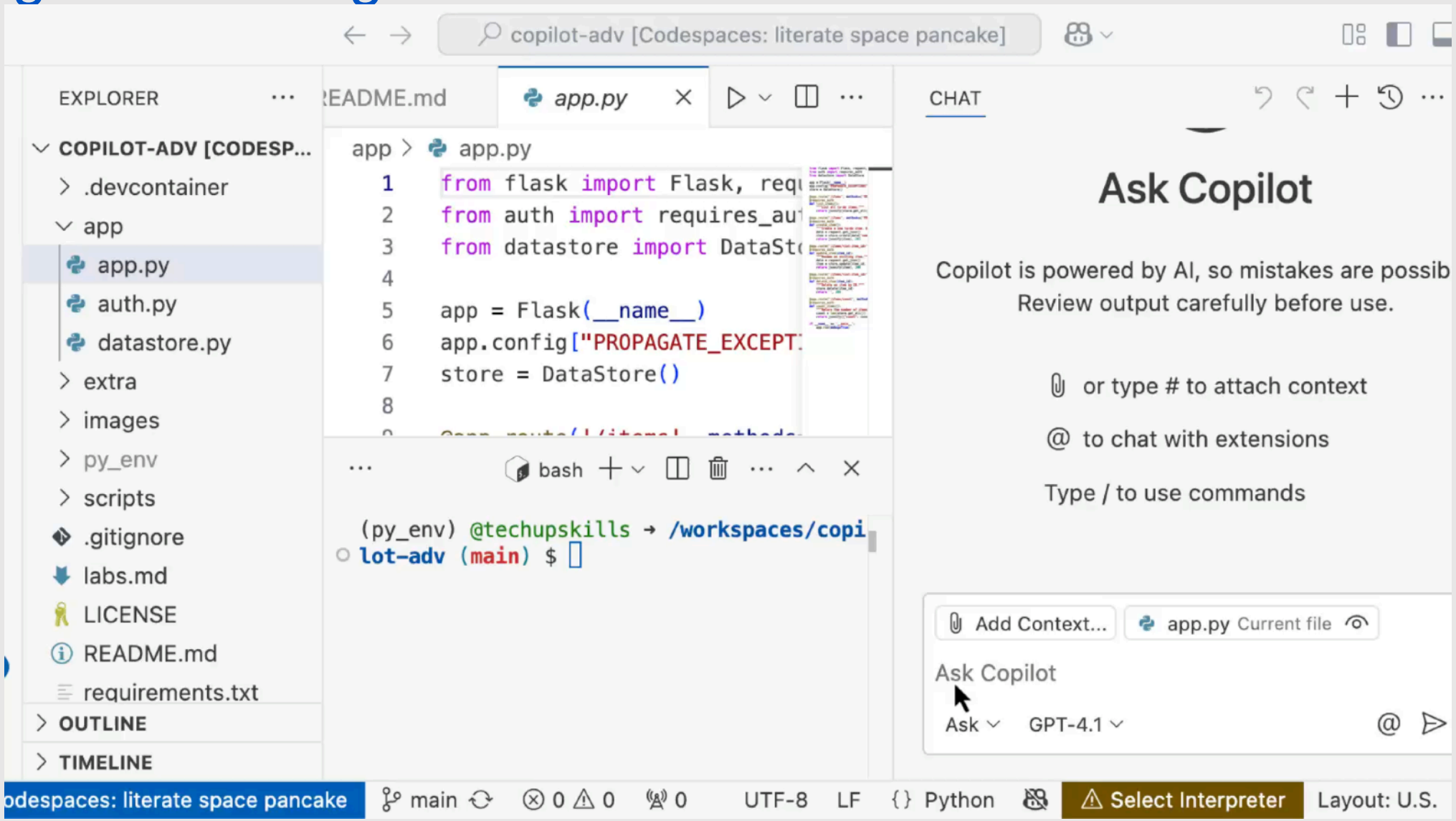
Dependencies

- **Spring Boot Starter Web**: For building web, including RESTful, applications using Spring MVC.
- **Spring Boot Starter Test**: For testing Spring Boot applications with libraries including JUnit, Hamcrest, and Mockito.
- **Spring Boot Starter Data JPA**: For using Spring Data JPA with Hibernate.

Simplifying Maintenance w/ Explaining/Onboarding

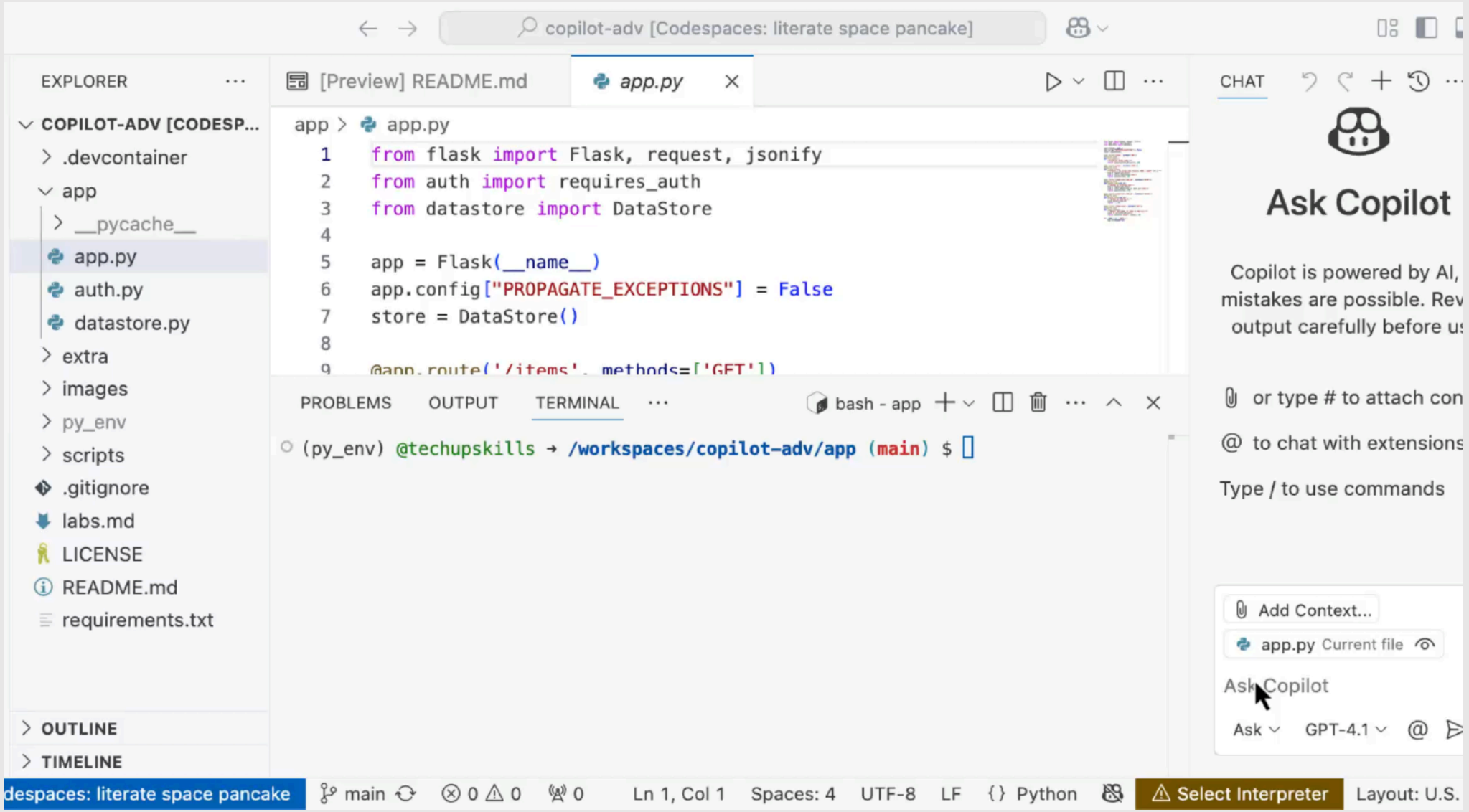
Simplifying Onboarding

- Onboarding is a key aspect of helping devs get going with code
- Simple to have AI generate an onboarding guide



Explaining Code

- Key for taking over/assisting with code bases
- Useful for learning new frameworks/ languages
- Using built-in “explain” functionality
- Can also have it explain how to see/run functionality



That's all - thanks!

Contact: training@getskillsnow.com

techskillstransformations.com
getskillsnow.com