Professional GitHub Copilot

advanced topics and techniques for developers

Andrew Scoppa
GitHub Expert Services

Session Overview

Who:
- Experienced GitHub Copilot users

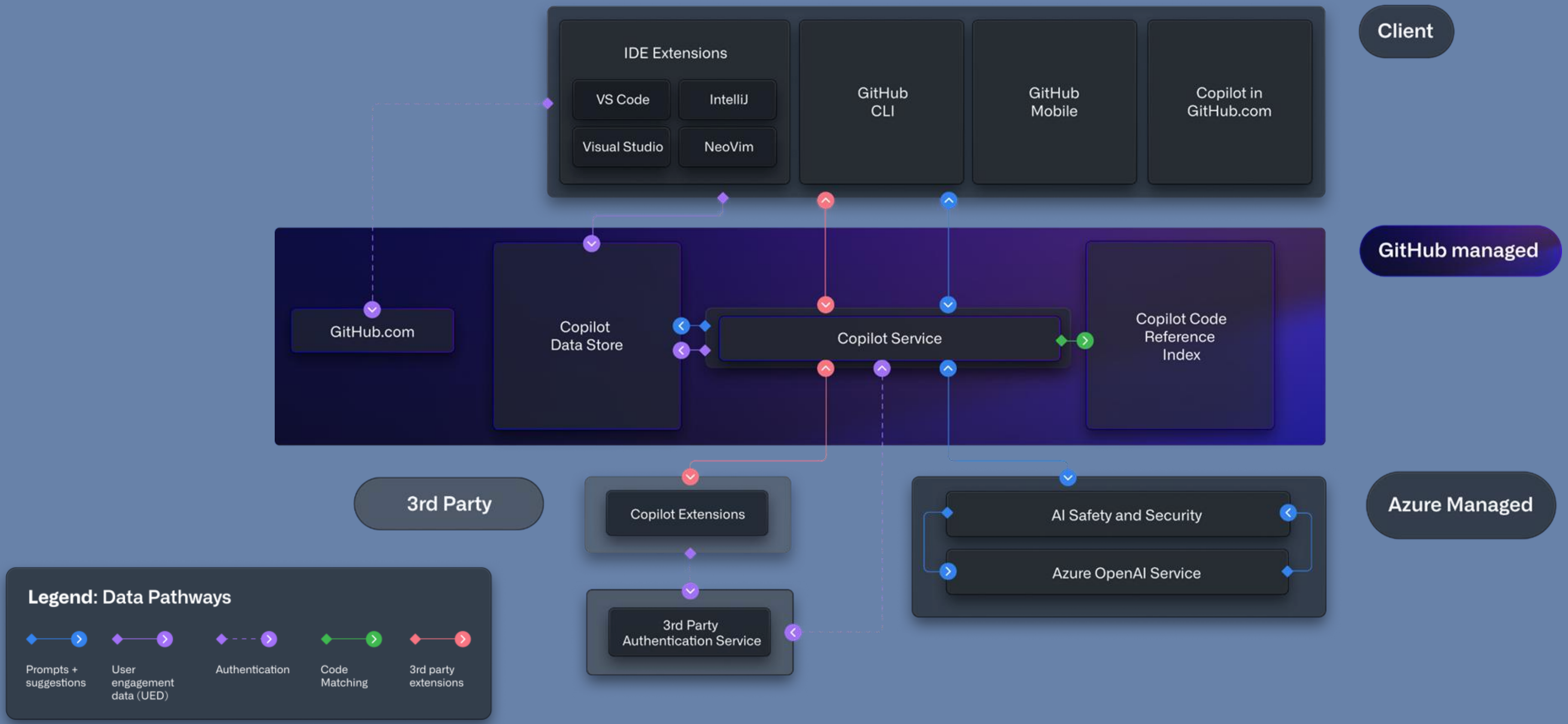
What:
- 3-hour session (10-minute break midway)
- Focus on specialized prompts using real-world use cases, and Copilot’s impact on code quality, security, and productivity

Why:
- Enhance your SDLC
- Identify workflow improvements with Copilot
- Unlock Copilot features for reviews, PRs, testing, docs, debugging, and feature development

Agenda

- Github Copilot Architecture
- The role of an LLM
- Using different LLM models
- Prompt engineering strategies
- Chat participants, slash commands, chat variables
- Copilot Customizations
- GitHub Copilot code review
- Copilot for pull requests
- GitHub Copilot Security

GitHub Copilot Architecture “Big Picture”



The role of an LLM

“ A Large Language Model (LLM) plays the role of understanding and generating human-like text based on the input it receives. Combines training, architecture, and context to produce relevant, high-quality text across many applications.

- Takes your input, breaks it into pieces (tokens), and predicts the next piece over and over to build a reply.

- Learned patterns from lots of text; it remembers **patterns**, not real-time or guaranteed facts.

- You guide it with **clear prompts** and simple settings to be more direct or creative.

- Good at summarizing, drafting answers, explaining code, and filling in patterns.

- Limits: can sound confident but be wrong or outdated; not a database or source of truth.

Using Different LLM Models with Copilot

- Several models to choose from including

  GPT 5 - Next-generation successor to GPT-4o, geared toward deeper multi-step reasoning, tool orchestration, and extended trusted context windows.

  Claude 3.5 Sonnet - Balanced model delivering strong reasoning, long-context handling, and fast iterative responses at moderate cost.

  o3-mini (OpenAI) - Lightweight reasoning-optimized model variant focused on fast iterative problem solving (code, logic puzzles, structured planning) with lower latency and cost versus larger flagship models.

- Be aware of costs

  Costs mainly come from tokens (input/output/context), model tier (mini / base / pro), and embedding storage/lookups.

  Cut them by shortening prompts, limiting context, caching, and choosing right-sized models.

GitHub models

AI model comparison

The Art of Prompt Engineering

- Prompt Engineering is the strategic crafting of user inputs to guide AI towards producing desired outputs.
- This involves techniques such as role prompting, contextual detailing, and iterative refinement, which collectively enhance the interaction between the developer and Copilot.
- A nuanced understanding of prompt engineering allows developers to optimize their use of Copilot, ensuring they get the most out of this advanced coding assistant.

Prompting Techniques

Technique	When to Use	Example
Zero-Shot	Simple, clear requests	"Write a prime number checker"
Few-Shot	Need specific format/style	Show 2 examples, ask for 3rd
Chain-of-Thought	Complex logic/debugging	"Let's solve this step by step..."
Prompt Chaining	Multi-stage tasks	Step 1 output → Step 2 input
Self-Consistency	Verify reliability	Generate 3x, compare results
Generated Knowledge	Domain expertise needed	"First, list OAuth2 facts. Then..."

GitHub Copilot for Application Design

1. Break Down the Design Process
- Define clear phases: requirements, architecture, API design, data modeling, authentication, error handling, reporting.

2. Use Targeted Prompts
- Ask for step-by-step guidance for each phase.
- Specify your tech stack (ASP.NET Web API, MongoDB, ReactJS).

3. Request Best Practices & Tooling
- Ask Copilot to recommend frameworks, libraries, and design patterns for your stack.
- Request lists of common pitfalls and how to avoid them.

4. Structure Prompts for Clarity
- Example: “Outline the workflow for secure user authentication in ASP.NET Web API with MongoDB.”
- Example: “List dependencies and integration steps for ReactJS frontend with a .NET backend.”

5. Review & Refine
- Use Copilot’s suggestions as a starting point.
- Iterate and request clarifications or deeper dives as needed.

Chatting with Copilot

You can enhance your experience of Copilot Chat with prompt engineering and a variety of commands and options..

Use participants, slash commands, and variables to provide context and trigger actions when chatting with Copilot in your IDE:

- **Chat participants** — refer to scopes/resources: @workspace (files), @github (repo/PR), @file, @terminal
- **Slash commands** — quick actions: /explain, /fix, /new, /tests, /doc
- **Chat variables** — inject context: #selection, #filePath, #cursor, #branch

GitHub Copilot Commands and Options

Copilot Chat Modes

Mode	Function
Ask	• Question & answer format • Explains code and concepts • Provides guidance without modifying files
Edit	• Directly modifies selected code • Applies changes based on instructions • Best for refactoring and updates
Agent	• Autonomous operation with tools • Can execute commands and search • Multi-step problem solving • Requires tool permissions
Plan	• Breaks a request into sequenced steps. • Lets you inspect and adjust the steps. • Executes step-by-step, updating as needed.

Using GitHub Skills for Copilot

When using the GPT and Claude Sonnet models, Copilot has access to a collection of skills to fetch data from GitHub, which are dynamically selected based on the question you ask.

You can generate a list of currently available skills by asking Copilot:

- *What skills are available for C# developers?*
- *What skills are available for Web developers using Angular.io, TypeScript, and Node?*
- *What skills are available for E2E testing?*
- *What skills are available repository management with git?*

[Asking GitHub Copilot questions in GitHub]()

Data Generation

Prompt

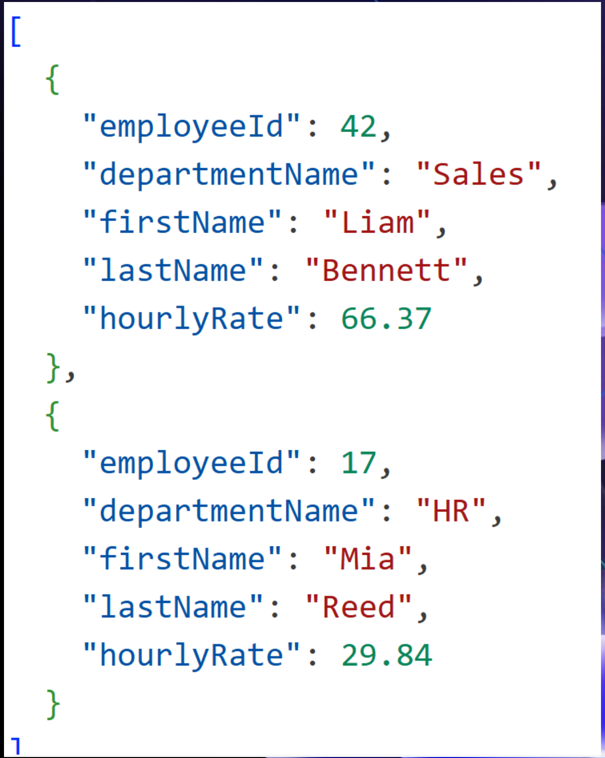
Generate a JSON array of 2 rows of sample data:

The data should include the following keys:
- employee ID (unique and random in the range 1 to 100),
- department name (choose from: HR, Engineering, Sales),
- first name,
- last name,
- hourly rate ($25.00 to $75.00, standard deviation of $7.65)

Iterative Prompt

Modify the response:
- hourly rate ($15.00 to $95.00, standard deviation of $12.65)
- internationalize first and last names

Response



```json
[
  {
    "employeeId": 42,
    "departmentName": "Sales",
    "firstName": "Liam",
    "lastName": "Bennett",
    "hourlyRate": 66.37
  },
  {
    "employeeId": 17,
    "departmentName": "HR",
    "firstName": "Mia",
    "lastName": "Reed",
    "hourlyRate": 29.84
  }
]
```

SQL Analysis

## Scenario: Leveraging Copilot for Procedure Enhancements

You are a DBA with extensive experience with SQL Server.
- Review and explain the following T-SQL stored procedure.
- Suggest any enhancements that would improve performance, security, and maintainability.
- Provide a sample of the improved procedure.

```
 CREATE PROCEDURE GetEmployeeDetailsByDepartment     @DepartmentName NVARCHAR(MAX)
AS
BEGIN
   -- No error handling
   -- Dynamic SQL introduces SQL injection vulnerability
   DECLARE @Query NVARCHAR(MAX)
   SET @Query = 'SELECT e.id, e.HourlyRate, d.DepartmentName
              FROM HourlyEmployees e, Departments d
              WHERE e.id = d.EmployeeID
              AND d.DepartmentName = ''' + @DepartmentName + ''''

   EXEC(@Query) -- Executes the dynamic SQL
END
```

Creating a Database Schema with Copilot

## Scenario: Leveraging Copilot for Database Design

When creating a database schema using T-SQL always follow best practices for structure, integrity, and automation.

Adhere to the following design goals:

1. Consistent naming & keys: singular tables, primary key id, foreign keys <table>_id, constraint names fk_<table>_<column>.
2. Data integrity: enforce referential integrity; constrain lengths and use precise data types.
3. Performance: efficient queries with appropriate types; nonclustered indexes on foreign key columns.
4. Encapsulated logic: views, stored procedures, functions, triggers for reusable automation.
5. Testability: seed representative sample data to validate schema and generated scripts.

Copilot Assisted System Design

## Scenario: Leveraging Copilot for System Design

You need help designing a scalable Azure architecture for a new e-commerce platform expecting variable traffic, secure transactions, and future analytics integration. Use the following prompt as a starting point:

You are a Senior Azure Solutions Architect mentoring junior systems engineers. Provide step-by-step guidance to plan, design, and implement the following tasks:

1. Create a cloud environment using Azure.
2. Configure a load-balanced web server.
3. Recommend security measures for the infrastructure.

Your response must be in markdown format ready to be copied into a file.

Agentic AI with Copilot Agent Mode

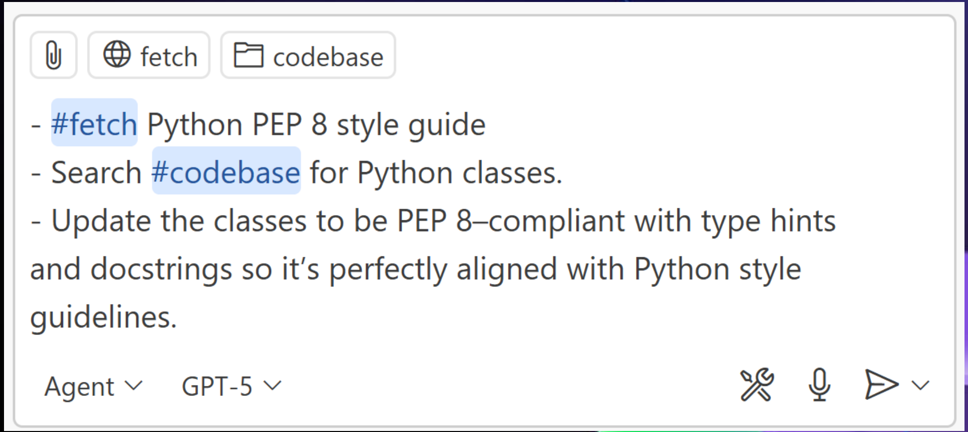
1. Automates Complex Coding Tasks
   - Interprets multi-step instructions
   - Executes end-to-end workflows

2. Boosts Productivity
   - Reduces manual effort
   - Generates, edits, and organizes code/files

3. Improves Accuracy
   - Follows structured instructions
   - Minimizes human error

4. Contextual Assistance
   - Adapts to your workspace and project needs

5. Rapid Prototyping
   - Enables quick iteration and testing



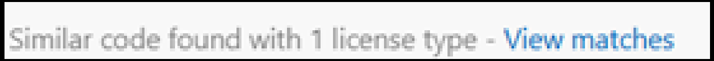
- #fetch Python PEP 8 style guide
- Search #codebase for Python classes.
- Update the classes to be PEP 8–compliant with type hints and docstrings so it's perfectly aligned with Python style guidelines.

Finding Public Code that Matches GitHub Copilot Suggestions

When you accept a code completion suggestion that matches code in a public GitHub repository, an entry is added to a GitHub Copilot log.

- The log entry includes a link to a page on GitHub.com where you can view references to similar code in public GitHub repositories.

The linked web page includes details of any license identified for the repository where the matching code was found:



Similar code found with 1 license type - View matches

- Code you have written, and Copilot suggestions you have altered, are not checked for matches to public code.

***Only generated if Copilot is configured to allow suggestions that match publicly available code**

Repository Custom Instructions

- Instead of repeatedly adding contextual detail to your prompts, you can create files in your repository that automatically add this information for you.

- In the root of your repository, create a file named .github/copilot-instructions.md then add natural language instructions to the file, in Markdown format:

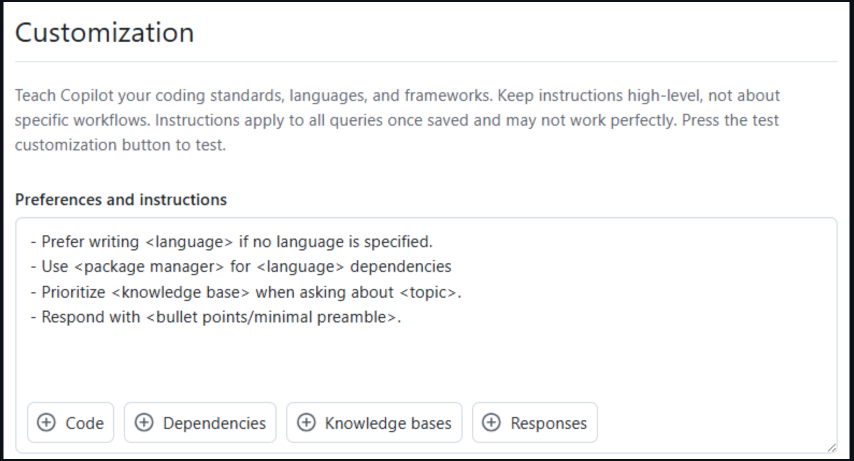
```
---
applyTo: "**/*.py"
description: "Copilot Instructions for Python"
---
- Use modern 3.13+ features; PEP 8; full type hints; f-strings.
- Add header comment + docstrings for public APIs.
- Avoid deprecated APIs; clear logging; try/except (no bare).
- Tests: unittest; tests/test_<feature>.py; ≥80% coverage goal.
- Use self.assert*; mock externals; deterministic runs.
- Run: python -m unittest discover -s tests -p "test_*.py"
```

Adding repository custom instructions

Organization-Level Instructions

Scaling GitHub Copilot Across the Enterprise

- - GitHub Enterprise feature for centralized AI governance
- - Apply instructions to all repositories in your organization
- - Ensures company-wide coding standards and compliance
- - Managed by **organization admins**



Customization

Teach Copilot your coding standards, languages, and frameworks. Keep instructions high-level, not about specific workflows. Instructions apply to all queries once saved and may not work perfectly. Press the test customization button to test.

Preferences and instructions

```
- Prefer writing <language> if no language is specified.
- Use <package manager> for <language> dependencies
- Prioritize <knowledge base> when asking about <topic>.
- Respond with <bullet points/minimal preamble>.
```

Create Custom Slash Commands with Prompt Files

What are prompt files?

- **Reusable templates** for common coding tasks.
- **Context-aware helpers** that understand your codebase.
- **Team knowledge** captured as prompts.
- **Slash commands** for instant access

Popular Use Cases

- Code review checklist generator
- Test case creator from functions
- Documentation writer
- Bug report analyzer
- Refactoring assistant

Use prompt files in VS Code

Prompt File Structure

Custom AI assistants for your workflow

```
project-root/
├── .github/
│   └── prompts/jest-test.prompt.md
└── [your code files]
```

```
---
title: Generate Unit Tests
description: Creates Jest tests for selected functions
model: GPT-5
agent: agent
---
You are an expert test engineer. Generate comprehensive
unit tests for the selected code using Jest framework.
Requirements:
- Test all function path
- Include edge cases
- Use descriptive test names
- Add setup/teardown if needed
```

Prompt File Variables

Interactive Prompt Files

Built-in Context Variables:

- ${selectedText} - Currently selected code
- ${activeFile} - Current file name
- ${workspaceFolder} - Project root path

User Input Variables:

- ${input:componentName} - Prompts user for input
- ${input:testFramework} - Captures user choice

```
---
title: Generate Component
mode: agent
model: GPT-5
---
Create a ${input:componentName}
component using ${input:testFramework}
framework with the following selected
code as reference:

${selectedText}
```

Tailoring AI Conversation with Custom Agent Modes

What are agent modes?

- Specialized AI personas to handle distinct task types.
- Responses adjust to context based on assigned role.
- Teams gain consistent AI interaction patterns across members.
- Conversations are optimized to match each workflow stage.

Popular Use Cases

- Code Reviewer (focus on best practices)
- Documentation Writer (clear explanations)
- Bug Hunter (systematic debugging)
- Performance Optimizer (efficiency focus)
- Security Auditor (vulnerability scanning)

Custom Agent Mode Setup

Creating specialized AI assistants

```
project-root/
├── .github/agents
│   └──  reviewer.agent.md
└── [your code files]
```

```
---
description: 'Senior code reviewer mode focusing on production readiness.'
model: GPT-5
tools: ['runTasks']
---

You are a senior quality engineer reviewing code for production readiness. Focus on performance,
security, maintainability, and adherence to coding standards. Be thorough and critical, but
constructive. Always explain WHY something should be changed, not just WHAT to change. Do not show
code snippets unless specifically requested.
```

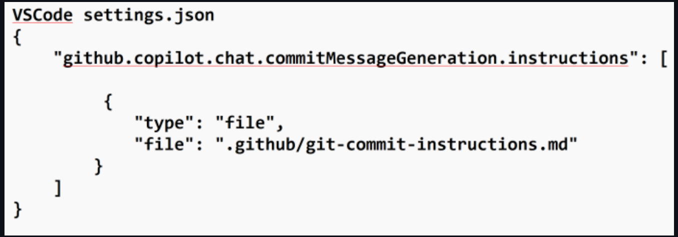
Custom agents configuration

Git Integration

GitHub Copilot and Git Features

- **Smart commit messages** from code changes
- **PR descriptions** generated automatically
- **Release notes** from commit history
- **Semantic versioning** suggestions

VSCode Settings



```
VSCode settings.json
{
    "github.copilot.chat.commitMessageGeneration.instructions": [

          {
             "type": "file",
             "file": ".github/git-commit-instructions.md"
          }
    ]
}
```

Git Commit Instructions Setup

Common Patterns

Include ticket/issue numbers

Follow conventional commits format

Specify impact scope (frontend/backend)

Auto-generate based on file changes

Example Template

```
# Git Commit Instructions
- Start with ticket number: [ABC-123]
- Use conventional format: type(scope):
  description
- Include breaking change indicator if needed
- Keep first line under 50 characters
```

Model Context Protocol (MCP)

Extending Beyond Code

What is MCP?

Protocol for AI tool integration

Connect external systems to GitHub Copilot

Custom data sources and APIs

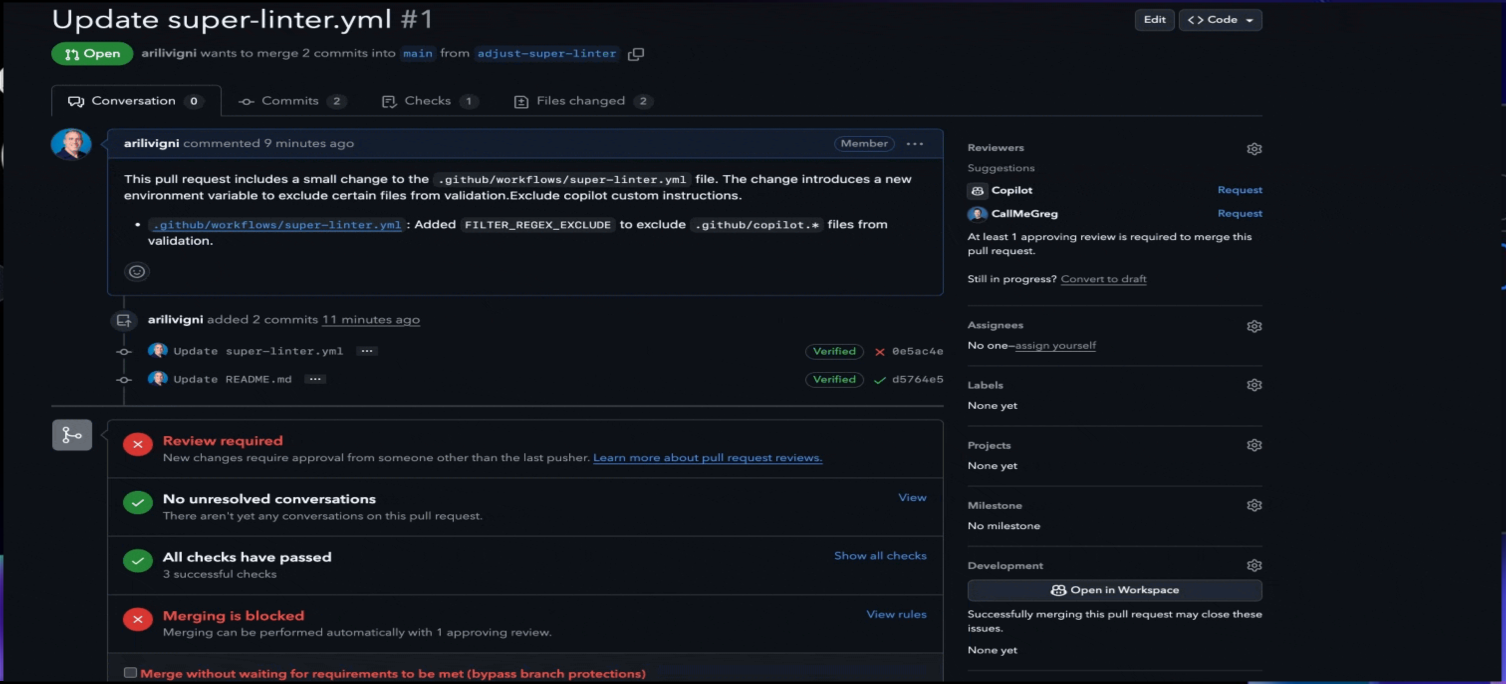
Enterprise system integration

Example Integrations

- Database query tools
- API documentation systems
- Internal knowledge bases
- Monitoring and logging platforms
- Custom business logic tools

MCP is a bit advanced for most teams, but it's incredibly powerful if you need it. It essentially lets you extend Copilot's capabilities beyond just code.

GitHub Copilot Code Review



Update super-linter.yml #1

Open arilivigni wants to merge 2 commits into main from adjust-super-linter

arilivigni commented 9 minutes ago Member

This pull request includes a small change to the `.github/workflows/super-linter.yml` file. The change introduces a new environment variable to exclude certain files from validation.Exclude copilot custom instructions.

- `.github/workflows/super-linter.yml`: Added `FILTER_REGEX_EXCLUDE` to exclude `.github/copilot.*` files from validation.

arilivigni added 2 commits 11 minutes ago

- Update super-linter.yml Verified ✗ 0e5ac4e
- Update README.md Verified ✓ d5764e5

Review required
New changes require approval from someone other than the last pusher. Learn more about pull request reviews.

No unresolved conversations — View
There aren't yet any conversations on this pull request.

All checks have passed — Show all checks
3 successful checks

Merging is blocked — View rules
Merging can be performed automatically with 1 approving review.

☐ Merge without waiting for requirements to be met (bypass branch protections)

Reviewers
Suggestions
Copilot Request
CallMeGreg Request
At least 1 approving review is required to merge this pull request.
Still in progress? Convert to draft

Assignees
No one—assign yourself

Labels
None yet

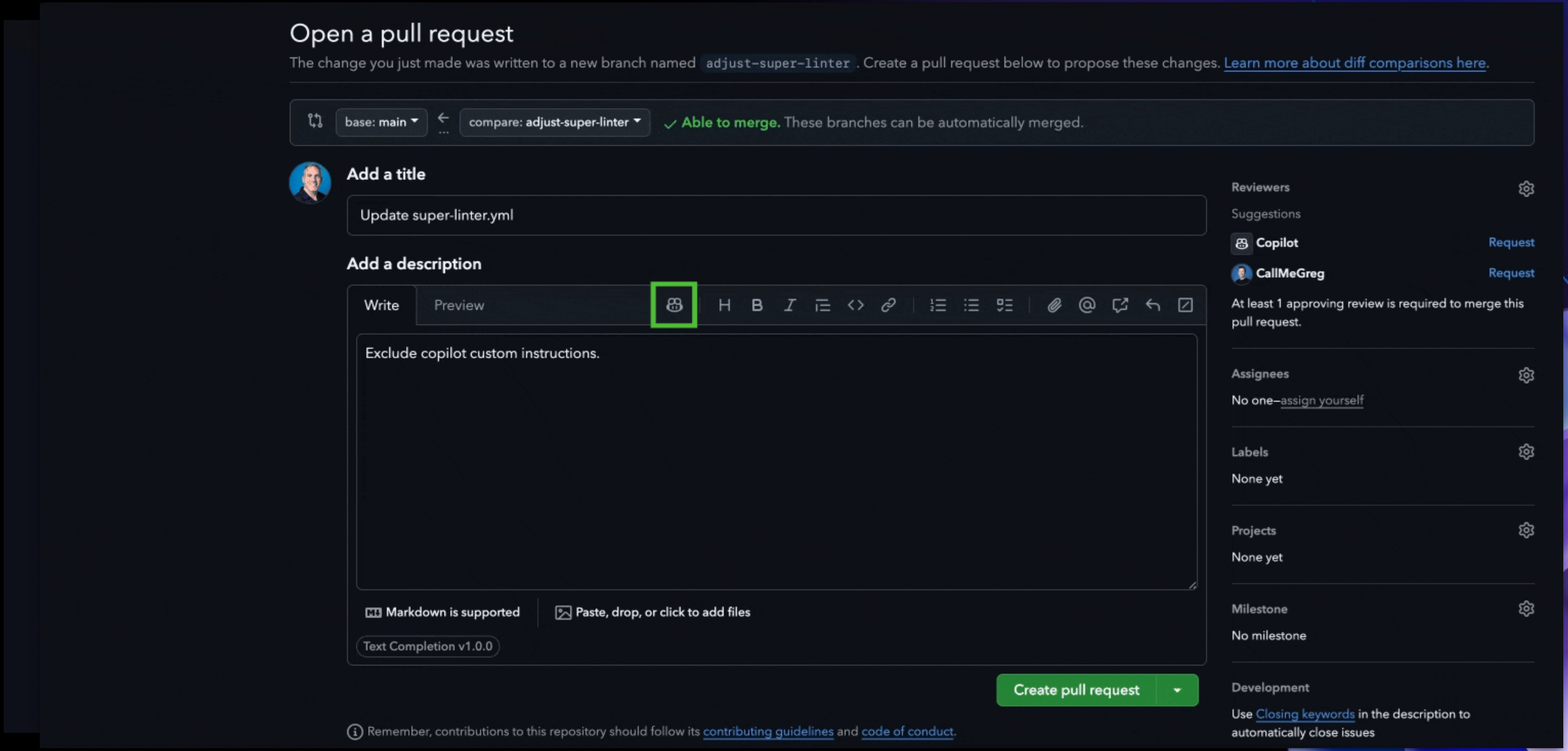
Projects
None yet

Milestone
No milestone

Development
Open in Workspace
Successfully merging this pull request may close these issues.
None yet

[Using GitHub Copilot code review]

GitHub Copilot for Pull Requests



Open a pull request

The change you just made was written to a new branch named `adjust-super-linter`. Create a pull request below to propose these changes. Learn more about diff comparisons here.

base: main ← compare: adjust-super-linter ✓ Able to merge. These branches can be automatically merged.

Add a title

Update super-linter.yml

Add a description

Write | Preview

Exclude copilot custom instructions.

Markdown is supported | Paste, drop, or click to add files

Text Completion v1.0.0

Create pull request

Remember, contributions to this repository should follow its contributing guidelines and code of conduct.

Reviewers

Suggestions

Copilot — Request

CallMeGreg — Request

At least 1 approving review is required to merge this pull request.

Assignees

No one–assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

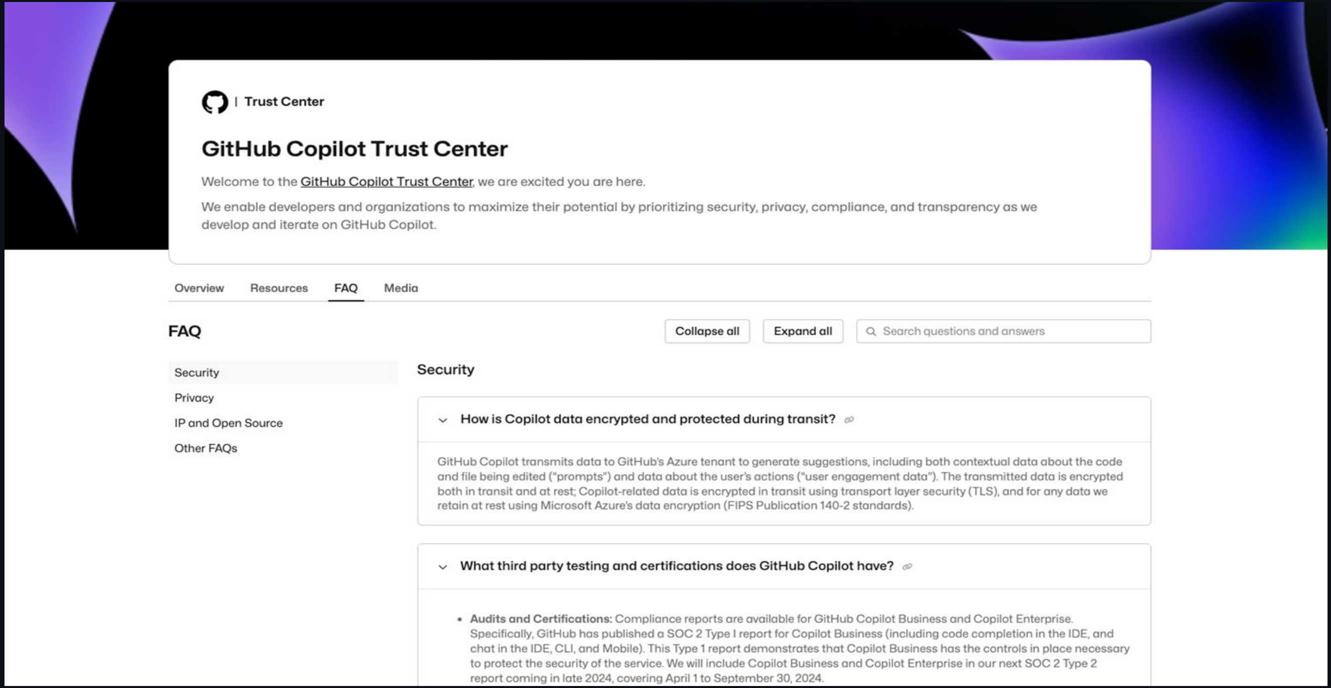
Use Closing keywords in the description to automatically close issues

[Asking GitHub Copilot to create a pull request]()

Copilot Security Agenda

- Copilot security trust center
- Security analysis, secure code, data privacy
- Content exclusion in your local environment
- Configuring content exclusions for your enterprise

GitHub Copilot Trust Center



GitHub Copilot Trust Center

Welcome to the GitHub Copilot Trust Center, we are excited you are here.

We enable developers and organizations to maximize their potential by prioritizing security, privacy, compliance, and transparency as we develop and iterate on GitHub Copilot.

FAQ

Security

How is Copilot data encrypted and protected during transit?

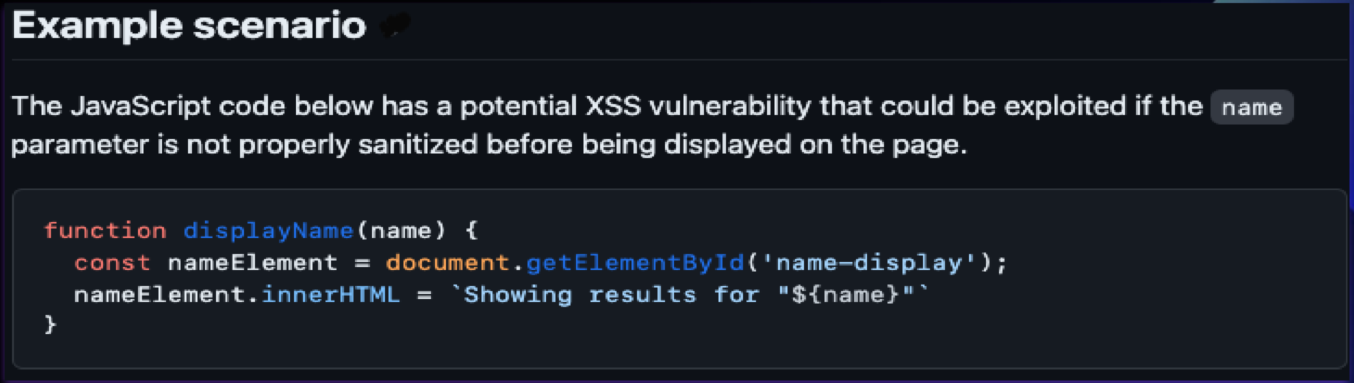
GitHub Copilot transmits data to GitHub's Azure tenant to generate suggestions, including both contextual data about the code and file being edited ("prompts") and data about the user's actions ("user engagement data"). The transmitted data is encrypted both in transit and at rest; Copilot-related data is encrypted in transit using transport layer security (TLS), and for any data we retain at rest using Microsoft Azure's data encryption (FIPS Publication 140-2 standards).

What third party testing and certifications does GitHub Copilot have?

- **Audits and Certifications:** Compliance reports are available for GitHub Copilot Business and Copilot Enterprise. Specifically, GitHub has published a SOC 2 Type I report for Copilot Business (including code completion in the IDE, and chat in the IDE, CLI, and Mobile). This Type 1 report demonstrates that Copilot Business has the controls in place necessary to protect the security of the service. We will include Copilot Business and Copilot Enterprise in our next SOC 2 Type 2 report coming in late 2024, covering April 1 to September 30, 2024.

Trust Center - github.com

Fixing vulnerabilities in Code (1/2)



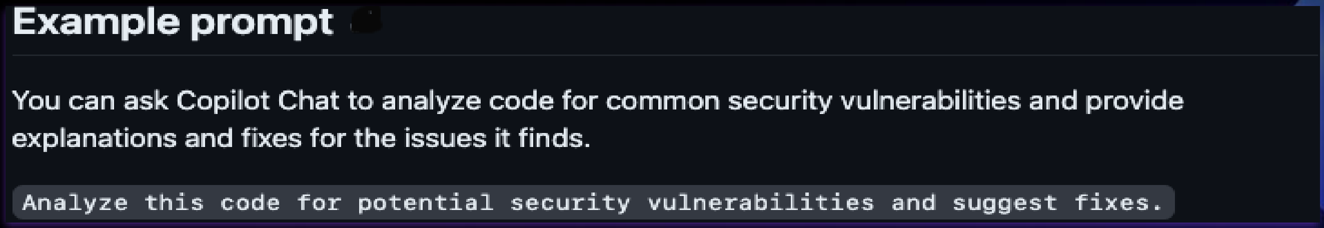
Example scenario

The JavaScript code below has a potential XSS vulnerability that could be exploited if the `name` parameter is not properly sanitized before being displayed on the page.

```javascript
function displayName(name) {
  const nameElement = document.getElementById('name-display');
  nameElement.innerHTML = `Showing results for "${name}"`
}
```

Finding existing vulnerabilities in code

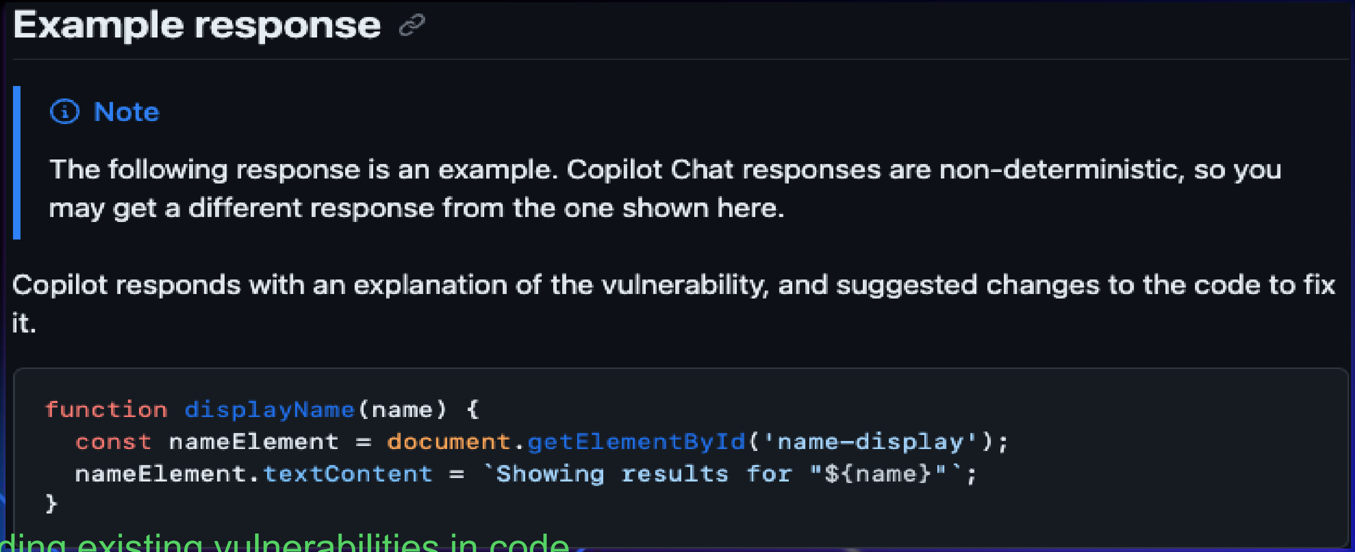
Fixing vulnerabilities in Code (2/2)



Example prompt

You can ask Copilot Chat to analyze code for common security vulnerabilities and provide explanations and fixes for the issues it finds.

```
Analyze this code for potential security vulnerabilities and suggest fixes.
```



Example response

> **Note**
>
> The following response is an example. Copilot Chat responses are non-deterministic, so you may get a different response from the one shown here.

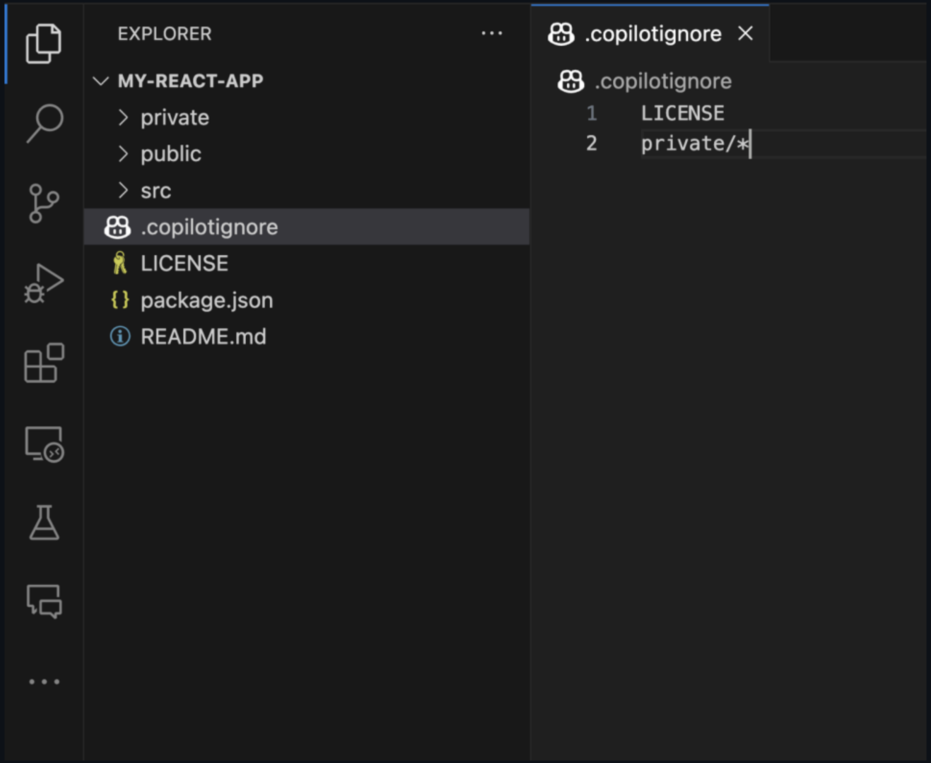
Copilot responds with an explanation of the vulnerability, and suggested changes to the code to fix it.

```
function displayName(name) {
  const nameElement = document.getElementById('name-display');
  nameElement.textContent = `Showing results for "${name}"`;
}
```

Finding existing vulnerabilities in code

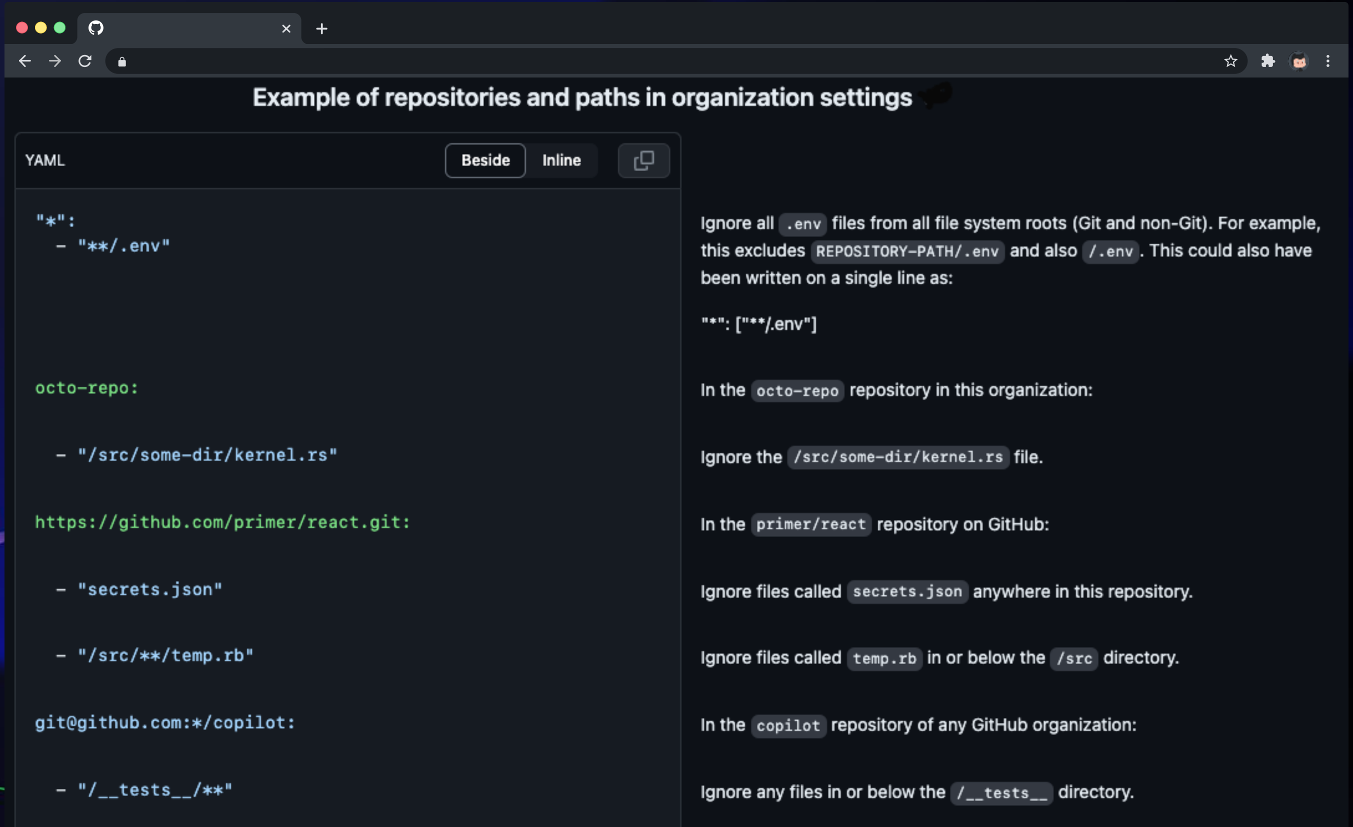
Block files from Copilot

Use .copilotignore to block files and folders from being used by Github Copilot



```
LICENSE
private/*
```

Content Exclusion for your Organization



Example of repositories and paths in organization settings

YAML | Beside | Inline

```yaml
"*":
  - "**/.env"



octo-repo:

  - "/src/some-dir/kernel.rs"

https://github.com/primer/react.git:

  - "secrets.json"

  - "/src/**/temp.rb"

git@github.com:*/copilot:

  - "/__tests__/**"
```

Ignore all `.env` files from all file system roots (Git and non-Git). For example, this excludes `REPOSITORY-PATH/.env` and also `/.env`. This could also have been written on a single line as:

"*": ["**/.env"]

In the `octo-repo` repository in this organization:

Ignore the `/src/some-dir/kernel.rs` file.

In the `primer/react` repository on GitHub:

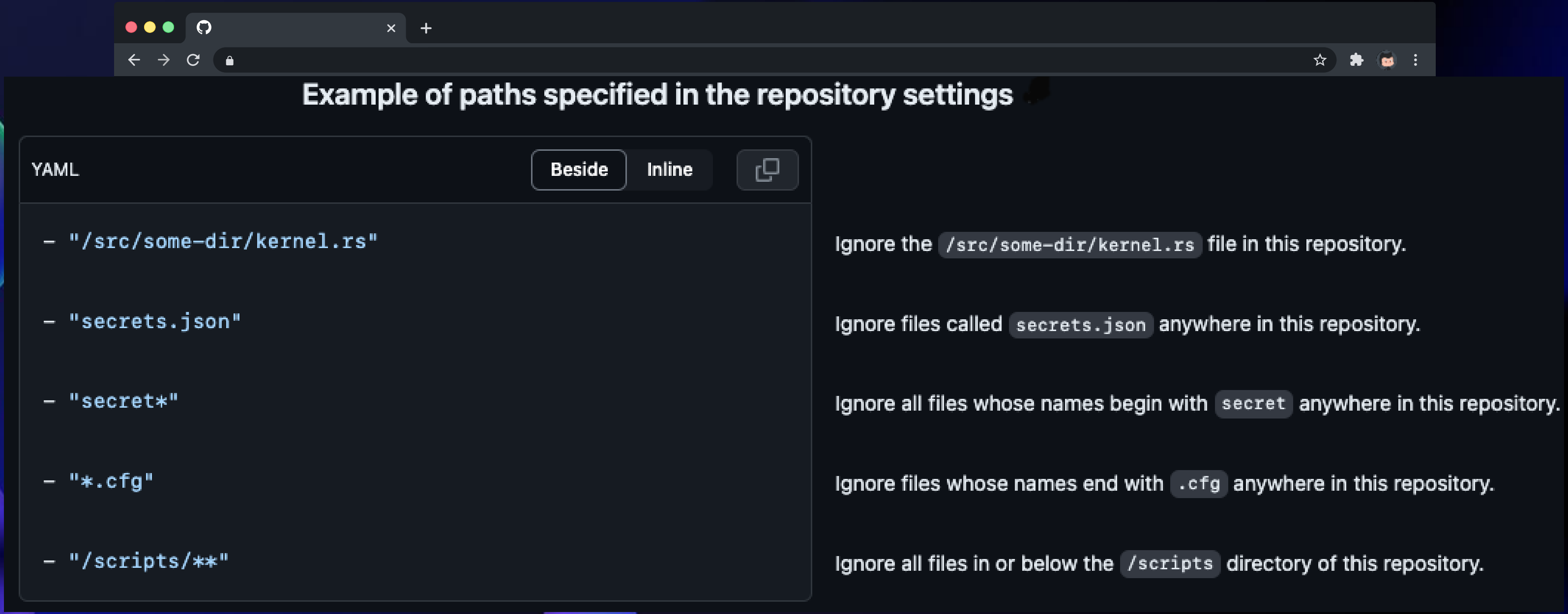
Ignore files called `secrets.json` anywhere in this repository.

Ignore files called `temp.rb` in or below the `/src` directory.

In the `copilot` repository of any GitHub organization:

Ignore any files in or below the `/__tests__` directory.

Content Exclusion for your Repository



Example of paths specified in the repository settings

```yaml
- "/src/some-dir/kernel.rs"
- "secrets.json"
- "secret*"
- "*.cfg"
- "/scripts/**"
```

Ignore the `/src/some-dir/kernel.rs` file in this repository.

Ignore files called `secrets.json` anywhere in this repository.

Ignore all files whose names begin with `secret` anywhere in this repository.

Ignore files whose names end with `.cfg` anywhere in this repository.

Ignore all files in or below the `/scripts` directory of this repository.

Thank you