

IV GitHub Copilot: Funcionalidades Avanzadas

Versión V1.1

Índice

- 01 Funcionalidades avanzadas
- 02 Tricks

Leyenda

A nivel de Contenido



Cuadros de Contexto



Problemas

Texto

Objetivos

Texto

Ejemplos explicativos



Detalles

A nivel de Tipo de Slide



Tipo "Curiosidades"



Tipo "Ejemplo"

Los recursos utilizados se han obtenido de

- <https://storyset.com> : Ilustraciones customizables gratuitas
- <https://lexica.art> : IA que genera imágenes
- <https://www.freepik.es>: Iconos

01 Funcionalidades avanzadas

01

01 Instrucciones personalizadas

01 Funcionalidades avanzadas

01.01

El problema de Copilot en equipos

Instrucciones personalizadas



Reduce la
mantenibilidad

Incrementa las
violaciones de
calidad

No hay un
estándar de
codificación

Incremento
del tiempo de
refactoring

Incremento
del tiempo de
revisión

Copilot es genérico...

Instrucciones personalizadas



Cosas que no conoce :

Normas de codificación de tu equipo

Configuraciones personales

Formas de hacer las cosas

Buenas prácticas

Conocimiento específico del dominio

Librerías "secretas"

...

¿Qué es una instrucción personalizada?

Instrucciones personalizadas



Permiten configurar GitHub Copilot para que se **adapte mejor** a tu **estilo de codificación, preferencias, lenguaje y necesidades específicas**, proporcionando **sugerencias más relevantes**

Características:

- *Funcionan a partir de una versión V1.98*
- *Configuración de tipo de proyecto o tecnologías preferidas*
- *Personalización del tono o estilo del código generado*
- *Configuración del tipo de proyectos o tecnologías preferidas*
- *Se pueden modificar desde la interfaz del usuario del editor compatible*
- *Establecer preferencias como nombre del autor, estructura del código, uso de comentarios, etc.*
- *Existe la opción por defecto en : .github/copilot-instructions.md*
- *Puedes utilizar uno o varios ficheros de instrucciones usando .instructions.md*

Importante: Estas instrucciones se envían en cada solicitud que se realiza

Objetivo: Hacer que GitHub Copilot entienda mejor el contexto y las preferencias del desarrollador, mejorando la calidad y utilidad de sus sugerencias

Ejemplo: Es como entrar a un asistente personal para que conozca cómo te gusta trabajar: si prefieres código limpio y comentado, con variables en español. Copilot intentará seguir esas indicaciones

¿Qué suele incluir estas instrucciones personalizadas?

Instrucciones personalizadas



Estilos de codificación

Convenciones de lenguaje

Convenciones de nomenclatura

Convenciones de framework

Convenciones de herramientas

Convenciones de patrones

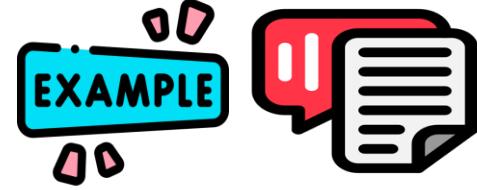
Reglas de análisis estático (más importantes)

Buenas prácticas

...

Ejemplo de fichero de instrucciones

Instrucciones personalizadas



Method names should be descriptive with mixed case and an underscore between words

Always use hungarian notation for variable names

Use 2 spaces for indentation

Use single quotes for strings Use double quotes for docstrings

Prefix private class members with underscore (_)

Use ALL_CAPS for constants

For methods and classes, place the curly braces on a different line from the method or class name.

For error handling:

- Add try/catch blocks to all methods
- Use specific exceptions
- Avoid using generic exceptions
- Use custom exceptions for specific cases
- Log errors with a logger
- Always log errors with contextual information

Add triple slash comments at the top of each class and method. Include a brief description of the class or method, its parameters, and return values. After saving the above file, I entered the following prompt:

El problema de los entornos copo de nieve

Instrucciones personalizadas



Seguir directrices = coherencia

Instrucciones personalizadas



Documentación

Instrucciones personalizadas



https://code.visualstudio.com/docs/copilot/copilot-customization#_types-of-custom_instructions

The screenshot shows a dark-themed web browser window displaying the Visual Studio Code documentation. The URL in the address bar is https://code.visualstudio.com/docs/copilot/copilot-customization#_types-of-custom_instructions. The main content area is titled "Customize AI responses in VS Code". It discusses how to use custom instructions and prompt files to tailor AI responses. A sidebar on the left lists various documentation categories like Overview, Setup, Get Started, Configure, Edit Code, Build, Debug, Test, Source Control, Terminal, and GitHub Copilot. The GitHub Copilot section is currently active. The right sidebar contains links for RSS Feed, Ask questions, Follow @code, Request features, Report issues, and Watch videos.

Existen muchas posibilidades de configuración según las necesidades:

- Fichero por defecto centralizado -> [.github/copilot-instructions.md](#)
 - All-in-one
- Referenciar dentro del fichero a otro mediante un enlace local de Markdown -> [XXX](./path/fichero)
- Varios ficheros por defecto -> [.github/instructions/](#)
 - Múltiples ficheros usando .instructions.md
- Varios ficheros específicos -> [./docs/](#)
- Por funcionalidad: **code review, commit, pull request, etc.**
- Configuración a nivel
 - IDE
 - Repositorio
 - Profile
- ...

Documentación

Instrucciones personalizadas



https://code.visualstudio.com/docs/copilot/copilot-customization#_types-of-custom_instructions

The screenshot shows a dark-themed web browser window displaying the Visual Studio Code documentation. The URL in the address bar is https://code.visualstudio.com/docs/copilot/copilot-customization#_types-of-custom_instructions. The main content area features a large heading "Customize AI responses in VS Code". Below it, a paragraph explains that Chat in VS Code can generate code matching coding practices by storing context in files. It then lists three ways to customize AI responses: "Custom instructions", "Prompt files (experimental)", and "MCP Servers". Each method has a corresponding "Example scenarios" section. On the left, a sidebar navigation menu includes links for Overview, SETUP, GET STARTED, CONFIGURE, EDIT CODE, BUILD, DEBUG, TEST, and GITHUB COPilot, with GITHUB COPilot currently selected. On the right, a sidebar titled "IN THIS ARTICLE" lists "Custom instructions", "Prompt files (experimental)", "Centrally manage instructions and prompt files in VS Code", "Settings", and "Related content". A footer at the bottom of the page includes links for RSS Feed, Ask questions, Follow @code, Request features, Report issues, and Watch videos.

Try [MCP servers](#) to extend agent mode in VS Code!

Customize AI responses in VS Code

Chat in Visual Studio Code can give you responses and generate code that matches your coding practices and project requirements, if you give it the right context. Instead of repeatedly adding this information in every chat prompt, you can store this context in files and automatically include it in every chat request. In this article, you learn how to use custom instructions and prompt files to customize AI responses in VS Code.

There are three main ways to customize AI responses in Visual Studio Code:

- **Custom instructions:** Define common guidelines or rules for tasks like generating code, performing code reviews, or generating commit messages. Custom instructions describe the conditions in which the AI should perform operate (*how* a task should be done). Learn how to [define custom instructions](#). VS Code can also help you [generate a custom instructions file for your workspace](#) that matches your coding practices and project requirements.
- **Prompt files:** Define reusable prompts for common tasks like generating code or performing a code review. Prompt files are standalone prompts that you can run directly in chat. They describe the task to be performed (*what* should be done). Optionally, you can include tasks-specific guidelines about how the task should be performed, or you can reference custom instructions in the prompt file. Learn how to [create prompt files](#).

▶ Example scenarios

▶ Example scenarios

IN THIS ARTICLE

- Custom instructions
- Prompt files (experimental)
- Centrally manage instructions and prompt files in VS Code
- Settings
- Related content

RSS Feed

Ask questions

Follow @code

Request features

Report issues

Watch videos

Documentación

Instrucciones personalizadas



<https://docs.github.com/es/copilot/how-tos/custom-instructions/adding-repository-custom-instructions-for-github-copilot>

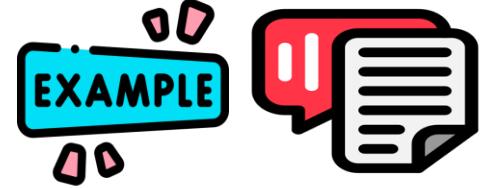
The screenshot shows a dark-themed web browser window displaying the GitHub Copilot documentation. The URL in the address bar is <https://docs.github.com/es/copilot/how-tos/custom-instructions/adding-repository-custom-instructions-for-github-copilot>. The page title is "Incorporación de instrucciones personalizadas del repositorio para GitHub Copilot". The main content area includes a note about the feature being in public preview and a section about repository custom instructions for Copilot. A sidebar on the left lists various GitHub Copilot topics, and a sidebar on the right lists related articles under "En este artículo".

En este artículo

- About repository custom instructions for Copilot
- Prerequisites for repository custom instructions
- Creating a repository custom instructions file
- Writing effective repository custom instructions
- Repository custom instructions in use
- Enabling or disabling repository custom instructions

Ejemplo de Guía de Estilo

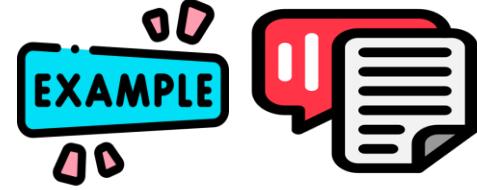
Instrucciones personalizadas



```
# Code Style Guide
## General Guidelines
- Keep the code simple and readable.
- Use Prettier for code formatting.
## CSS
- Use class names that are descriptive and follow a consistent naming convention.
- Avoid using IDs for styling.
- Organize CSS properties in a logical order.
## Tailwind CSS
- Use utility-first classes to build components.
- Avoid using custom CSS when possible.
- Group related classes together for better readability.
## JavaScript
- Use `const` and `let` instead of `var`.
- Prefer arrow functions for anonymous functions.
- Use template literals for string concatenation.
- Always use semicolons.
- Follow the Prettier configuration for formatting.
## Vue.js
- Use single-file components (SFCs) with the `.vue` extension.
- Organize component structure in this order: `
```

Ejemplo de Guía de Commits

Instrucciones personalizadas



Commit Style Guide

We follow the [Conventional Commits](<https://www.conventionalcommits.org/en/v1.0.0/>) style for our commit messages.

Here are some examples:

- `feat: add new user authentication module`
- `fix: resolve issue with data fetching.`
- `docs: update README with installation instructions`
- `style: format code with Prettier`
- `refactor: improve performance of data processing`
- `test: add unit tests for user service`
- `chore: update dependencies`

commit-style.md

Configuración en el fichero settings.json de VSC

Instrucciones personalizadas



Commit Style Guide

We follow the [Conventional Commits](<https://www.conventionalcommits.org/en/v1.0.0/>) style for our commit messages.
Here are some examples:

```
{  
  "github.copilot.chat.codeGeneration.instructions": [  
    {  
      "file": "docs/code-style.md"// import instructions from file `code-style.md`  
    }  
  ],  
  "github.copilot.chat.commitMessageGeneration.instructions": [  
    {  
      "file": "docs/commit-message.md"// import instructions from file `commit-message.md`  
    }  
  ]  
}
```

Documentation-Driven Development

Instrucciones personalizadas



Conocimiento del dominio

Intención arquitectónica

Patrones de uso

Requerimientos

Restricciones

Best Practices

Instrucciones personalizadas



Mejora enormemente las sugerencias

Mantiene la coherencia

Hay que dedicarle tiempo a definir el contexto y mantenerlo

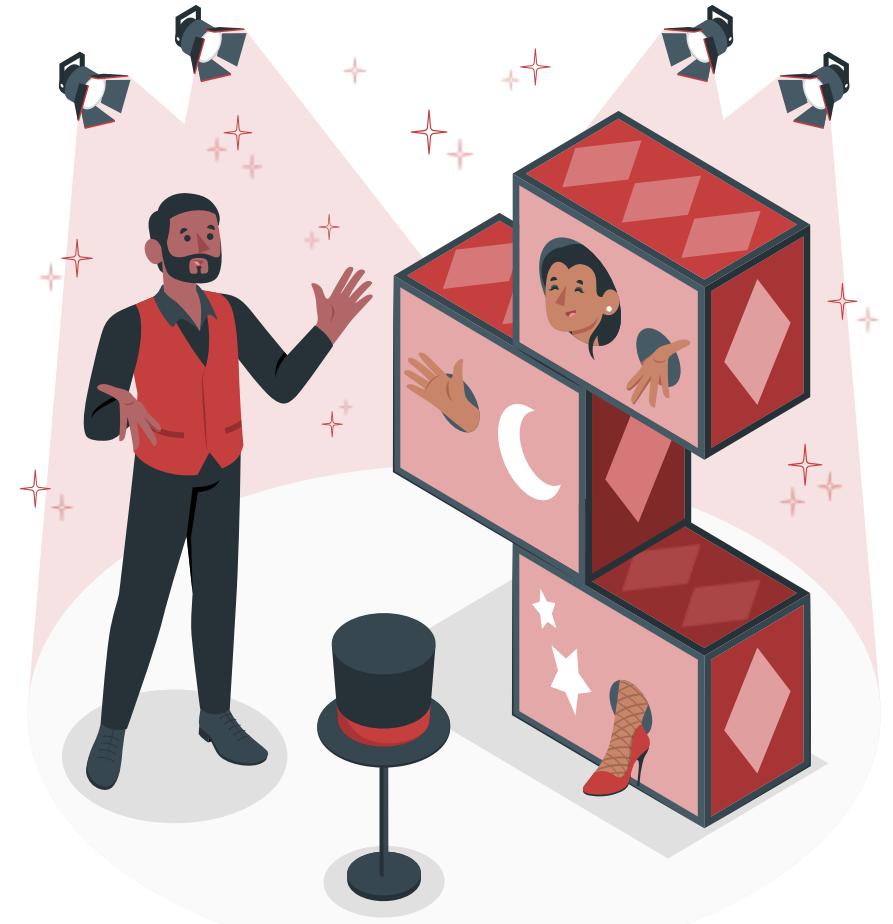
Ir perfeccionando poco a poco y de forma controlada

Añadir las instrucciones personalizadas en el repositorio para facilitarlos al equipo

Reduce las alucinaciones

Requieren control bajo el control de versiones

Acelera los OnBoarding y las transferencias de conocimiento



02 Ficheros de Prompts

01 Funcionalidades
avanzadas

01.02

¿Qué es un fichero de Prompt?

Ficheros de Prompts



Ficheros diseñados para **almacenar instrucciones o indicaciones** que orientan a GitHub Copilot en **tareas específicas**. Sirven como contexto estructurado para que el asistente genere código más preciso y adaptado al propósito deseado

Importante: Son prompts autocontenido

Características:

- *Suelen estar en formato .md, .txt o .prompt*
- *Contienen instrucciones detalladas, ejemplos de entrada/salida, estructuras deseadas o reglas de estilo*
- *Pueden ser reutilizados por todo un equipo para mantener coherencia en los resultados*
- *Ideales para tareas repetitivas, generación de código boilerplate o configuración de plantillas*
- *Se ejecutan desde la interfaz del chat*
- *Se almacenan en el workspace : .github/prompts*

Hay de varios tipos: a nivel de workspace o a nivel de usuario

Objetivo: Definir contexto estable y reutilizables por parte de los desarrolladores para tareas específicas

Ejemplo: Es como tener una “chuleta inteligente” predefinida para que Copilot trabaje con un enfoque específico

Documentación

Ficheros de Prompts



https://code.visualstudio.com/docs/copilot/copilot-customization#_types-of-custom_instructions

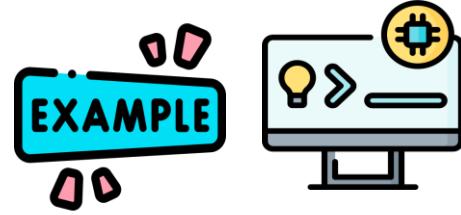
The screenshot shows the Visual Studio Code documentation website. The main content is titled "Prompt files (experimental)". It explains that prompt files are reusable prompts for common tasks like generating code or performing a code review. A "Tip" section notes that prompt files can reuse common guidelines and include task-specific instructions. The page also mentions that VS Code supports two types of scopes for prompt files: workspace prompt files (available within the workspace) and user prompt files (available across multiple workspaces). Below this, there are "Prompt file examples" with links to "Example: generate a React form component" and "Example: perform a security review of a REST API". The left sidebar contains navigation links for Overview, SETUP, GET STARTED, CONFIGURE, EDIT CODE, BUILD, DEBUG, TEST, SOURCE CONTROL, TERMINAL, GITHUB COPILOT, and MCP Servers.

Existen muchas posibilidades de configuración según las necesidades:

- Varios ficheros por defecto -> [.github/prompts/](#)
 - Múltiples ficheros usando .prompt.md
- Referenciar dentro del fichero a otro mediante un enlace local de Markdown -> [XXX](./path/fichero)
- Varios ficheros específicos -> [./docs/](#)
- Por funcionalidad: **code review, commit, pull request, etc.**
- Configuración a nivel
 - IDE
 - Repositorio
 - Profile
- ...
- Sincronizar los prompts mediante **Settings Sync**

Ejemplo de fichero de prompt

Ficheros de Prompts



```
---
```

```
mode: 'agent'
```

```
tools: ['markdownlint', 'codebase']
```

```
description: 'Update the documentation README.md file with new changes.'
```

```
--
```

You are a documentation specialist.

Requirements for the README update:

- Use the existing README structure as a guide.
- Ensure the new section is consistent with the current style and formatting.
- Update the README.md file with the new changes.
- Use markdownlint to ensure the updated README.md file is properly formatted.

Uso de variables en los prompts

Ficheros de Prompts



Tipo de variable	Ejemplos
Workspace	`\${workspaceFolder}`, `\${workspaceFolderBasename}`
Selección	`\${selection}`, `\${selectedText}`
Contexto del fichero	`\${file}`, `\${fileBasename}`, `\${fileDirname}`, `\${fileBasenameNoExtension}`
Entrada	`\${input:variableName}`, `\${input:variableName:placeholder}`

Las variables de tipo “Entrada” se refieren con su valor desde el Chat

Configuración en el fichero settings.json de VSC

Ficheros de Prompts



```
{  
  ...  
  "chat.promptFilesLocations": {  
    ".github/prompts": false,  
    "setup/**/prompts": true  
  },  
  ...  
}
```

Best Practices

Ficheros de Prompts



Acelera los workflows

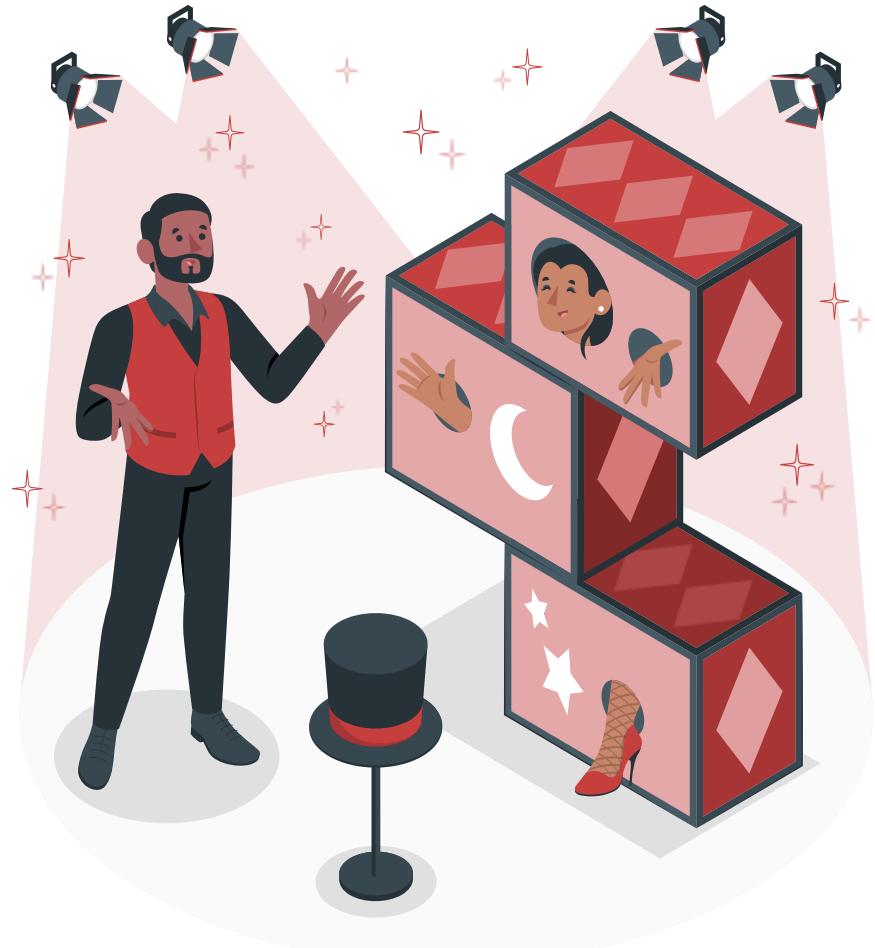
Mantiene la coherencia

Hay que dedicarle tiempo a definir los prompts

Ir perfeccionando poco a poco y de forma controlada

Acelera los OnBoarding y las transferencias de conocimiento

Hay que documentar y versionar estos prompts



03 MCP para GitHub Copilot

01 Funcionalidades
avanzadas

01.03

¿Qué es MCP (Model Context Protocol)?

MCP para GitHub Copilot



Estándar abierto que facilita la **integración** de forma **fluida entre aplicaciones LLM y otros sistemas** (fuentes de datos externas, herramientas y servidores)

Se basa en **establecer servidores MCP** que **actúan** como **intermediario** en un LLM y el otro sistema

Características:

- *Define un conjunto de reglas y especificaciones*
- *Genera un contexto Enriquecido al proporcionar a los modelos de lenguaje*
- *Mayor precisión al conectarse a fuentes de datos verificadas (reduce la probabilidad de errores y alucinaciones)*
- *Mayor personalización del comportamiento de los LLMs (adaptación al proyecto o a la empresa)*
- *Automatización de tareas complejas*

Objetivo: Crea una única solución para todos los modelos que facilite añadir nuevas funcionalidades

Ejemplo: Es como tener un USB-C pero para conectar cosas en los LLMs

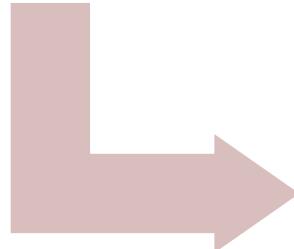
Origen del MCP

MCP para GitHub Copilot



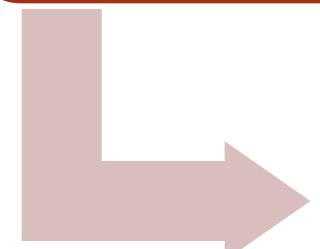
Fase 1

- Uso de los LLMs para realizar preguntas o solicitar acciones en base al conocimiento con el que han sido entrenados
- Explotación de su conocimiento, pero no realizan tareas/acciones (ejemplo : escribe un email pero no lo envía)



Fase 2

- Necesidad de integrar los LLMs con otras herramientas y/o fuentes de datos
- Integración vía API
- Estrategias de “function calling”
- Cada herramienta tiene su propia API y se hace necesario un desarrollo de integración específico para cada caso



Fase 3

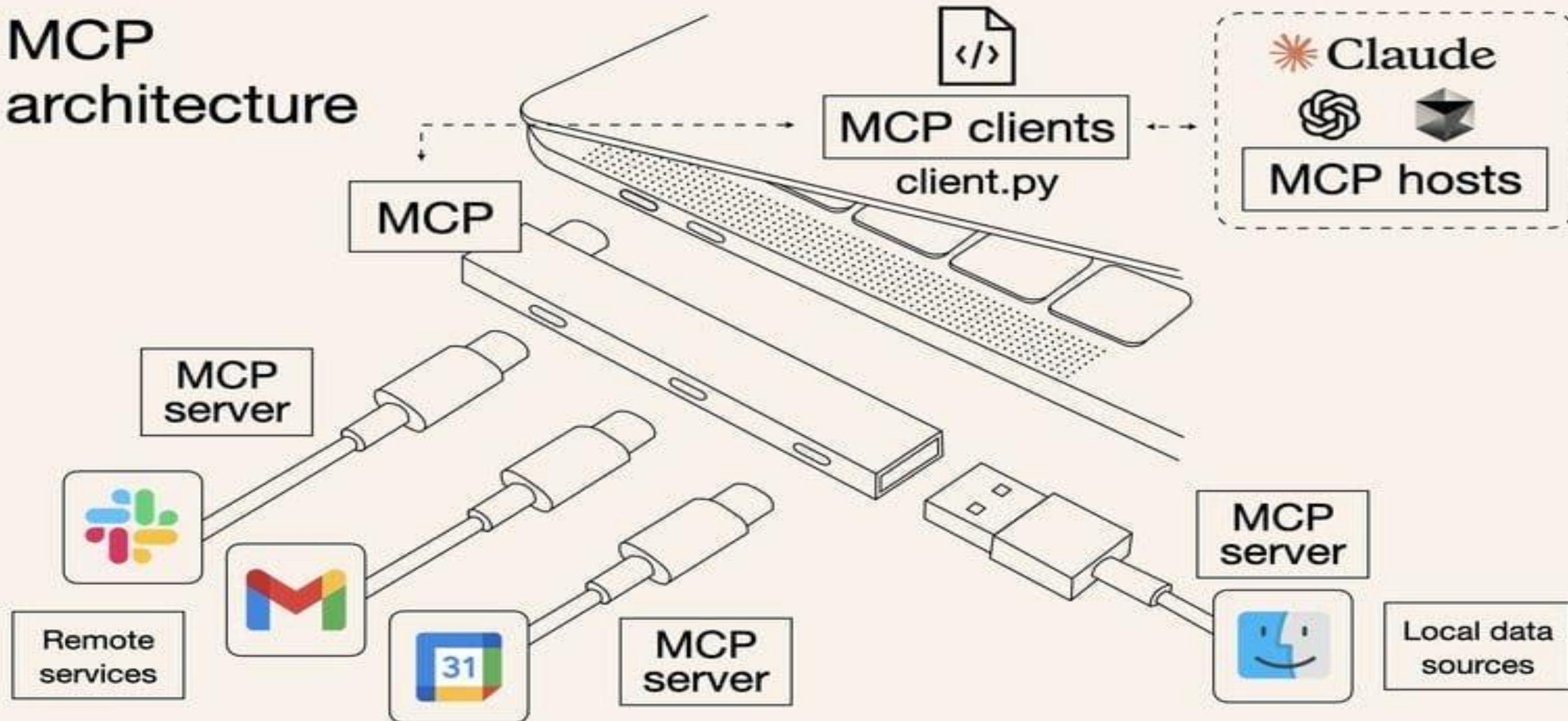
- Necesidad de no repetir errores como en desarrollo
- Necesidad de mejorar los contextos de trabajo de los LLMs
- Necesidad de disponer un protocolo de comunicaciones para que todos lo hagan de la misma manera

Funcionamiento de MCP

MCP para GitHub Copilot



MCP architecture



¿Qué es MCP para GitHub Copilot?

MCP para GitHub Copilot



Capacidad del **asistente de código** de usar la **tecnología MCP** que facilita **disponer** de **mejoras** muy importantes en el entorno de trabajo : más inteligencia, mayor número de funcionalidades , etc.

Importante: Supone un nuevo cambio de paradigma a los desarrolladores

Características:

- *Proporcionar mayor contexto a la herramienta -> Contexto adicional*
- *Acceso a información disponible en base de datos, APIs externas, sobre documentación, Wikis, etc.*
- *Integración con diversas herramientas*

Objetivo: Mejorar la eficiencia de los desarrolladores

Ejemplo: Es como hacer que nuestro “becario junior” se convierta en un superhéroe

Requerimientos

MCP para GitHub Copilot



VS Code
(últimas
versiones)

GitHub
Copilot

Soporte a
lenguajes y
paquetes

Docker
(optional)

- Python para paquetes PIP
- Node.js para paquetes NPX
- TypeScript para paquetes TS

Ejemplo de repositorio de Servidores MCP “Populares”

MCP para GitHub Copilot



<https://github.com/modelcontextprotocol/servers>

The screenshot shows the GitHub repository page for "Model Context Protocol servers". The page includes a README section with links to various MCP SDKs and reference servers like Everything, Fetch, Filesystem, Git, Memory, and SequentialThinking. It also features a "Languages" chart showing TypeScript as the primary language.

Model Context Protocol servers

This repository is a collection of *reference implementations* for the [Model Context Protocol](#) (MCP), as well as references to community built servers and additional resources.

The servers in this repository showcase the versatility and extensibility of MCP, demonstrating how it can be used to give Large Language Models (LLMs) secure, controlled access to tools and data sources. Typically, each MCP server is implemented with an MCP SDK:

- [C# MCP SDK](#)
- [Java MCP SDK](#)
- [Kotlin MCP SDK](#)
- [Python MCP SDK](#)
- [TypeScript MCP SDK](#)

Note: Lists in this README are maintained in alphabetical order to minimize merge conflicts when adding new items.

★ Reference Servers

These servers aim to demonstrate MCP features and the official SDKs.

- [Everything](#) - Reference / test server with prompts, resources, and tools
- [Fetch](#) - Web content fetching and conversion for efficient LLM usage
- [Filesystem](#) - Secure file operations with configurable access controls
- [Git](#) - Tools to read, search, and manipulate Git repositories
- [Memory](#) - Knowledge graph-based persistent memory system
- [SequentialThinking](#) - Dynamic and reflective problem solving through thought sequences

+ 583 contributors

Languages

Language	Percentage
TypeScript	41.7%
JavaScript	28.7%
Python	26.9%
Dockerfile	2.7%

Documentación

MCP para GitHub Copilot



<https://docs.github.com/es/enterprise-cloud@latest/copilot/how-tos/context/model-context-protocol/extending-copilot-chat-with-mcp>

The screenshot shows a dark-themed web browser window displaying the GitHub Documentation for the Enterprise Cloud version. The left sidebar is titled "GitHub Copilot" and includes sections like "Get started", "Concepts", "How-tos", "Configurar", "Completions", "Chat", "Agents", "Modelos de IA", "Context", "Espacios de Copilot", and "Uso de MCP". The "Extensión de Copilot Chat con MCP" section is currently selected and highlighted with a blue border. The main content area has a breadcrumb navigation path: "GitHub Copilot / How-tos / Context / Uso de MCP /". The title of the article is "Extensión de Copilot Chat con el protocolo de contexto de modelo (MCP)". A sub-section header "Información general" is present. On the right side, there is a sidebar titled "En este artículo" with links to "Información general", "Requisitos previos", "Configuración de servidores MCP en Eclipse", "Creación de un servidor MCP", and "Información adicional".

Extensión de Copilot Chat con el protocolo de contexto de modelo (MCP)

Aprende a usar el Protocolo de contexto de modelo (MCP) para ampliar Copilot Chat.

Eclipse JetBrains IDEs Visual Studio Code Xcode

Nota:

- La compatibilidad con MCP se encuentra actualmente en versión preliminar pública y está sujeta a cambios.
- La compatibilidad con MCP está disponible en Copilot Chat para Visual Studio Code, JetBrains, Eclipse y Xcode.
- Los [Términos de licencia de la versión preliminar de GitHub](#) se aplican a tu uso de este producto.

Información general

The Model Context Protocol (MCP) is an open standard that defines how applications share context with large language models (LLMs). MCP provides a standardized way to connect AI models to different data sources and tools, enabling them to work together more effectively.

MCP Toolkit de Docker

MCP para GitHub Copilot



A partir de

- Docker Desktop 4.38.0 se incluye “Ask Gordon” (El asistente IA diseñado por Docker)
- Docker Desktop 4.41.0 se incluye la opción “MCP Toolkit”
- Docker Desktop 4.42.0 se incluye el comando : docker mcp -help

A screenshot of the Docker Desktop application window. The title bar says "docker desktop PERSONAL". The left sidebar has a "MCP Toolkit" section highlighted with a "BETA" badge. Other sections include "Containers", "Images", "Volumes", "Builds", "Docker Hub", "Docker Scout", and "Extensions". The main content area is titled "MCP Toolkit" with a "Give feedback" link. It shows "Servers (0)", "Catalog (134)", "Clients", and "OAuth". A central message says "No MCP servers" with a "Add MCP servers" button. At the bottom, there's a "Highlighted" section with cards for "Atlassian", "Docker Hub", and "DuckDuckGo", each with a "+" button. The footer shows "Engine running", resource usage (RAM 1.06 GB, CPU 0.38%), disk usage (Disk: 20.49 GB used / limit 58.37 GB), a terminal icon, and an "Update" button.

Catalogo de MCP de Docker

MCP para GitHub Copilot



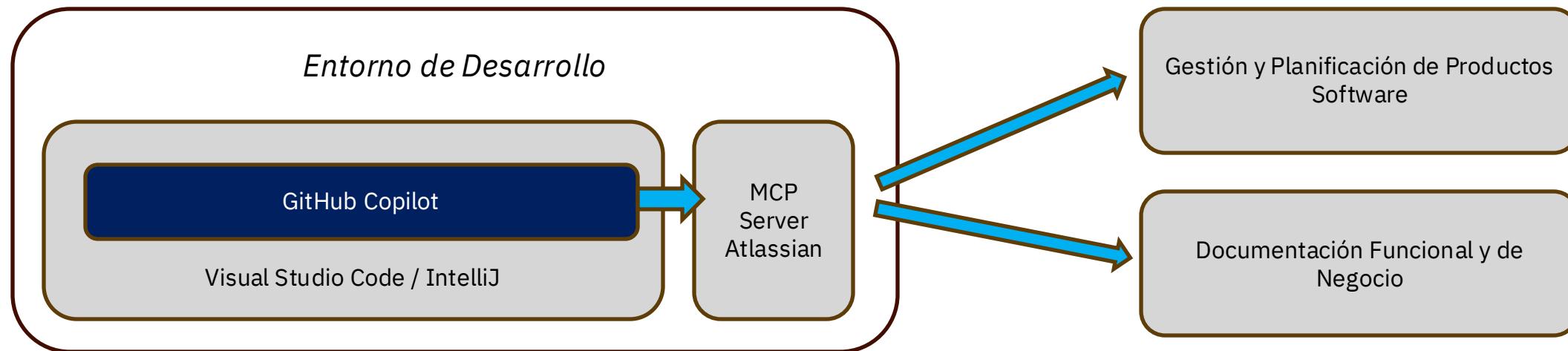
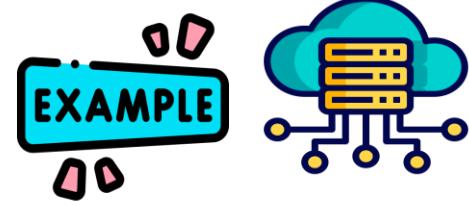
<https://hub.docker.com/mcp>

The screenshot shows a dark-themed web browser window displaying the MCP hub website at hub.docker.com/mcp. The page features a large, blurred background image with a gradient from purple to teal. In the center, the text "Access the largest library of secure, containerized MCP servers" is displayed in white. At the top, there is a navigation bar with the "mcp hub" logo, an "Explore" link, and buttons for "Contribute", "Sign In", and a user icon. Below the main title, there is a search bar with the placeholder "Search by MCP name or use case". Under the heading, the word "Use cases" is visible, followed by a grid of twelve cards, each representing a different use case with an icon and a brief description:

- Create a new Jira issue
- Get recent messages from a channel
- List all subscriptions in Stripe
- Create a coupon in Stripe
- Retrieve the transcript of a YouTube video
- Commit changes to a repository
- List issues in a GitHub repository
- Create a new issue in a GitHub repository
- Run a SQL query
- Search DuckDuckGo
- List all blobs in a Storage container
- Create a new collection in a database
- Perform an Elasticsearch search
- Search Loki logs for elevated error patterns
- Append content to a file in a vault

Ejemplo de MCP Atlassian “Básico”

MCP para GitHub Copilot



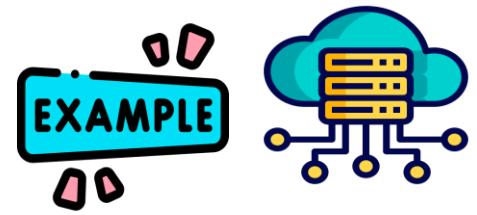
Jira Software



Confluence

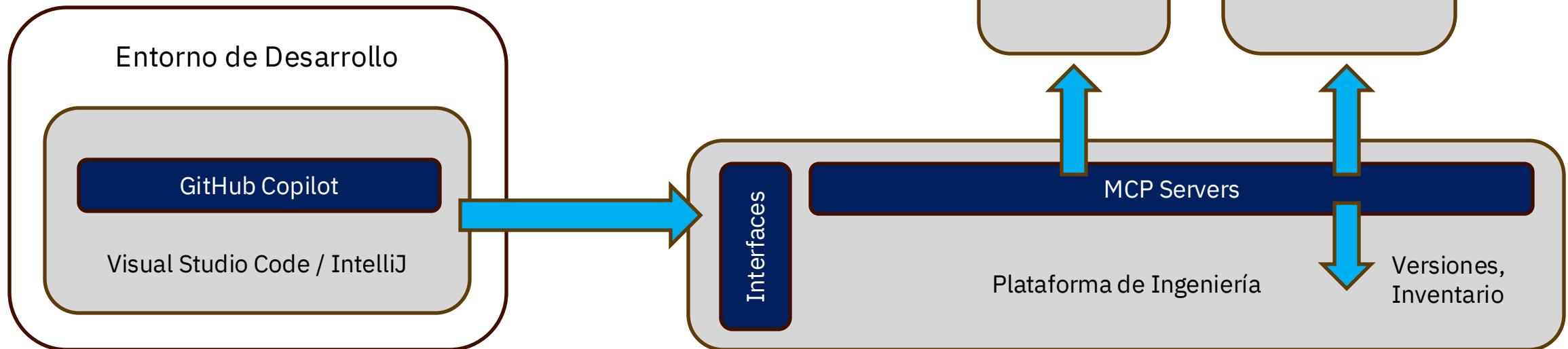
Ejemplo de MCP Atlassian “Avanzado”

MCP para GitHub Copilot



Retos a nivel empresarial

- Gobierno de MCPs
- Impersonación - Credenciales



Impacto en la Industria

MCP para GitHub Copilot



Interoperabilidad

Extensibilidad

Personalización
del contexto

Automatización
de tareas
complejas

Acceso a fuentes
de datos Externas

Acceso a
herramientas
desde el Chat

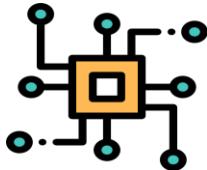
Curva de
aprendizaje inicial

04 Agente de Codificación

01 Funcionalidades
básicas

01.04

¿Qué es el agente de codificación Copilot?



Agente de codificación

Capacidad avanzada del asistente de IA para facilitar la **incorporación** y el **uso** de una **IA autónoma** que pueda realizar cambios en el código automáticamente, realizar tareas complejas, trabajar en incidencias de GitHub y crear pull request

Versión previa pública (Public Preview)

Características:

- *Basada en IA multimodal y con contexto ampliado*
- *Compresión de contexto del proyecto completo*
- *Uso del protocolo MCP*
- *Navegación del código*
- *Crear/modificar múltiples archivos*
- *Sugerencias con capacidad de ejecución tras aprobación*
- *Etc,*

Objetivo: mejorar la experiencia del desarrollador (automatizar tareas, ejecutar flujos de codificación asistida, ahorrar tiempo al desarrollador, colaboración humana – IA, ejecutar tareas completas de codificación, etc.)

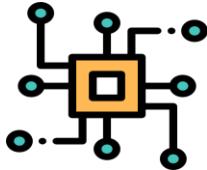
Ejemplo: Es como tener un desarrollador junior que entiendo lo que quieras hacer, propone una solución completa y te pregunta si puede implementarla paso a paso

¿Cómo se usa?

- Asignar una incidencia de GitHub a "GitHub Copilot" (como asignarla a un compañero de equipo).
- O usar los comandos @github en el chat de Copilot.
- El agente analiza la incidencia, crea un plan e implementa los cambios.
- Crea un borrador de PR con explicaciones detalladas de los cambios.

Copilot Edits

Agente de codificación

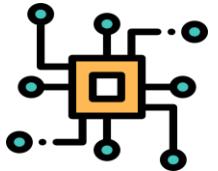


Capacidad del asistente de IA para facilitar el cambio de modelos IA dentro de la herramienta

Modo	Nivel de Control	IDE Support	Puntos fuertes	Entrada de usuario requerida
Edit	Alto (eliges los archivos)	VS Code, JetBrains	Cambios específicos y refactorización	Los archivos a modificar
Agent	Bajo (la IA decide)	VS Code, Visual Studio	Tareas complejas, cambios de arquitectura	Describir el objetivo

Documentación del modo Agente

Agente de codificación



- <https://code.visualstudio.com/docs/copilot/chat/chat-agent-mode>

The screenshot shows a dark-themed web browser window for `code.visualstudio.com`. The URL bar displays the site's address. The main content area features a large heading "Use agent mode in VS Code". To the left is a sidebar with navigation links for various VS Code features like Overview, SETUP, GET STARTED, CONFIGURE, EDIT CODE, BUILD, DEBUG, TEST, SOURCE CONTROL, TERMINAL, and GITHUB COPILOT. Under GITHUB COPILOT, the "Agent Mode" link is highlighted in blue. The main content includes sections for "Prerequisites" (with a bulleted list) and "Why use agent mode?" (with a detailed paragraph and a bulleted list). On the right side, there's a sidebar titled "IN THIS ARTICLE" containing a list of topics such as Prerequisites, Why use agent mode?, Enable agent mode in VS Code, Use agent mode, Agent mode tools, Manage tool approvals, Accept or discard edits, Revert edits, Interrupt an agent mode request, Use instructions to get AI edits that follow your coding style, Settings, Frequently asked questions, and Related resources. At the bottom right, there are links for RSS Feed, Ask questions, Follow @code, Request features, Report issues, and Watch videos.

Try [MCP servers](#) to extend agent mode in VS Code!

Use agent mode in VS Code

With chat *agent mode* in Visual Studio Code, you can use natural language to specify a high-level task, and let AI autonomously reason about the request, plan the work needed, and apply the changes to your codebase. Agent mode uses a combination of code editing and tool invocation to accomplish the task you specified. As it processes your request, it monitors the outcome of edits and tools, and iterates to resolve any issues that arise.

Prerequisites

- Install the latest version of [Visual Studio Code](#)
- Access to [Copilot](#). [Copilot Free plan](#) and get a monthly limit of completions and chat interactions.

Why use agent mode?

Agent mode is optimized for making autonomous edits across multiple files in your project. It is particularly useful for complex tasks that require not only code edits but also the invocation of tools and terminal commands. You can use agent mode to:

- Refactor parts of your codebase, such as "refactor the app to use a Redis cache".
- Plan and implement new features, such as "add a login form to the app using OAuth for authentication".

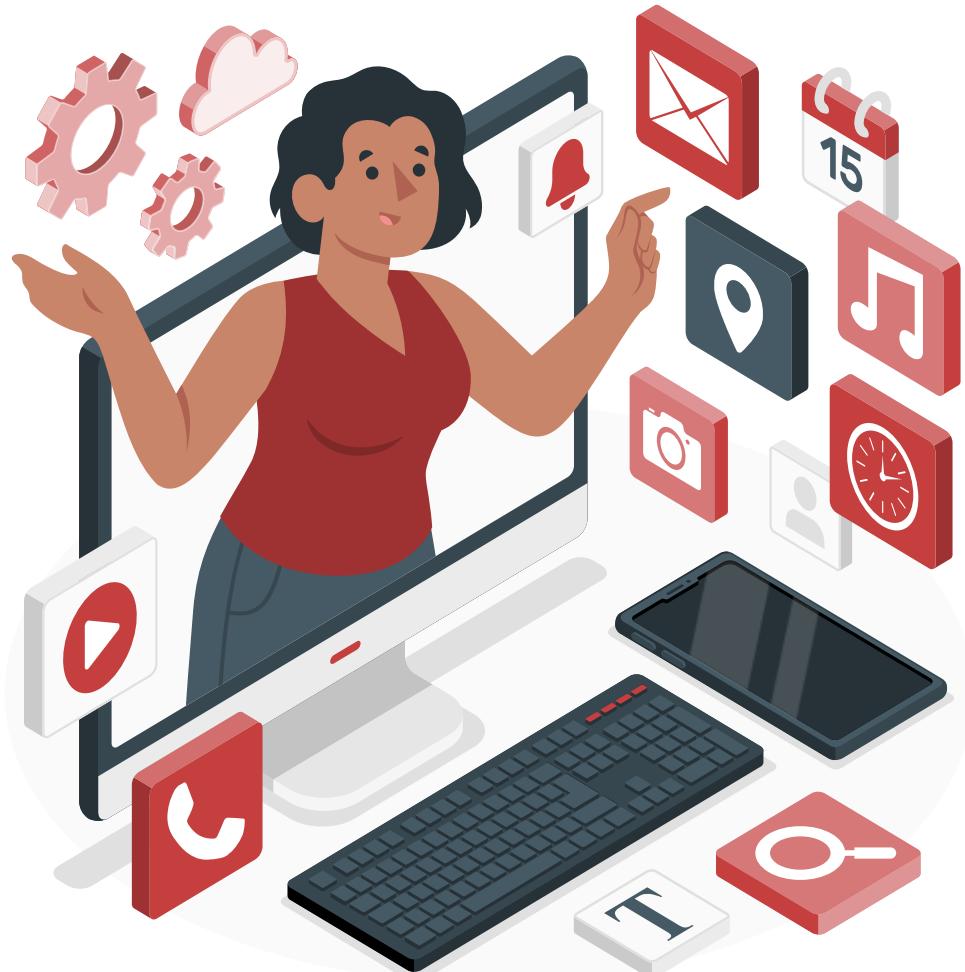
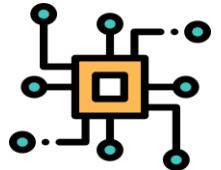
IN THIS ARTICLE

Prerequisites
Why use agent mode?
Enable agent mode in VS Code
Use agent mode
Agent mode tools
Manage tool approvals
Accept or discard edits
Revert edits
Interrupt an agent mode request
Use instructions to get AI edits that follow your coding style
Settings
Frequently asked questions
Related resources

RSS Feed
Ask questions
Follow @code
Request features
Report issues
Watch videos

Extensión de funcionalidad

Agente de codificación



Herramientas Built-in

Herramientas MCP

Herramientas proporcionadas
por extensiones

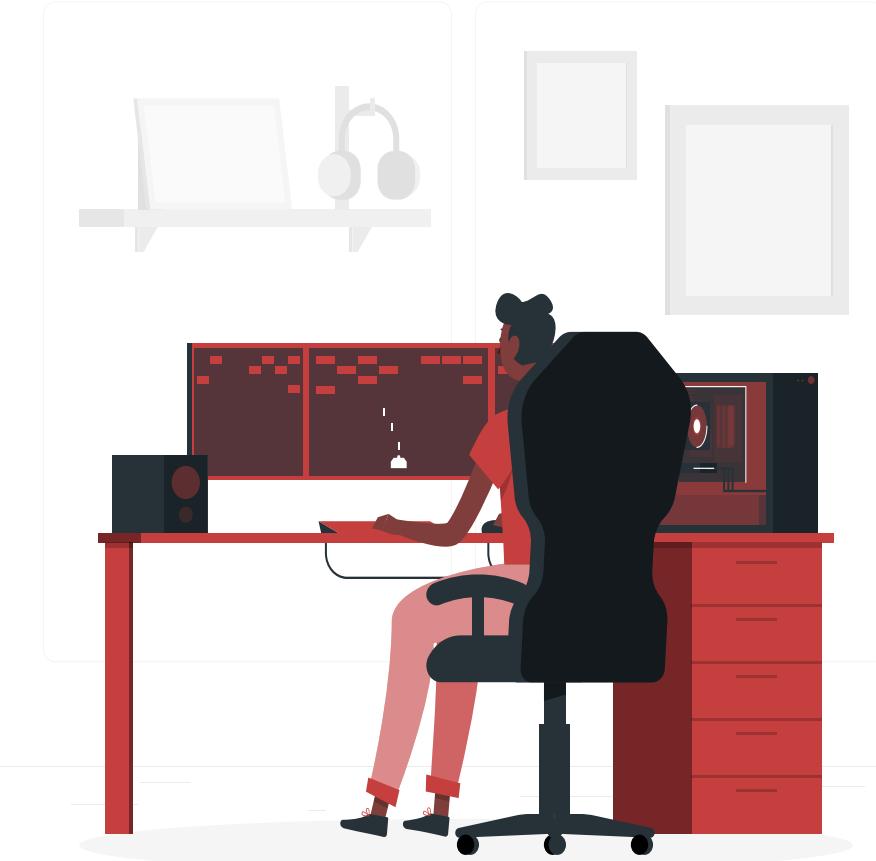
02 Trucos

02

Nomenclatura legible para una IA



Trucos



Mejor usar nombres específicos que genéricos

Mejor usar nombres con detalles

Mejor usar nombre con tipo

...



Crear guías de convenciones

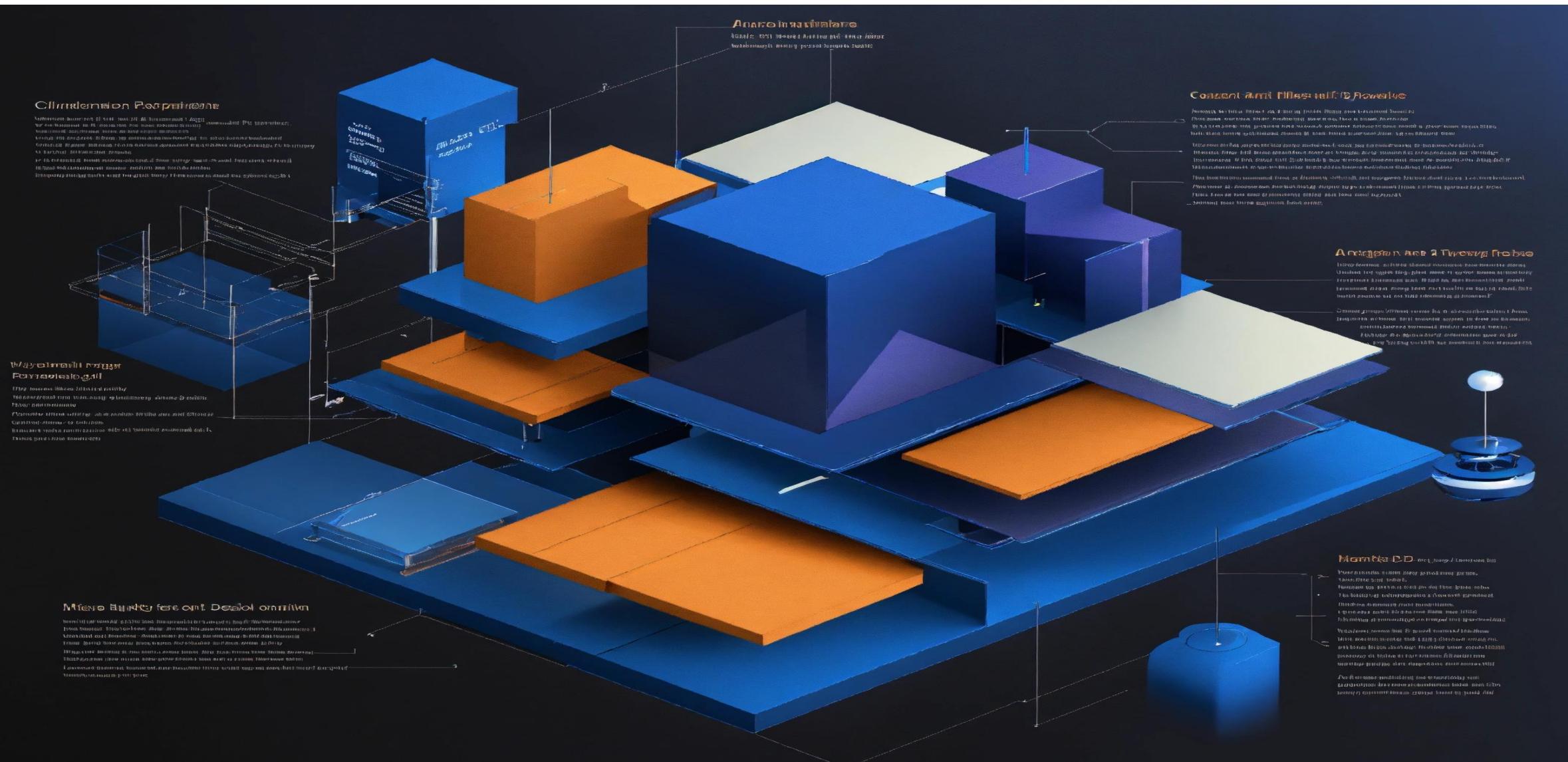
Trucos





High-Level Architecture First

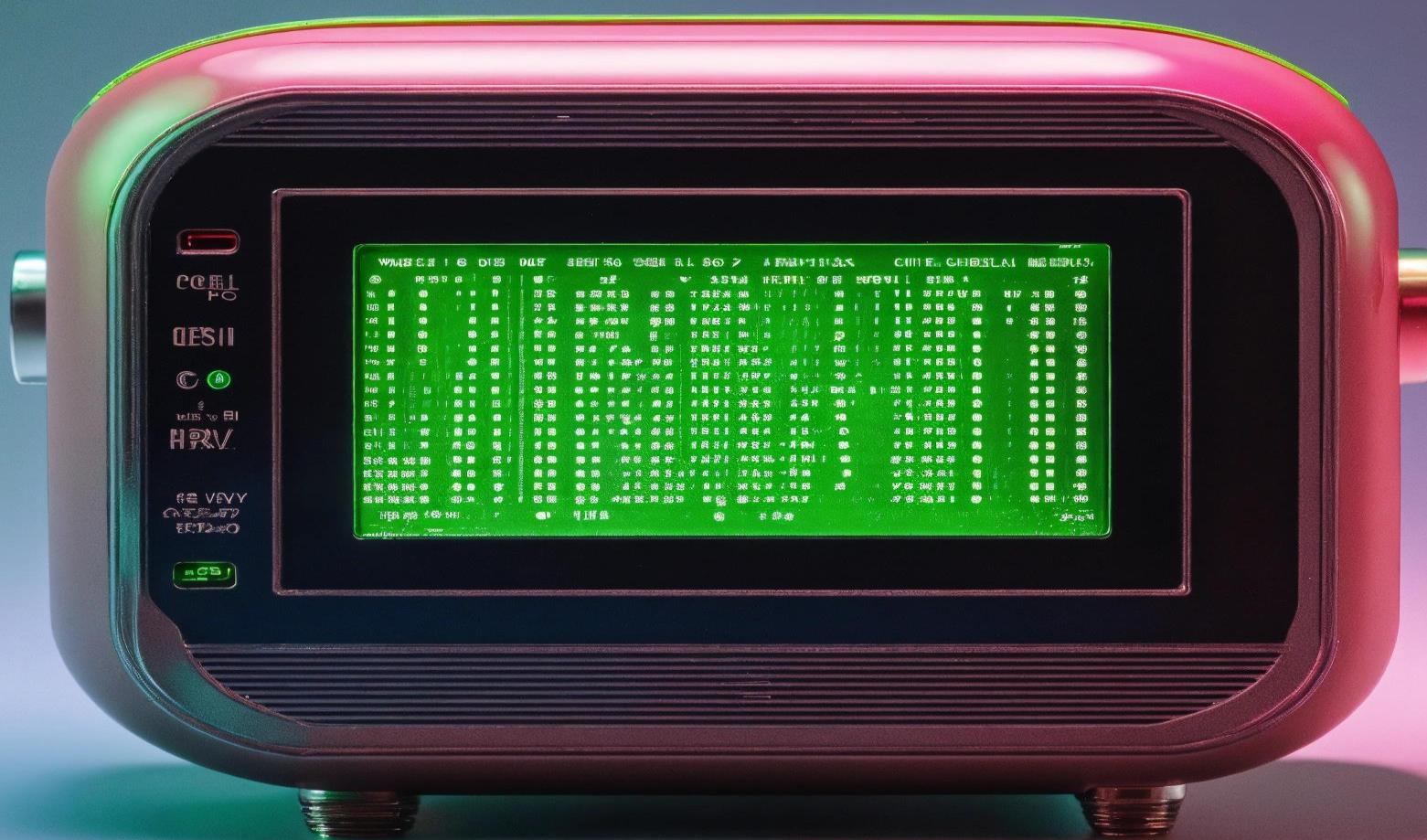
Trucos





Trabajar con fragmentos de código

Trucos



Editar e iteración

Trucos



Las sugerencias de código aceptadas se pueden seguir editando

Afrontar lo desconocido

Trucos



Uso como herramienta de autoaprendizaje para programadores

Fijar los archivos que necesitas

Trucos



The screenshot shows a VS Code interface with the following details:

- Title Bar:** Untitled (Workspace)
- Explorer:** Shows a tree view of the workspace structure, including a folder named "workspace-github-copilot" which contains "examples", "exercises", and "01-completion-code". Inside "01-completion-code", there are four files: "exerc-01-01-code-completion.md", "exerc-01-02-shortcuts.md", "exerc-01-03-com... 9+, M", and "exerc-01-04-comment-to-...".
- Editor:** The main editor area displays a Markdown file with the following content:

```
1 # Ejercicio: Finalización de código
2 ## Requisitos previos
3 - Cuenta de GitHub con acceso a Copilot
4 - VS Code con extensión de GitHub Copilot
5
6 ## Estimación de tiempo
7 Tiempo estimado en complementarse: 5-10 minutos
8
9 ## Instrucciones
10 Completar la función calculateSum(a, b) utilizando las sugerencias de GitHub Copilot
11
12 Definiremos la estructura de una función determinada y veremos la sugerencia propuesta
13
14 ````bash
15 function calculateSum(var1, var2) {
16     // Enter your code here
17 }
18 ````
```

Line 28: Según el lenguaje de programación, especificar el tipo de datos de los parámetros y el tipo de retorno de la
- Bottom Status Bar:** Shows file names (main*, exerc-01-01-code-completion.md, exerc-01-03-comment-to-code.md), line numbers (Ln 28, Col 179), and other status information.

Directorio de contexto

Trucos



Repository utilizado para **recopilar** el **contexto** de **todo** el **código base** para facilitar a GitHub Copilot el **acceso** a un contexto preciso **durante** el **desarrollo**

Estrategia:

1. *Crear un directorio de contexto: Directorio que contiene todos los archivos necesarios para que GitHub Copilot recuerde el contexto y/o reglas auxiliares. Se aconsejan que sean del mismo lenguaje*
2. *Cerrar todos los desarrollos que no están relacionados con lo que se está desarrollando actualmente*
3. *Mantener abiertos los archivos relacionados con lo que se está desarrollando actualmente*
4. *(Opcional) Añadir este directorio a .gitignore*

Problema de contexto

- Copilot al no tener acceso al contexto completo del código base realizará sugerencias inexactas -> pero calidad, mayor tiempo a realizar ajustes, etc.
- Copilot requiere mantener los archivos relacionados en pestañas adyacentes -> incremento del tiempo en la búsqueda y consulta de archivos
- Copilot puede perder la coherencia de código por la falta de contexto
- Problema del límite del contexto



Técnica: Quick Q&A



Trucos

Técnica de **chateo rápido** con GitHub Copilot que permite **interactuar** rápidamente **desde el editor** para obtener una respuesta breve

Aprovecha la estrategia de usar los comentarios para interactuar



Propuesta 1

```
# me: How can I protect this endpoint?  
# copilot:
```



Propuesta 2

```
# q: How can I protect this endpoint?  
# a:
```



Propuesta 3

```
# Roles: copilot  
# Expert in Python with 15+ years of experience  
# Role: me  
# Mid-level engineer  
#  
# me: How can I protect this endpoint?  
# copilot:
```

knowmad mood



Spain · Portugal · Italy · United Kingdom · United States · Uruguay. Morocco

Email: example@knowmadmood.com
www.knowmadmood.com