Guía de Características de GitHub Copilot

📋 Conceptos Principales

1. Chat Instructions (Instrucciones de Chat)

Definición: Directrices que definen **CÓMO** debe realizarse el trabajo. Se aplican automáticamente a todas las conversaciones del workspace.

Características:

- ✅ Aplicación automática e invisible
- ✅ Alcance: Todo el workspace o subcarpetas específicas
- ✅ Propósito: Establecer reglas y estándares de desarrollo
- ✅ Permanente: No necesitas invocarlas manualmente

Cuándo usar:

- Estándares de código del equipo
- Convenciones de nomenclatura
- Patrones arquitectónicos a seguir
- Reglas de documentación

---

2. Prompt Files (Archivos de Prompt)

Definición: Plantillas reutilizables que definen **QUÉ** debe hacerse para tareas específicas. Se invocan bajo demanda.

Características:

- ✅ Activación manual mediante comandos `/nombre`
- ✅ Alcance: Tarea específica
- ✅ Propósito: Automatizar tareas repetitivas
- ✅ Reutilizable: Compartible entre proyectos

Cuándo usar:

- Tareas repetitivas específicas
- Generación de código con plantilla

- Revisiones de código especializadas
- Comandos personalizados del equipo

📁 Estructura de Directorios

```
mi-proyecto/
├── .github/
│   ├── copilot-instructions.md          # Instrucciones globales del proyecto
│   ├── instructions/                    # Instrucciones por área/tecnología
│   │   ├── backend-dotnet.instructions.md
│   │   ├── frontend-angular.instructions.md
│   │   └── database.instructions.md
│   └── prompts/                         # Archivos de prompts reutilizables
│       ├── review-api.prompt.md
│       ├── generate-service.prompt.md
│       └── create-test.prompt.md
├── src/
└── README.md
```

💡 Ejemplos Prácticos

Ejemplo 1: Chat Instructions - Archivo Único

Ubicación: `.github/copilot-instructions.md`

Contenido:

```
markdown
```

```
# Instrucciones de Desarrollo del Proyecto

## Backend (.NET)

**Arquitectura:**
- Aplicar principios SOLID
- Separación de capas: Controllers → Services → Repositories
- Usar inyección de dependencias para todo

**Código:**
- DTOs para todas las transferencias de datos
- Validaciones con FluentValidation
- Comentarios XML en miembros públicos
- async/await para operaciones I/O

**Logging y Errores:**
- ILogger con contexto estructurado
- ProblemDetails para respuestas de error
- Nunca exponer stack traces en producción

**Testing:**
- Código testeable con interfaces
- Configuración externalizada (appsettings.json)

## Frontend (React)

- Componentes funcionales con hooks
- TypeScript estricto (strict mode)
- Styled-components para estilos
- React Query para estado del servidor
- Nomenclatura: PascalCase para componentes, camelCase para funciones

## Base de Datos

- Migraciones versionadas con nombre descriptivo
- Índices en columnas de búsqueda frecuente
- Constraints y relaciones siempre explícitas
- Nunca usar SELECT * en queries
```

Ejemplo 2: Chat Instructions - Múltiples Archivos

Ubicación: `.github/instructions/backend-dotnet.instructions.md`

Contenido:

markdown

Instrucciones Backend .NET

Estructura de Proyecto

Todos los proyectos backend deben seguir esta estructura:
- API/ (Controllers, Middleware)
- Application/ (Services, DTOs, Validators)
- Domain/ (Entities, Interfaces)
- Infrastructure/ (Repositories, DbContext)

Reglas de Código

1. ****Controllers:****
   - Solo orquestación, sin lógica de negocio
   - Retornar ActionResult<T>
   - Atributos de r