GitHub Copilot Chat Cookbook

This comprehensive cookbook provides practical examples and patterns for maximizing your productivity with GitHub Copilot Chat. Each scenario includes optimized prompts and context usage for best results.

Core Chat Patterns

Scenario	Example Prompt	Context Tips
Debugging Errors	"I'm encountering an 'Invalid JSON' error in my application. Can you analyze the code and suggest fixes?"	Use `#selection` or `#file` to provide error context
Handling API Rate Limits	"My application is hitting API rate limits. What strategies can I implement with exponential backoff and queuing?"	Include `#file` with your API client code
Exploring Feature Implementations	"How can I implement a dark mode feature in my React application with TypeScript and Tailwind CSS?"	Use `#codebase` for existing theme structure
Incorporating User Feedback	"Users requested search functionality. Help me design a scalable search with debouncing and pagination."	Provide `#file` with current component structure
Improving Code Readability	"Can you refactor this function to enhance readability while maintaining performance?"	Always use `#selection` for targeted refactoring
Resolving Lint Errors	"I'm getting ESLint errors. Please analyze and fix them while explaining the violations."	Include `#terminalLastCommand` for error output
Optimizing Performance	"How can I optimize this database query? Consider indexing, caching, and query structure."	Use `#selection` with the specific query

Scenario	Example Prompt	Context Tips
Applying Design Patterns	"Implement the Observer pattern for this event system using modern TypeScript features."	Provide `#file` with existing class structure
Designing Data Access Layers	"Structure a clean data access layer with repository pattern and dependency injection."	Use `#codebase` for existing architecture
Decoupling Business Logic	"Separate business logic from this controller using clean architecture principles."	Include `#file` with current implementation

Advanced Chat Scenarios

Scenario	Example Prompt	Best Practices
Cross-Language Integration	"Generate TypeScript types from this Python Pydantic model for consistent API contracts."	Use `#file` for both source and target files
Migration Strategies	"Help me migrate from REST to GraphQL while maintaining backward compatibility."	Provide `#codebase` for current API structure
Security Code Review	"/doc Review this authentication code for OWASP Top 10 vulnerabilities and suggest improvements."	Use `#selection` with security-sensitive code
Performance Profiling	"Analyze this algorithm's time complexity and suggest optimizations for large datasets."	Include `#selection` with algorithm implementation
Accessibility Enhancement	"Make this React component WCAG 2.1 AA compliant with proper ARIA attributes and keyboard navigation."	Use `#file` with existing component
CI/CD Pipeline Optimization	"Optimize this GitHub Actions workflow for faster builds and better caching strategies."	Provide `#file` with current workflow
Database Schema Evolution	"Design a safe database migration strategy for this schema change in production."	Include `#file` with current and target schemas
Microservices Communication		Use `#codebase` for service architecture

Scenario	Example Prompt	Best Practices
	"Implement resilient service-to-service communication with circuit breakers and retries."	

Modern Development Workflows

Scenario	Example Prompt	Slash Commands
Test-Driven Development	"/tests Generate comprehensive unit tests for this service class with edge cases and mocks."	Use `/tests` with `#selection`
Documentation Generation	"/doc Create API documentation for this REST endpoint with OpenAPI 3.0 specification."	Use `/doc` with `#file`
Code Architecture Review	"/review Analyze this module's architecture and suggest improvements for maintainability."	Use `/review` with `#codebase`
Dependency Updates	"Help me update this package.json safely, checking for breaking changes and compatibility."	Use `#file` with package files
Environment Configuration	"Set up Docker containerization for this Node.js app with multi-stage builds and security best practices."	Provide `#codebase` context
Error Handling Patterns	"Implement comprehensive error handling with custom exceptions and proper logging."	Use `#selection` with error-prone code
Code Quality Automation	"Set up pre-commit hooks with formatting, linting, and testing for this TypeScript project."	Include `#file` with current tooling
Performance Monitoring	"Add observability to this service with metrics, tracing, and structured logging."	Use `#file` with service implementation

Context-Aware Prompting

Using Hash Context Effectively

```
# Best practices for context usage:

1. **#selection** - For targeted code analysis and refactoring
2. **#file** - When working with entire file context
3. **#codebase** - For architecture-level decisions
4. **#terminal** - When debugging command-line issues
5. **#editor** - For understanding current workspace state
```

Workspace Integration Patterns

Use Case	Context Strategy
New Feature Development	Use `#codebase` + specific `#file` references for existing patterns
Bug Investigation	Combine `#terminalLastCommand` + `#selection` for error context
Code Review Preparation	Use `/review` with `#file` or `#selection` for targeted analysis
Refactoring Sessions	Start with `#codebase` overview, then `#selection` for specific areas
API Integration	Use `#file` with existing service patterns + external documentation

Advanced Prompting Techniques

Multi-Step Conversations

```
1. **Context Setting**: "I'm working on a React e-commerce app with
TypeScript. #codebase"
2. **Specific Request**: "Help me implement cart functionality with local
storage persistence. #file:components/Cart.tsx"
3. **Follow-up**: "Now add optimistic updates for better UX when items are
added/removed."
```

```
4. **Testing**: "/tests Generate tests for the cart functionality we just
created."
```

Iterative Improvement

```
1. **Initial Implementation**: "Create a basic user authentication system"
2. **Security Enhancement**: "Add JWT refresh token rotation and secure
cookie handling"
3. **Error Handling**: "Implement comprehensive error handling with user-
friendly messages"
4. **Testing**: "Generate integration tests for the complete auth flow"
```

Language-Specific Patterns

TypeScript/JavaScript

- Emphasize type safety and modern ES features
- Request proper error handling with typed exceptions
- Ask for performance considerations (bundle size, runtime)

Python

- Focus on Pythonic patterns and PEP compliance
- Request proper virtual environment and dependency management
- Emphasize testing with pytest and type hints

Java

- Request modern Java features (records, sealed classes, pattern matching)
- Focus on Spring Boot best practices and dependency injection
- Emphasize proper exception handling and logging

Go

- Request idiomatic Go patterns and error handling
- Focus on concurrency patterns and goroutine management
- Emphasize proper package structure and interfaces

Troubleshooting Common Issues

Problem	Solution
Generic Responses	Provide more specific context with `#file` or `#selection`
Outdated Suggestions	Specify current versions: "using React 18 with concurrent features"
Missing Dependencies	Include `#file:package.json` or requirements file in context
Inconsistent Code Style	Reference your style guide: "following Airbnb ESLint config"
Poor Error Messages	Use `#terminalLastCommand` to show actual error output

Remember: The more specific and contextual your prompts, the better Copilot Chat can assist you!