Documentation Generation with GitHub Copilot

Generate comprehensive documentation using AI-assisted patterns and context-aware suggestions.

🎯 Overview

GitHub Copilot excels at generating documentation by understanding code context and producing clear, consistent explanations. This guide covers patterns for generating various types of documentation.

📝 Inline Code Comments

Single-Line Comments

```
// Start typing a comment and let Copilot complete
// Calculate the total expense for a given category
public double calculateCategoryTotal(String category) {
    // Copilot will suggest implementation based on the comment
}
```

Multi-Line Documentation

Java (Javadoc):

```
/**
 * Type /** and press Enter above a method
 * Copilot generates complete Javadoc with:
 * - Description
 * - @param annotations
 * - @return description
 * - @throws exceptions
 */
public Expense createExpense(ExpenseDTO dto) throws ValidationException {
    // ...
}
```

Python (Docstrings):

```
def create_expense(self, dto: ExpenseDTO) -> Expense:
    """
    Type triple quotes and Copilot generates:
    - Description
    - Args section
    - Returns section
    - Raises section
    """
    pass
```

💬 Using Chat for Documentation

Generate Method Documentation

```
Prompt: /doc

Select a method and use /doc to generate documentation
```

Explain Complex Code

```
Prompt: /explain

Select code block → /explain
Copilot provides detailed explanation of logic
```

Generate README Content

```
Prompt: Generate a README.md for this project that includes:
- Project overview
- Installation instructions
- Usage examples
- API documentation
- Contributing guidelines

#codebase
```

📄 Documentation Patterns

Pattern 1: Class-Level Documentation

```
Prompt: Add comprehensive class documentation for
#file:ExpenseService.java
Include:
- Class purpose
- Dependencies
- Usage examples
- Thread safety notes
```

Pattern 2: API Documentation

```
Prompt: Generate OpenAPI/Swagger documentation for all endpoints in
#file:ExpenseController.java
```

Pattern 3: Architecture Documentation

```
Prompt: @workspace Create an architecture overview document explaining:
- Project structure
- Layer responsibilities
- Data flow
- Key design decisions
```

🔧 Practical Exercises

Exercise 1: Document a Service Class

1. Open an undocumented service class
2. Select the entire class
3. Use: `/doc`
4. Review and refine the generated documentation

Exercise 2: Generate README

1. Open Chat panel
2. Prompt: `@workspace Generate a comprehensive README.md for this project`
3. Copy output to README.md
4. Customize sections as needed

Exercise 3: Inline Comments

1. Write a complex method
2. Add comment `//` at each logical step
3. Let Copilot suggest explanatory comments
4. Accept or modify suggestions

✅ Best Practices

- **Be Specific**: Include what sections you want in documentation
- **Use Context**: Reference files with `#file` for accurate docs
- **Review Output**: Always verify generated documentation for accuracy
- **Maintain Style**: Use `/doc` consistently for uniform documentation
- **Update Regularly**: Regenerate docs when code changes significantly

---

🔗 Related Resources

- Slash Commands - `/doc`, `/explain` usage
- Hash Context Variables - `#file`, `#codebase` for context
- Custom Instructions - Define documentation style