02-copilot-chat-modes

Lab 2 - Copilot Chat

In this comprehensive lab, you'll explore the three main GitHub Copilot Chat modes: Ask mode, Edit mode, and Agent mode. You'll learn how each mode serves different purposes and their strengths and limitations.

- **Ask Mode**: Ideal for understanding code, generating ideas, and getting explanations. It excels at answering questions about your codebase and suggesting improvements without making direct changes.
- **Edit Mode**: Best for making small code changes, predecessor of Agent Mode. It allows you to directly modify code in your editor based on your prompts.
- **Agent Mode**: Designed for complex, multi-step tasks that require reasoning, planning, and the use of external tools. It can autonomously navigate your codebase, make changes, and validate them.

The goal of these exercises are to come in contact with different aspects of AI Augmented engineering.
You'll start with exploring and understanding code in Ask mode, then move to making some direct changes in edit mode and finally see how agent mode can excel in certain tasks and project wide changes.

Overview of Exercises

Ex 1-2: Ask Mode

Ex 3: Edit Mode

Ex 4-5-6: Agent Mode

Prerequisites

Pet Clinic Setup:
Go to github.com/VinkVdB/demo-aiaegh-petclinic and clone the repository.

Available Implementations:
Choose your preferred implementation

- 🟡 **Java** (Spring Boot) - `./java`
- 🟣 **.NET** (ASP.NET Core) - `./dotnet`

Steps to get started:

1. Open the corresponding `.code-workspace` file with VS Code
2. Use added run configurations or docker to run the application
3. Navigate to localhost:8080 (Java) or localhost:5172 (.NET)

Disclaimer: You can use your own project for a more realistic experience. However for code exploration I recommend taking a project you're not familiar with.

Exercise 1: Ask Mode - Explore the codebase with @workspace

Objective

In Ask mode, use the `@workspace` chat participant to understand the structure and architecture of your project.

Instructions

1. Start a Copilot Chat window.

2. Start a Copilot Chat window and ensure you're in Ask mode.

3. Use the `@workspace` chat participant to explore your project:
   - "What is the overall architecture of this project?"
   - "What technologies and frameworks are being used?"
   - "How is the project structured? What are the main modules/components?"
   - "Where is the main entry point of the application?"
4. Ask more specific questions about key components:
   - "What is covered in the tests?"

- "What is the data model structure?"
- "How are API endpoints organized?"

5. Ask Copilot to create an exploratory diagram (Mermaid or PlantUML):
   - "Generate a ... diagram showing the high-level architecture ..."
   - "Create a ... class diagram for the main entities ..."
   - "Show the data flow through the application as a ... flowchart"

Observe

- Are the answers contextually accurate?
- Would this help onboard a new team member faster?
- How does the quality of answers change when you're more specific vs. general in your questions?
- How accurate and useful is the generated diagram?

Exercise 2: Ask Mode - Use chat participants and slash commands

Objective

Learn to leverage specialized chat participants and slash commands for targeted assistance.

Instructions

1. Try `/help` to see available commands.
2. Experiment with different chat participants:
   - Use `@vscode` to ask about IDE configuration and settings
   - Use `@github` for repository and version control questions
3. Try various slash commands:
   - `/explain` on the `DatabaseInitializer` / `ApplicationDbInitializer` class
   - `/fix` on code with potential issues
   - `/tests` to generate unit tests for existing code
4. Use chat variables for specific context:
   - `#file` to reference the current file
   - `#selection` for selected code
   - `#function` for the current function

Observe

- How do different chat participants provide specialized knowledge?
- Which slash commands are most useful for your workflow?
- How does using chat variables improve the relevance of responses?

Exercise 3: Edit Mode: Add Gender to the Pet class

Edit mode enables direct code modifications across multiple files. Unlike Ask mode, it can actually change your code in place.

Objective

Use Edit mode to make structural changes to your codebase, including modifying data models and related code.

Instructions

1. Switch to Edit mode in Copilot Chat.

2. Provide context about what you want to change:
   - "I want to add a Gender field to the Pet class/entity"
   - Include the Pet class file in your context
   - Specify the type of gender field (enum, string, etc.)
3. Let Copilot propose the changes and apply them.
4. Ask for additional related changes:
   - "Update any DTOs or view models that use Pet"
   - "Add validation for the gender field"
   - "Update any database migration files if needed"
5. Update the tests:

- "Update all unit tests to include the new Gender field"
- "Add test cases for gender validation"
- "Update any integration tests that create Pet objects"

6. Run the tests and observe the results.
7. If tests fail, ask Edit mode to fix them:
   - "Fix the failing test in [test file name]"
   - "The test is failing because [describe the error]"

Observe

- How many files does Edit mode modify in a single operation?
- Does it understand the relationships between related classes?
- Does it maintain the coding style of the existing code?
- What happens when you provide more vs. less context?
- Does Edit mode correctly identify which tests need updating?
- Are the generated test cases comprehensive?
- How does it handle test data generation for the new field?
- What issues do you encounter?

Exercise 4: Agent Mode - Generate an exploratory Mermaid diagram (revisited)

Agent mode combines the capabilities of Ask and Edit modes with autonomous tool usage. It can plan, execute, and verify complex multi-step changes.

Objective

Compare how Agent mode approaches diagram generation differently from Ask mode.

Instructions

1. Switch to Agent mode

2. ask for the same diagram you generated in Lab 2:
   - "Generate an exploratory Mermaid diagram of this project's architecture"

Observe

- What tools does Agent mode use in its process?
- How does the approach differ from Ask mode?
- Is the output more accurate or detailed?
- What's different about the Agent mode experience?

Exercise 5: Add gender to the Pet class (revisited)

Objective

Compare Agent mode's approach to the same structural change you made in Edit mode.

Instructions

1. Reset your Pet class to its original state (or work on a different branch).
2. Ask Agent mode to add the gender field:
   - "Add a Gender field to the Pet class and update all related code and tests"
3. Let Agent mode work autonomously, observe its process.
4. Ask it to run tests and fix any issues it encounters.

Observe

- Which files does it reference?
- Does it validate its changes differently than Edit mode?
- How does the final result compare to your Edit mode implementation?
- Does Claude Sonnet 4 handle the multi-step process better?

Exercise 6: Agent Mode - Project-wide change

Objective

Use Agent mode for a complex, multi-step change that requires planning, exploration, and validation.

Options

Choose one of these project-wide changes (or design your own):

Option A: Add a new Veterinarian registration screen

- Create a new Veterinarian entity/model
- Implement CRUD operations
- Create UI components for registration
- Add navigation and routing
- Include appropriate validation and error handling

Option B: Add appointment scheduling functionality

- Create Appointment entity with Pet and Veterinarian relationships
- Implement scheduling logic
- Create UI for booking and viewing appointments
- Add calendar/date picker components

Option C: Test and validate the UI

- Let Agent mode manually test the UI
- Identify if every flow works as expected
- Create a report of issues found and suggest fixes

Option D: Custom

- Propose your own complex, multi-step change that affects multiple files and components

Instructions

1. Describe your chosen change to Agent mode in natural language. Don't copy paste the above instructions verbatim.
2. Ask it to create a plan before implementing.
3. Review and refine the plan if needed.
4. Let Agent mode implement the changes step by step.
5. Ask it to test the implementation and create a report.

Observe

- How does Agent mode break down complex requirements?
- What tools does it use for discovery, implementation, and validation?
- How does it handle dependencies between different parts of the change?
- Does it test its own work? How?
- What errors or issues does it encounter and how does it resolve them?

Deliverable

Add a section `Lab 2 — Copilot Chat` to your Markdown lab journal. For each exercise, include:

- Your evaluation:
  - What worked well?
  - What challenges did you face?
  - How do the modes compare for different tasks?
- A short conclusion of the value provided by each mode for the tasks performed
- Any interesting observations or insights you gained