GitHub Copilot Custom Chat Modes ⚡

This repository includes four specialized custom chat modes designed specifically for the NFCU GitHub Copilot training program. These modes complement the built-in chat modes with project-specific expertise and educational focus.

🎯 Understanding Custom Chat Modes

Custom Chat Modes allow you to:

- **Specialized Expertise**: Access domain-specific knowledge and patterns
- **Consistent Behavior**: Maintain consistent AI persona across conversations
- **Project Context**: Leverage project-specific conventions and practices
- **Educational Focus**: Provide structured learning experiences

VS Code Custom Chat Modes Overview

Custom chat modes in VS Code are defined using `.chatmode.md` files placed in the `.github/chatmodes/` directory. Each file defines:

- **Mode Name and Description**: Clear identification and purpose
- **Specialized Instructions**: Domain-specific behavior and knowledge
- **Response Patterns**: Consistent formatting and approach
- **Context Integration**: Project-specific patterns and conventions

Built-In vs Custom Modes

- **Built-In Modes** (Ask, Edit, Agent): General-purpose modes for common development tasks
- **Custom Modes**: Specialized assistants tailored to specific technologies, workflows, or learning objectives

> ***Note**: See Copilot Built-In Modes for detailed information about Ask, Edit, and Agent modes.*

📁 NFCU Training Project Chat Modes

This repository includes four specialized chat modes located in `.github/chatmodes/`:

1. SDLC Modes (`sdlc-modes.chatmode.md`) 🔄

Purpose: Software Development Lifecycle specialized assistance with phase-specific guidance

Key Features:

- Mode switching for different development phases (`/mode development` , `/mode review` , etc.)
- Context-aware responses based on SDLC stage
- Quality gates and best practices enforcement
- Educational explanations for each phase

Usage Examples:

```
@sdlc-modes /mode development - Create a REST controller for user
management
@sdlc-modes /mode review - Analyze this code for security
vulnerabilities
@sdlc-modes /mode production - Prepare this code for production deployment
@sdlc-modes /mode learning - Explain Spring Boot dependency injection
@sdlc-modes /mode rapid - Quick prototype for data validation
```

Example Interaction:

```
User: @sdlc-modes /mode development Create a user service with validation

Response:
🔧 **Development Mode Active**

I'll create a comprehensive UserService following Spring Boot best
practices:

[Provides complete service implementation with proper patterns]

This implementation demonstrates:
- Constructor-based dependency injection
- Proper transaction management
- Input validation with Bean Validation
- Custom exception handling
- DTO pattern for data transfer
```

2. Spring Boot Expert (`spring-boot-expert.chatmode.md`) 🍃

Purpose: Specialized Spring Boot development assistance for the task-manager project

Key Features:

- Spring Boot 3.2.3 + Java 21 expertise
- Project-specific configuration patterns (Maven, H2 database, Thymeleaf)
- Package convention awareness (`com.taskmanager.app`)
- Security implementation guidance (currently disabled for training)
- Testing strategies with Spring Boot Test

Usage Examples:

```
@spring-boot-expert Create a service layer for task management with
validation
@spring-boot-expert How do I implement JWT authentication in this project?
@spring-boot-expert Generate integration tests for the UserController
@spring-boot-expert Add caching to the user service using Spring Cache
```

Example Interaction:

```
User: @spring-boot-expert Create a TaskService with proper validation

Response:
🍃 **Spring Boot Expert**

I'll create a comprehensive TaskService following the project's Spring
Boot patterns:

[Provides complete service implementation with project-specific patterns]

**Key Spring Boot Features Used:**
- @Service with @Transactional for data consistency
- Constructor injection (Spring recommended approach)
- @Validated for method-level validation
- Custom exceptions for business logic
- DTO pattern for clean data transfer
```

3. Selenium Expert (`selenium-expert.chatmode.md`) 🧪

Purpose: UI test automation expertise for the ui-tests project

Key Features:

- Selenium WebDriver 4.27.0 + TestNG framework expertise
- WebDriverManager integration for automatic driver setup
- Page Object Model implementation patterns
- Cross-browser testing strategies
- Package convention awareness (`com.kavinschool.app`)

Usage Examples:

```
@selenium-expert Create a page object for the user registration form
@selenium-expert Generate TestNG test suite configuration for parallel
execution
@selenium-expert Implement data-driven testing with Excel integration
@selenium-expert Help me debug flaky tests and add proper waits
```

Example Interaction:

```
User: @selenium-expert Create a comprehensive page object for login
functionality

Response:
🧪 **Selenium Expert**

I'll create a robust LoginPage following Selenium best practices and
project patterns:

[Provides complete Page Object with proper locators, waits, and methods]

**Selenium Best Practices Implemented:**
- Page Object Model with PageFactory initialization
- Explicit waits for reliable test execution
- Fluent interface for method chaining
- Proper exception handling for element interactions
- Cross-browser compatible locator strategies
```

4. Training Instructor (`training-instructor.chatmode.md`) �🏫

Purpose: Educational assistance optimized for GitHub Copilot training delivery

Key Features:

- 1-day course structure awareness (Morning: Foundations & Backend, Afternoon: Context & Web Layer)
- Progressive learning approach from basic to advanced concepts
- Hands-on exercise creation and assessment
- Multi-language support (Java, JavaScript, Python)
- Course material integration and cross-references

Usage Examples:

```
@training-instructor Create a hands-on exercise for Copilot Chat
participants
@training-instructor Explain the difference between built-in and custom
chat modes
@training-instructor Generate assessment questions for Spring Boot with
Copilot
@training-instructor Design a progressive exercise series for GitHub
Copilot features
```

Example Interaction:

```
User: @training-instructor Create a Day 2 exercise for advanced Copilot
features

Response:
🧑‍🏫 **Training Instructor**

## Day 2 Advanced Exercise: Multi-Context Development

### Learning Objectives:
- Master @workspace, #file, and #selection context usage
- Apply custom chat modes effectively
- Implement complex features using Copilot assistance

### Exercise Structure:
[Provides structured learning activities with clear objectives and
assessments]

### Assessment Criteria:
- Effective context usage demonstration
- Code quality and best practices adherence
- Understanding of Copilot capabilities and limitations
```

🚀 Accessing Custom Chat Modes

Mode Activation

```
# Basic Usage Pattern
@[mode-name] [your question or request]

# Available Modes in This Project
@sdlc-modes - Software Development Lifecycle assistance with phase
switching
@spring-boot-expert - Spring Boot specialized help
@selenium-expert - UI testing and automation expertise
@training-instructor - Educational and training support
```

Mode Selection Guidelines

Scenario	Recommended Mode	Why
Building REST APIs	`@spring-boot-expert`	Framework-specific expertise
Writing UI tests	`@selenium-expert`	Testing patterns and best practices
Learning new concepts	`@training-instructor`	Structured educational approach
Development process	`@sdlc-modes`	Phase-appropriate guidance
Code reviews	`@sdlc-modes /mode review`	Quality assurance focus

🛠️ Custom Mode Implementation Details

File Structure and Location

Custom chat modes are stored in `.github/chatmodes/` with the following structure:

```
---
description: Brief description of the mode's purpose
tools: [] # Available tools (optional)
---
```

```
# Mode instructions in Markdown format
Detailed instructions for AI behavior, patterns, and responses
```

NFCU Training Project Structure

```
.github/chatmodes/
├── sdlc-modes.chatmode.md          # Multi-phase development assistance
├── spring-boot-expert.chatmode.md   # Spring Boot specialized help
├── selenium-expert.chatmode.md      # UI testing expertise
└── training-instructor.chatmode.md  # Educational support
```

Creating Custom Modes

1. **Create File**: Place `.chatmode.md` files in `.github/chatmodes/`
2. **Define Metadata**: Add YAML front matter with description
3. **Write Instructions**: Provide detailed AI behavior guidelines
4. **Test and Refine**: Iterate based on usage and feedback

💡 Best Practices for Custom Modes

Effective Mode Usage

1. **Choose Specific Modes**: Use domain-specific modes for specialized tasks
2. **Combine with Context**: Use modes with #file, #selection, @workspace
3. **Consistent Interaction**: Stay within the same mode for related tasks
4. **Progressive Learning**: Use training-instructor for new concepts first

Mode-Specific Tips

SDLC Modes

```
# Effective patterns
@sdlc-modes /mode development Implement user authentication
@sdlc-modes /mode review #file:UserController.java Analyze for security
issues
@sdlc-modes /mode production Prepare deployment configuration
```

Spring Boot Expert

```
# Leverage project context
@spring-boot-expert #file:pom.xml Add caching dependencies and
configuration
@spring-boot-expert Create integration tests for the UserService following
project patterns
```

Selenium Expert

```
# Testing scenarios
@selenium-expert Create data-driven tests for user registration
@selenium-expert #file:GoogleTest.java Refactor using Page Object Model
@selenium-expert Generate TestNG XML for parallel test execution
```

Training Instructor

~~~
# Educational queries
@training-instructor Explain GitHub Copilot context usage with examples
@training-instructor Create a hands-on exercise for Spring Boot
development
@training-instructor Assess understanding of Selenium best practices
## 🚨 Common Pitfalls and Solutions

### Mode Selection Issues

**Problem**: Using the wrong mode for your task type
**Solution**: Reference the mode selection guidelines and choose based on
domain expertise needed

**Example**:
```markdown
# Less Effective
@sdlc-modes How do I configure Maven dependencies?

# More Effective
@spring-boot-expert How do I add caching dependencies to this Spring Boot
project?
~~~

Context Integration

Problem: Not providing enough project context

Solution: Combine modes with appropriate context references

Example:

```
# Better Context Usage
@spring-boot-expert #file:UserController.java Add validation following
project patterns
@selenium-expert @workspace Create page objects for all forms in the
application
```

Mode Consistency

Problem: Switching modes mid-conversation without clear reason
Solution: Stay within appropriate mode for related tasks, switch intentionally

🔧 Troubleshooting Custom Modes

Mode Not Available

1. **Check File Location**: Ensure `.chatmode.md` files are in `.github/chatmodes/`
2. **Verify File Format**: Check YAML front matter and markdown structure
3. **Restart VS Code**: Reload window after adding new modes

Inconsistent Behavior

1. **Review Mode Instructions**: Ensure instructions are clear and specific
2. **Test with Examples**: Use concrete examples to validate mode behavior
3. **Iterate and Refine**: Update mode instructions based on usage feedback

Performance Issues

1. **Simplify Instructions**: Keep mode instructions focused and concise
2. **Optimize Context**: Use appropriate context without overloading
3. **Check Connectivity**: Ensure stable internet connection for AI responses

🎓 Training Integration

Course Usage

These custom modes are designed for progressive skill building:

1. **Morning**: Use `@training-instructor` for foundational concepts
2. **Afternoon**: Apply `@spring-boot-expert` and `@selenium-expert` for hands-on development
3. **Advanced**: Leverage `@sdlc-modes` for complete development workflows

Best Learning Practices

- Start with `@training-instructor` for new concepts
- Use domain-specific modes for practical implementation
- Apply `@sdlc-modes` for process-oriented development
- Combine modes with project context for realistic scenarios

Remember: Custom Chat Modes provide specialized AI assistance tailored to your specific development contexts. The modes in this repository are designed specifically for the NFCU GitHub Copilot training program, demonstrating how to create focused, domain-specific AI assistants that enhance productivity and learning outcomes. Use them as templates for creating your own specialized modes in your projects.