# VS Code Workspace Guide

## 📚 Table of Contents

---

## What is a VS Code Workspace?

A **VS Code Workspace** is a collection of one or more folders (project directories) that are opened together in Visual Studio Code. Workspaces enable you to:

- Work with multiple related projects simultaneously
- Share settings, extensions, and configurations across projects
- Maintain consistent development environment
- Enable cross-project code navigation and search

### Workspace Types

1. **Single-Folder Workspace**: Opening one folder in VS Code
2. **Multi-Root Workspace**: Opening multiple folders in a single VS Code window (saved as `.code-workspace` file)

---

## Multi-Root Workspaces

Multi-root workspaces are powerful when you have:

- **Microservices architectures** (frontend + backend)
- **Monorepos** with multiple packages

- **Training environments** with reference code and exercises
- **Cross-project dependencies** that need simultaneous development

### Workspace File Structure

A `.code-workspace` file is a JSON configuration that defines:

```
{
  "folders": [
    {
      "name": "Training Materials",
      "path": "../copilot-United"
    },
    {
      "name": "Recipe App Exercise",
      "path": "../../Copilot-Exercises/recipe-sharing-app"
    }
  ],
  "settings": {
    "editor.formatOnSave": true,
    "files.autoSave": "afterDelay"
  },
  "extensions": {
    "recommendations": [
      "github.copilot",
      "github.copilot-chat"
    ]
  }
}
```

---

## Adding Folders to Your Workspace

### Method 1: Using the Menu (Beginner-Friendly)

1. **Open VS Code** with your first project folder
2. **File → Add Folder to Workspace...**
3. Browse and select the folder you want to add
4. Click **Add** to include it in the current window
5. **File → Save Workspace As...** to save the configuration

### Method 2: Using the Command Palette (Fast)

1. Press `Cmd+Shift+P` (Mac) or `Ctrl+Shift+P` (Windows/Linux)
2. Type: `Workspaces: Add Folder to Workspace`
3. Select the folder to add
4. Save workspace with `Cmd+Shift+P` → `Workspaces: Save Workspace As`

### Method 3: Edit Workspace File Directly (Advanced)

1. Open the `.code-workspace` file in your editor
2. Add new folder entries to the `folders` array:

```
{
  "folders": [
    {
      "name": "Project 1",
      "path": "/absolute/path/to/project1"
    },
    {
      "name": "Project 2",
      "path": "../relative/path/to/project2"
    }
  ]
}
```

1. Save and reload VS Code to apply changes

### Method 4: Drag and Drop

1. Simply **drag a folder** from Finder/Explorer into the VS Code explorer panel
2. Choose **Add Folder to Workspace**

---

## GitHub Copilot and Workspace Context

### How Copilot Uses Workspace Context

GitHub Copilot is **context-aware** and leverages your entire workspace to provide better suggestions:

## 1. Code Understanding Across Projects

When you have multiple folders in a workspace, Copilot:

- ✅ Reads code from all workspace folders
- ✅ Understands patterns and conventions across projects
- ✅ Suggests code that matches your multi-project architecture
- ✅ References types, functions, and classes from other folders

## 2. Cross-Project Pattern Recognition

Example: Training workspace with reference code

```
Workspace:
├── copilot-United/          # Training materials + examples
|   ├── java-exercise/       # Java Spring Boot examples
|   ├── python-exercise/     # Python FastAPI examples
|   └── dotnet-exercise/     # .NET examples
└── recipe-sharing-app/      # Your current project
    ├── RecipeApp.Api/       # .NET API
    └── RecipeApp.Client/    # Blazor client
```

**Copilot can**:

- Reference patterns from `dotnet-exercise/` when working in `RecipeApp.Api/`
- Suggest similar controller patterns you used in example projects
- Adapt testing strategies from training materials
- Apply naming conventions from reference code

## 3. Using the `#codebase` Context

With a multi-root workspace, the `#codebase` context becomes more powerful:

```
# In Copilot Chat:
"Review #codebase for all API controller patterns"

# Copilot searches BOTH:
- copilot-United/dotnet-exercise/Controllers/
- recipe-sharing-app/RecipeApp.Api/Controllers/
```

### 4. Custom Instructions Integration

Each workspace root can have its own `.github/copilot-instructions.md`:

```
copilot-United/.github/copilot-instructions.md
  → Training-specific instructions
  → Educational code patterns
  → Four-track course structure

recipe-sharing-app/.github/copilot-instructions.md
  → Project-specific architecture
  → Recipe app conventions
  → Development guidelines
```

**Copilot merges these instructions** when providing suggestions!

## Practical Workflow Examples

### Example 1: Learning from Reference Code

**Scenario**: You're implementing a new controller in your recipe app and want to follow Spring Boot patterns from training materials.

```
// In recipe-sharing-app/RecipeApp.Api/Controllers/RecipesController.cs

// Prompt: "Create a controller method to get recipes by category"

// Copilot references:
// - copilot-United/dotnet-exercise/Controllers/ patterns
// - Your existing RecipesController.cs structure
// - RESTful conventions from training materials

[HttpGet("category/{category}")]
public async Task<ActionResult<IEnumerable<RecipeDto>>>
GetRecipesByCategory(string category)
{
    // Generated code follows patterns from both projects
}
```

### Example 2: Consistent Testing Patterns

**Scenario**: Write tests using AAA pattern demonstrated in training materials.

```
# In recipe-sharing-app/tests/test_recipes.py

# Copilot references:
# - copilot-United/python-exercise/tests/ for AAA pattern
# - Your existing test structure

def test_create_recipe_success():
    # Arrange (learned from training materials)
    recipe_data = {"name": "Test Recipe"}

    # Act
    response = client.post("/api/recipes", json=recipe_data)

    # Assert
    assert response.status_code == 201
```

### Example 3: Documentation Style Consistency

```
# In recipe-sharing-app/docs/API.md

# Copilot Chat prompt:
"Generate API documentation using the style from #codebase"

# Copilot analyzes:
# - copilot-United/docs/ markdown patterns
# - Existing documentation structure
# - Emoji headers (🎯, 💬, ✏️) from training materials

# Result: Consistent documentation style across workspace
```

---

## Best Practices

### 🎯 Workspace Organization

#### 1. Logical Grouping

```
✅ Good:
- Training materials + practice exercises
- Backend + Frontend projects
- Shared libraries + dependent services
```

```
❌ Avoid:
- Unrelated projects in same workspace
- Too many folders (>5-7 roots)
```

### 2. Named Folders

```
{
  "folders": [
    {
      "name": "📚 Training Materials",  // Clear, descriptive names
      "path": "../copilot-United"
    },
    {
      "name": "🚀 Recipe App",
      "path": "."
    }
  ]
}
```

### 3. Relative Paths When Possible

```
// Prefer relative paths for portability
"path": "../shared-library"

// Instead of absolute paths
"path": "/Users/username/projects/shared-library"
```

## 🤖 Maximizing Copilot Context

### 1. Strategic Folder Addition

Add folders that provide **relevant context**:

```
Working on .NET project?
✅ Add: Reference .NET training materials
✅ Add: Shared DTOs/models project
❌ Don't add: Unrelated Java projects
```

### 2. Use Custom Instructions

Create `.github/copilot-instructions.md` in each root:

```
## Project: Recipe Sharing App
- Backend: .NET 10 Web API
- Frontend: Blazor WebAssembly
- Database: SQLite + EF Core
- Architecture: Repository pattern
```

### 3. Leverage Hash Contexts

```
# Reference specific workspace folders
#file:copilot-United/dotnet-exercise/Controllers/ExpenseController.cs

# Search across entire workspace
#codebase repository pattern implementation

# Work with current selection
#selection optimize this query
```

### 4. Consistent Conventions

When Copilot sees patterns across your workspace:

- **Naming conventions** (PascalCase, camelCase)
- **File organization** (Controllers/, Services/, Models/)
- **Testing patterns** (AAA, naming conventions)
- **Documentation style** (markdown headers, code examples)

It will **automatically apply these patterns** to new code!

---

## Real-World Example

### Current Workspace Setup

Based on your current environment:

```
{
  "folders": [
    {
      "name": "📚 Copilot Training (copilot-United)",
      "path": "/Users/kangs/github/copilot-United"
    },
    {
      "name": "🚀 Recipe Sharing App",
      "path": "/Users/kangs/Copilot-Exercises/recipe-sharing-app"
    }
  ]
}
```

## How Copilot Uses This Setup

### Context Flow Diagram

```
┌─────────────────────────────────────────────┐
│  Copilot Context Engine                     │
├──────────────────────────────────────────── ┤
│                                             │
│  📚 copilot-United/                         │
│     ├── .github/copilot-instructions.md    │
│     │   → Training patterns                │
│     │   → Educational guidelines           │
│     │                                      │
│     ├── dotnet-exercise/                   │
│     │   → Controller patterns              │
│     │   → Repository examples              │
│     │   → Testing templates                │
│     │                                      │
│     └── docs/                              │
│         → Documentation style              │
│         → Best practices                   │
│                                            │
│  🚀 recipe-sharing-app/                      │
│     ├── .github/copilot-instructions.md     │
│     │   → Project architecture              │
│     │   → Recipe app specifics              │
│     │                                       │
│     └── RecipeApp.Api/                      │
│         ↑↑↑ Copilot combines both contexts   │
│                                              │
└──────────────────────────────────────────── ┘
```

### Practical Benefits

1. **When writing controllers in RecipeApp.Api**:
2. Copilot references `dotnet-exercise/` controller patterns
3. Applies repository pattern from training materials

4. Uses consistent error handling approaches

5. **When creating tests**:

6. Follows AAA pattern from training examples
7. Uses similar naming conventions

8. Applies test organization strategies

9. **When generating documentation**:

10. Matches markdown style from `copilot-United/docs/`
11. Uses emoji headers consistently

12. Follows documentation templates

13. **When using Copilot Chat**:

```text "Create a new RecipeController following patterns from #codebase"

Copilot analyzes: ✓ copilot-United/dotnet-exercise/Controllers/*.cs ✓ recipe-sharing-app/RecipeApp.Api/Controllers/*.cs ✓ Both copilot-instructions.md files

Result: Controller that follows both training patterns and project-specific conventions ```

---

## 🚀 Quick Start Checklist

- [ ] Open your primary project in VS Code
- [ ] Add reference/training folders: **File** → **Add Folder to Workspace**
- [ ] Save workspace: **File** → **Save Workspace As...**
- [ ] Create `.github/copilot-instructions.md` in each root folder
- [ ] Test Copilot context: Use `#codebase` in Copilot Chat
- [ ] Verify cross-project suggestions work
- [ ] Commit `.code-workspace` file to version control

---

## 📖 Additional Resources

- VS Code Multi-Root Workspaces
- GitHub Copilot Documentation
- Copilot Custom Instructions
```

- Training Materials: copilot-United/day2/Copilot-Hash-Context.md

---

## 💡 Pro Tips

### Context Window Management

Copilot has context limits. Prioritize quality over quantity:

```
✅ Do: Add 2-3 highly relevant project folders
❌ Don't: Add 10+ unrelated repositories

✅ Do: Use #file to target specific examples
❌ Don't: Rely on Copilot to search everything
```

### Workspace Settings Precedence

Settings override order:

1. **User Settings** (global)
2. **Workspace Settings** ( `.code-workspace` )
3. **Folder Settings** ( `.vscode/settings.json` in each root)

### Performance Considerations

- Large workspaces may slow down indexing
- Exclude unnecessary folders in `.code-workspace` :

```
{
  "folders": [...],
  "settings": {
    "files.exclude": {
      "**/node_modules": true,
      "**/bin": true,
      "**/obj": true
    }
  }
}
```