GitHub Copilot Project Setup Commands & Tips 🚀

This guide provides essential GitHub Copilot commands and techniques to accelerate your Spring Boot Task Manager project setup. Use these commands to leverage AI assistance throughout the development process, making your setup faster and more efficient.

🎯 Prerequisites

Before starting, ensure you have completed the installation guides:

- **macOS Users**: Project 1 Installation Guide (macOS)
- **Windows Users**: Project 1 Installation Guide (Windows)

Required Extensions:

- ✅ GitHub Copilot
- ✅ GitHub Copilot Chat
- ✅ Java Extension Pack
- ✅ Spring Boot Extension Pack

🛠️ Essential Copilot Slash Commands for Setup

`/new` - Create New Files and Projects

The `/new` command is your best friend for rapid project scaffolding:

Create New Spring Boot Project Structure

```
/new Create a Spring Boot 3.2.3 project with the following structure:
- Maven project with JDK 21
- Dependencies: web, data-jpa, h2, thymeleaf, security, validation
- Package structure: com.taskmanager.app
- Include controller, service, repository, entity packages
```

Generate Configuration Files

```
/new Create application.properties for Spring Boot task manager with:
- H2 database configuration for development
- Server port 8080
- Thymeleaf configuration
- Spring Security disabled for development
```

Create Maven POM Template

```
/new Generate a complete pom.xml for Spring Boot 3.2.3 with:
- JDK 21 configuration
- Task manager dependencies: web, jpa, h2, thymeleaf, bootstrap webjars
- Spring Boot Maven plugin configuration
```

`/generate` - Generate Boilerplate Code

Perfect for creating repetitive code structures:

Generate Entity Classes

```
/generate Create a complete JPA User entity with:
- Long id (auto-generated)
- String firstName, lastName, email
- LocalDateTime createdAt, updatedAt
- Proper JPA annotations and validation
- Constructors, getters, setters
```

Generate Repository Interfaces

```
/generate Create JPA repository interfaces for:
- UserRepository extending JpaRepository
- TaskRepository with custom query methods
- Include method signatures for common operations
```

Generate Service Layer

```
/generate Create service classes with:
- UserService with CRUD operations
- Proper exception handling
```

```
- Validation logic
- Transaction annotations
```

`/explain` - Understand Generated Code

Use this to understand complex configurations:

```
/explain What does this Spring Boot configuration do?
```

```
/explain How do these JPA annotations work together?
```

```
/explain What is the purpose of this Maven dependency?
```

`/fix` - Resolve Setup Issues

When you encounter errors during setup:

```
/fix This Maven compilation error
```

```
/fix These Spring Boot startup issues
```

```
/fix This JPA entity mapping problem
```

`/optimize` - Improve Project Configuration

```
/optimize This Maven pom.xml for better performance
```

```
/optimize These application.properties for development
```

```
/optimize This Spring Boot configuration class
```

🎨 Copilot Chat Commands for Project Setup

Project Initialization Prompts

Complete Project Scaffolding

```
I need to create a Spring Boot 3.2.3 task management application with:
- Maven build system using JDK 21
- Package structure: com.taskmanager.app
- Entities: User, Task, Category
- Thymeleaf frontend with Bootstrap
- H2 database for development
- Spring Security (configurable)

Generate the complete project structure with all necessary files.
```

Database Setup

```
Set up H2 database configuration for a Spring Boot task manager:
- In-memory database for development
- Console access enabled
- JPA auto-configuration
- Sample data initialization
```

Frontend Setup

```
Configure Thymeleaf with Bootstrap 5.3.2 using WebJars:
- Create base template with navbar
- Include Bootstrap CSS and JavaScript
- Set up responsive layout
- Add jQuery 3.7.1 support
```

Dependency Configuration

```
Update my Spring Boot pom.xml to include:
- Spring Web, JPA, Security, Validation
- H2 database and PostgreSQL drivers
- Thymeleaf template engine
- Bootstrap and jQuery WebJars
- SpringDoc OpenAPI for API documentation
```

```
- DevTools for development

Ensure all versions are compatible with Spring Boot 3.2.3
```

🔧 Advanced Copilot Techniques for Setup

Context-Aware Generation

Using File Context

```
#file:pom.xml Add the missing dependencies for a complete task management application
```

```
#file:application.properties Configure this for H2 database with console access
```

Using Selection Context

```
#selection Explain this Spring Boot configuration section
```

```
#selection Generate corresponding test configuration for this setup
```

Using Terminal Context

```
#terminal Help me resolve this Maven build error
```

```
#terminal Explain what this Spring Boot startup log means
```

Multi-Step Project Setup

Phase 1: Foundation

```
Create the foundation for a Spring Boot task manager:
1. Generate main application class
2. Create basic configuration
```

```
3. Set up package structure
4. Add essential dependencies
```

Phase 2: Data Layer

```
Based on the existing project structure, create:
1. User and Task entities with JPA annotations
2. Repository interfaces with custom queries
3. Database initialization scripts
4. Entity relationship mappings
```

Phase 3: Web Layer

```
Building on the data layer, generate:
1. REST controllers for User and Task
2. Thymeleaf templates for basic UI
3. Form handling and validation
4. Static resources setup
```

💡 Pro Tips for Efficient Setup

1. Start with Comments

Write detailed comments first, then let Copilot generate the implementation:

```
/**
 * Main Spring Boot application class for Task Manager
 * Enables JPA repositories and web MVC
 * Configures component scanning for com.taskmanager.app
 */
// Let Copilot complete the class
```

2. Use Descriptive Prompts

Instead of: "Create a user class" Use: "Create a JPA User entity with validation, timestamps, and email uniqueness constraint"

3. Leverage Existing Code Context

Keep related files open in VS Code so Copilot can use them as context for generating complementary code.

4. Iterative Refinement

```
// First iteration
Create a basic Spring Boot controller

// Second iteration
Add validation and exception handling to this controller

// Third iteration
Add Swagger documentation to these endpoints
```

5. Test-Driven Generation

```
Generate unit tests for UserService, then create the service
implementation to pass these tests
```

🚀 Quick Setup Workflow

Step 1: Project Structure

```
/new Create complete Spring Boot 3.2.3 Maven project structure for task
manager with:
- Standard directory layout
- pom.xml with all required dependencies
- application.properties with H2 configuration
- Main application class with proper annotations
```

Step 2: Core Entities

```
Generate JPA entities for task management system:
- User (id, firstName, lastName, email, password, timestamps)
- Task (id, title, description, status, priority, dueDate, user
relationship)
```

```
- TaskStatus enum (PENDING, IN_PROGRESS, COMPLETED, CANCELLED)
- Proper JPA annotations and validation
```

Step 3: Data Access Layer

```
Create repository interfaces extending JpaRepository:
- UserRepository with findByEmail method
- TaskRepository with findByUserAndStatus methods
- Include custom query annotations where needed
```

Step 4: Business Logic Layer

```
Generate service classes with business logic:
- UserService with CRUD operations and email validation
- TaskService with task management operations
- Proper exception handling and transaction management
```

Step 5: Web Layer

```
Create Spring MVC controllers:
- TaskController with REST endpoints
- UserController with user management
- Proper request/response DTOs
- Input validation and error handling
```

Step 6: Frontend Templates

```
Generate Thymeleaf templates with Bootstrap:
- Base layout template with navigation
- Task list and form templates
- User management pages
- Responsive design with Bootstrap components
```

🐛 Common Setup Issues & Copilot Solutions

Maven Build Failures

```
/fix Maven compilation error: package com.taskmanager.app does not exist
```

Spring Boot Startup Issues

```
#terminal /fix Spring Boot application failed to start - explain this
error log
```

Database Connection Problems

```
/fix H2 database connection refused - check application.properties
configuration
```

Dependency Conflicts

```
/optimize Resolve Maven dependency conflicts in this pom.xml
```

Template Engine Issues

```
/fix Thymeleaf template not found error - check template location and
controller mapping
```

📚 Command Reference Quick Card

Command	Purpose	Example
`/new`	Create new files/ projects	`/new Spring Boot controller with CRUD operations`
`/generate`	Generate boilerplate code	`/generate JPA entity with validation`

Command	Purpose	Example
`/explain`	Understand code/ configs	`/explain this Maven dependency`
`/fix`	Resolve issues	`/fix compilation errors`
`/optimize`	Improve code/config	`/optimize database configuration`
`#file:`	Reference specific file	`#file:pom.xml add missing dependency`
`#selection`	Use selected code	`#selection explain this configuration`
`#terminal`	Reference terminal output	`#terminal help with this error`

🎓 Best Practices

DO:

- ✅ **Write descriptive prompts** - Be specific about what you want
- ✅ **Use context references** - Reference files and selections
- ✅ **Iterate and refine** - Start simple, then enhance
- ✅ **Review generated code** - Understand what Copilot creates
- ✅ **Test frequently** - Verify each step works before moving on

DON'T:

- ❌ **Accept code blindly** - Always review and understand generated code
- ❌ **Skip documentation** - Use `/explain` to understand complex parts
- ❌ **Ignore errors** - Use `/fix` to resolve issues immediately
- ❌ **Work in isolation** - Use existing code as context for better suggestions
- ❌ **Rush the process** - Take time to understand each component

🚀 Getting Started Checklist

Before your first session, complete this checklist:

- [ ] ✅ **Installation Complete**: JDK 21, Maven, Git, VS Code installed
- [ ] ✅ **Extensions Installed**: GitHub Copilot, Copilot Chat, Java Extension Pack
- [ ] ✅ **Copilot Active**: Green Copilot icon visible in VS Code status bar
- [ ] ✅ **Project Directory**: Created workspace folder for task manager project
- [ ] ✅ **Terminal Access**: Can run `java -version`, `mvn -version`, `git --version`
- [ ] ✅ **Practice Commands**: Tried basic `/new` and `/generate` commands
- [ ] ✅ **Chat Panel**: Can open and interact with Copilot Chat

📞 Getting Help

If you encounter issues during setup:

1. **Use Copilot Chat**: Describe your problem and paste error messages
2. **Reference Context**: Use `#terminal` to share error outputs
3. **Ask for Explanations**: Use `/explain` to understand error messages
4. **Get Fixes**: Use `/fix` with specific error descriptions

Example Help Request:

```
I'm getting this error when running mvn spring-boot:run:
#terminal [ERROR] Failed to execute goal org.springframework.boot:spring-
boot-maven-plugin

/fix this Spring Boot Maven execution error and explain what caused it
```

🎉 Ready to Start

With these commands and techniques, you're equipped to efficiently set up your Spring Boot Task Manager project using GitHub Copilot. Remember:

- **Start with structure**, then add functionality
- **Use descriptive prompts** for better suggestions
- **Leverage context** from existing files
- **Test and validate** each step

- **Ask Copilot for help** when stuck

Happy coding with GitHub Copilot! 🚀

💡 **Remember**: The goal is to learn both GitHub Copilot and Spring Boot development. Use AI assistance to accelerate learning, not replace understanding!