.NET Conf China 2025

改变世界 改变自己

2025 年 11 月 30 日 | 中国 上海

用.NET拥抱Spec驱动开发

About Me

张海

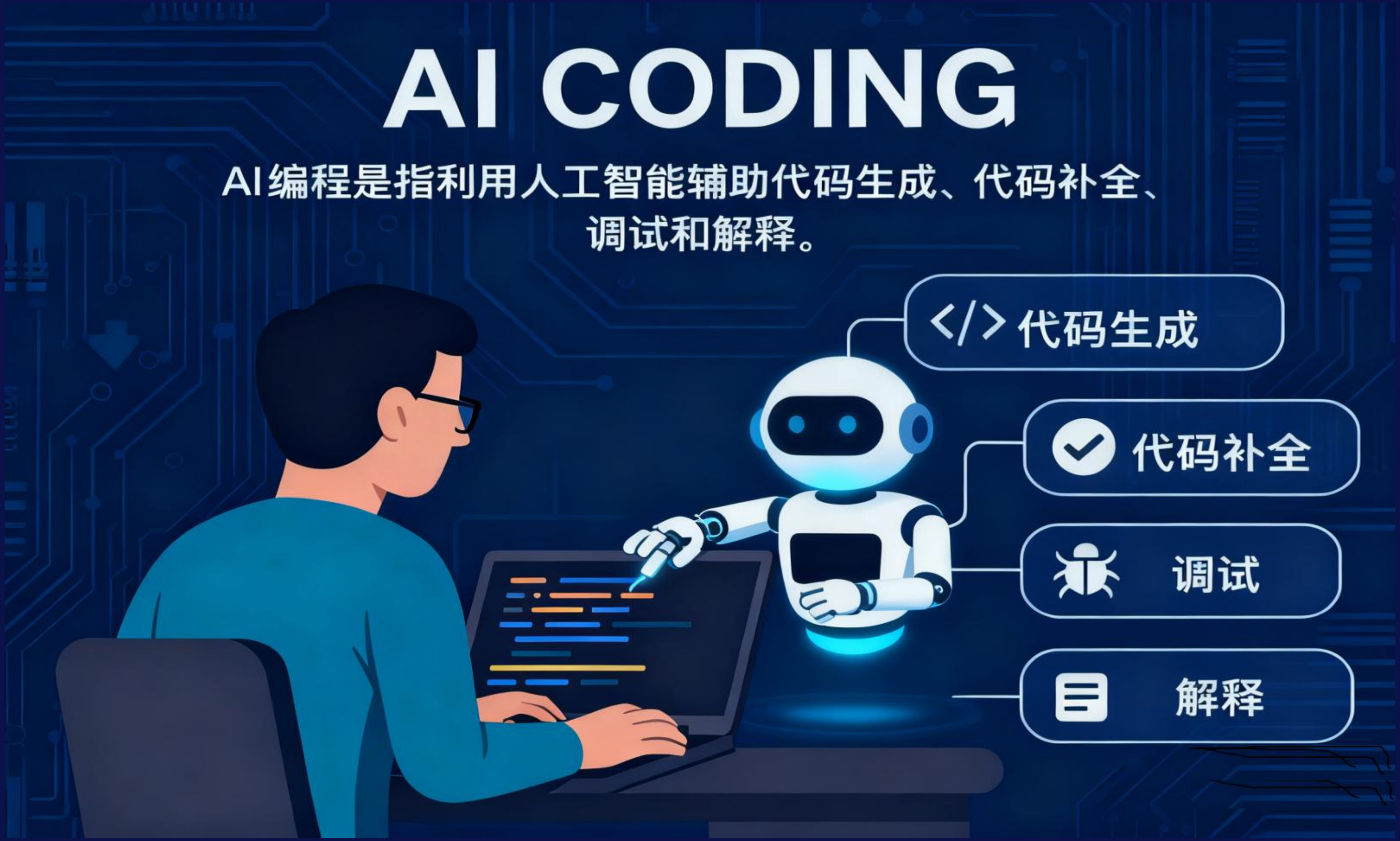
神州数码 资深架构师

公众号 MyIO 作者

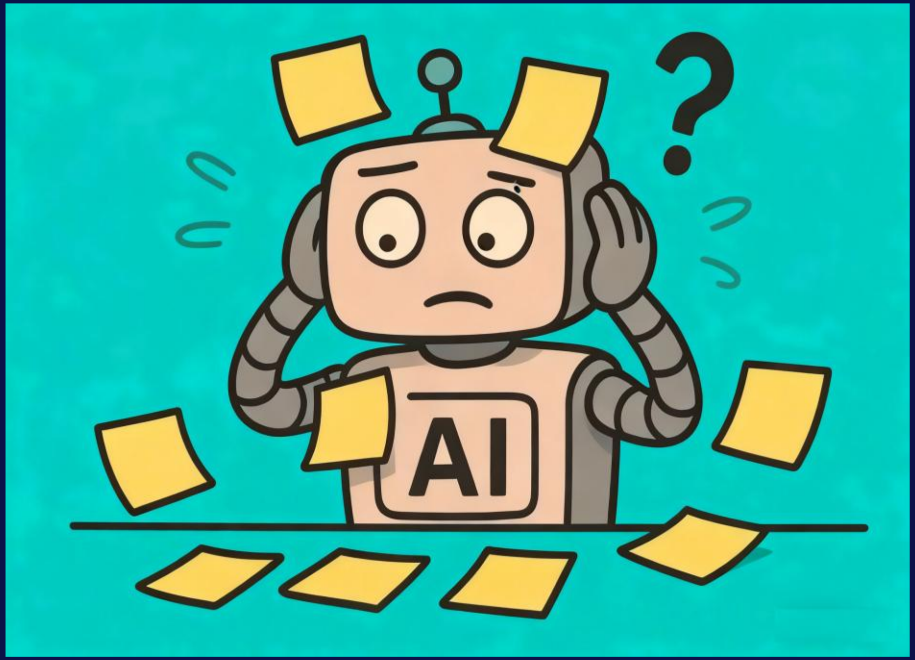
现任 Microsoft MVP (开发方向)

.Net Conf China 2022 讲师

AI Coding



遇到的问题



- 上下文遗忘
- 需求理解偏差
- 工程质量不高

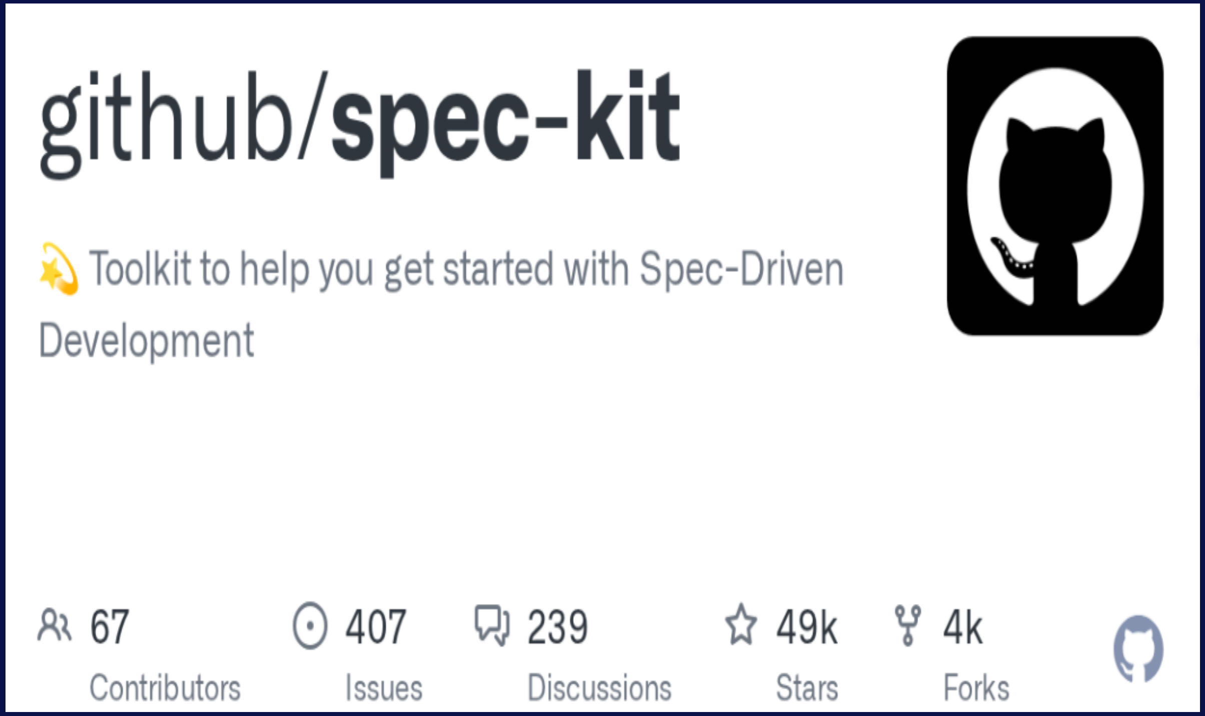
Spec Driven Development

“规格/规范驱动开发（SDD）”是一种软件开发方法，其中Spec（即对系统应具备功能的正式描述）是整个开发过程的首要依据和核心准则。

Spec Driven Development

KIRO

IDE



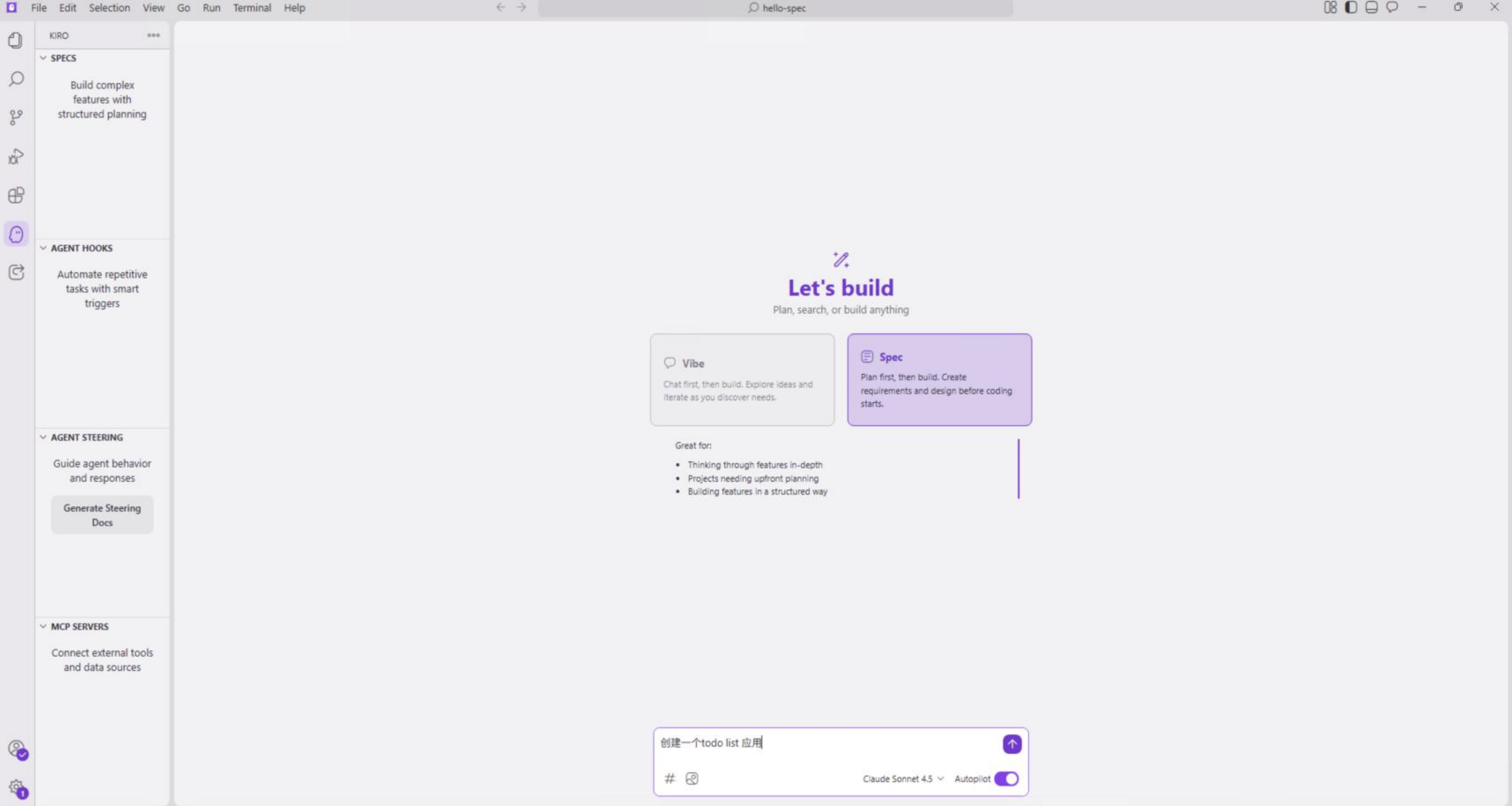
github/**spec-kit**

💫 Toolkit to help you get started with Spec-Driven Development

67	407	239	49k	4k
Contributors	Issues	Discussions	Stars	Forks

CLI

Kiro



SPECS

Build complex features with structured planning

AGENT HOOKS

Automate repetitive tasks with smart triggers

AGENT STEERING

Guide agent behavior and responses

MCP SERVERS

Connect external tools and data sources

Let's build

Plan, search, or build anything

Vibe

Chat first, then build. Explore ideas and iterate as you discover needs.

Spec

Plan first, then build. Create requirements and design before coding starts.

Great for:

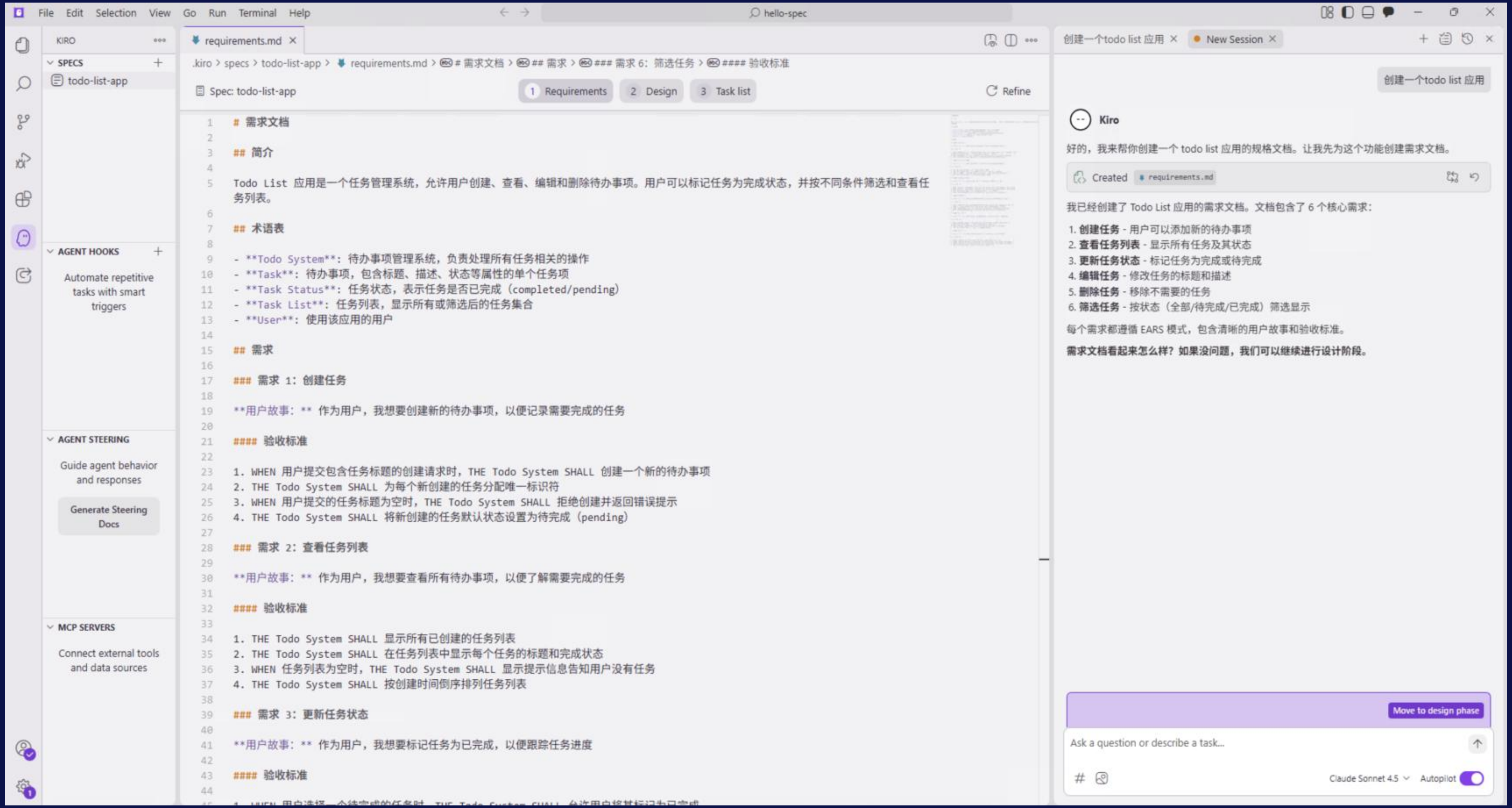
- Thinking through features in-depth
- Projects needing upfront planning
- Building features in a structured way

创建一个todo list 应用

Claude Sonnet 4.5

Autopilot

Kiro


File Edit Selection View Go Run Terminal Help
hello-spec
KIRO
SPECS
todo-list-app
AGENT HOOKS
Automate repetitive tasks with smart triggers
AGENT STEERING
Guide agent behavior and responses
Generate Steering Docs
MCP SERVERS
Connect external tools and data sources
requirements.md
.kiro › specs › todo-list-app › requirements.md › # 需求文档 › ## 需求 › ### 需求 6：筛选任务 › #### 验收标准
Spec: todo-list-app
1 Requirements
2 Design
3 Task list
Refine
创建一个todo list 应用
New Session
Move to design phase
Ask a question or describe a task...
Claude Sonnet 4.5
Autopilot

```
# 需求文档

## 简介

Todo List 应用是一个任务管理系统，允许用户创建、查看、编辑和删除待办事项。用户可以标记任务为完成状态，并按不同条件筛选和查看任务列表。

## 术语表

- **Todo System**: 待办事项管理系统，负责处理所有任务相关的操作
- **Task**: 待办事项，包含标题、描述、状态等属性的单个任务项
- **Task Status**: 任务状态，表示任务是否已完成（completed/pending）
- **Task List**: 任务列表，显示所有或筛选后的任务集合
- **User**: 使用该应用的用户

## 需求

### 需求 1: 创建任务

**用户故事: ** 作为用户，我想要创建新的待办事项，以便记录需要完成的任务

#### 验收标准

1. WHEN 用户提交包含任务标题的创建请求时，THE Todo System SHALL 创建一个新的待办事项
2. THE Todo System SHALL 为每个新创建的任务分配唯一标识符
3. WHEN 用户提交的任务标题为空时，THE Todo System SHALL 拒绝创建并返回错误提示
4. THE Todo System SHALL 将新创建的任务默认状态设置为待完成（pending）

### 需求 2: 查看任务列表

**用户故事: ** 作为用户，我想要查看所有待办事项，以便了解需要完成的任务

#### 验收标准

1. THE Todo System SHALL 显示所有已创建的任务列表
2. THE Todo System SHALL 在任务列表中显示每个任务的标题和完成状态
3. WHEN 任务列表为空时，THE Todo System SHALL 显示提示信息告知用户没有任务
4. THE Todo System SHALL 按创建时间倒序排列任务列表

### 需求 3: 更新任务状态

**用户故事: ** 作为用户，我想要标记任务为已完成，以便跟踪任务进度

#### 验收标准

```

创建一个todo list 应用

Kiro

好的，我来帮你创建一个 todo list 应用的规格文档。让我先为这个功能创建需求文档。

Created `requirements.md`

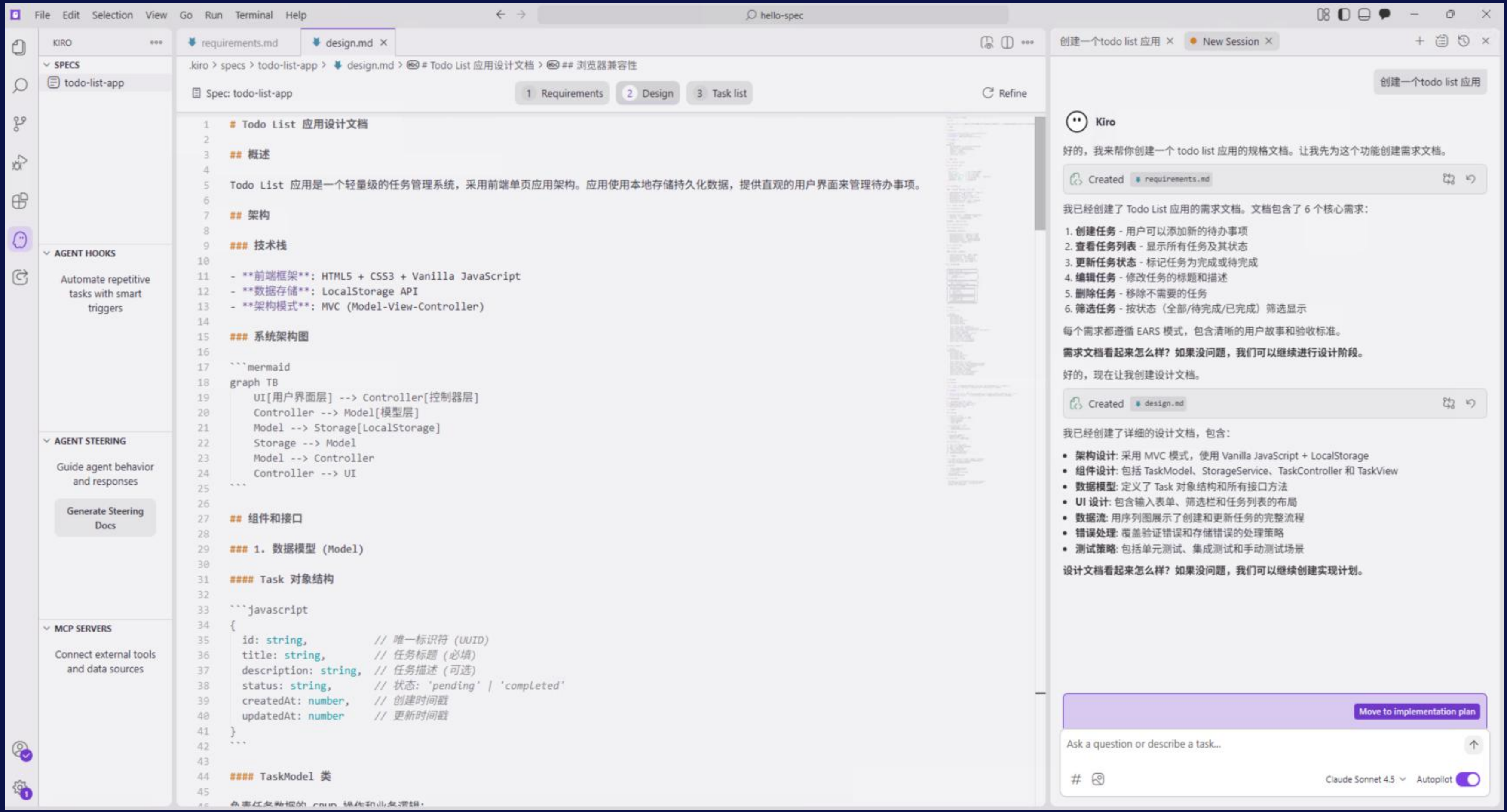
我已经创建了 Todo List 应用的需求文档。文档包含了 6 个核心需求：

1. **创建任务** - 用户可以添加新的待办事项
2. **查看任务列表** - 显示所有任务及其状态
3. **更新任务状态** - 标记任务为完成或待完成
4. **编辑任务** - 修改任务的标题和描述
5. **删除任务** - 移除不需要的任务
6. **筛选任务** - 按状态（全部/待完成/已完成）筛选显示

每个需求都遵循 EARS 模式，包含清晰的用户故事和验收标准。

需求文档看起来怎么样？如果没问题，我们可以继续进行设计阶段。

Kiro


File Edit Selection View Go Run Terminal Help
hello-spec
KIRO
SPECS
todo-list-app
AGENT HOOKS
Automate repetitive tasks with smart triggers
AGENT STEERING
Guide agent behavior and responses
Generate Steering Docs
MCP SERVERS
Connect external tools and data sources
requirements.md
design.md
.kiro > specs > todo-list-app > design.md > # Todo List 应用设计文档 > ## 浏览器兼容性
Spec: todo-list-app
1 Requirements
2 Design
3 Task list
Refine
创建一个todo list 应用
New Session
Move to implementation plan
Ask a question or describe a task...
#
Claude Sonnet 4.5
Autopilot

~~~
# Todo List 应用设计文档

## 概述

Todo List 应用是一个轻量级的任务管理系统，采用前端单页应用架构。应用使用本地存储持久化数据，提供直观的用户界面来管理待办事项。

## 架构

### 技术栈

- **前端框架**: HTML5 + CSS3 + Vanilla JavaScript
- **数据存储**: LocalStorage API
- **架构模式**: MVC (Model-View-Controller)

### 系统架构图

```mermaid
graph TB
    UI[用户界面层] --> Controller[控制器层]
    Controller --> Model[模型层]
    Model --> Storage[LocalStorage]
    Storage --> Model
    Model --> Controller
    Controller --> UI
```

## 组件和接口

### 1. 数据模型 (Model)

#### Task 对象结构

```javascript
{
  id: string,          // 唯一标识符 (UUID)
  title: string,       // 任务标题 (必填)
  description: string, // 任务描述 (可选)
  status: string,      // 状态: 'pending' | 'completed'
  createdAt: number,   // 创建时间戳
  updatedAt: number    // 更新时间戳
}
```

#### TaskModel 类
~~~

创建一个todo list 应用

Kiro

好的，我来帮你创建一个 todo list 应用的规格文档。让我先为这个功能创建需求文档。

Created `requirements.md`

我已经创建了 Todo List 应用的需求文档。文档包含了 6 个核心需求：

1. **创建任务** - 用户可以添加新的待办事项
2. **查看任务列表** - 显示所有任务及其状态
3. **更新任务状态** - 标记任务为完成或待完成
4. **编辑任务** - 修改任务的标题和描述
5. **删除任务** - 移除不需要的任务
6. **筛选任务** - 按状态（全部/待完成/已完成）筛选显示

每个需求都遵循 EARS 模式，包含清晰的用户故事和验收标准。

需求文档看起来怎么样？如果没问题，我们可以继续进行设计阶段。

好的，现在让我创建设计文档。

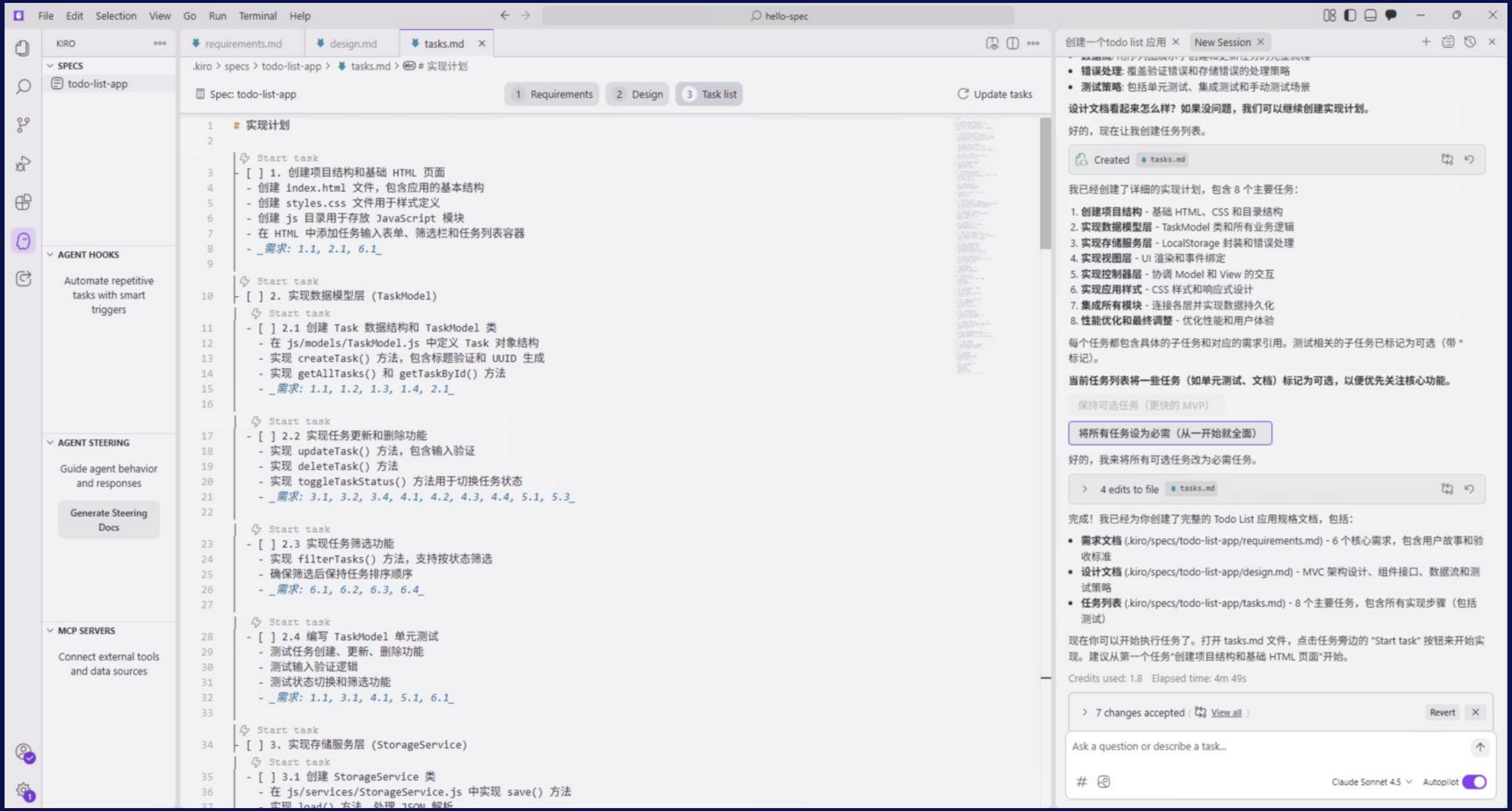
Created `design.md`

我已经创建了详细的设计文档，包含：

- **架构设计**: 采用 MVC 模式，使用 Vanilla JavaScript + LocalStorage
- **组件设计**: 包括 TaskModel、StorageService、TaskController 和 TaskView
- **数据模型**: 定义了 Task 对象结构和所有接口方法
- **UI 设计**: 包含输入表单、筛选栏和任务列表的布局
- **数据流**: 用序列图展示了创建和更新任务的完整流程
- **错误处理**: 覆盖验证错误和存储错误的处理策略
- **测试策略**: 包括单元测试、集成测试和手动测试场景

设计文档看起来怎么样？如果没问题，我们可以继续创建实现计划。

Kiro



```
# 实现计划

- [ ] 1. 创建项目结构和基础 HTML 页面
  - 创建 index.html 文件，包含应用的基本结构
  - 创建 styles.css 文件用于样式定义
  - 创建 js 目录用于存放 JavaScript 模块
  - 在 HTML 中添加任务输入表单、筛选栏和任务列表容器
  - _需求: 1.1, 2.1, 6.1_

- [ ] 2. 实现数据模型层 (TaskModel)
  - [ ] 2.1 创建 Task 数据结构和 TaskModel 类
    - 在 js/models/TaskModel.js 中定义 Task 对象结构
    - 实现 createTask() 方法，包含标题验证和 UUID 生成
    - 实现 getAllTasks() 和 getTaskById() 方法
    - _需求: 1.1, 1.2, 1.3, 1.4, 2.1_

  - [ ] 2.2 实现任务更新和删除功能
    - 实现 updateTask() 方法，包含输入验证
    - 实现 deleteTask() 方法
    - 实现 toggleTaskStatus() 方法用于切换任务状态
    - _需求: 3.1, 3.2, 3.4, 4.1, 4.2, 4.3, 4.4, 5.1, 5.3_

  - [ ] 2.3 实现任务筛选功能
    - 实现 filterTasks() 方法，支持按状态筛选
    - 确保筛选后保持任务排序顺序
    - _需求: 6.1, 6.2, 6.3, 6.4_

  - [ ] 2.4 编写 TaskModel 单元测试
    - 测试任务创建、更新、删除功能
    - 测试输入验证逻辑
    - 测试状态切换和筛选功能
    - _需求: 1.1, 3.1, 4.1, 5.1, 6.1_

- [ ] 3. 实现存储服务层 (StorageService)
  - [ ] 3.1 创建 StorageService 类
    - 在 js/services/StorageService.js 中实现 save() 方法
```

- **错误处理**: 覆盖验证错误和存储错误的处理策略
- **测试策略**: 包括单元测试、集成测试和手动测试场景

设计文档看起来怎么样？如果没问题，我们可以继续创建实现计划。

好的，现在让我创建任务列表。

Created `tasks.md`

我已经创建了详细的实现计划，包含 8 个主要任务：

1. **创建项目结构** - 基础 HTML、CSS 和目录结构
2. **实现数据模型层** - TaskModel 类和所有业务逻辑
3. **实现存储服务层** - LocalStorage 封装和错误处理
4. **实现视图层** - UI 渲染和事件绑定
5. **实现控制器层** - 协调 Model 和 View 的交互
6. **实现应用样式** - CSS 样式和响应式设计
7. **集成所有模块** - 连接各层并实现数据持久化
8. **性能优化和最终调整** - 优化性能和用户体验

每个任务都包含具体的子任务和对应的需求引用。测试相关的子任务已标记为可选（带 * 标记）。

当前任务列表将一些任务（如单元测试、文档）标记为可选，以便优先关注核心功能。

保持可选任务（更快的 MVP）

将所有任务设为必需（从一开始就全面）

好的，我来将所有可选任务改为必需任务。

4 edits to file `tasks.md`

完成！我已经为你创建了完整的 Todo List 应用规格文档，包括：

- **需求文档** (.kiro/specs/todo-list-app/requirements.md) - 6 个核心需求，包含用户故事和验收标准
- **设计文档** (.kiro/specs/todo-list-app/design.md) - MVC 架构设计、组件接口、数据流和测试策略
- **任务列表** (.kiro/specs/todo-list-app/tasks.md) - 8 个主要任务，包含所有实现步骤（包括测试）

现在你可以开始执行任务了。打开 tasks.md 文件，点击任务旁边的 "Start task" 按钮来开始实现。建议从第一个任务"创建项目结构和基础 HTML 页面"开始。

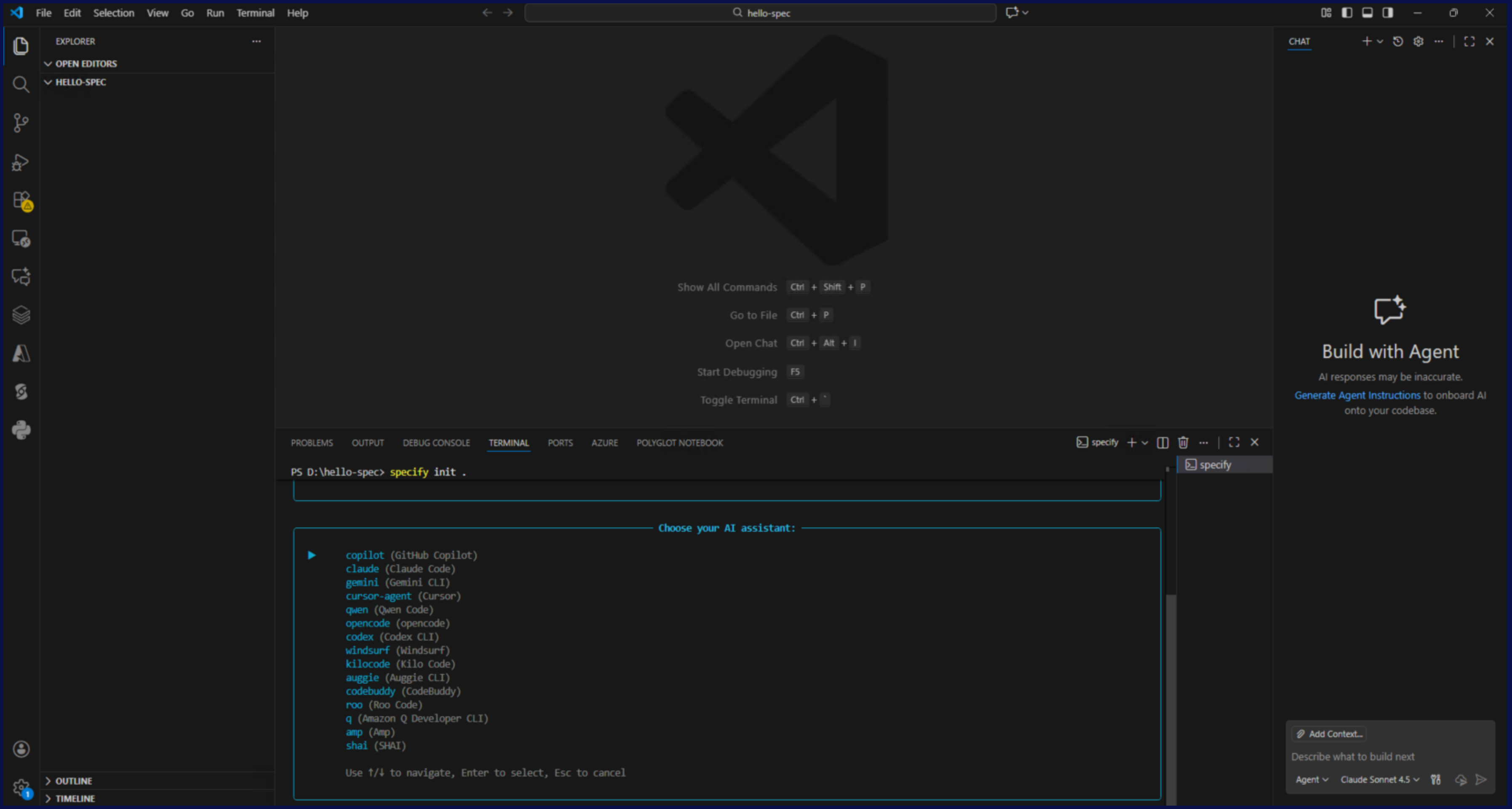
Credits used: 1.8 Elapsed time: 4m 49s

7 changes accepted (View all) Revert

Ask a question or describe a task...

Claude Sonnet 4.5 Autopilot

Spec Kit



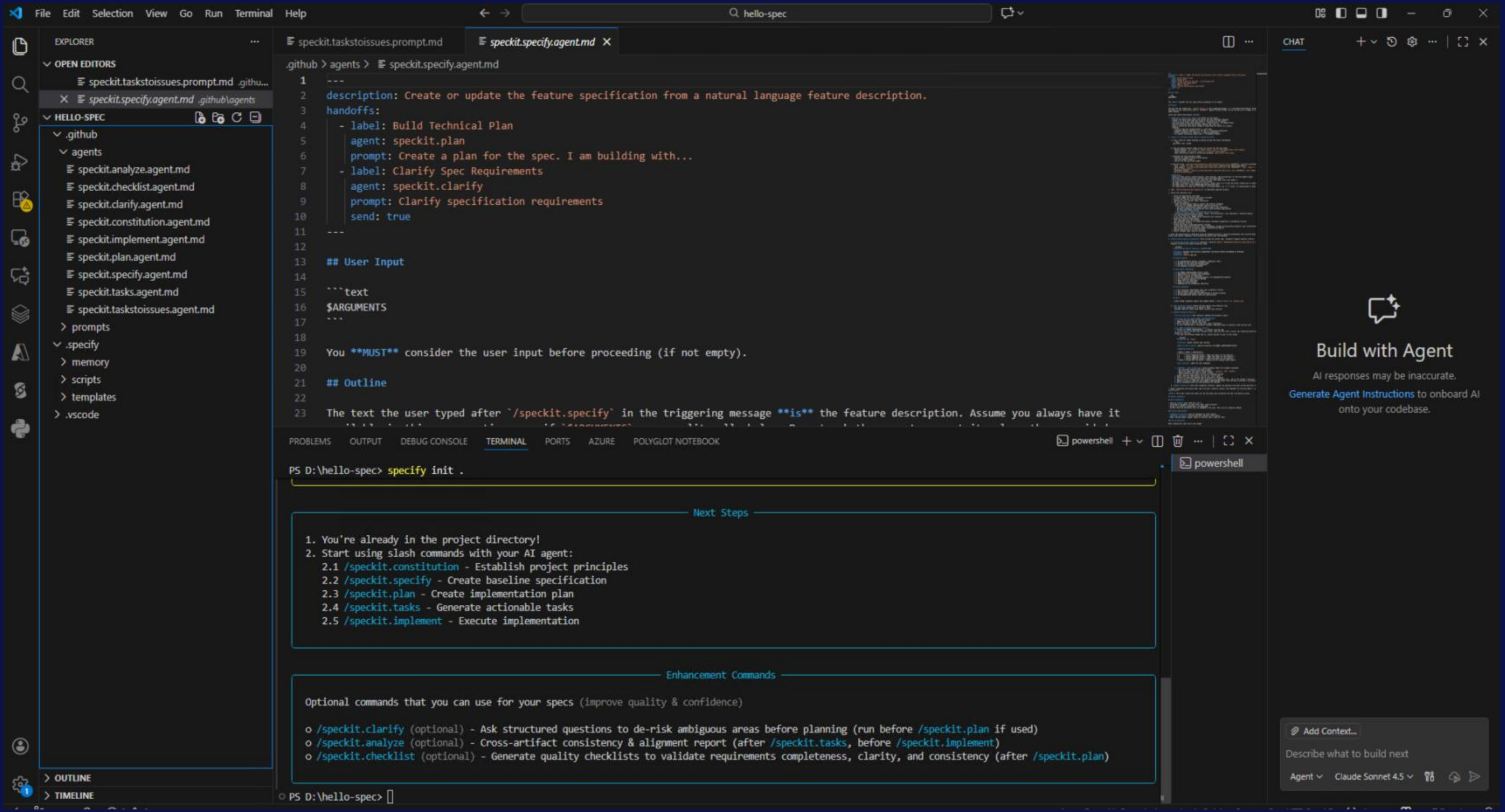
```
PS D:\hello-spec> specify init .

──────────────────── Choose your AI assistant: ────────────────────

 ▶    copilot (GitHub Copilot)
      claude (Claude Code)
      gemini (Gemini CLI)
      cursor-agent (Cursor)
      qwen (Qwen Code)
      opencode (opencode)
      codex (Codex CLI)
      windsurf (Windsurf)
      kilocode (Kilo Code)
      auggie (Auggie CLI)
      codebuddy (CodeBuddy)
      roo (Roo Code)
      q (Amazon Q Developer CLI)
      amp (Amp)
      shai (SHAI)

      Use ↑/↓ to navigate, Enter to select, Esc to cancel
```

Build with Agent

AI responses may be inaccurate.

Generate Agent Instructions to onboard AI onto your codebase.

Spec Kit


File Edit Selection View Go Run Terminal Help
hello-spec
EXPLORER
OPEN EDITORS
speckit.taskstoissues.prompt.md .githu...
speckit.specify.agent.md .github\agents
HELLO-SPEC
.github
agents
speckit.analyze.agent.md
speckit.checklist.agent.md
speckit.clarify.agent.md
speckit.constitution.agent.md
speckit.implement.agent.md
speckit.plan.agent.md
speckit.specify.agent.md
speckit.tasks.agent.md
speckit.taskstoissues.agent.md
prompts
.specify
memory
scripts
templates
.vscode
OUTLINE
TIMELINE
speckit.taskstoissues.prompt.md
speckit.specify.agent.md
.github > agents > speckit.specify.agent.md
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE POLYGLOT NOTEBOOK
powershell
CHAT
Add Context...
Describe what to build next
Agent
Claude Sonnet 4.5

~~~
---
description: Create or update the feature specification from a natural language feature description.
handoffs:
  - label: Build Technical Plan
    agent: speckit.plan
    prompt: Create a plan for the spec. I am building with...
  - label: Clarify Spec Requirements
    agent: speckit.clarify
    prompt: Clarify specification requirements
    send: true
---

## User Input

```text
$ARGUMENTS
```

You **MUST** consider the user input before proceeding (if not empty).

## Outline

The text the user typed after `/speckit.specify` in the triggering message **is** the feature description. Assume you always have it
~~~

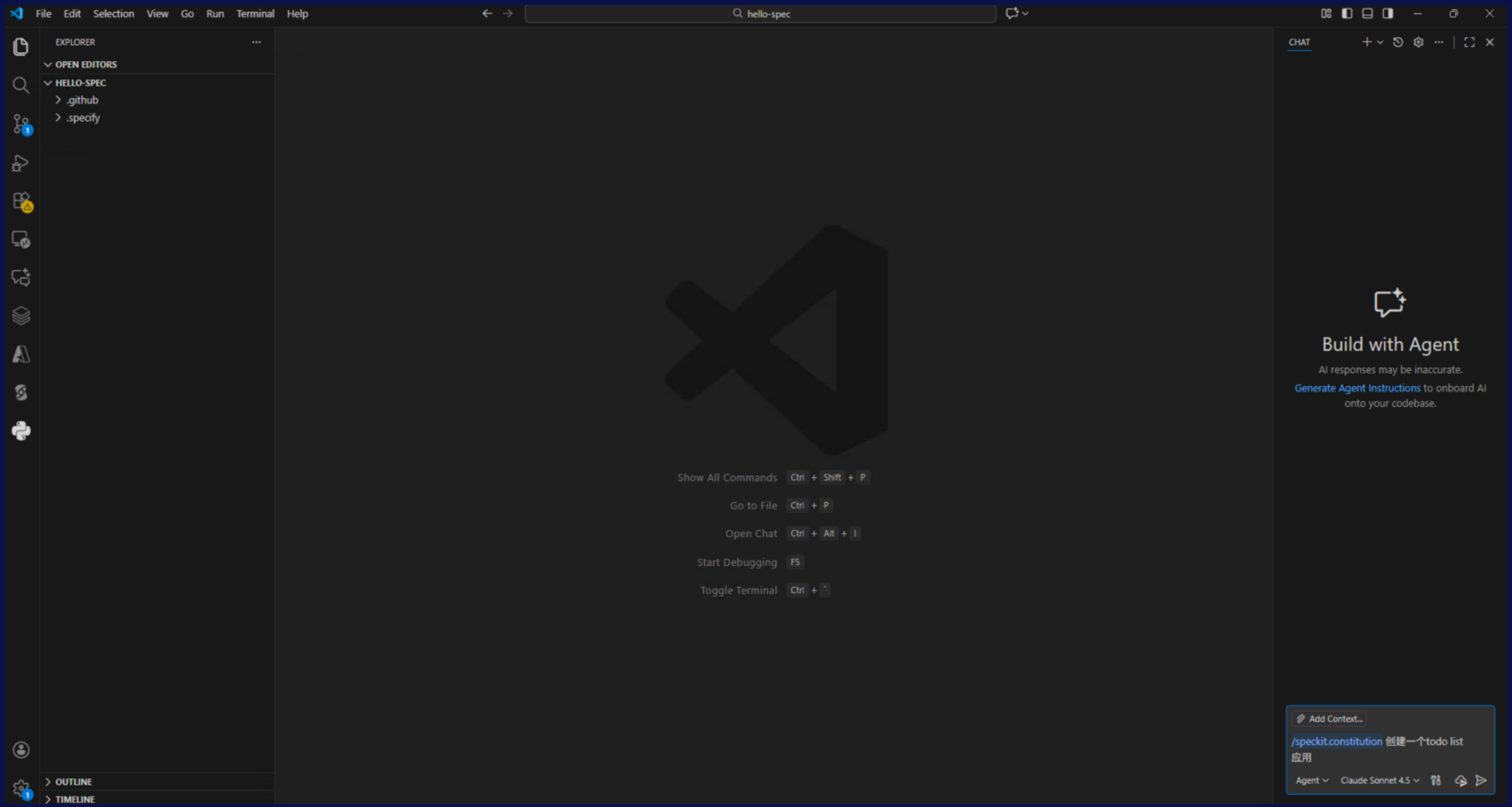
```
PS D:\hello-spec> specify init .

                                        Next Steps

 1. You're already in the project directory!
 2. Start using slash commands with your AI agent:
    2.1 /speckit.constitution - Establish project principles
    2.2 /speckit.specify - Create baseline specification
    2.3 /speckit.plan - Create implementation plan
    2.4 /speckit.tasks - Generate actionable tasks
    2.5 /speckit.implement - Execute implementation

                                  Enhancement Commands

 Optional commands that you can use for your specs (improve quality & confidence)

 o /speckit.clarify (optional) - Ask structured questions to de-risk ambiguous areas before planning (run before /speckit.plan if used)
 o /speckit.analyze (optional) - Cross-artifact consistency & alignment report (after /speckit.tasks, before /speckit.implement)
 o /speckit.checklist (optional) - Generate quality checklists to validate requirements completeness, clarity, and consistency (after /speckit.plan)

PS D:\hello-spec> 
```

Build with Agent

AI responses may be inaccurate.

Generate Agent Instructions to onboard AI onto your codebase.

Spec Kit



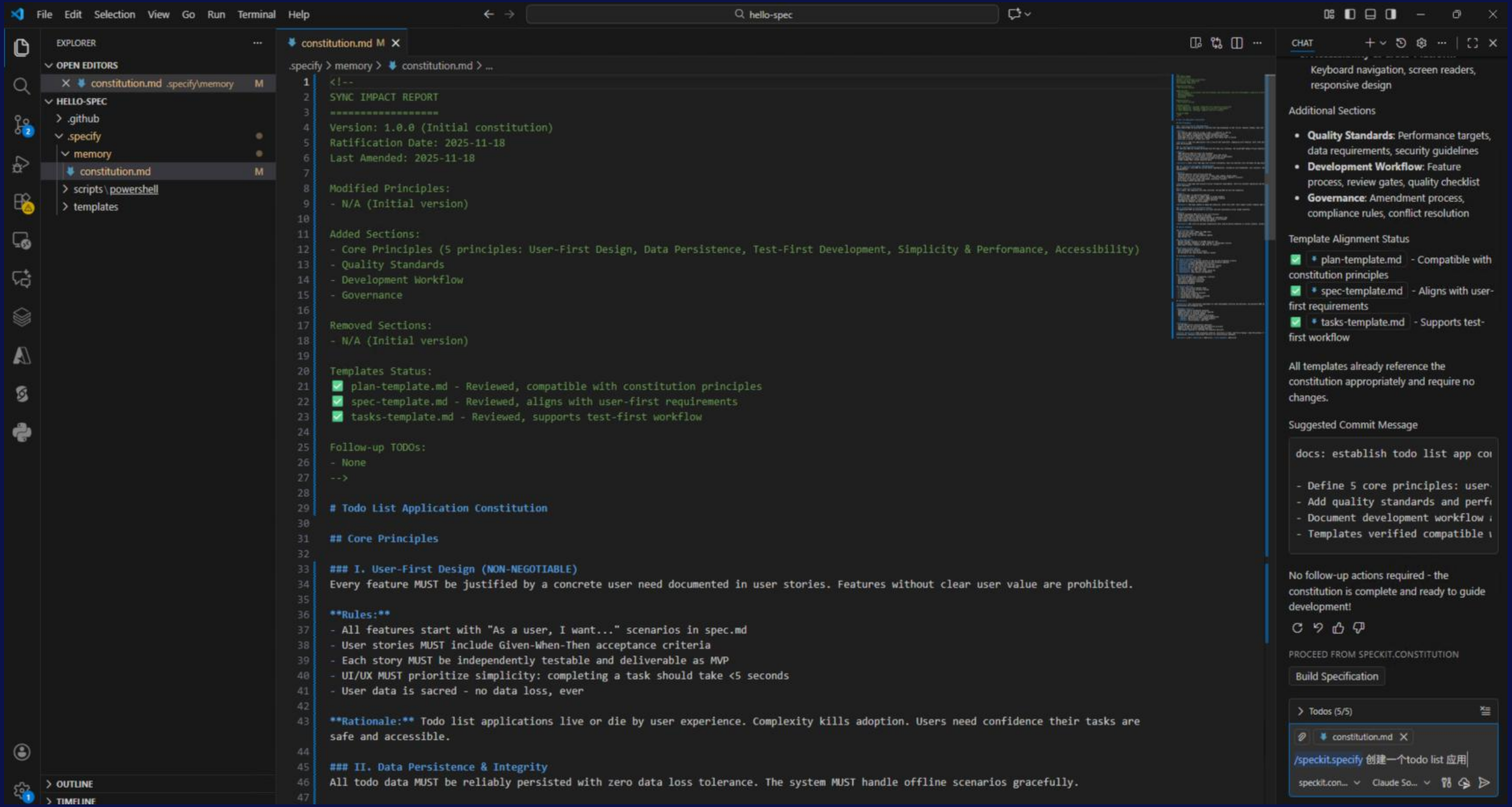
Build with Agent

AI responses may be inaccurate.

Generate Agent Instructions to onboard AI onto your codebase.

/speckit.constitution 创建一个todo list 应用

Spec Kit



```
<!--
SYNC IMPACT REPORT
==================
Version: 1.0.0 (Initial constitution)
Ratification Date: 2025-11-18
Last Amended: 2025-11-18

Modified Principles:
- N/A (Initial version)

Added Sections:
- Core Principles (5 principles: User-First Design, Data Persistence, Test-First Development, Simplicity & Performance, Accessibility)
- Quality Standards
- Development Workflow
- Governance

Removed Sections:
- N/A (Initial version)

Templates Status:
✅ plan-template.md - Reviewed, compatible with constitution principles
✅ spec-template.md - Reviewed, aligns with user-first requirements
✅ tasks-template.md - Reviewed, supports test-first workflow

Follow-up TODOs:
- None
-->

# Todo List Application Constitution

## Core Principles

### I. User-First Design (NON-NEGOTIABLE)
Every feature MUST be justified by a concrete user need documented in user stories. Features without clear user value are prohibited.

**Rules:**
- All features start with "As a user, I want..." scenarios in spec.md
- User stories MUST include Given-When-Then acceptance criteria
- Each story MUST be independently testable and deliverable as MVP
- UI/UX MUST prioritize simplicity: completing a task should take <5 seconds
- User data is sacred - no data loss, ever

**Rationale:** Todo list applications live or die by user experience. Complexity kills adoption. Users need confidence their tasks are safe and accessible.

### II. Data Persistence & Integrity
All todo data MUST be reliably persisted with zero data loss tolerance. The system MUST handle offline scenarios gracefully.

```

Keyboard navigation, screen readers, responsive design

Additional Sections

- **Quality Standards**: Performance targets, data requirements, security guidelines
- **Development Workflow**: Feature process, review gates, quality checklist
- **Governance**: Amendment process, compliance rules, conflict resolution

Template Alignment Status

✅ plan-template.md - Compatible with constitution principles
✅ spec-template.md - Aligns with user-first requirements
✅ tasks-template.md - Supports test-first workflow

All templates already reference the constitution appropriately and require no changes.

Suggested Commit Message

```
docs: establish todo list app co

- Define 5 core principles: user
- Add quality standards and perf
- Document development workflow
- Templates verified compatible
```

No follow-up actions required - the constitution is complete and ready to guide development!

PROCEED FROM SPECKIT.CONSTITUTION

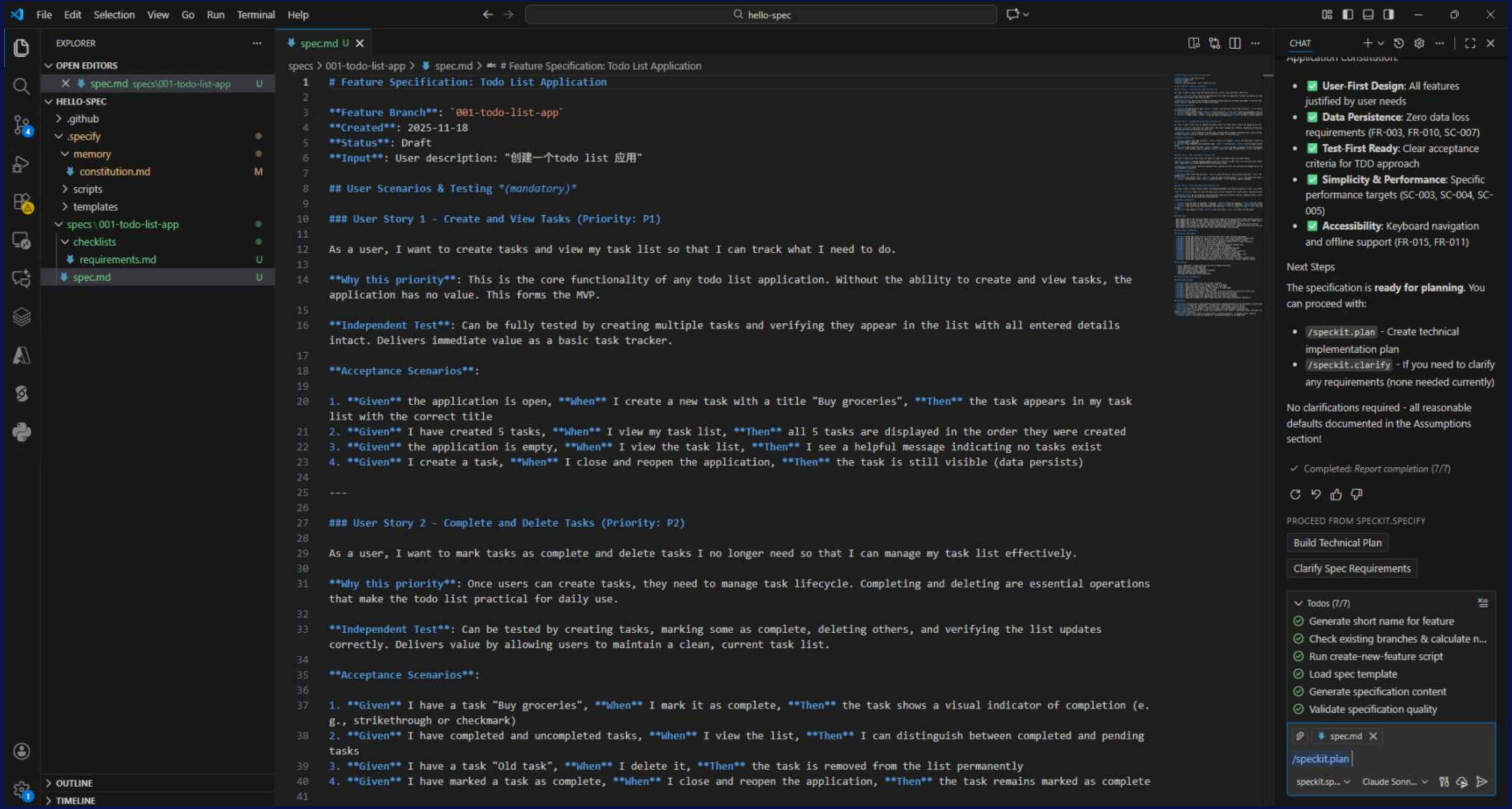
Build Specification

Todos (5/5)

constitution.md

/speckit.specify 创建一个todo list 应用

Spec Kit



```
# Feature Specification: Todo List Application

**Feature Branch**: `001-todo-list-app`
**Created**: 2025-11-18
**Status**: Draft
**Input**: User description: "创建一个todo list 应用"

## User Scenarios & Testing *(mandatory)*

### User Story 1 - Create and View Tasks (Priority: P1)

As a user, I want to create tasks and view my task list so that I can track what I need to do.

**Why this priority**: This is the core functionality of any todo list application. Without the ability to create and view tasks, the application has no value. This forms the MVP.

**Independent Test**: Can be fully tested by creating multiple tasks and verifying they appear in the list with all entered details intact. Delivers immediate value as a basic task tracker.

**Acceptance Scenarios**:

1. **Given** the application is open, **When** I create a new task with a title "Buy groceries", **Then** the task appears in my task list with the correct title
2. **Given** I have created 5 tasks, **When** I view my task list, **Then** all 5 tasks are displayed in the order they were created
3. **Given** the application is empty, **When** I view the task list, **Then** I see a helpful message indicating no tasks exist
4. **Given** I create a task, **When** I close and reopen the application, **Then** the task is still visible (data persists)

---

### User Story 2 - Complete and Delete Tasks (Priority: P2)

As a user, I want to mark tasks as complete and delete tasks I no longer need so that I can manage my task list effectively.

**Why this priority**: Once users can create tasks, they need to manage task lifecycle. Completing and deleting are essential operations that make the todo list practical for daily use.

**Independent Test**: Can be tested by creating tasks, marking some as complete, deleting others, and verifying the list updates correctly. Delivers value by allowing users to maintain a clean, current task list.

**Acceptance Scenarios**:

1. **Given** I have a task "Buy groceries", **When** I mark it as complete, **Then** the task shows a visual indicator of completion (e.g., strikethrough or checkmark)
2. **Given** I have completed and uncompleted tasks, **When** I view the list, **Then** I can distinguish between completed and pending tasks
3. **Given** I have a task "Old task", **When** I delete it, **Then** the task is removed from the list permanently
4. **Given** I have marked a task as complete, **When** I close and reopen the application, **Then** the task remains marked as complete
```

- ✅ **User-First Design**: All features justified by user needs
- ✅ **Data Persistence**: Zero data loss requirements (FR-003, FR-010, SC-007)
- ✅ **Test-First Ready**: Clear acceptance criteria for TDD approach
- ✅ **Simplicity & Performance**: Specific performance targets (SC-003, SC-004, SC-005)
- ✅ **Accessibility**: Keyboard navigation and offline support (FR-015, FR-011)

Next Steps

The specification is **ready for planning**. You can proceed with:

- `/speckit.plan` - Create technical implementation plan
- `/speckit.clarify` - If you need to clarify any requirements (none needed currently)

No clarifications required - all reasonable defaults documented in the Assumptions section!

✓ Completed: *Report completion* (7/7)

PROCEED FROM SPECKIT.SPECIFY

Build Technical Plan

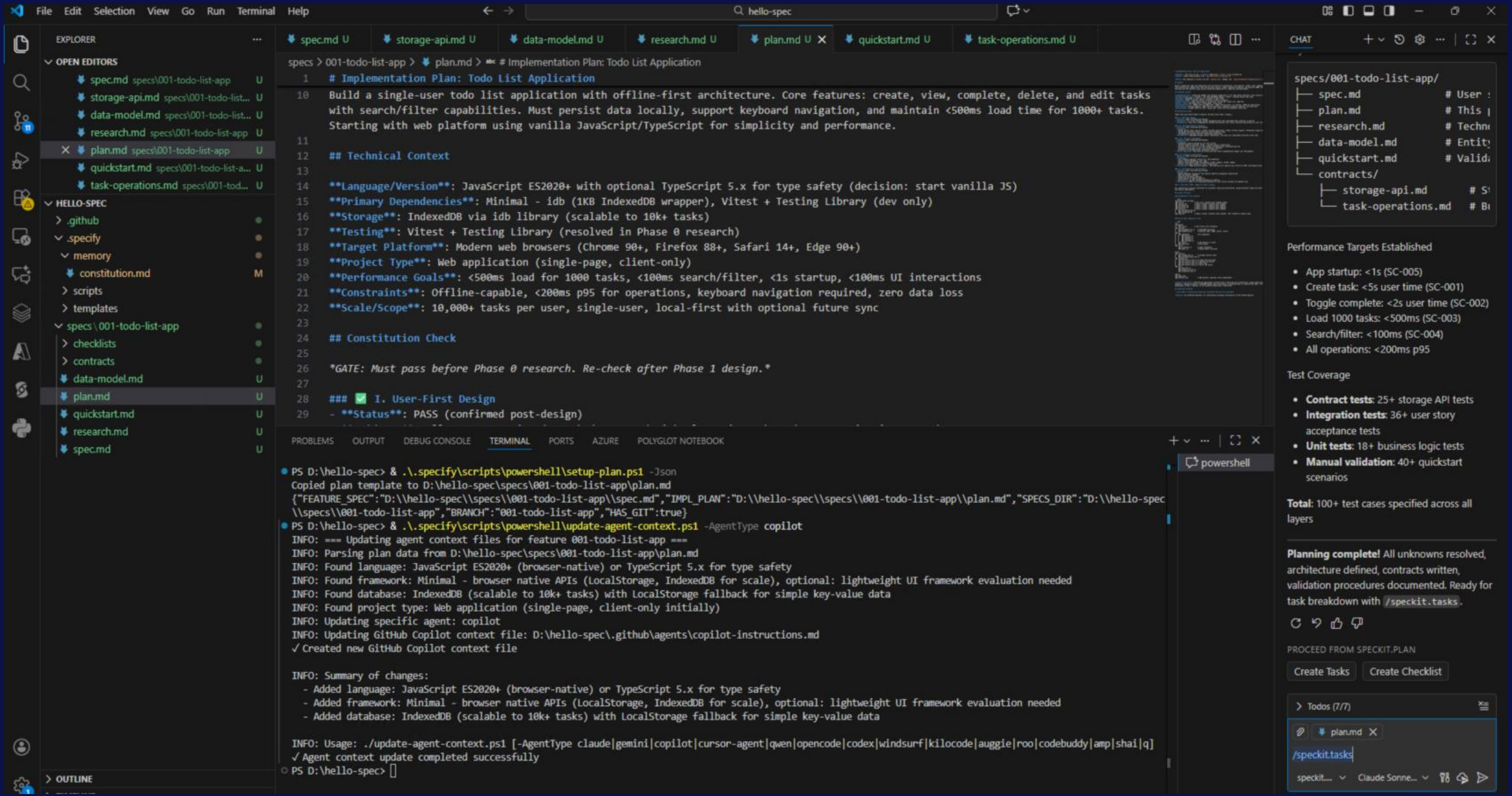
Clarify Spec Requirements

Todos (7/7)
- ✓ Generate short name for feature
- ✓ Check existing branches & calculate n...
- ✓ Run create-new-feature script
- ✓ Load spec template
- ✓ Generate specification content
- ✓ Validate specification quality

spec.md

/speckit.plan

Spec Kit



```
# Implementation Plan: Todo List Application
Build a single-user todo list application with offline-first architecture. Core features: create, view, complete, delete, and edit tasks
with search/filter capabilities. Must persist data locally, support keyboard navigation, and maintain <500ms load time for 1000+ tasks.
Starting with web platform using vanilla JavaScript/TypeScript for simplicity and performance.

## Technical Context

**Language/Version**: JavaScript ES2020+ with optional TypeScript 5.x for type safety (decision: start vanilla JS)
**Primary Dependencies**: Minimal - idb (1KB IndexedDB wrapper), Vitest + Testing Library (dev only)
**Storage**: IndexedDB via idb library (scalable to 10k+ tasks)
**Testing**: Vitest + Testing Library (resolved in Phase 0 research)
**Target Platform**: Modern web browsers (Chrome 90+, Firefox 88+, Safari 14+, Edge 90+)
**Project Type**: Web application (single-page, client-only)
**Performance Goals**: <500ms load for 1000 tasks, <100ms search/filter, <1s startup, <100ms UI interactions
**Constraints**: Offline-capable, <200ms p95 for operations, keyboard navigation required, zero data loss
**Scale/Scope**: 10,000+ tasks per user, single-user, local-first with optional future sync

## Constitution Check

*GATE: Must pass before Phase 0 research. Re-check after Phase 1 design.*

### ✅ I. User-First Design
- **Status**: PASS (confirmed post-design)
```

```
PS D:\hello-spec> & .\.specify\scripts\powershell\setup-plan.ps1 -Json
Copied plan template to D:\hello-spec\specs\001-todo-list-app\plan.md
{"FEATURE_SPEC":"D:\\hello-spec\\specs\\001-todo-list-app\\spec.md","IMPL_PLAN":"D:\\hello-spec\\specs\\001-todo-list-app\\plan.md","SPECS_DIR":"D:\\hello-spec
\\specs\\001-todo-list-app","BRANCH":"001-todo-list-app","HAS_GIT":true}
PS D:\hello-spec> & .\.specify\scripts\powershell\update-agent-context.ps1 -AgentType copilot
INFO: === Updating agent context files for feature 001-todo-list-app ===
INFO: Parsing plan data from D:\hello-spec\specs\001-todo-list-app\plan.md
INFO: Found language: JavaScript ES2020+ (browser-native) or TypeScript 5.x for type safety
INFO: Found framework: Minimal - browser native APIs (LocalStorage, IndexedDB for scale), optional: lightweight UI framework evaluation needed
INFO: Found database: IndexedDB (scalable to 10k+ tasks) with LocalStorage fallback for simple key-value data
INFO: Found project type: Web application (single-page, client-only initially)
INFO: Updating specific agent: copilot
INFO: Updating GitHub Copilot context file: D:\hello-spec\.github\agents\copilot-instructions.md
✓ Created new GitHub Copilot context file

INFO: Summary of changes:
  - Added language: JavaScript ES2020+ (browser-native) or TypeScript 5.x for type safety
  - Added framework: Minimal - browser native APIs (LocalStorage, IndexedDB for scale), optional: lightweight UI framework evaluation needed
  - Added database: IndexedDB (scalable to 10k+ tasks) with LocalStorage fallback for simple key-value data

INFO: Usage: ./update-agent-context.ps1 [-AgentType claude|gemini|copilot|cursor-agent|qwen|opencode|codex|windsurf|kilocode|auggie|roo|codebuddy|amp|shai|q]
✓ Agent context update completed successfully
PS D:\hello-spec> 
```

```
specs/001-todo-list-app/
├── spec.md              # User
├── plan.md              # This
├── research.md          # Techn
├── data-model.md        # Entit
├── quickstart.md        # Valid
└── contracts/
    ├── storage-api.md       # S
    └── task-operations.md   # B
```

Performance Targets Established

- App startup: <1s (SC-005)
- Create task: <5s user time (SC-001)
- Toggle complete: <2s user time (SC-002)
- Load 1000 tasks: <500ms (SC-003)
- Search/filter: <100ms (SC-004)
- All operations: <200ms p95

Test Coverage

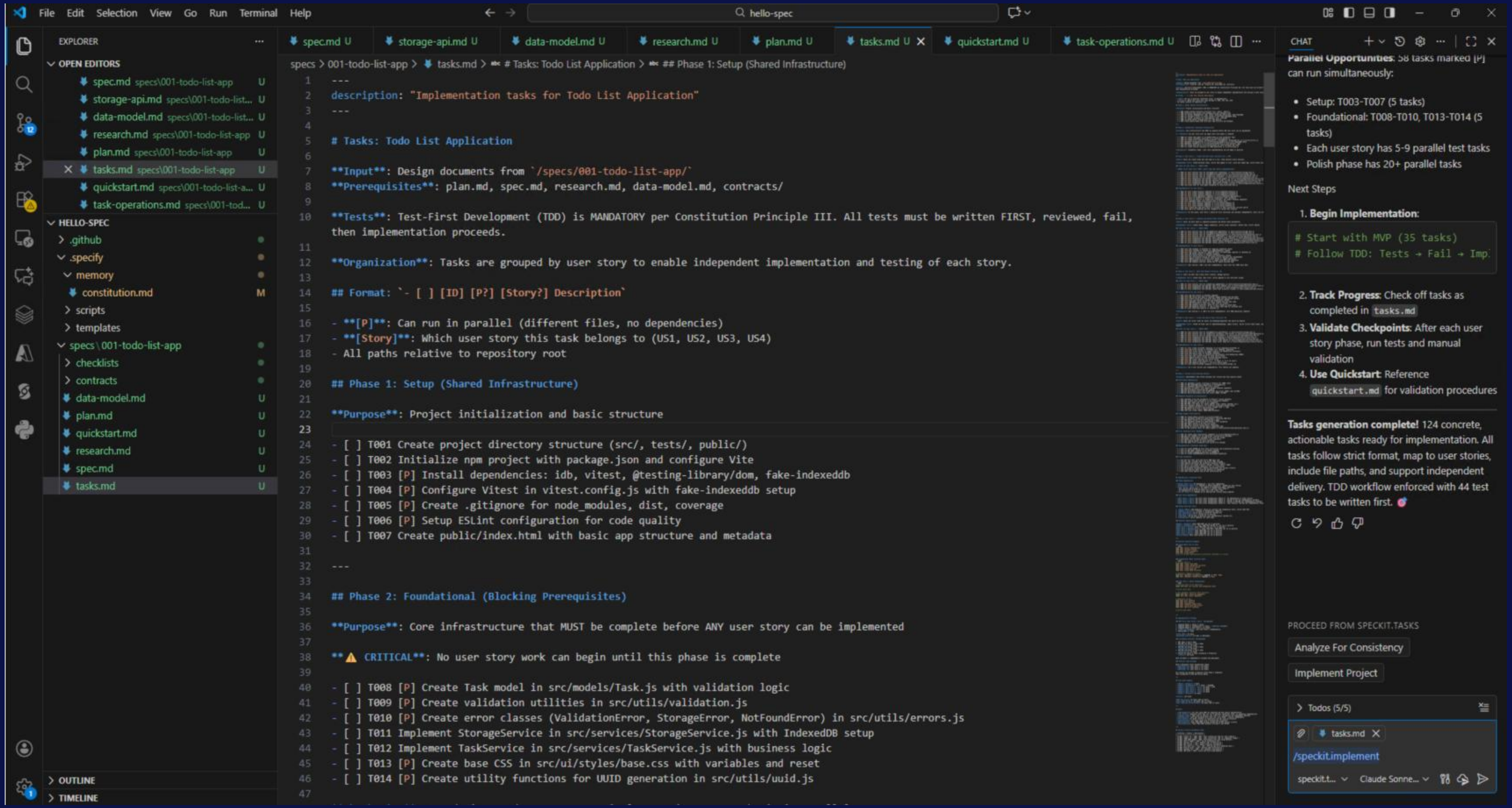
- **Contract tests**: 25+ storage API tests
- **Integration tests**: 36+ user story acceptance tests
- **Unit tests**: 18+ business logic tests
- **Manual validation**: 40+ quickstart scenarios

Total: 100+ test cases specified across all layers

Planning complete! All unknowns resolved, architecture defined, contracts written, validation procedures documented. Ready for task breakdown with `/speckit.tasks`.

/speckit.tasks

Spec Kit


File
Edit
Selection
View
Go
Run
Terminal
Help
hello-spec
EXPLORER
OPEN EDITORS
spec.md specs\001-todo-list-app U
storage-api.md specs\001-todo-list... U
data-model.md specs\001-todo-list... U
research.md specs\001-todo-list-app U
plan.md specs\001-todo-list-app U
tasks.md specs\001-todo-list-app U
quickstart.md specs\001-todo-list-a... U
task-operations.md specs\001-tod... U
HELLO-SPEC
.github
.specify
memory
constitution.md M
scripts
templates
specs \ 001-todo-list-app
checklists
contracts
data-model.md U
plan.md U
quickstart.md U
research.md U
spec.md U
tasks.md U
OUTLINE
TIMELINE
spec.md U
storage-api.md U
data-model.md U
research.md U
plan.md U
tasks.md U
quickstart.md U
task-operations.md U
specs > 001-todo-list-app > tasks.md > # Tasks: Todo List Application > ## Phase 1: Setup (Shared Infrastructure)
CHAT

```
---
description: "Implementation tasks for Todo List Application"
---

# Tasks: Todo List Application

**Input**: Design documents from `/specs/001-todo-list-app/`
**Prerequisites**: plan.md, spec.md, research.md, data-model.md, contracts/

**Tests**: Test-First Development (TDD) is MANDATORY per Constitution Principle III. All tests must be written FIRST, reviewed, fail, then implementation proceeds.

**Organization**: Tasks are grouped by user story to enable independent implementation and testing of each story.

## Format: `- [ ] [ID] [P?] [Story?] Description`

- **[P]**: Can run in parallel (different files, no dependencies)
- **[Story]**: Which user story this task belongs to (US1, US2, US3, US4)
- All paths relative to repository root

## Phase 1: Setup (Shared Infrastructure)

**Purpose**: Project initialization and basic structure

- [ ] T001 Create project directory structure (src/, tests/, public/)
- [ ] T002 Initialize npm project with package.json and configure Vite
- [ ] T003 [P] Install dependencies: idb, vitest, @testing-library/dom, fake-indexeddb
- [ ] T004 [P] Configure Vitest in vitest.config.js with fake-indexeddb setup
- [ ] T005 [P] Create .gitignore for node_modules, dist, coverage
- [ ] T006 [P] Setup ESLint configuration for code quality
- [ ] T007 Create public/index.html with basic app structure and metadata

---

## Phase 2: Foundational (Blocking Prerequisites)

**Purpose**: Core infrastructure that MUST be complete before ANY user story can be implemented

**⚠️ CRITICAL**: No user story work can begin until this phase is complete

- [ ] T008 [P] Create Task model in src/models/Task.js with validation logic
- [ ] T009 [P] Create validation utilities in src/utils/validation.js
- [ ] T010 [P] Create error classes (ValidationError, StorageError, NotFoundError) in src/utils/errors.js
- [ ] T011 Implement StorageService in src/services/StorageService.js with IndexedDB setup
- [ ] T012 Implement TaskService in src/services/TaskService.js with business logic
- [ ] T013 [P] Create base CSS in src/ui/styles/base.css with variables and reset
- [ ] T014 [P] Create utility functions for UUID generation in src/utils/uuid.js
```

Parallel Opportunities: 58 tasks marked [P] can run simultaneously:

- Setup: T003-T007 (5 tasks)
- Foundational: T008-T010, T013-T014 (5 tasks)
- Each user story has 5-9 parallel test tasks
- Polish phase has 20+ parallel tasks

Next Steps

1. **Begin Implementation**:

```
# Start with MVP (35 tasks)
# Follow TDD: Tests → Fail → Imp
```

2. **Track Progress**: Check off tasks as completed in `tasks.md`
3. **Validate Checkpoints**: After each user story phase, run tests and manual validation
4. **Use Quickstart**: Reference `quickstart.md` for validation procedures

Tasks generation complete! 124 concrete, actionable tasks ready for implementation. All tasks follow strict format, map to user stories, include file paths, and support independent delivery. TDD workflow enforced with 44 test tasks to be written first. 🎯

/speckit.implement

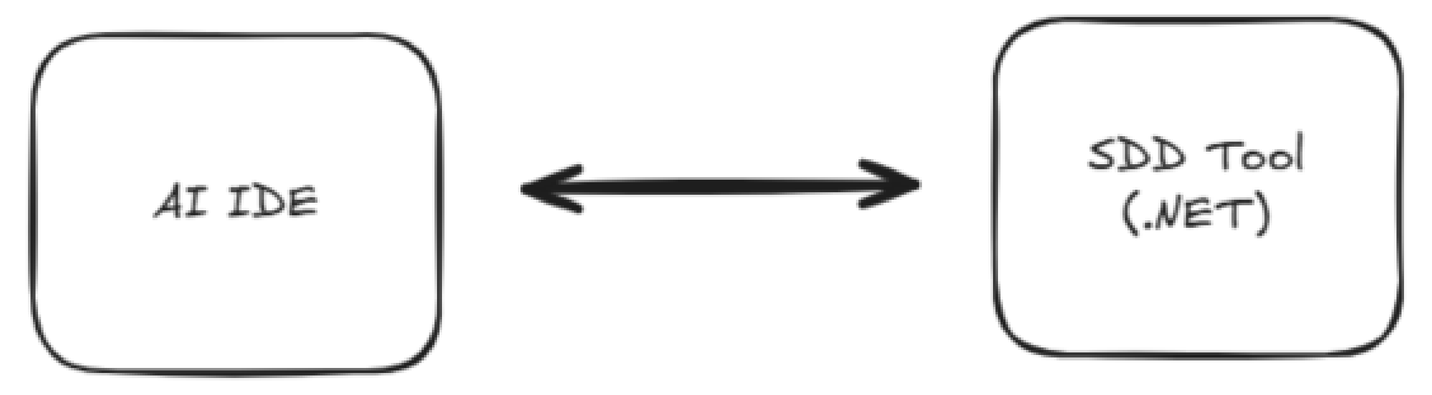
SDD 三要素

功能规格说明
描述需求与架构

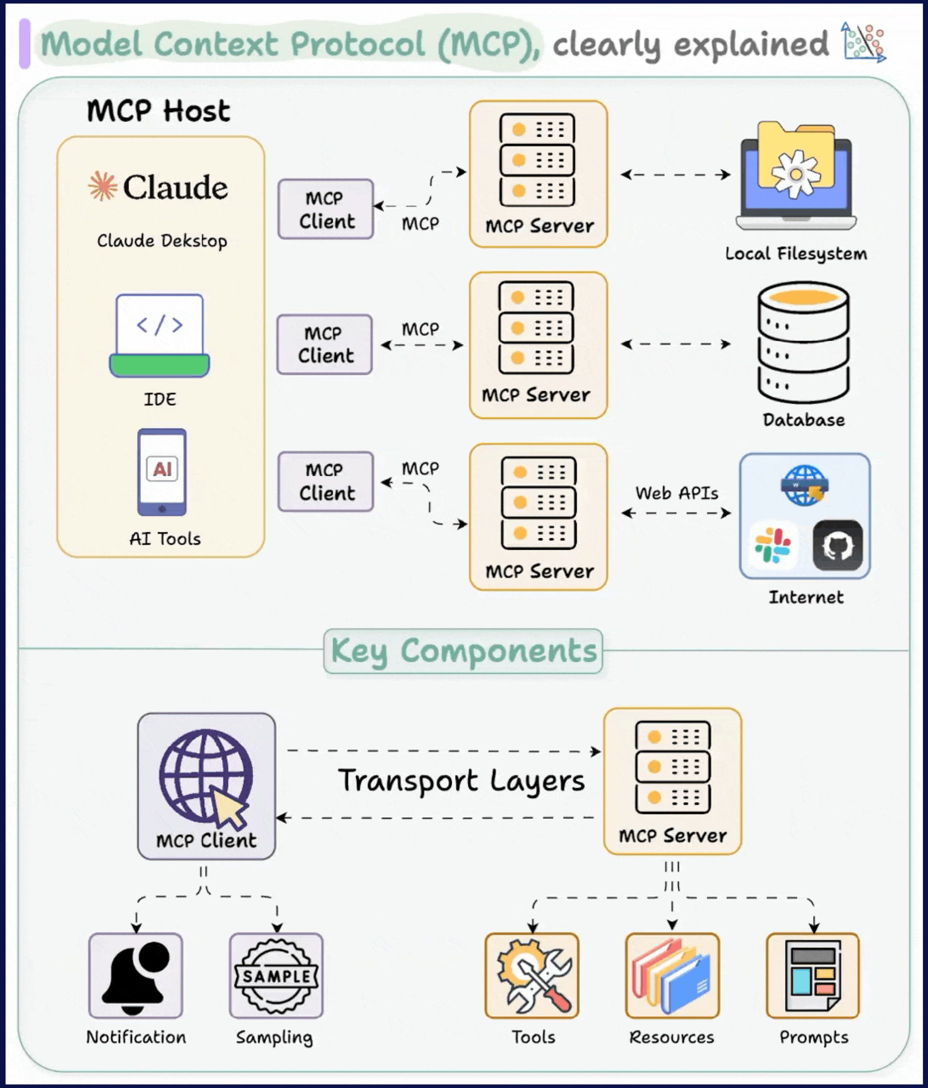
实施计划
推动迭代任务

定制提示词/工具
使工作流程规范化

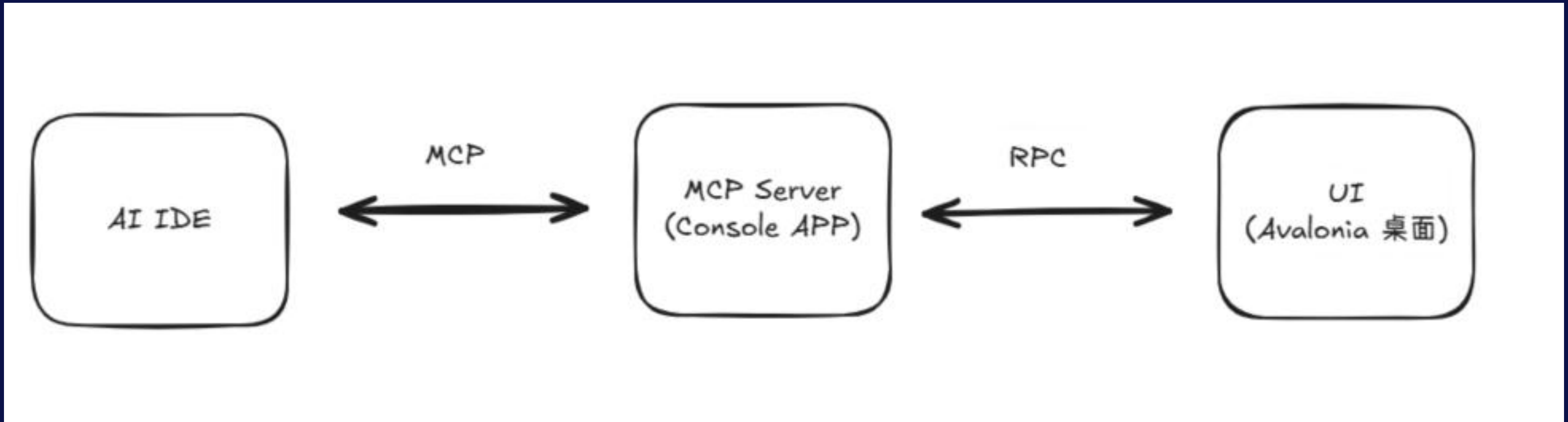
用.NET 实现 SDD



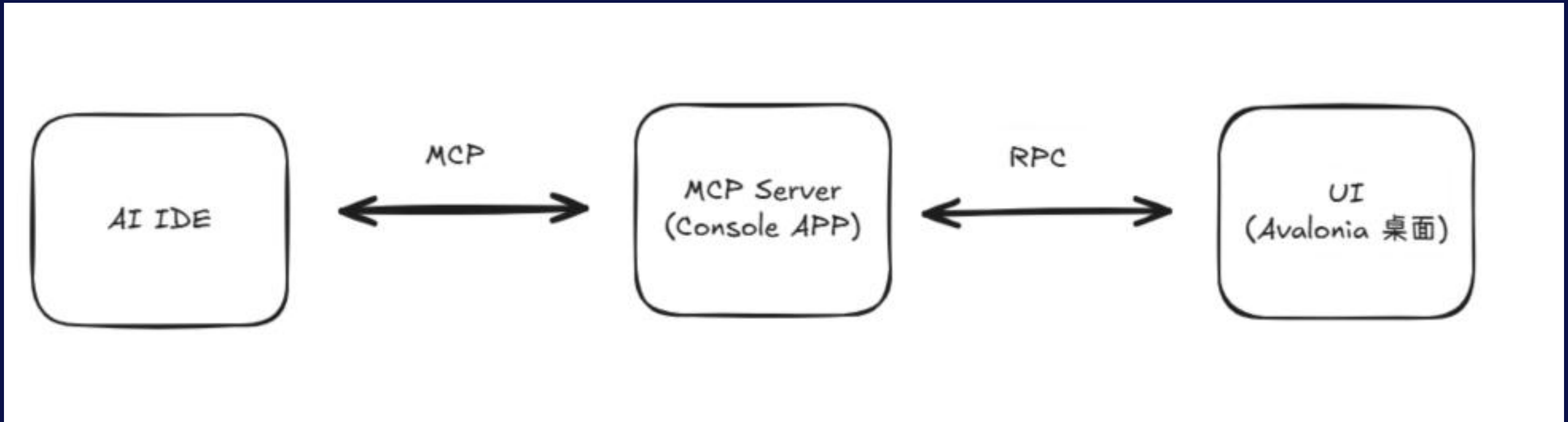
用.NET 实现 SDD

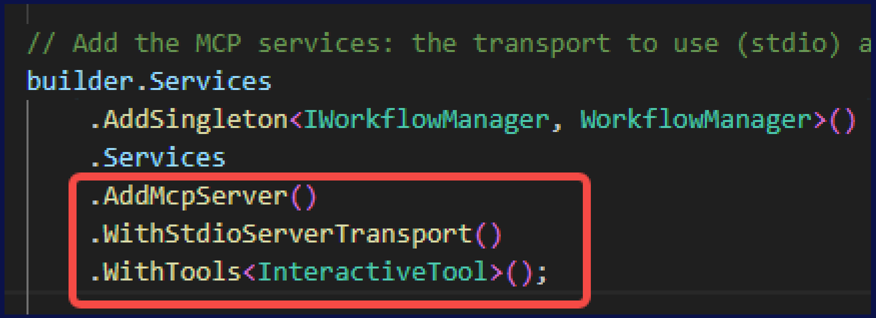

Model Context Protocol (MCP), clearly explained
MCP Host
Claude
Claude Dekstop
IDE
AI Tools
MCP Client
MCP
MCP Server
Local Filesystem
MCP Client
MCP
MCP Server
Database
MCP Client
MCP
MCP Server
Web APIs
Internet
Key Components
MCP Client
Transport Layers
MCP Server
Notification
Sampling
SAMPLE
Tools
Resources
Prompts

用.NET 实现 SDD

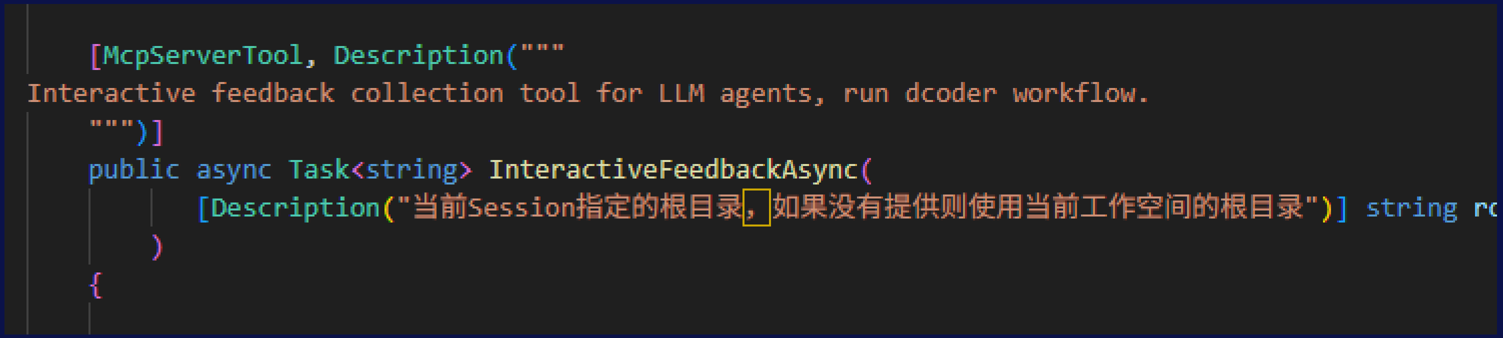


用.NET 实现 SDD



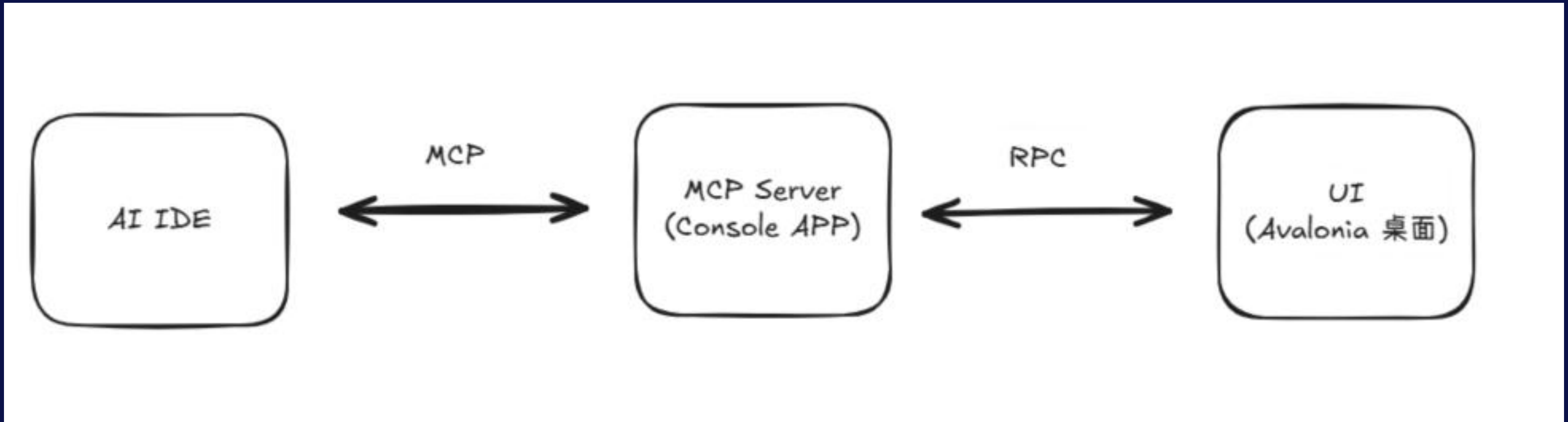


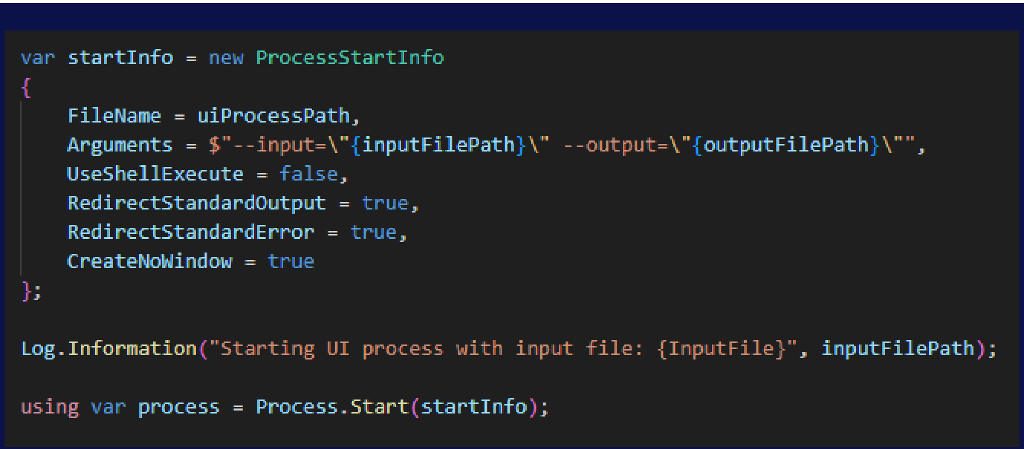
```
// Add the MCP services: the transport to use (stdio) a
builder.Services
    .AddSingleton<IWorkflowManager, WorkflowManager>()
    .Services
    .AddMcpServer()
    .WithStdioServerTransport()
    .WithTools<InteractiveTool>();
```



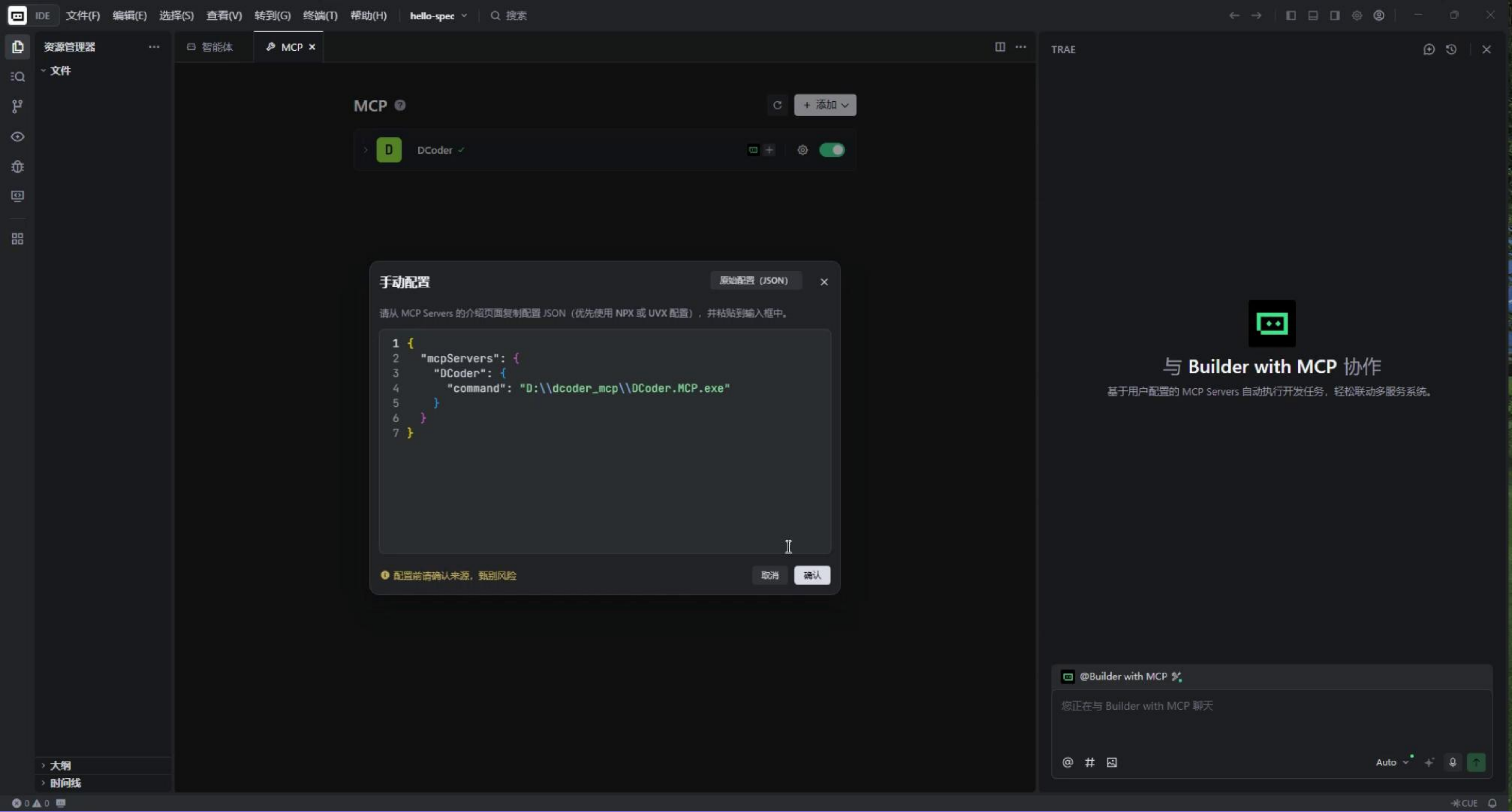
```
    [McpServerTool, Description("""
Interactive feedback collection tool for LLM agents, run dcoder workflow.
    """)]
    public async Task<string> InteractiveFeedbackAsync(
           [Description("当前Session指定的根目录，如果没有提供则使用当前工作空间的根目录")] string r
        )
    {
```

用.NET 实现 SDD





```
var startInfo = new ProcessStartInfo
{
    FileName = uiProcessPath,
    Arguments = $"--input=\"{inputFilePath}\" --output=\"{outputFilePath}\"",
    UseShellExecute = false,
    RedirectStandardOutput = true,
    RedirectStandardError = true,
    CreateNoWindow = true
};

Log.Information("Starting UI process with input file: {InputFile}", inputFilePath);

using var process = Process.Start(startInfo);
```



MCP

手动配置

请从 MCP Servers 的介绍页面复制配置 JSON（优先使用 NPX 或 UVX 配置），并粘贴到输入框中。

```
{
  "mcpServers": {
    "DCoder": {
      "command": "D:\\dcoder_mcp\\DCoder.MCP.exe"
    }
  }
}
```

配置前请确认来源，甄别风险

与 Builder with MCP 协作

基于用户配置的 MCP Servers 自动执行开发任务，轻松联动多服务系统。

您正在与 Builder with MCP 聊天

THANK YOU