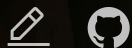


Angular - Agentic AI Engineering

Leveraging AI Tools for Modern Angular Development

Press Space for next page →



Modern Control Flow Syntax

From Structural Directives to Built-in Control Flow

Why Modern Control Flow?

The Angular team introduced a new control flow syntax in Angular 17 to address long-standing pain points with structural directives.



The Problem with Structural Directives

Old Approach

- Required importing CommonModule
- Runtime overhead from directive instantiation
- Complex syntax with microsyntax
- Type narrowing limitations
- Harder to read and maintain

New Approach

- No imports needed
- Compile-time optimization
- Clear, intuitive syntax
- Better type inference
- Improved developer experience

Real-World Impact

```
// Before: Component with multiple imports
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-book-item',
  standalone: true,
  imports: [CommonModule, RouterModule], // Extra dependency
  template: `
    <div *ngIf="book.cover">
      <!-- Template content -->
    </div>
    <div *ngIf="!book.cover">
      <!-- Fallback content -->
    </div>
  `,
})
```

Problem: Every component using `*ngIf`, `*ngFor` needs `CommonModule` import

Developer Experience Improvements

Before

```
<div *ngIf="loading; else content">
  Loading...
</div>

<ng-template #content>

  <div *ngFor="let item of items">
    {{ item }}
  </div>
</ng-template>
```

After

```
```html {1-3|4-8} @if (loading) {
 Loading...
} @else { @for (item of items; track item) {
}} ```
```

**Key Benefit:** More readable, less nesting, clearer intent

# Migration is Essential



## Future-Proof Your Angular Applications



### Performance

Up to 90% faster builds



### Bundle Size

Smaller production builds



### Type Safety

Better type inference

# Core Concepts: Modern Control Flow

## Three Essential Control Flow Blocks

- **@if** - Conditional rendering
- **@for** - List iteration
- **@switch** - Multiple conditions

# @if - Conditional Rendering

## New Syntax (@if)

```
<!-- Simple condition -->
@if (isVisible) {
 <div>Content</div>
}

<!-- With else -->
@if (condition) {
 <div>Primary content</div>
} @else {
 <div>Alternative content</div>
}
```

# @if - Multiple Branches

```
<!-- Old: Complex nested conditions -->
<div *ngIf="status === 'loading'">Loading...</div>
<div *ngIf="status === 'error'">Error occurred</div>
<div *ngIf="status === 'success'">
 <div *ngIf="data">{{ data }}</div>
 <div *ngIf="!data">No data</div>
</div>
```

```
<!-- New: Clear control flow -->
@if (status === 'loading') {
 <div>Loading...</div>
} @else if (status === 'error') {
 <div>Error occurred</div>
} @else if (status === 'success' && data) {
 <div>{{ data }}</div>
} @else {
 <div>No data</div>
}
```

**Benefit:** @else if chains eliminate complex nesting

# @for - List Iteration

## New Syntax (@for)

```
@for (item of items; track item.id) {
 <div>{{ item.name }}</div>
}

<!-- Track is mandatory! -->
@for (item of items; track $index) {
 <div>{{ item.name }}</div>
}
```

**Important:** track is mandatory in @for - prevents performance issues

# @for - Empty State Handling

## ✓ Advantages

- Single block for iteration and empty state
- No duplicate array checks
- Cleaner template structure

## 🎯 Use Cases

- Search results with "no results" message
- Shopping cart items
- Data tables with empty states

# @switch - Multiple Conditions

```
<!-- Old: Nested ngSwitch -->
<div [ngSwitch]="userRole">
 <div *ngSwitchCase="'admin'">Admin Dashboard</div>
 <div *ngSwitchCase="'user'">User Dashboard</div>
 <div *ngSwitchDefault>Guest View</div>
</div>
```

```
<!-- New: Clear switch syntax -->
@switch (userRole) { @case ('admin') {
<div>Admin Dashboard</div>
} @case ('user') {
<div>User Dashboard</div>
} @default {
<div>Guest View</div>
} }
```

# Implicit Variables

## Variable Reference

Variable	Description
<code>\$index</code>	Current index
<code>\$first</code>	Is first item
<code>\$last</code>	Is last item
<code>\$even</code>	Even index
<code>\$odd</code>	Odd index
<code>\$count</code>	Total items


# Type Inference Benefits

```
// Component with type-safe control flow
interface Book {
 id: string;
 title: string;
 author?: string;
}

@Component({
 template: `
 @if (book.author) {
 <!-- TypeScript knows author is defined here -->
 <p>Written by {{ book.author.toUpperCase() }}</p>
 }
 `,
})
```

## Comparison

 **Old \*ngIf:** Limited type narrowing, needed optional chaining

 **New @if:** Full type narrowing, TypeScript understands context

# Summary: Control Flow Blocks

## @if / @else if / @else

- ✓ Replace \*ngIf
- ✓ Multiple branches
- ✓ Type narrowing

## @for / @empty

- ✓ Replace \*ngFor
- ✓ Mandatory tracking
- ✓ Built-in empty state

## @switch / @case / @default

- ✓ Replace ngSwitch
- ✓ Multiple cases
- ✓ Default fallback

# Migration Guide: Step by Step

## Migration Strategy

1. Remove CommonModule imports
2. Replace structural directives with control flow blocks
3. Update trackBy functions to inline tracking
4. Test each component after migration
5. Run automated migration tools

# Step 1: Remove CommonModule

```
// Before
import { CommonModule } from '@angular/common';

@Component({
 selector: 'app-book-item',
 imports: [CommonModule, RouterModule],
 // ...
})

// After
// No CommonModule import needed!

@Component({
 selector: 'app-book-item',
 imports: [RouterModule], // Only keep what you need
 // ...
})
```

✓ **First Win:** Reduced bundle size immediately by removing CommonModule

## Step 2: Migrate \*ngIf to @if

### Real Example from book-item.component.ts


```
<!-- Before: Using *ngIf -->
<img
 *ngIf="book.cover"
 [src]="book.cover"
 [alt]="book.title"
 class="w-full h-full object-contain bg-gray-100"
/>

<!-- After: Using @if -->
@if (book.cover) {
<img
 [src]="book.cover"
 [alt]="book.title"
 class="w-full h-full object-contain bg-gray-100"
/>
}
```

## Step 2: Migrate Negated Conditions

```
<!-- Before: Negated *ngIf -->
<div
 *ngIf="!book.cover"
 class="w-full h-full bg-gray-100 flex items-center justify-center"
>
 No cover available
</div>

<!-- After: Using @if with negation -->
@if (!book.cover) {
 <div class="w-full h-full bg-gray-100 flex items-center justify-center">
 No cover available
 </div>
}
```

 **Tip:** Keep the same condition logic - just wrap in @if

 **Note:** Don't forget the closing brace }

# Step 3: Migrate \*ngFor to @for

## From book-list.component.ts

```
<!-- Before: *ngFor with trackBy function -->
<app-book-item *ngFor="let book of books; trackBy: trackById" [book]="book">
</app-book-item>
```

```
<!-- After: @for with inline track -->
@for (book of books; track book.id) {
<app-book-item [book]="book"></app-book-item>
}
```

```
// Before: Needed trackBy function in component
trackById(index: number, book: Book): string {
 return book.id;
}
```

```
// After: No function needed! Track directly in template
```

# Step 4: Complex Migration Example

## From book-list.component.ts - Loading State

```
<!-- Before: Complex nested structure -->
<div *ngIf="loading" class="flex justify-center items-center py-20">
 <div class="animate-pulse flex flex-col items-center">
 <div class="h-16 w-16 rounded-full border-4 /* ... */ animate-spin"></div>
 <p class="mt-4 text-gray-600">Loading books...</p>
 </div>
</div>

<div *ngIf="!loading" class="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-4">
 <!-- content -->
</div>

<!-- After: Clean @if/else structure -->
@if (loading) {
 <div class="flex justify-center items-center py-20">
 <div class="animate-pulse flex flex-col items-center">
 <div class="h-16 w-16 rounded-full border-4 /* ... */ animate-spin"></div>
 <p class="mt-4 text-gray-600">Loading books...</p>
 </div>
 </div>
} @else {
 <div class="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-4">
```

# Step 5: Migrate Empty States

## From book-list.component.ts - Empty State

```
<!-- Before: Separate empty check -->
<div *ngIf="filteredBooks.length === 0" class="col-span-full">
 <!-- empty state content -->
</div>
<app-book-item
 *ngFor="let book of filteredBooks; trackBy: trackById"
 [book]="book"
></app-book-item>

<!-- After: Built-in @empty -->
@for (book of filteredBooks; track book.id) {
 <app-book-item [book]="book"></app-book-item>
} @empty {
 <div class="col-span-full flex flex-col items-center">
 <svg xmlns="http://www.w3.org/2000/svg" class="h-16 w-16 text-gray-400 mb-4">
 <!-- SVG content -->
 </svg>
 <p class="text-xl font-medium text-gray-600 mb-2">
 {{ searchTerm ? 'No books match your search' : 'No books available' }}
 </p>
 </div>
}
```

## Step 6: Complex Nested Migration

<!-- Before: Deep nesting nightmare -->

```
<div *ngIf="user">
 <div *ngIf="user.isAdmin">
 <div *ngIf="user.permissions?.canEdit">
 Admin Editor
 </div>
 <div *ngIf="!user.permissions?.canEdit">
 Admin Viewer
 </div>
 </div>
 <div *ngIf="!user.isAdmin">
 User
 </div>
</div>
```

<!-- After: Flattened with @else if -->

```
@if (user?.isAdmin && user.permissions?.canEdit) {
 Admin Editor
} @else if (user?.isAdmin) {
 Admin Viewer
} @else if (user) {
 User
} @else {
```

# Common Migration Gotchas

## Common Mistakes

```
<!-- Forgot track -->
@for (item of items) {
<div>{{ item }}</div>
}

<!-- Wrong syntax -->
@if book.title {
<h1>{{ book.title }}</h1>
}
```

## Correct Usage

```
<!-- Always include track -->
@for (item of items; track item.id) {
<div>{{ item }}</div>
}

<!-- Parentheses required -->
@if (book.title) {
<h1>{{ book.title }}</h1>
}
```

# Automated Migration

## Angular CLI Migration Tool

```
ng generate @angular/core:control-flow
```



### What it does

- Automatically converts `*ngIf` to `@if`
- Converts `*ngFor` to `@for` with tracking
- Handles `*ngSwitch` conversions
- Removes unnecessary imports



### What to check after

- Verify track expressions are correct
- Test complex nested conditions
- Check TypeScript compilation
- Run unit tests

# Migration Checklist

- ☐ Remove CommonModule from imports
- ☐ Replace \*ngIf with @if blocks
- ☐ Convert \*ngFor to @for with tracking
- ☐ Update \*ngSwitch to @switch
- ☐ Test each component
- ☐ Run automated migration tool
- ☐ Verify TypeScript compilation
- ☐ Run full test suite

# What If: Edge Cases & Advanced Patterns

## **Exploring Beyond the Basics**

Let's dive into complex scenarios, edge cases, and creative solutions with modern control flow

# What If: Complex Async Patterns?

```
<!-- Traditional Observable approach -->
<div *ngIf="data$ | async as data; else loading">
 <div *ngIf="data.items?.length > 0; else empty">
 <div *ngFor="let item of data.items">{{ item.name }}</div>
 </div>
 <ng-template #empty> No items found </ng-template>
</div>
<ng-template #loading>Loading...</ng-template>

<!-- Modern approach with signals -->
@if (loading()) {
 <div>Loading...</div>
} @else if (error()) {
 <div>Error: {{ error() }}</div>
} @else { @for (item of items(); track item.id) {
 <div>{{ item.name }}</div>
} @empty {
 <div>No items found</div>
} }
```

# What If: Dynamic Component Loading?

```
// Combining @if with dynamic components
@Component({
 template: `
 @switch (componentType) {
 @case ("chart") {
 <app-chart-component [data]="data" />
 }
 @case ("table") {
 <app-table-component [data]="data" />
 }
 @case ("grid") {
 <app-grid-component [data]="data" />
 }
 @default {
 <div>Unknown component type</div>
 }
 }
 `,
})
export class DynamicViewComponent {
 componentType = signal<string>("chart");
 data = signal<any[]>([]);

 switchView(type: string) {
```

# What If: Nested Loops with Filtering?

```
<!-- Complex nested structure -->
@for (category of categories(); track category.id) {
 <h2>{{ category.name }}</h2>
 @for (item of category.items; track item.id) { @if (item.isActive && item.price
 > minPrice) {
 <div class="item">{{ item.name }} - ${{ item.price }}</div>
 } } @empty {
 <p>No active items in this category</p>
 }
```

```
<!-- Alternative: Pre-filter in component -->
@for (category of filteredCategories(); track category.id) {
 <h2>{{ category.name }}</h2>
 @for (item of category.activeItems; track item.id) {
 <div class="item">{{ item.name }} - ${{ item.price }}</div>
 }
```

**Best Practice:** Pre-filter data in computed signals for better performance

# What If: Form Validation States?

```
<!-- Multi-state form field validation -->
<div class="form-field">
 <input
 #emailInput
 type="email"
 [(ngModel)]="email"
 (blur)="validateEmail()"
 />

 @if (emailInput.touched) { @if (!email()) {
 Email is required
 } @else if (!isValidEmail(email())) {
 Invalid email format
 } @else if (isCheckingAvailability()) {
 Checking availability...
 } @else if (!isEmailAvailable()) {
 Email already taken
 } @else {
 ✓ Email available
 } }
</div>
```

# What If: Recursive Templates?

```
<!-- Tree structure with recursive rendering -->
@Component({ selector: 'app-tree-node', template: `
<div class="node">

 @if (hasChildren()) { {{ expanded() ? '▼' : '►' }} } {{ node.name }}

 @if (expanded() && hasChildren()) {
 <div class="children">
 @for (child of node.children; track child.id) {
 <app-tree-node [node]="child" />
 }
 </div>
 }
</div>
` }) export class TreeNodeComponent { @Input() node!: TreeNode; expanded =
signal(false); hasChildren = computed(() => this.node.children &&
this.node.children.length > 0); toggle() { this.expanded.update(v => !v); } }
```

# What If: Performance Optimization?

## Avoid

```
<!-- Expensive computation in template -->
@for (item of items; track item.id) {
 @if (complexCalculation(item) > threshold) {
 <div>{{ item.name }}</div>
 }
}
```

```
// Use computed signals for derived state
processedItems = computed(() =>
 this.items().map((item) => ({
 ...item,
 meetsThreshold: this.complexCalculation(item) > this.threshold(),
 })),
);
```

## Prefer

```
<!-- Pre-computed values -->
@for (item of processedItems(); track item.id) {
 @if (item.meetsThreshold) {
 <div>{{ item.name }}</div>
 }
}
```

# What If: Server-Side Rendering (SSR)?

```
<!-- SSR-aware conditional rendering -->
@if (isPlatformBrowser()) {
 <!-- Browser-only content -->
 <div class="interactive-chart">
 <canvas #chartCanvas></canvas>
 </div>
} @else {
 <!-- SSR fallback -->
 <div class="chart-placeholder">
 <div class="placeholder-content">Chart Preview</div>
 </div>
}

<!-- Deferred loading for better performance -->
@defer (on viewport) {
 <app-heavy-component />
} @placeholder {
 <div class="skeleton-loader"></div>
} @loading (minimum 200ms) {
 <div class="spinner"></div>
}
```

# What If: Migration Challenges?

## Challenge

Complex template with multiple ng-templates

```
<ng-container *ngTemplateOutlet="
 isAdmin ? adminTpl : userTpl;
 context: { data: userData }
></ng-container>
```

## Solution

Use @if with component composition

```
@if (isAdmin) {
 <app-admin-view [data]="userData" />
} @else {
 <app-user-view [data]="userData" />
}
```

# What If: Future Possibilities?



## Upcoming Features

- • @defer for lazy loading components
- • @placeholder for loading states
- • @error for error boundaries
- • Enhanced type inference



## Best Practices Evolution

- • Signals + Control Flow = Reactive templates
- • Computed properties for complex conditions

# Creative Applications

## Dynamic Forms

```
@for (field of formFields(); track field.id) {
 @switch (field.type) {
 @case ('text') {
 <input type="text" />
 }
 @case ('select') {
 <select><!-- options --></select>
 }
 }
}
```

## Wizard Steps

```
@switch (currentStep()) {
 @case (1) {
 <app-step-one />
 }
 @case (2) {
 <app-step-two />
 }
 @case (3) {
 <app-step-three />
 }
}
```



# Hands-On Exercise

## Migrate to Modern Control Flow Syntax

Transform your Angular components from structural directives to modern control flow

# Why Model Context Protocol?

## The Problem with AI Development Today

### Current AI Limitations

- **No direct tool access** - AI can't run commands
- **Limited context** - Can't access your project files
- **Static responses** - No dynamic interaction with your environment
- **Manual copy-paste** - You have to implement AI suggestions manually



# The AI Development Gap

## What We Want vs. What We Have

### What We Want

- AI that understands our project structure
- Direct integration with development tools
- Seamless code generation and execution
- Context-aware suggestions

### What We Have

- Generic AI responses
- Manual implementation of suggestions
- No project context
- Disconnected development workflow

**The Gap:** AI assistants are powerful but disconnected from your actual development environment.

# Real-World Pain Points

## Common Developer Frustrations

### 1. "Generate a component"

- AI suggests code
- You copy-paste manually
- You run CLI commands yourself
- **Time wasted:** 5-10 minutes per component

### 2. "Fix this error"

- AI doesn't see your actual error logs
- Generic solutions that don't fit your setup
- **Result:** Trial and error debugging

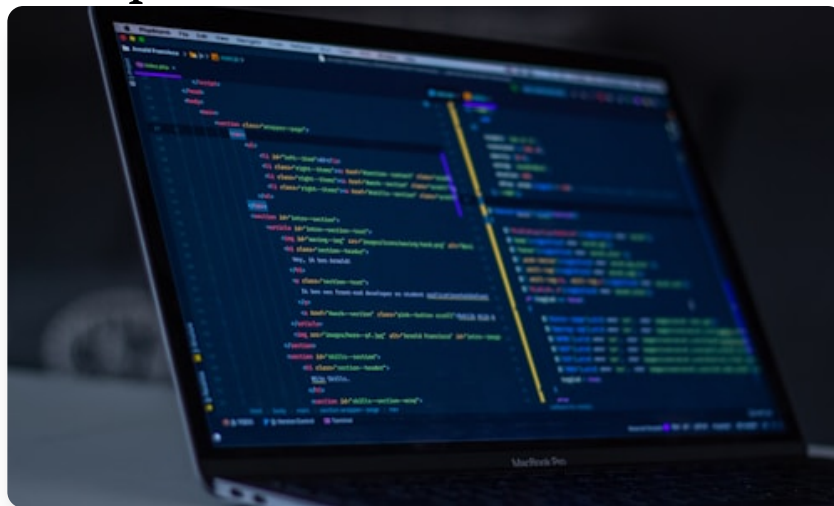
### 3. "Migrate to new Angular version"

# The MCP Solution

## Bridge the Gap Between AI and Development

### What MCP Enables

- **Direct tool access** - AI can run your CLI commands
- **Project context** - AI understands your file structure
- **Dynamic interaction** - Real-time feedback and execution
- **Seamless workflow** - AI becomes part of your development process



**The Vision:** AI assistants that are truly integrated into your development environment, not just helpful

# Why This Matters for Angular

## Specific Angular Development Benefits Component Generation

- AI can run `ng generate component` directly
- No more manual file creation and imports
- Consistent project structure maintained

## Migration Assistance

- AI can execute Angular update commands
- Real-time feedback on migration progress
- Automatic handling of breaking changes

## Testing Integration

# The Bigger Picture

## Beyond Angular - Universal AI Integration

### Current State

- Each AI tool has its own integration
- Fragmented development experience
- Vendor lock-in concerns
- Limited extensibility

### MCP Vision

- **Standardized protocol** for AI integration
- **Universal compatibility** across AI assistants
- **Extensible architecture** for custom tools
- **Open ecosystem** of MCP servers

**The Future:** A world where AI assistants seamlessly integrate with any development tool through a common protocol.

# Ready to Build the Future?

## Let's Create Your First MCP Server

**Next:** Understanding what MCP actually is and how it works

`<carbon:arrow-right class="inline-block w-8 h-8" />`

# What is Model Context Protocol?

A Standardized Framework for AI Integration

## Core Concept

**MCP** is a protocol that enables AI assistants to interact with external tools and data sources in a standardized way.



# MCP Architecture

## Three Core Components

### Resources

*Like GET endpoints*

- Provide data to AI
- Read-only information
- Project files, configurations
- Real-time data streams

### Tools

*Like POST endpoints*

- Enable AI to perform actions
- Execute commands
- Modify files
- Interact with external systems

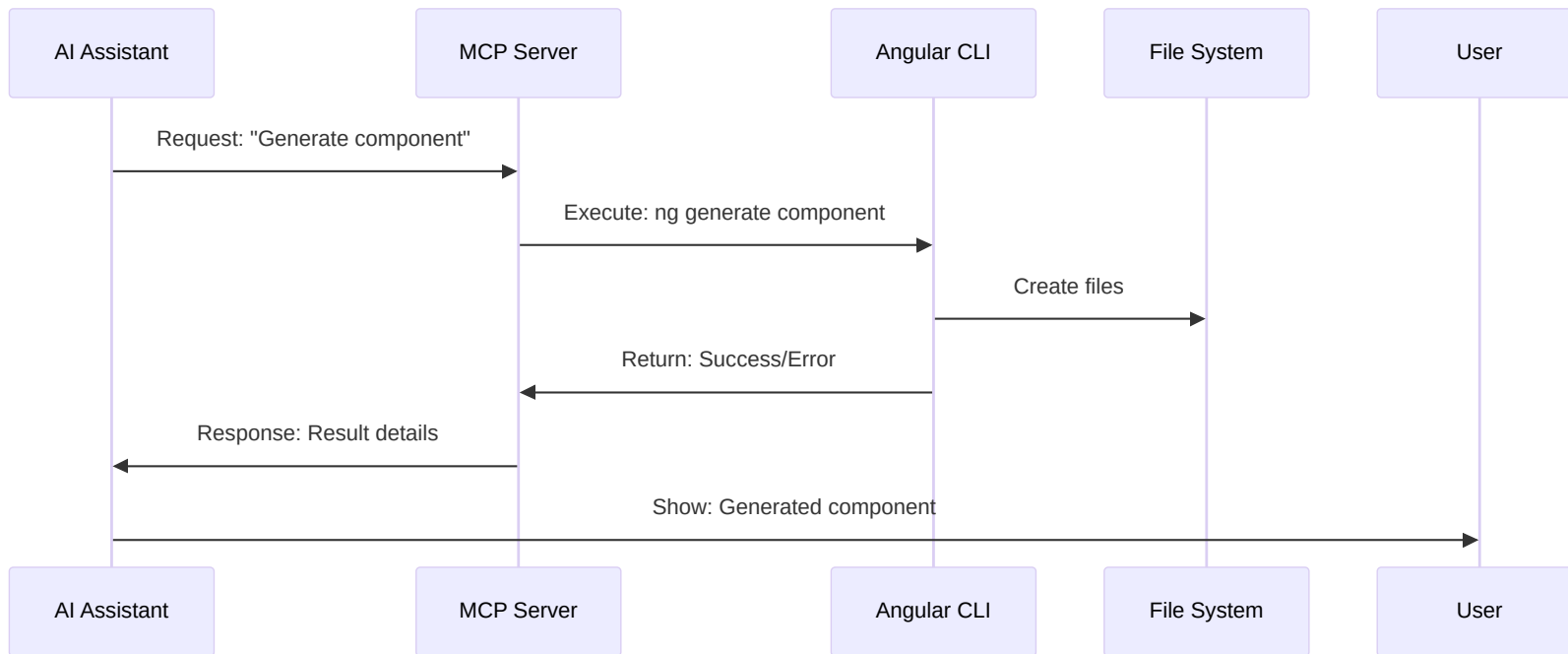
### Prompts

*Reusable templates*

- Pre-defined interaction patterns
- Common workflows
- Best practice templates
- Context-aware suggestions

# MCP Communication Flow

## How AI and Tools Interact



# MCP Server vs Client

## The Two Sides of MCP

### MCP Server

- Exposes tools and resources
- Handles AI requests
- Executes actual commands
- Manages security and permissions

```
// Server registers tools
server.registerTool(
 "generate_component",
 {
 // Tool definition
 },
 async (params) => {
 // Tool implementation
 },
);
```

### MCP Client

*AI Assistant (Cursor, Copilot)*

- Connects to MCP servers
- Makes requests to tools
- Receives responses
- Integrates with user interface

```
// Client uses tools
const result = await mcpClient.callTool("generate_component",
 name: "user-profile",
);
```

# TypeScript SDK Overview

## Building MCP Servers Made Easy

### Core Classes

```
import { McpServer } from "@modelcontextprotocol/sdk/server/mcp.js";
import { StdioServerTransport } from "@modelcontextprotocol/sdk/server/stdio.js";
```

### Key Features

- **Type Safety** - Full TypeScript support
- **Transport Layer** - stdio, SSE, WebSocket support
- **Schema Validation** - Zod integration for input validation
- **Error Handling** - Built-in error management
- **Protocol Compliance** - Follows MCP specification

# MCP Transport Options

## How AI and MCP Servers Communicate

### STDIO Transport

*Most Common*

- Standard input/output streams
- Simple process communication
- Works with any AI assistant
- Easy to debug and test

```
const transport = new StdioServerTransport();
```

### SSE Transport

*Web-based*

- Server-Sent Events over HTTP
- Real-time communication
- Web application integration
- Browser compatibility

```
const transport = new SSEServerTransport();
```

**For our Angular MCP server, we'll use STDIO transport for maximum compatibility.**

# MCP Message Format

## Standardized Communication Protocol

### Request Format

```
{
 "jsonrpc": "2.0",
 "id": 1,
 "method": "tools/call",
 "params": {
 "name": "generate_component",
 "arguments": {
 "name": "user-profile",
 "path": "src/app/components"
 }
 }
}
```

### Response Format

```
{
 "jsonrpc": "2.0",
 "id": 1,
 "result": {
 "content": [
]
 }
 }
}
```

# Security Considerations

## Safe AI Integration

### What MCP Enables

- Controlled tool access
- Permission-based execution
- Audit logging
- Sandboxed environments

### Security Best Practices

- Validate all inputs
- Limit tool capabilities
- Use least privilege principle
- Monitor tool usage

**Important:** MCP servers run with the same permissions as the user - implement proper validation and security measures.

# MCP Ecosystem

## Growing Community and Tools

### Official MCP Servers

- **Filesystem** - File operations
- **Git** - Version control
- **Database** - Data access
- **Web Search** - Information retrieval

### Community Servers

- **Docker** - Container management
- **AWS** - Cloud operations
- **Slack** - Team communication
- **Custom tools** - Project-specific

# Ready to Build?

## Now You Understand the Foundation

**Next:** Let's implement your first MCP server

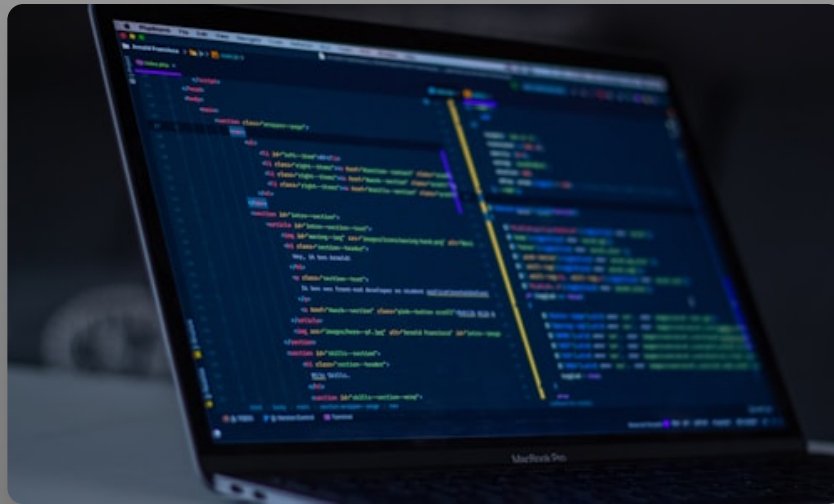
`<carbon:arrow-right class="inline-block w-8 h-8" />`

# How to Build an MCP Server

## Step-by-Step Implementation

### What We'll Build

- Angular CLI integration
- Component generation tool
- Error handling
- TypeScript implementation



# Project Setup

## Initialize Your MCP Server

### 1. Install Dependencies

```
npm install @modelcontextprotocol/sdk zod
npm install --save-dev @types/node
```

### 2. Add Build Script

```
{
 "scripts": {
 "build:angular-mcp": "tsc --project tools/angular-mcp/tsconfig.json"
 }
}
```

### 3. Create Directory Structure

```
tools/
├─ angular-mcp/
│ ├─ index.ts
│ └─ tsconfig.json
```

# TypeScript Configuration

## Set Up Compilation

**Create** `tools/angular-mcp/tsconfig.json`

```
{
 "compilerOptions": {
 "target": "ES2022",
 "module": "NodeNext",
 "moduleResolution": "NodeNext",
 "outDir": "../dist/angular-mcp",
 "rootDir": ".",
 "esModuleInterop": true,
 "forceConsistentCasingInFileNames": true,
 "strict": true,
 "skipLibCheck": true,
 "resolveJsonModule": true,
 "types": ["node"]
 },
 "include": ["index.ts"],
 "exclude": ["node_modules", "dist"]
}
```

# Basic MCP Server Structure

## Foundation Implementation

### Import Required Modules

```
import { McpServer } from "@modelcontextprotocol/sdk/server/mcp.js";
import { StdioServerTransport } from "@modelcontextprotocol/sdk/server/stdio.js";
import { exec as execSync } from "child_process";
import { dirname } from "path";
import { promisify } from "util";
import { z } from "zod";

const exec = promisify(execSync);
```

### Initialize Server

```
const server = new McpServer({
 name: "workshops-de-angular-mcp",
 version: "1.0.0",
});
```

# Register Your First Tool

## Component Generation Tool Tool Registration

```
server.registerTool(
 "generate_component",
 {
 title: "Generate Angular Component",
 description: "Creates a new Angular component using the Angular CLI",
 inputSchema: {
 name: z.string().describe("Component name"),
 path: z.string().optional().describe("Target path or project"),
 },
 },
 async ({ name, path }: { name: string; path?: string }) => {
 // Tool implementation
 },
);
```

# Implement Tool Logic

## Angular CLI Integration Command Construction

```
async ({ name, path }: { name: string; path?: string }) => {
 // Remove src/app prefix if present
 path = path?.replace(/^src\/app\/?/, "");

 // Construct CLI command
 const target = path ? `${path}/${name}` : name;
 const cliCommand = `npx @angular/cli generate component ${target} --standalone --flat --skip-tests --inline-style --inlin

 // Execute command
 const result = await exec(cliCommand, {
 cwd: dirname(dirname(__dirname)),
 });

 return {
 content: [
 {
 type: "text",
 text: `✅ Component generated successfully:\n${result.stdout}`,
 },
],
 },
}
```

# Error Handling

## Graceful Failure Management Try-Catch Implementation

```
try {
 const result = await exec(cliCommand, {
 cwd: dirname(dirname(__dirname)),
 });
 return {
 content: [
 {
 type: "text",
 text: `✅ Component generated successfully:\n${result.stdout}`,
 },
],
 };
} catch (error: unknown) {
 return {
 content: [
 {
 type: "text",
 text: `❌ CLI Error: ${error instanceof Error ? error.message : "Unknown error"}`,
 },
],
 };
}
```

# Start the Server

## Connect and Listen

### Transport Setup

```
const transport = new StdioServerTransport();

server
 .connect(transport)
 .then(() => {
 console.log("MCP server started");
 })
 .catch((error) => {
 console.error("Error connecting to MCP server:", error);
 });
```

## Complete Server File

```
// All previous code combined
// Save as tools/angular-mcp/index.ts
```

# Build and Test

## Compile Your MCP Server

### 1. Build the Server

```
npm run build.angular-mcp
```

### 2. Verify Output

```
dist/angular-mcp/
└─ index.js
```

### 3. Test Manually

```
node dist/angular-mcp/index.js
```

**The compiled JavaScript file is what AI assistants will actually execute.**

# AI Assistant Integration

## Connect to Cursor IDE

### 1. Create MCP Configuration

Create or edit `~/.cursor/mcp.json` :

```
{
 "mcpServers": {
 "workshops-de-angular-mcp": {
 "command": "node",
 "args": ["/path/to/your/project/dist/angular-mcp/index.js"],
 "cwd": "/path/to/your/project"
 }
 }
}
```

### 2. Reload Cursor

- Open Command Palette ( `Cmd+Shift+P` )
- Run "Developer: Reload Window"

# Test Your Integration

## Verify Everything Works

### 1. Ask AI to Generate Component

```
"Generate a component called user-profile in the src/app/components directory"
```

### 2. Expected Behavior

- AI makes MCP tool call
- Your server executes Angular CLI
- Component files are created
- AI shows success message

### 3. Troubleshooting

- Check MCP server logs

# Congratulations!

## You've Built Your First MCP Server

**Next:** Explore advanced features and extensions

`<carbon:arrow-right class="inline-block w-8 h-8" />`

# What If We Extend MCP?

## Creative Applications and Advanced Features

### Beyond Basic Tools

- Migration automation
- Testing integration
- Performance monitoring
- Custom workflows



### Advanced features

# Advanced Angular Tools

## Extending Your MCP Server Migration Tool

```
server.registerTool(
 "run_migration",
 {
 title: "Run Angular Migration",
 description: "Execute Angular update migrations",
 inputSchema: {
 version: z.string().describe("Target Angular version"),
 packages: z.array(z.string()).optional(),
 },
 },
 async ({ version, packages }) => {
 const command = `npx @angular/cli update @angular/core@${version}`;
 // Implementation
 },
);
```

## Testing Tool

# Resource Integration

## Providing Context to AI Project Structure Resource

```
server.registerResource(
 "project_structure",
 {
 name: "Angular Project Structure",
 description: "Current project file organization",
 mimeType: "application/json",
 },
 async () => {
 const structure = await getProjectStructure();
 return {
 contents: [
 {
 uri: "project://structure",
 mimeType: "application/json",
 text: JSON.stringify(structure, null, 2),
 },
],
 };
 },
);
```

# Custom Prompts

## Reusable AI Workflows

### Component Generation Prompt

```
server.registerPrompt(
 "generate_component_with_tests",
 {
 name: "Generate Component with Tests",
 description: "Create a component with comprehensive test setup",
 arguments: [
 {
 name: "componentName",
 description: "Name of the component to generate",
 required: true,
 },
 {
 name: "features",
 description: "Features to include (forms, routing, etc.)",
 required: false,
 },
],
 },
 async (args) => {
 return {
```

# Multi-Tool Workflows

## Complex Development Tasks

### Full Feature Implementation

```
server.registerTool(
 "implement_feature",
 {
 title: "Implement Complete Feature",
 description: "Generate component, service, tests, and documentation",
 inputSchema: {
 featureName: z.string(),
 componentType: z.enum(["page", "widget", "form"]),
 includeTests: z.boolean().default(true),
 includeDocs: z.boolean().default(true),
 },
 },
 async ({ featureName, componentType, includeTests, includeDocs }) => {
 const steps = [];

 // 1. Generate component
 steps.push(await generateComponent(featureName, componentType));

 // 2. Generate service if needed
 if (componentType === "form") {
```

# Integration with External Services

## Beyond Angular CLI

### Git Integration

```
server.registerTool(
 "commit_changes",
 {
 title: "Commit Generated Code",
 description: "Automatically commit generated components",
 inputSchema: {
 message: z.string(),
 files: z.array(z.string()),
 },
 },
 async ({ message, files }) => {
 // Git operations
 },
);
```

### Monitoring Integration

```
server.registerTool(
 "analyze_performance",
 {
 title: "Analyze Component Performance",
 description: "Run performance analysis on components",
 inputSchema: {
 componentPath: z.string(),
 metrics: z.array(z.string()),
 },
 },
 async ({ componentPath, metrics }) => {
 // Performance analysis
 },
);
```

### Deployment Integration

```
server.registerTool(
 "deploy_component",
 {

```

# Advanced Error Handling

## Robust Error Management

### Retry Logic

```
async function executeWithRetry<T>(
 operation: () => Promise<T>,
 maxRetries: number = 3,
) : Promise<T> {
 for (let attempt = 1; attempt <= maxRetries; attempt++) {
 try {
 return await operation();
 } catch (error) {
 if (attempt === maxRetries) {
 throw new Error(
 `Operation failed after ${maxRetries} attempts: ${error.message}`,
);
 }

 // Exponential backoff
 await new Promise((resolve) =>
 setTimeout(resolve, Math.pow(2, attempt) * 1000),
);
 }
 }
}
```

# Security Enhancements

## Production-Ready Security Input Sanitization

```
function sanitizeComponentName(name: string): string {
 // Remove dangerous characters
 return name.replace(/[^a-zA-Z0-9- _]/g, "");
}

function validatePath(path: string): boolean {
 // Prevent directory traversal
 return !path.includes("..") && !path.startsWith("/");
}
```

## Permission System

```
interface ToolPermissions {
 canGenerateComponents: boolean;
 canModifyFiles: boolean;
 canExecuteCommands: boolean;
 allowedPaths: string[];
}

async function checkPermissions(tool: string, user: string): Promise<boolean> {
```

# Performance Optimization

## Efficient MCP Server Caching Strategy

```
class ProjectCache {
 private cache = new Map<string, { data: any; timestamp: number }>();
 private ttl = 5 * 60 * 1000; // 5 minutes

 get(key: string): any | null {
 const entry = this.cache.get(key);
 if (!entry) return null;

 if (Date.now() - entry.timestamp > this.ttl) {
 this.cache.delete(key);
 return null;
 }

 return entry.data;
 }

 set(key: string, data: any): void {
 this.cache.set(key, { data, timestamp: Date.now() });
 }
}
```

# Future Possibilities

## What's Next for MCP?

### **AI-Driven Development**

- Automatic code optimization
- Intelligent refactoring suggestions
- Performance bottleneck detection
- Security vulnerability scanning

### **Team Collaboration**

- Shared MCP servers
- Team-specific tools
- Code review automation
- Knowledge sharing prompts

### **Ecosystem Integration**

- VS Code extensions
- JetBrains plugins
- CI/CD pipeline integration
- Cloud deployment tools

### **Advanced Analytics**

- Development metrics
- Code quality insights
- Team productivity tracking
- Project health monitoring

# Your Next Steps

## Extend Your MCP Server

### Immediate Extensions

1. **Add migration tool** - Implement the workshop task
2. **Create resource endpoints** - Expose project information
3. **Add validation** - Improve error handling
4. **Test thoroughly** - Ensure reliability

### Advanced Features

1. **Multi-project support** - Handle multiple Angular projects
2. **Custom schematics** - Integrate with Angular schematics
3. **Team sharing** - Deploy MCP server for your team
4. **Monitoring** - Add logging and metrics

# Ready for the Task?

## Implement Your Migration Tool

**Your turn:** Add the Angular migration tool to your MCP server

`<carbon:arrow-right class="inline-block w-8 h-8" />`

# Workshop Task

## Building an Angular Migration MCP Tool

### Your Mission

Add a migration tool to your MCP server that can execute Angular migrations from [angular.dev/update-guide](https://angular.dev/update-guide)



Migration task

# Sources and References

## Model Context Protocol (MCP)

# MCP Official Documentation

## Core Resources

### 1. Model Context Protocol Documentation

- <https://modelcontextprotocol.io/>
- Official MCP specification and guides

### 2. MCP TypeScript SDK

- <https://github.com/modelcontextprotocol/typescript-sdk>
- Official SDK for building MCP servers and clients

### 3. MCP Server Examples

- <https://github.com/modelcontextprotocol/servers>
- Community-maintained MCP server implementations

### 4. MCP Client Examples

# AI Assistant Integration

## Platform-Specific Guides

### 1. Cursor IDE MCP Integration

- <https://cursor.sh/docs>
- Official Cursor documentation for MCP setup

### 2. GitHub Copilot MCP Support

- <https://docs.github.com/en/copilot>
- GitHub's AI assistant integration options

### 3. VS Code MCP Extensions

- VS Code Marketplace
- Community extensions for MCP support

### 4. IntelliJ AI Assistant

# Angular-Specific Resources

## Development Tools

### 1. Angular CLI Documentation

- <https://angular.io/cli>
- Official Angular CLI commands and options

### 2. Angular Update Guide

- <https://angular.dev/update-guide>
- Migration guides and update instructions

### 3. Angular Schematics

- <https://angular.io/guide/schematics>
- Custom code generation with Angular CLI

### 4. Angular DevTools

# Community Resources

## Tutorials and Examples

### 1. **"Building Your First MCP Server"**

- Author: MCP Community
- Source: GitHub Examples
- Focus: Step-by-step MCP server creation

### 2. **"Integrating AI with Development Tools"**

- Author: Various Contributors
- Source: Dev.to
- Focus: Real-world MCP implementations

### 3. **"Angular CLI Automation with MCP"**

- Author: Community

# Tools and Extensions

## Development Environment

### 1. Node.js MCP Runtime

- <https://nodejs.org/>
- Required runtime for TypeScript MCP servers

### 2. TypeScript Compiler

- <https://www.typescriptlang.org/>
- Compile TypeScript MCP servers to JavaScript

### 3. Zod Validation Library

- <https://zod.dev/>
- Schema validation for MCP tool parameters

### 4. VS Code TypeScript Extension

# Control Flow Syntax Migration

# Angular Documentation

## Official Angular Resources

### 1. Angular Control Flow Guide

- <https://angular.dev/guide/templates/control-flow>
- Comprehensive guide to @if, @for, @switch, and @defer

### 2. Angular Migration Guide

- <https://angular.dev/update-guide>
- Step-by-step migration instructions for different Angular versions

### 3. Angular Blog - Control Flow Announcement

- <https://blog.angular.dev/introducing-angular-v17-4d7033312e4b>
- Official announcement and rationale for new control flow syntax

### 4. Angular Performance Best Practices

# Community Resources

## Articles and Tutorials

### 1. **"Migrating to Angular's New Control Flow Syntax"**

- Author: Angular Team
- Source: Angular Blog
- Focus: Step-by-step migration guide

### 2. **"Performance Benefits of Built-in Control Flow"**

- Author: Minko Gechev
- Source: Medium
- Focus: Bundle size and runtime performance analysis

### 3. **"Angular 17: The Renaissance of Angular"**

- Author: Angular Community

# Code Examples

## Repository References

### 1. Angular Official Examples

- <https://github.com/angular/angular>
- Location: packages/core/test/acceptance/control\_flow\_spec.ts
- Contains comprehensive test cases for control flow

### 2. Angular DevTools

- <https://github.com/angular/angular-devtools>
- Shows how to debug components with new control flow

### 3. Angular CLI Schematics

- <https://github.com/angular/angular-cli>
- Location: packages/schematics/angular/migrations/

# Video Resources

## Recommended Talks and Tutorials

### 1. **"What's New in Angular 17" - ng-conf**

- Speaker: Angular Team
- Duration: 45 minutes
- Focus: Complete overview of Angular 17 features

### 2. **"Control Flow Deep Dive" - Angular Connect**

- Speaker: Alex Rickabaugh
- Duration: 30 minutes
- Focus: Technical implementation details

### 3. **"Migrating Large Applications" - YouTube**

- Channel: Angular University

# Tools and Extensions

## Development Tools

### 1. Angular Language Service

- VS Code Extension
- Provides IntelliSense for new control flow syntax

### 2. Angular DevTools

- Chrome/Firefox Extension
- Debug and profile components with control flow

### 3. ESLint Angular

- <https://github.com/angular-eslint/angular-eslint>
- Linting rules for control flow best practices

### 4. Prettier Angular

# Thank You!

## Continue Learning



Official Angular Documentation

---



Angular GitHub Repository

---



Angular Discord Community

---



Follow @angular on Twitter

---