

Table of contents

Angular Series	2
Starter	4
Components And Data Binding	8

Angular Series


Chapter 1: Introduction

This is a guide to helping you learn angular from basics to advanced concepts. Involves understanding the essentials of building with the framework, developing, testing and deployment of applications build on Angular.

Prerequisites

- Node.js (<https://nodejs.org/en>)
- Text Editor
- Angular CLI

Installing Angular CLI

```
 npm install -g @angular/cli
```


Creating a new project

With angular CLI installed, run `ng new <project-name>` in your terminal.

- Example:

```
 ng new todolist
```

- You will be presented with some configuration options for your project. Use the arrow and enter keys to navigate and select which options you desire.
- After you select the configuration options and the CLI runs through the setup, you should see the following message:

```
 $ ✓ Packages installed successfully.
```

\$ Successfully initialized git.

- You are ready to run the project locally

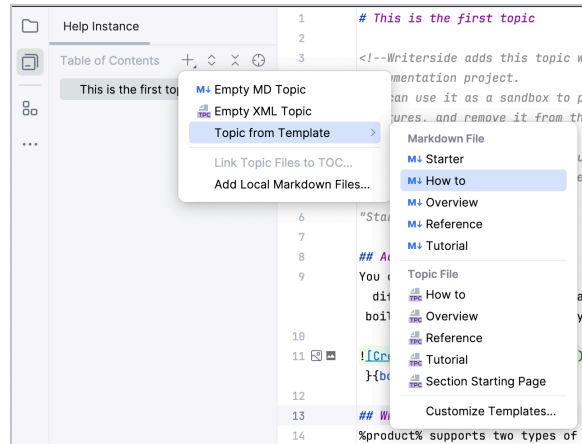
Running project locally

- Switch to your new Angular project `cd todolist`
- Run `npm start` or `ng serve`
- Once everything is successful open the project on the browser on port `4200` or visit the path `http://localhost:4200`

Starter

Add new topics

You can create empty topics, or choose a template for different types of content that contains some boilerplate structure to help you get started:



Create new topic options

Write content

Writerside supports two types of markup: Markdown and XML. When you create a new help article, you can choose between two topic types, but this doesn't mean you have to stick to a single format. You can author content in Markdown and extend it with semantic attributes or inject entire XML elements.

Inject XML

For example, this is how you inject a procedure:

Inject a procedure

1. Start typing and select a procedure type from the completion suggestions:



completion suggestions for procedure

2. Press `Tab` or `Enter` to insert the markup.

Add interactive elements

Tabs

To add switchable content, you can make use of tabs (inject them by starting to type `tab` on a new line):

Markdown

```
![Alt Text](new_topic_options.png){ width=450 }
```

Semantic markup

```

```

Collapsible blocks

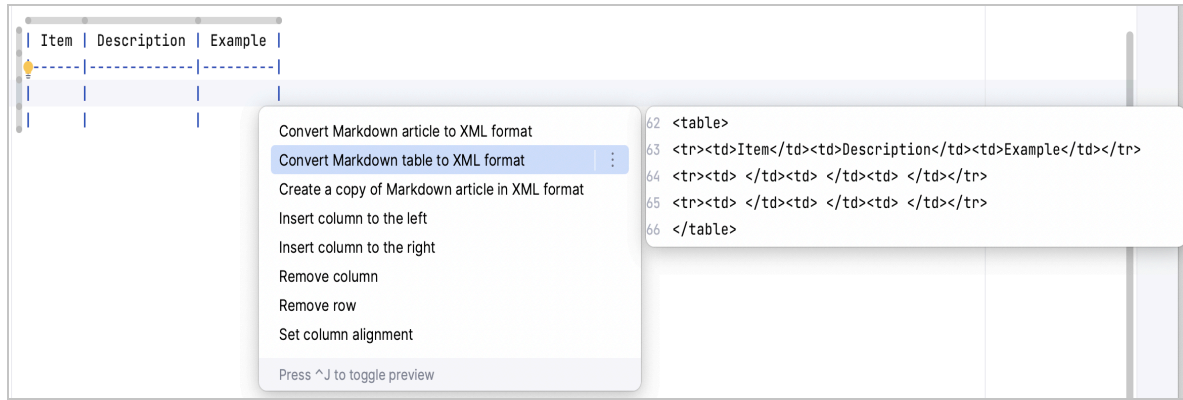
Apart from injecting entire XML elements, you can use attributes to configure the behavior of certain elements. For example, you can collapse a chapter that contains non-essential information:

Supplementary info

Content under a collapsible header will be collapsed by default, but you can modify the behavior by adding the following attribute: `default-state="expanded"`

Convert selection to XML

If you need to extend an element with more functions, you can convert selected content from Markdown to semantic markup. For example, if you want to merge cells in a table, it's much easier to convert it to XML than do this in Markdown. Position the caret anywhere in the table and press **Alt + Enter**:



Convert table to XML

Feedback and support

Please report any issues, usability improvements, or feature requests to our YouTrack project (<https://youtrack.jetbrains.com/newIssue?project=WRS>) (you will need to register).

You are welcome to join our public Slack workspace (https://jb.gg/WRS_Slack). Before you do, please read our Code of conduct (<https://plugins.jetbrains.com/plugin/20158-writerside/docs/writerside-code-of-conduct.html>). We assume that you've read and acknowledged it before joining.

You can also always email us at writerside@jetbrains.com (<mailto:writerside@jetbrains.com>).

See also

Writerside documentation

Markup reference (<https://plugins.jetbrains.com/plugin/20158-writerside/docs/markup-reference.html>)

Reorder topics in the TOC (<https://plugins.jetbrains.com/plugin/20158-writerside/docs/manage-table-of-contents.html>)

Build and publish (<https://plugins.jetbrains.com/plugin/20158-writerside/docs/local-build.html>)

Configure Search (<https://plugins.jetbrains.com/plugin/20158-writerside/docs/configure-search.html>)

Components And Data Binding

- A **component** is a single unit of a user interface.
- An **application** is built by combining different components.
- Executing the `ng new` command creates a new component, which will act as the root component.


Each component has:

- An **HTML template**: Defines the structure of the component or what will be rendered.
- A **CSS file**: Defines the styling.
- A **TypeScript file**: Defines behavior.
- A **testing specification file**: Used for writing unit tests.

Creating A Component

Through the Terminal:

- To create a new component, run:

•  `ng generate component <componentname>`

- Or the short hand way

•  `ng g c <componentname>`

Component Decorator

- A component is a normal class, but to convert it into a component, we add the `@Component` decorator, which takes an object with the following properties:

1. Selector:

- Used like a CSS selector to select an element.

Example:

- selector: 'app-home'

2. templateUrl:

- Link to the component template or HTML page.
- An alternative is using **template**.

3. styleUrls:

- An array that contains the links to the styles the component uses.
- An alternative is using **styles**.

4. Standalone:

- A feature that allows a component to import its dependencies directly without needing NgModule
- Takes in a boolean value, true or false

5. Imports:

- An array that takes in a list of all dependencies including, directives, pipes, components etc.

Data Binding

- **Data binding** creates communication between the component and the HTML, making the DOM interactive.

Text Interpolation

- **Interpolation** displays string values dynamically into the HTML template using double curly braces - `{{}}`.

Example:

- In the template file: `<p>Programming language: {{language}}</p>`
- in the component file, have a property language in the class i.e

```
export class FirstComponent {  
  language = 'TypeScript'  
}
```

Property Binding

- **Property binding** is used to set the value of a property for an element or directive.

Example:

- In the component:
-
-
- in the component file, have a property language in the class i.e

```
export class FirstComponent {  
  imageLink = 'linktoimage';  
  disabled = true;  
}
```

- In the template: `<td [attr.colspan]="colspanValue">A cell</td>`

Class Binding

- **Class binding** is used to add or remove CSS classes on an element.
- Classes can be added dynamically based on component data.

Example:

- In the component:

```
export class FirstComponent {  
  isSpecial = true;  
}
```

- In the template: `<div [class.special]="isSpecial">This div is special</div>`
- You can bind multiple classes using an object eg. `<div [ngClass]="{ special: isSpecial, 'another-class': anotherCondition }"></div>`

Style Binding

- **Style binding** is used to set inline styles of HTML elements dynamically.
- Similar to class binding, you can bind to style properties.

Example:

- In the component:

```
export class FirstComponent {  
  isActive = true;  
}
```

- In the template: `<div [style.color]="isActive ? 'red' : 'green'">This text is styled</div>`
- You can also bind multiple styles using an object: `<div [ngStyle]="{ 'font-weight': isActive ? 'bold' : 'normal', 'font-size': '20px' }"></div>`

Two-Way Data Binding

- **Two-way binding** allows data to flow both ways between the component and the view.
- Angular's `ngModel` directive facilitates this.

Example:

- In the component:

```
export class FirstComponent {  
  name = 'Angular';  
}
```

- In the template:

```
<input [(ngModel)]="name">  
<p>Hello, {{name}}!</p>
```