WSA | Forward looking IT finishing school

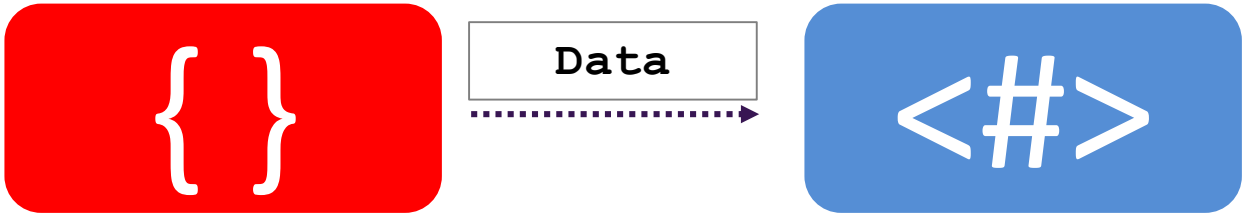
Data and Event Handling

Angular


WSA
Forward looking IT finishing school
Interpolation
(Connecting data with view)
www.webstackacademy.com

Interpolation

- At high level component can be broken into two parts:
  - View - Which takes care of the look & Feel (HTML + CSS)
  - Business logic - Which takes care of the user and server interactivity (TypeScript -> JavaScript)



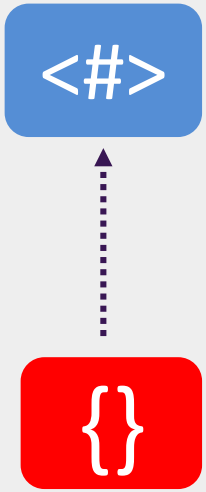
How to access data from into template?

Interpolation

- The major difference between writing them in a 'vanilla' fashion (how we have done during JavaScript module) and Angular is we have followed certain rules
- The main goal of these rules is to de-couple the **View (HTML + CSS)** and **Business Logic**
- This makes components re-usable, modular and maintainable which is required for SPA
- In real-world applications these two need to work together by communicating properties (variables, objects, arrays, etc..) from the component class to the template, you can use interpolation
- This is achieved with the notation `{{ propertyName }}` in the template. With this approach whenever we are changing the functionality, the template don't need to change.
- This way of connection is called as **Binding**

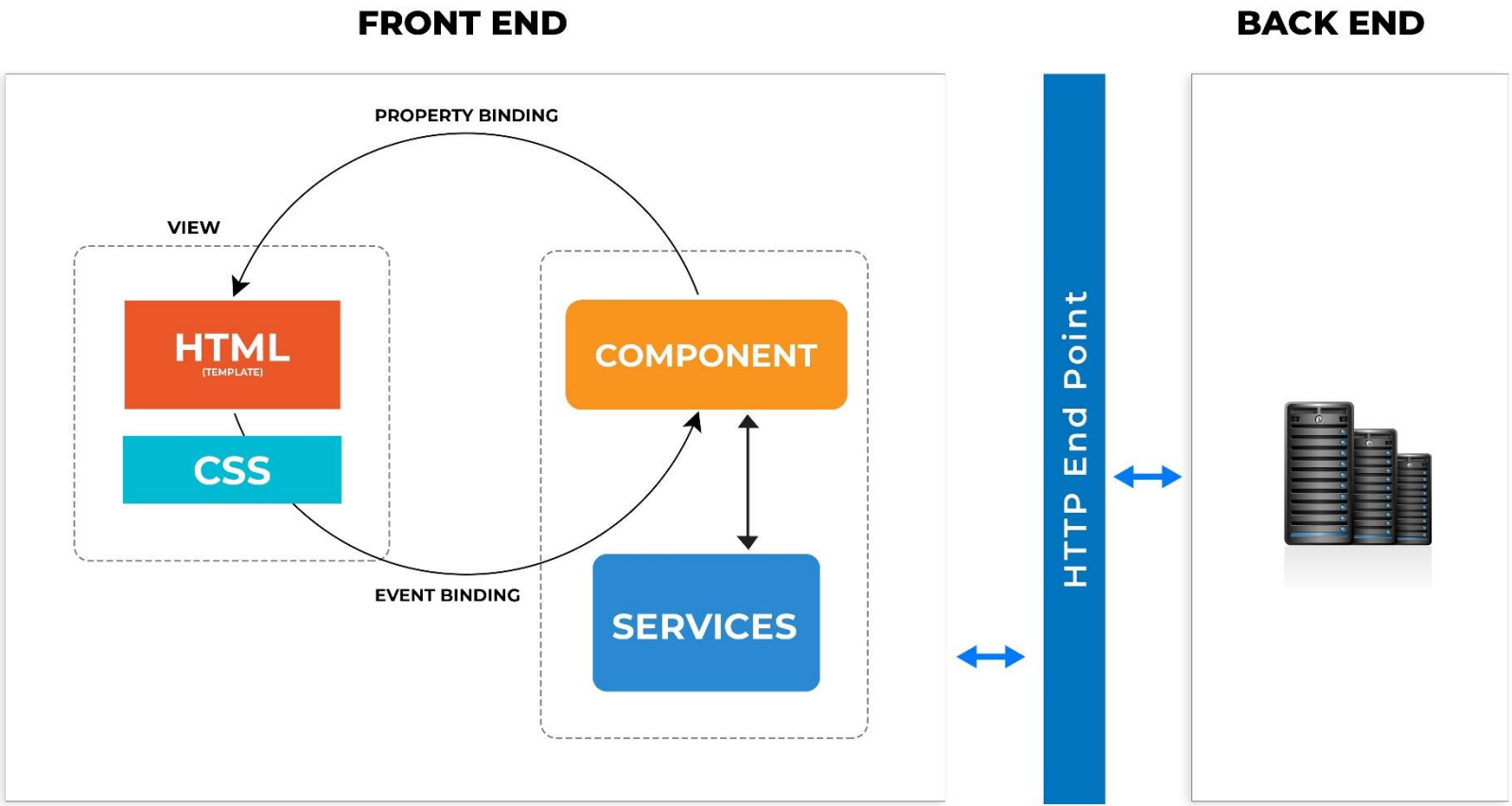
Interpolation

```
@Component({
selector: 'courses',
template: `
    <h2> {{title}}</h2>
    <ul> {{courses}}</ul>
`
})

export class CoursesComponent {

title = 'List of courses in WSA:';
courses = ["FullStack","FrontEnd","BackEnd"]

}
```




WSA
Forward looking IT finishing school
Binding
(Connecting data with view)
www.webstackacademy.com

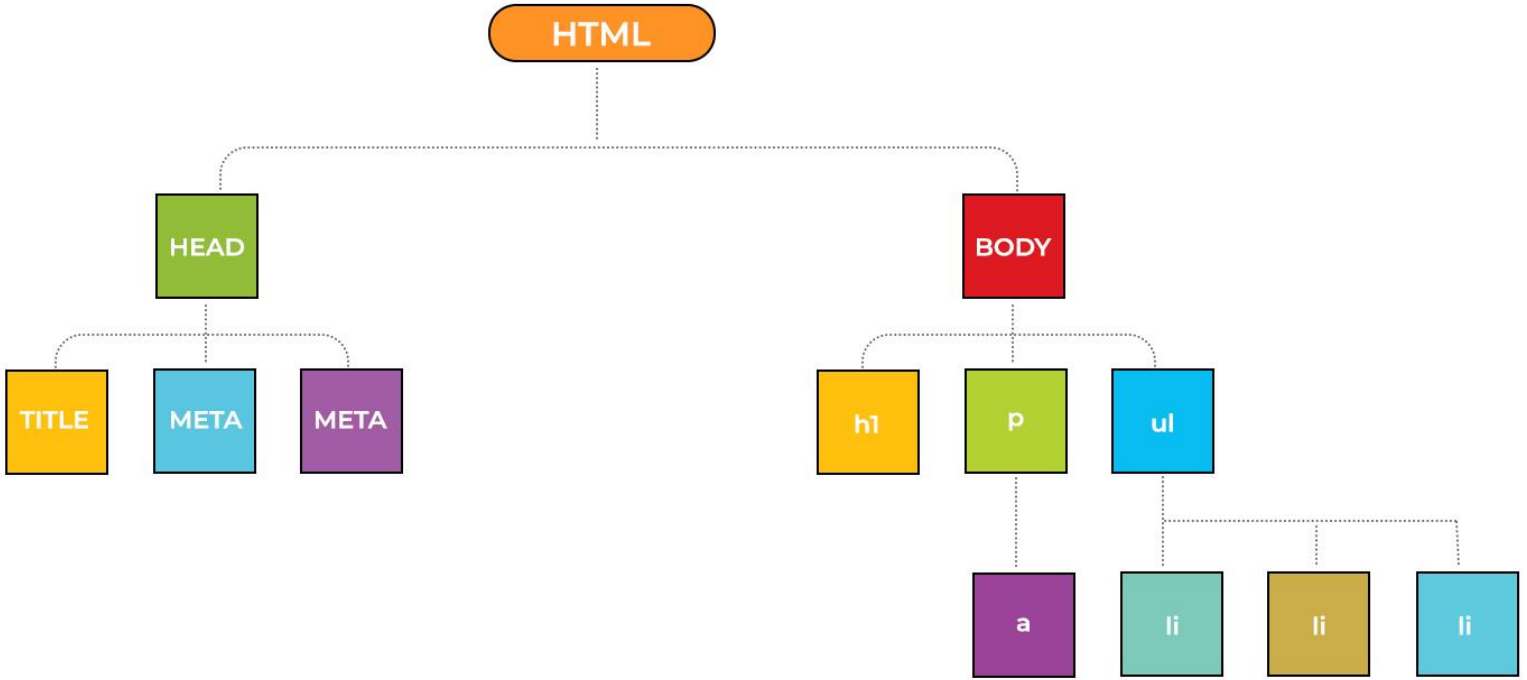
Binding – Big picture



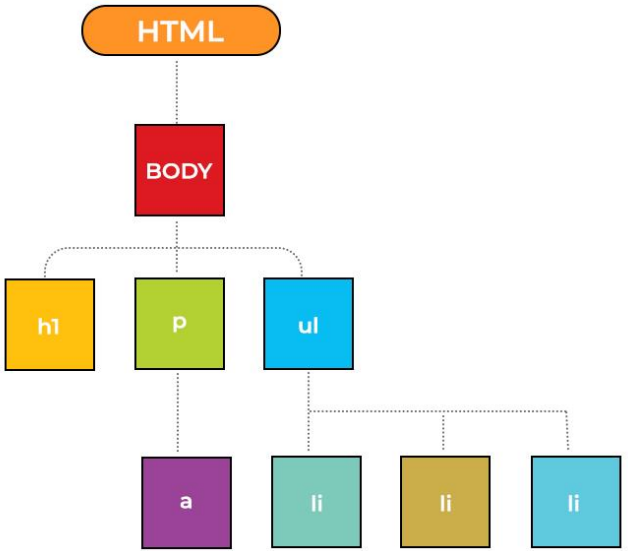
DOM - Revisiting

- The Document Object Model (DOM) is a cross-platform and language-independent application programming interface
- The DOM, is the API through which JavaScript interacts with content within a website
- The DOM API is used to access, traverse and manipulate HTML and XML documents
- The HTML is represented in a tree format with each node representing a property
- Nodes have “parent-child” relationship tracing back to the “root” <html>

DOM - Revisiting

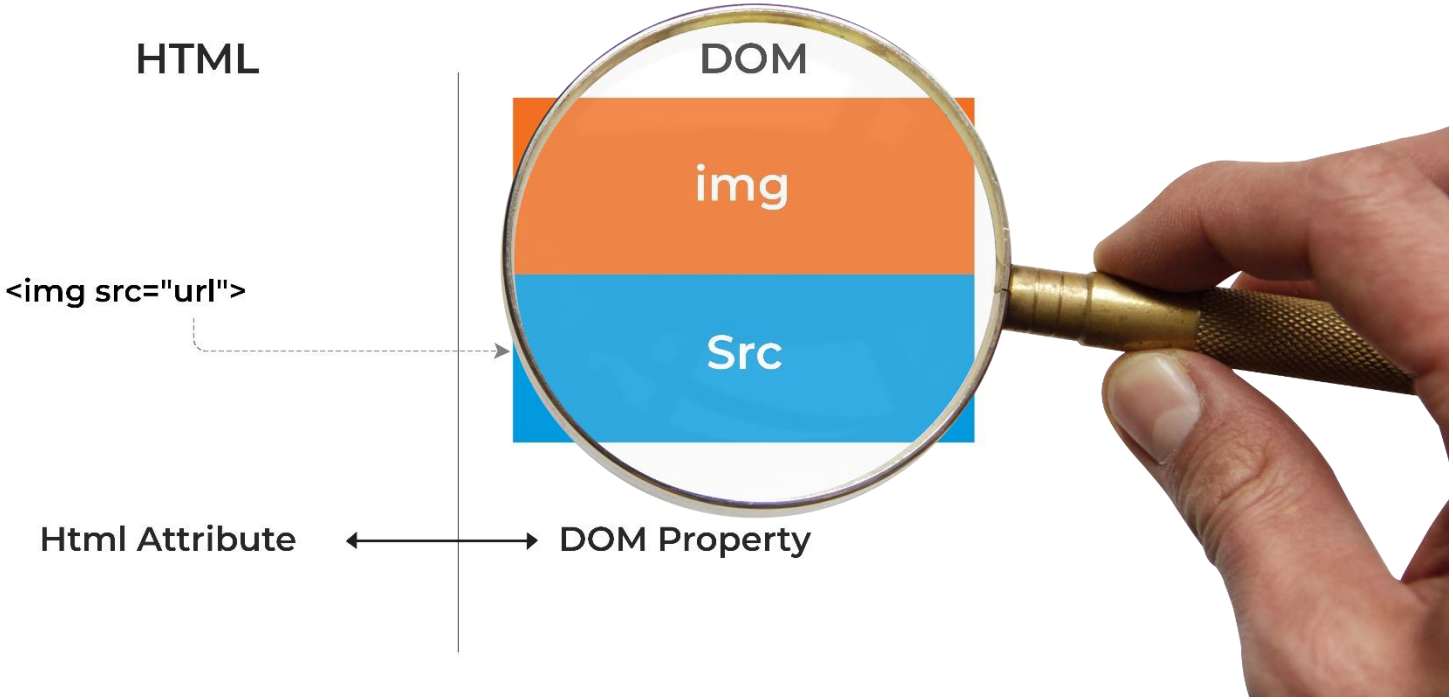


DOM and HTML




```
<html>

<body>


<h1> Courses in WSA </h1>
<ul>
   <li> FullStack web developer</li>
   <li> Frontend web developer </li>
   <li> Backend web developer </li>
<ul>

</body>

</html>
```

DOM and HTML – Zooming In!



Attribute Binding (One way: Business logic -> View)

```
<p><b>Example of property binding: Image</b></p>
<img [src]= "imageLink"/>
```


```
export class BindingComponent {
  imageLink = "http://www.testimage.com/side-image.png";
}
```

HTML - DOM element object

Following are some of the properties that can be used on all HTML elements

Property	Description
clientHeight	Returns the height of an element, including padding
Id	Sets or returns the value of the id attribute of an element
innerHTML	Sets or returns the content of an element
Style	Sets or returns the value of the style attribute of an element
textContent	Sets or returns the textual content of a node and its descendants
Title	Sets or returns the value of the title attribute of an element
nextSibling	Returns the next node at the same node tree level
nodeType	Returns the node type of a node


WSA
Forward looking IT finishing school
Event Binding
(Capturing events from the view)
www.webstackacademy.com

Event Binding

- In property binding, typically the changes done in the business logic (functionality part) is getting accessed from the view. It was achieved by the `[]notation`, in combination with the template (HTML) elements

- Event binding is the opposite of property binding where events from the template is sent back to the business logic in form of event notification functions.

- Similar to how it was done with vanilla JavaScript, each event to have a call-back function along with the event in order to notify in case of events

- User actions such as clicking a link, pushing a button, and entering text raise DOM events.

- This event binding is achieved with the `()notation`

Attribute Binding (One way: Business logic -> View)

```
<button (click)="mySaveFunction()">Save</button>
```


```
export class EbindingComponent  {

    mySaveFunction() {
      console.log ("Button was pressed");
   }
}
```

More Event Handling…

Event Filtering: Event handlers can be called on a filtering basis.

```
<input (keyup.enter)="myKeyPress()">
```

View variables: Variables can be defined in the view in order to send value to business logic. These variables are also called as **Template Variables**.

```
<input #userInput
(keyup.enter)="myKeyPressWithValue(userInput.value)">
```

Event Binding

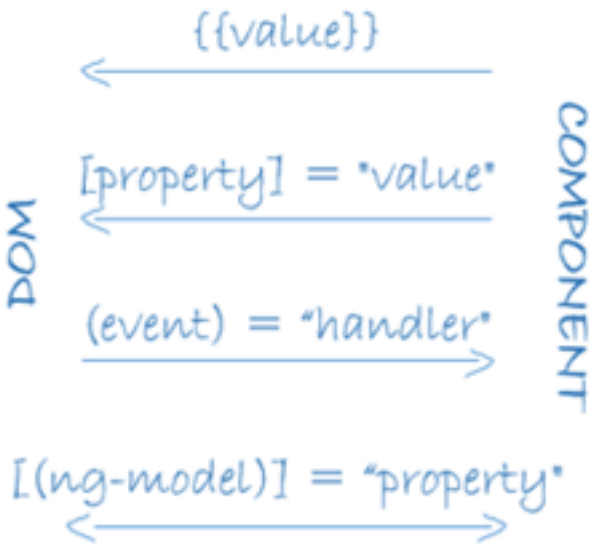
Following are some of the important events which is used in event binding use-cases

Methods	Description
Click	The event occurs when the user clicks on an element
Dbclick	The event occurs when the user double-clicks on an element
Keyup	The event occurs when the user releases a key
Mouseover	The event occurs when the pointer is moved onto an element, or onto one of its children
Submit	The event occurs when a form is submitted


WSA
Forward looking IT finishing school
Two way Binding
(Binding Business logic & View in both direction)
www.webstackacademy.com

Two way Binding

- One way binding in Angular is achieved in the following ways:
  - From View to Business Logic (Event binding) : **`()notation`**
  - From Business logic to view (Data binding) : **`[]notation`**
- They are also used in a combined manner. An event from the view triggers a change in a HTML attribute (View->Business Logic->View) [Ex: Changing another paragraph's background with a button click event]
- However handling two way binding is still not feasible with such approaches.
- There are some ways we have achieved (ex: $event), but parameter passing quite complex
- In order to achieve two way binding **`ngModel`** directive is used



What is ngModel directive?

- **ngModel** is one of the directives supported by Angular to achieve two-way binding
- A directive is a custom HTML element (provided by Angular) that is used to extend the power of HTML (More on this during next chapter)
- To set up two-way data binding with ngModel, you need to add an input in the template and achieve binding them using combination of **both square bracket and parenthesis**

```
<input [(ngModel)]="userInput" (keyup.enter)="userKeyPress()"/>
```

- With this binding whatever changes you are making in both view and business-logic get synchronized automatically without any explicit parameter passing

How to use ngModel?

- ngModel is part of the Forms Module in Angular. It is not imported by default, hence it need to be done explicitly by adding it into the **app.module.ts** file

```
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3  import { FormsModule } from '@angular/forms';
4
5  import { AppComponent } from './app.component';
6  import { BindingComponent } from './binding/binding.component';
7  import { EbindingComponent } from './ebinding/ebinding.component';
8  import { PipesComponent } from './pipes/pipes.component';
9  import { CustpipeComponent } from './custpipe/custpipe.component';
10 import { SummarypipePipe } from './summarypipe.pipe';
11 import { MyformatPipe } from './myformat.pipe';
12 import { TbindingComponent } from './tbinding/tbinding.component';
13
14 @NgModule({
15   declarations: [
16     AppComponent,
17     BindingComponent,
18     EbindingComponent,
19     PipesComponent,
20     CustpipeComponent,
21     SummarypipePipe,
22     MyformatPipe,
23     TbindingComponent
24   ],
25   imports: [
26     BrowserModule,
27     FormsModule
28   ],
```


WSA
Forward looking IT finishing school
Pipes
(Displaying data in a better way)
www.webstackacademy.com

Pipes

- Pipe takes input in the view and transforms into a better readable / understandable format
- Some of the standard pipes are supported, users can create their own pipes (**custom pipes**)
- Multiple pipes can be combined together to achieve the desired functionality
- Pipes are used along with text, from the template along with interpolation
- Pipes make application level handling of text much easier. Multiple pipes can also be combined together
- In case of multiple pipes, output of one pipe is given as input of another

```
Pipe usage: {{ userText | Pipe }}
```

Standard Pipes

Methods	Description
`Currency`	Formats the current input in form of currency (ex: currency:'INR')
`Date`	Transforms dates (shortDate | shortTime | fullDate)
`Decimal`	Formats decimal numbers (number: '2.2-4')
`Lowercase`	Converts input into lower-case
`Uppercase`	Converts input into upper case
`Percent`	Convers into percentage format (percent:'2.2-5')

Exercise

- Using data binding, change the following HTML attributes by changing appropriate DOM element:
  - Font size
  - Left Margin
  - Top Margin

- Using event binding, handle the following events by having event handlers
  - Mouse over
  - Double click
  - Change a HTML attribute of a text (ex: BackgroundColor) when the user click any of the mouse events. Demonstrate view->business logic -> view binding

- Handle the following standard pipes
  - Percentage
  - Currency

Creating a Custom Pipe - GUI

(Implementing your own pipe functionality)

Creating a pipe – GUI mode

- **Step-1: Creating / Adding your pipe TS file**
  - Open your Angular project folder (ex: myFirstApp) in your editor (ex: VS code). Go to `src/app` folder.
  - Add a new file in the following format: <pipe_name>.pipe.ts (ex: `summarypipe.pipe.ts`)
  - Add the following lines into your new pipe file. Details given as comments.

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'summarypipe'  // Pipe name
})
export class SummarypipePipe implements PipeTransform {

transform(value: any, args?: number): any {

  // Implement your pipe functionality here

}
```

Implements interface

- Implements interface ensures classes must follow a particular structure that is already defined
- Basically ensures the class is of same ‘shape’

```
interface Point {
  xValue : number;
  yValue : number;
}
```

```
class MyPoint implements Point {
    xValue: number;
    yValue: number;
}
```

```
class MyPoint implements Point {
    xValue: number;
    yValue: number;
    zValue: number;
}
```

PipeTransform Interface

- The PipeTransform interface is defined in Angular as follows
- The **transform** method is where the custom pipe functionality is implemented

```
interface PipeTransform {
        transform(value: any, ...args: any[]): any
}
```

Creating a pipe – GUI mode

- **Step-2: Make your pipe as a part of the module (app.module.ts file)**
  - Add your new pipe (ex: SummaryPipe) under the declarations sections of the module.
  - Ensure you import appropriate pipe files, similar to components

```
@NgModule({
declarations: [
AppComponent,
SummaryPipe
],
imports: [
BrowserModule,
],
providers: [],
bootstrap: [AppComponent]
})
```

Creating a pipe – GUI mode

- **Step-3: Invoking / Using the pipe**
  - Pipe need to be invoked by the component from its template file

```
<h2>Custom pipe - Example</h2>
   {{ myText | summarypipe }}
```

- **Step-4: Save all and compile. U should be able to use your custom pipe now**

Creating a Custom pipe - CLI

(CLI mode for custom pipes)

Creating a component – CLI mode

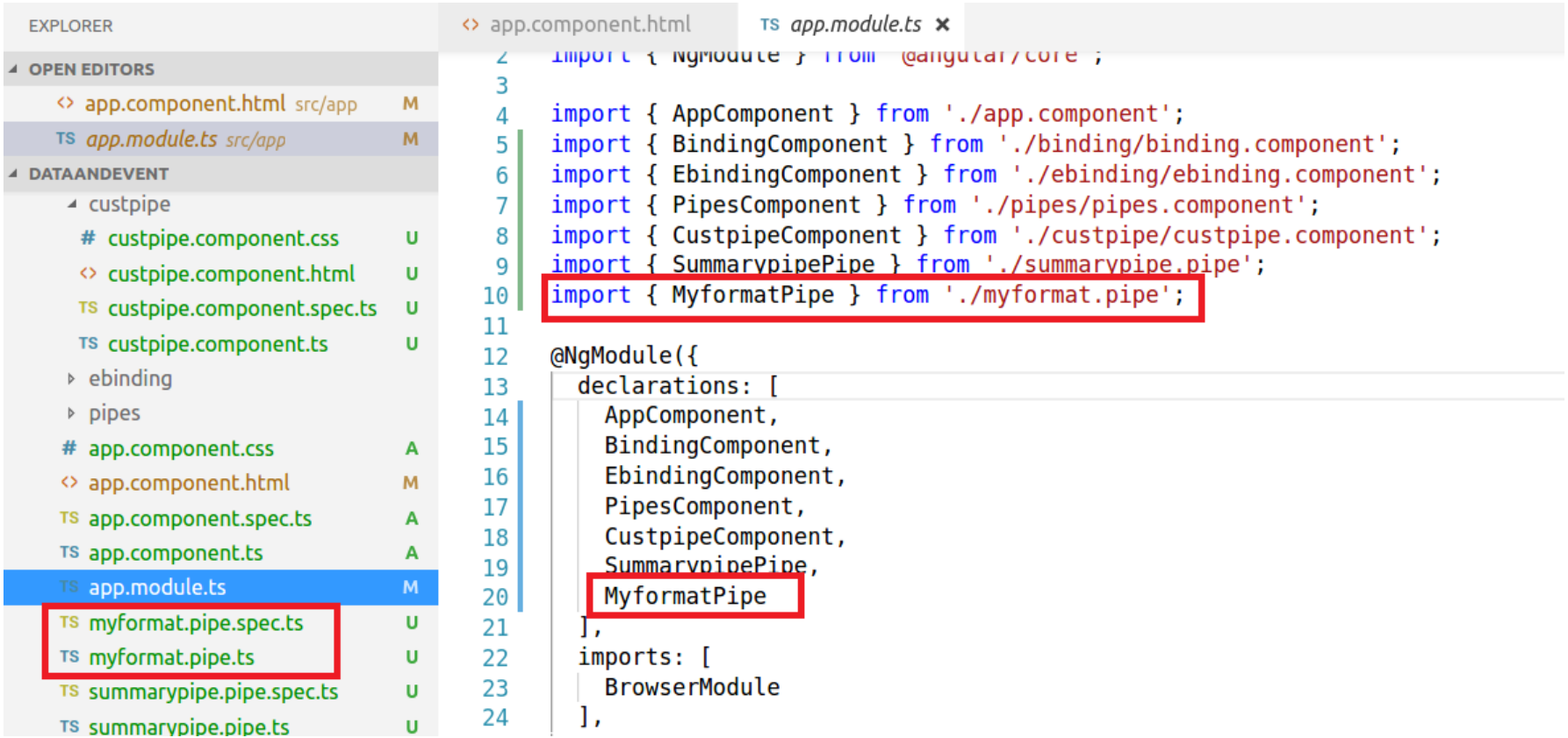
- Sometimes you might find it little hard to create a component using GUI. In case you miss out any steps, the app will not work.

- Angular CLI offers much more reliable way to create a pipe (similar to component and services). Try out the following command, which will not only generate a component but also make appropriate entries

```
$ ng g p myformat                    // Creates a new pipe
```

```
wsa@wsa-VirtualBox:~/Angular/DataAndEvent$ ng g p myformat
CREATE src/app/myformat.pipe.spec.ts (195 bytes)
CREATE src/app/myformat.pipe.ts (205 bytes)
UPDATE src/app/app.module.ts (799 bytes)
```

Creating a pipe – CLI mode



Exercise

- Create a custom pipe to find out the zodiac sign of the user
- Input is given as an entry in CSV file format as follows:

```
Rajan,9845123450,rajan@gmail.com,22/05/1990,Bangalore
```

- Here are the Zodiac signs for all the months:
  - Aries (March 21-April 19)
  - Taurus (April 20-May 20)
  - Gemini (May 21-June 20)
  - Cancer (June 21-July 22)
  - Leo (July 23-August 22)
  - Virgo (August 23-September 22)
  - Libra (September 23-October 22)
  - Scorpio (October 23-November 21)
  - Sagittarius (November 22-December 21)
  - Capricorn (December 22-January 19)
  - Aquarius (January 20 to February 18)
  - Pisces (February 19 to March 20)

WSA | Forward looking IT finishing school

WebStack Academy

#83, Farah Towers,
1st Floor, MG Road,
Bangalore – 560001

M: +91-809 555 7332
E: training@webstackacademy.com

WSA in Social Media:

www.webstackacademy.com