Table of Contents

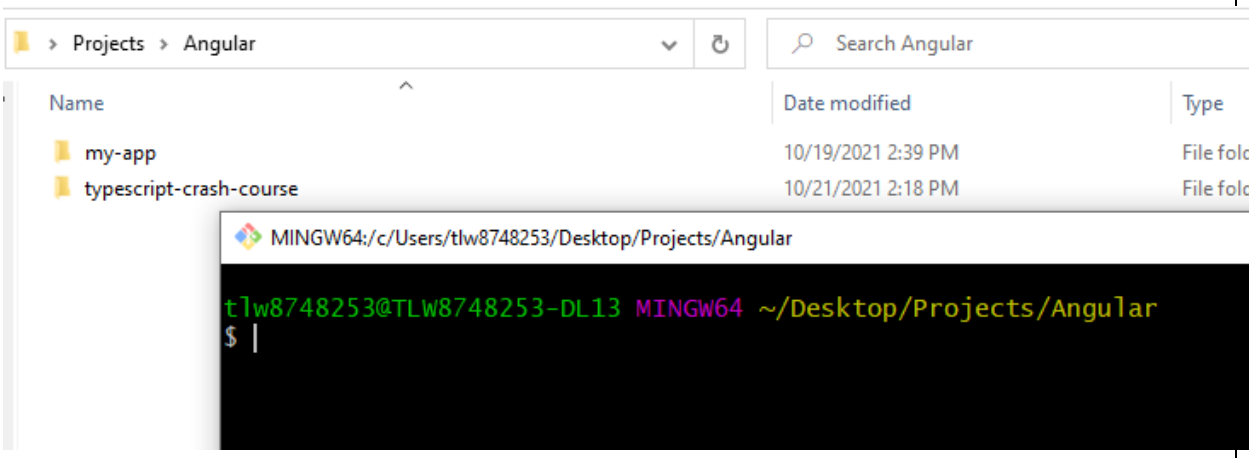
Aug 25th 2021 recording

One way and two way databinding

Appendix Aug 25th 2021: angular-one-way-databinding.md

Create new Angular Project

Open a Git Bash window in your Angular project folder.



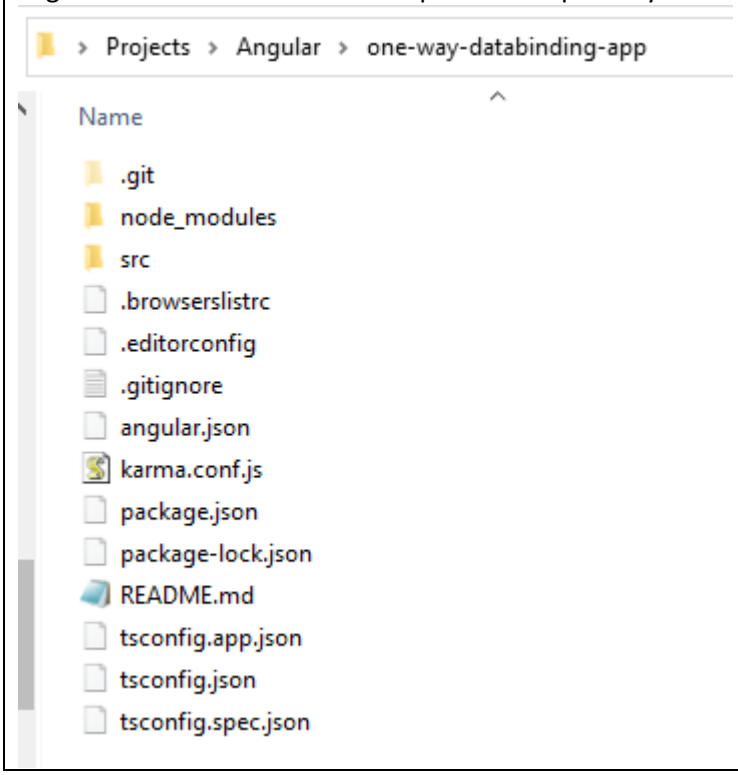
If not previously installed, install the Angular command line interface:

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

Create a new project:

```
ng new one-way-databinding-app
```

Once the command completes you will have all the initial files created for an Angular project including a .git folder which can be used to push to a repository.



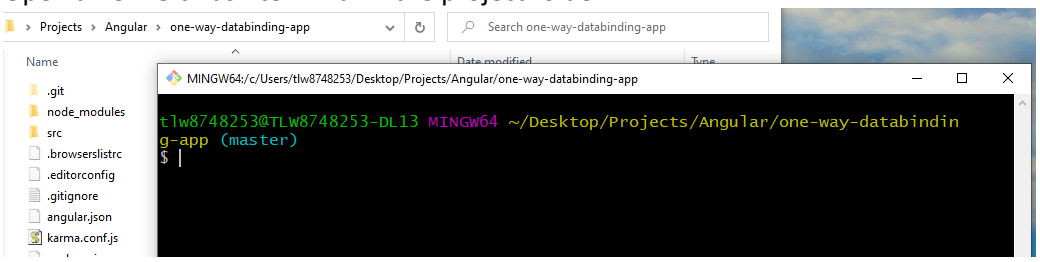
Open the new project in VS Code

Close the my-app folder and open the one-way-databinding-app folder.
Open index.html

Add bootstrap to the index.html
We will download and include the bootstrap files instead of running as an external link. This way if the external site is unavailable our project will still work.

https://getbootstrap.com/docs/5.0/getting-started/introduction/

Open a new Git Bash terminal in the project folder



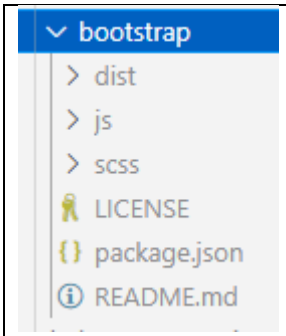
Install bootstrap:

```
npm install bootstrap
```

In the package.json file there should now be a bootstrap dependency.

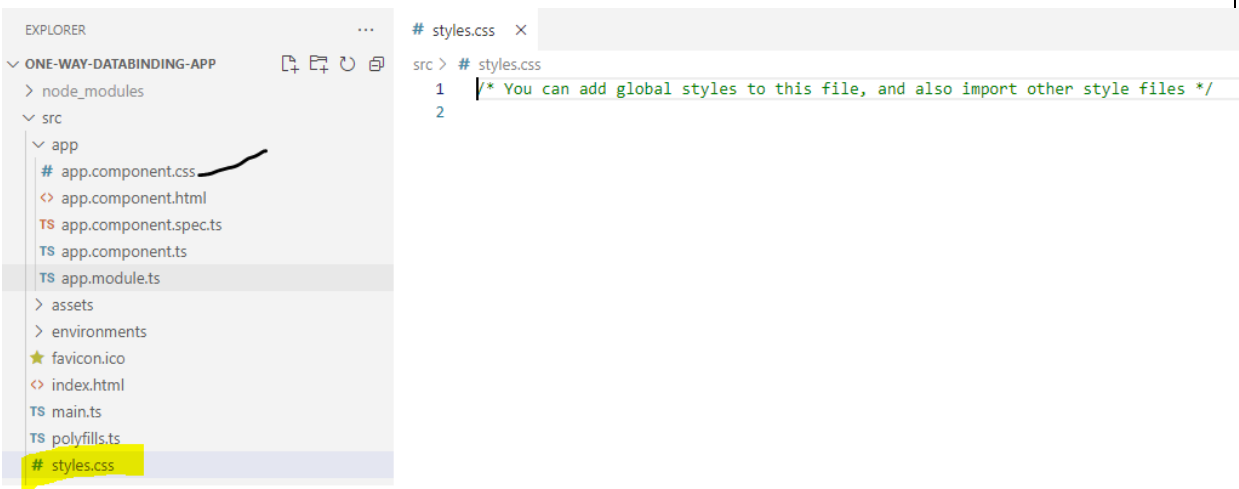
```
"dependencies": {
  "@angular/animations": "~12.2
  "@angular/common": "~12.2.0",
  "@angular/compiler": "~12.2.0
  "@angular/core": "~12.2.0",
  "@angular/forms": "~12.2.0",
  "@angular/platform-browser":
  "@angular/platform-browser-dy
  "@angular/router": "~12.2.0",
  "bootstrap": "^5.1.3",
  "rxjs": "~6.6.0",
  "tslib": "^2.3.0",
  "zone.js": "~0.11.4"
},
```

There is also a bootstrap folder under node_modules


bootstrap
dist
js
scss
LICENSE
package.json
README.md

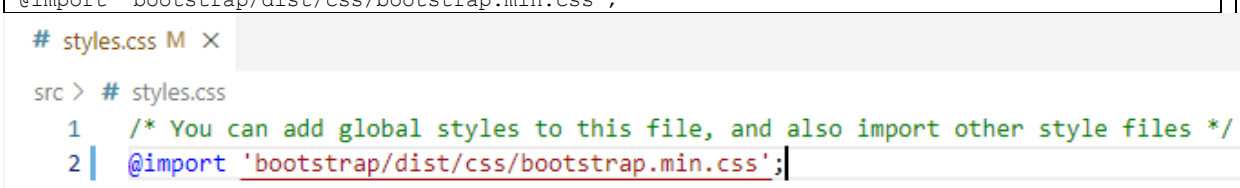
Apply bootstrap style globally

Open the global style file styles.css. Definitions here will cascade globally to all HTML files. Unlike definitions in <name>.components.css files which are local to that components HTML file.



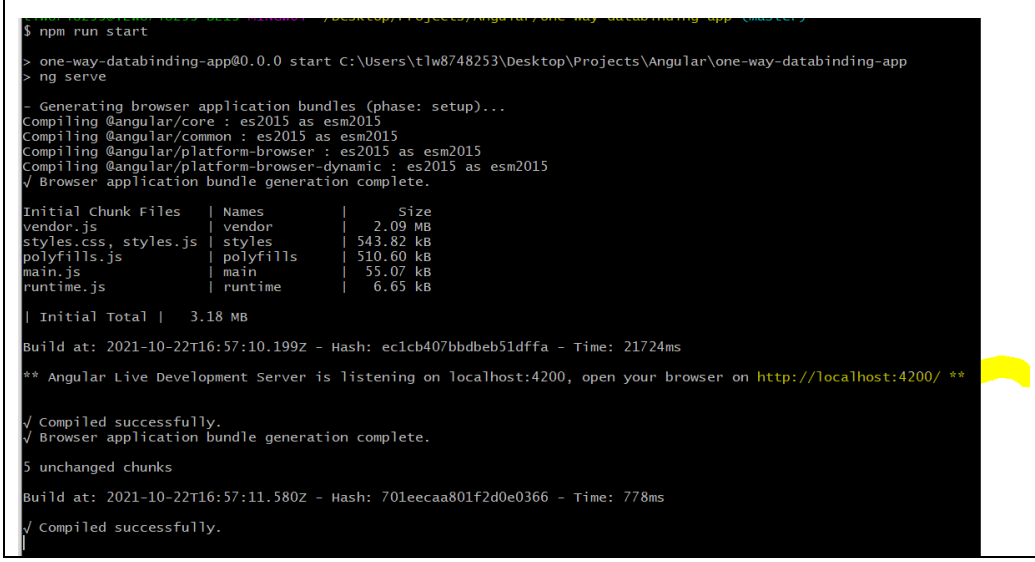
Import a bootstrap style in styles.css. Must give full path and filename:

```
@import 'bootstrap/dist/css/bootstrap.min.css';
```



Check the changes.
In the Git Bash terminal window:

```
npm run start
```

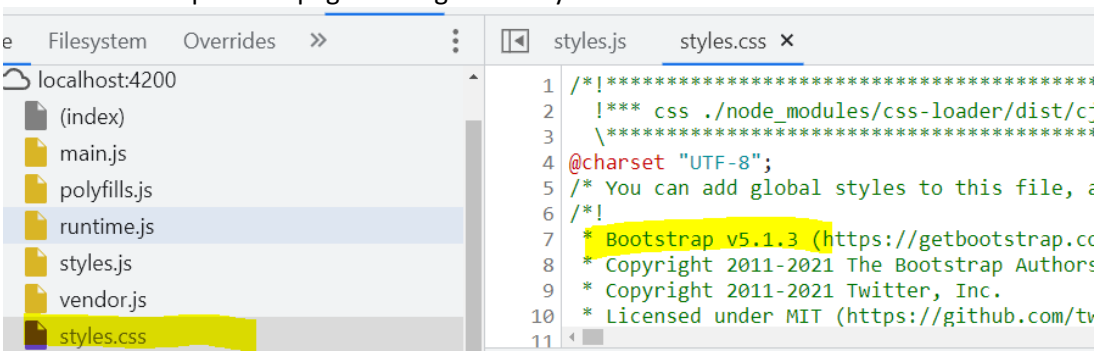


Open the webpage:

```
http://localhost:4200/
```

You only see slight changes between the original page and the one with the bootstrap style at this time.

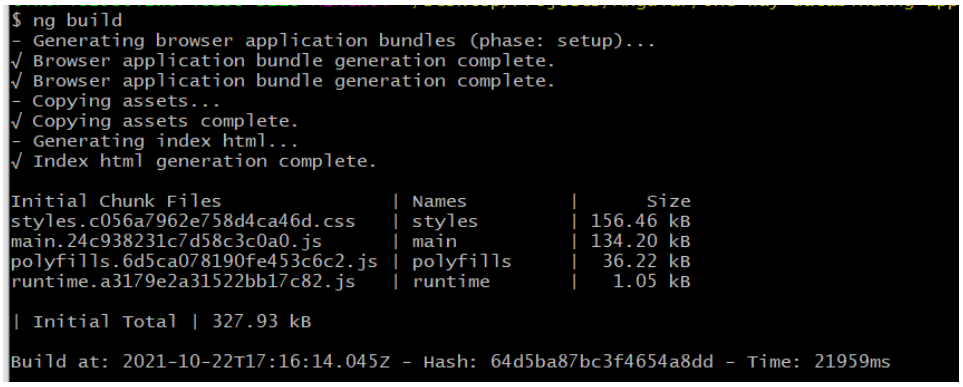
You can also inspect the page looking at the style.css file and see:



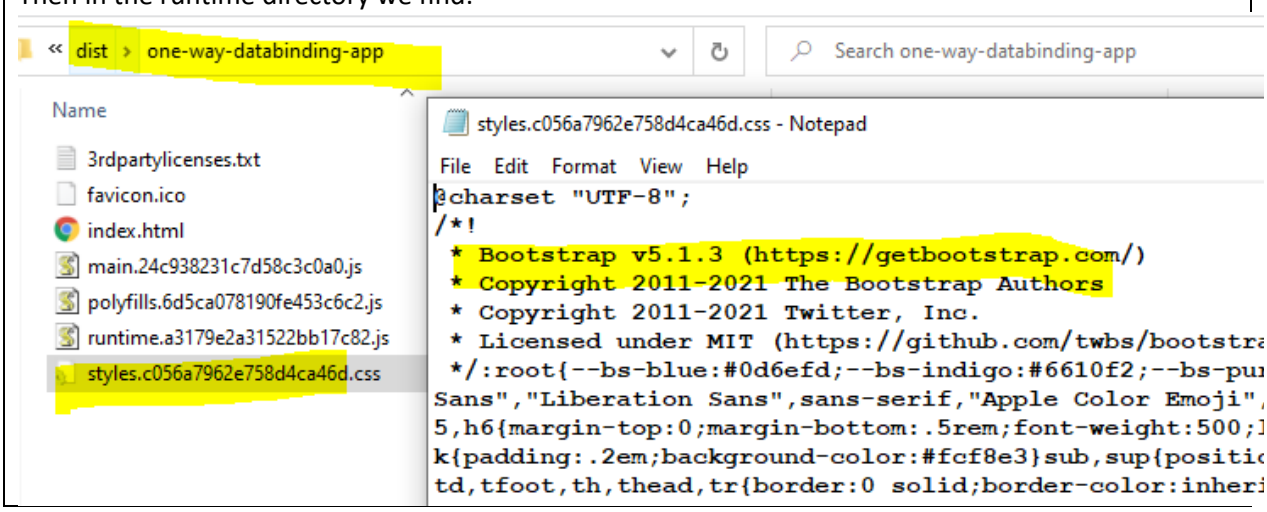
Now if you stop the application ctrl-c in the Git Bash terminal window then build the application:

```
ng build
```

This will include the bootstrap dependency in the runtime files.



Then in the runtime directory we find:



Start the server again.
In the Git Bash terminal window:

```
npm run start
```

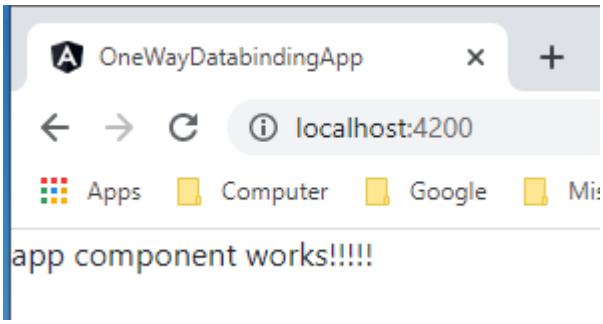
Update the project files

app.component.html

Remove all default code from app.component.html.
Add the following line:

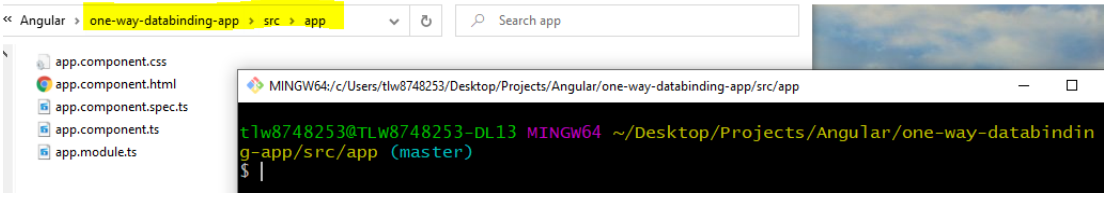
```
<p>app component works!!!!!</p>
```

Check the results:



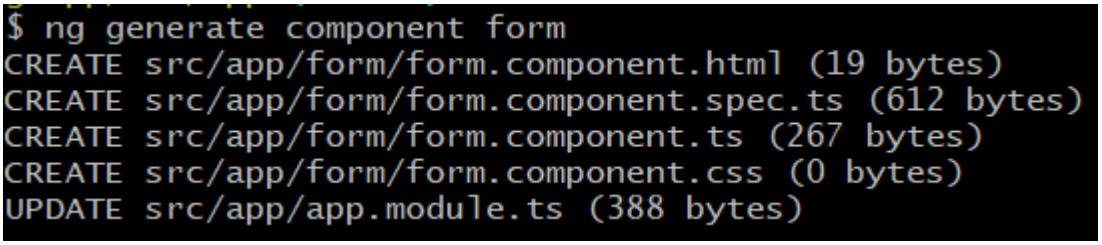
Create new component

Open a second Git Bash window in the project's src/app folder:

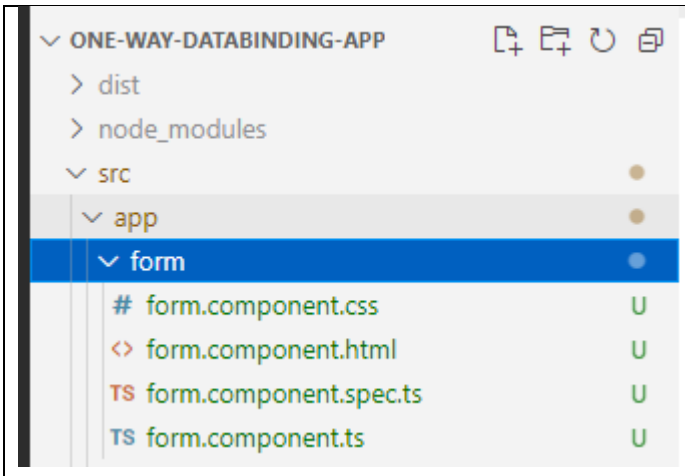


Create a form component

```
ng generate component form
```

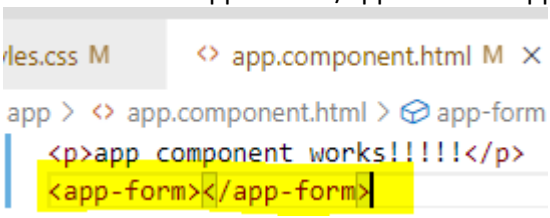


The form component files are created:

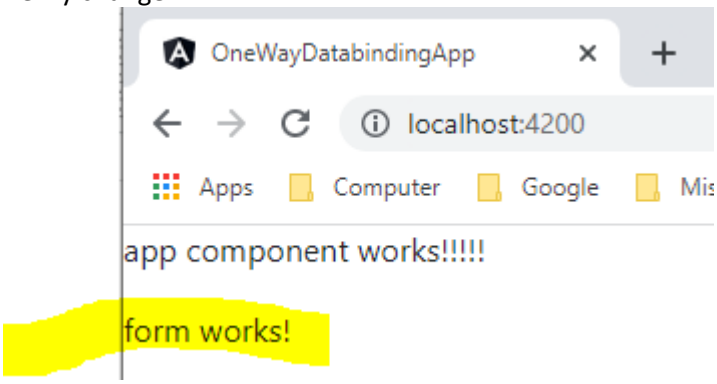


In the form.component.ts we find the selector element and find how to reference the new component:

Add reference <app-form></app-form> to app.component.html:



Verify change:



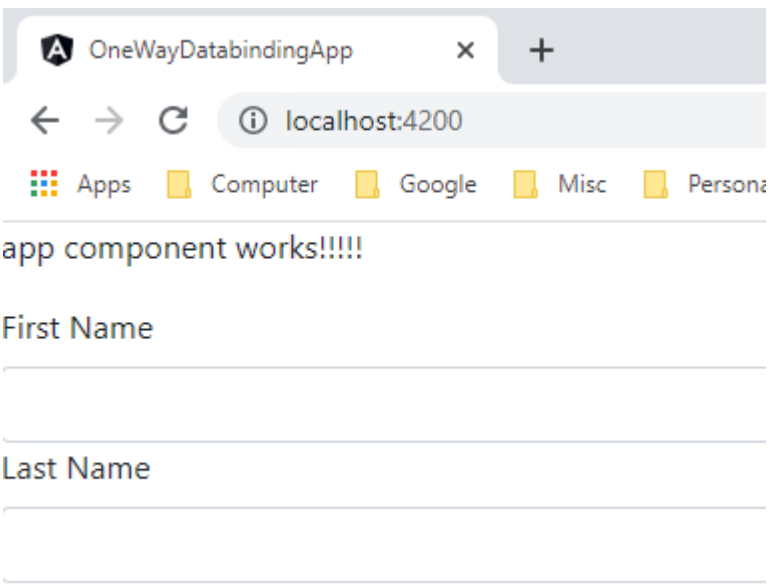
Update form.component.html and form.component.ts files

Once all the code in this section is complete it will demonstrate the following:

1. Event data binding (as we type into the input fields)
2. String Interpolation binding (as input is type it is sent to the output area)
3. Property binding (with the image display)

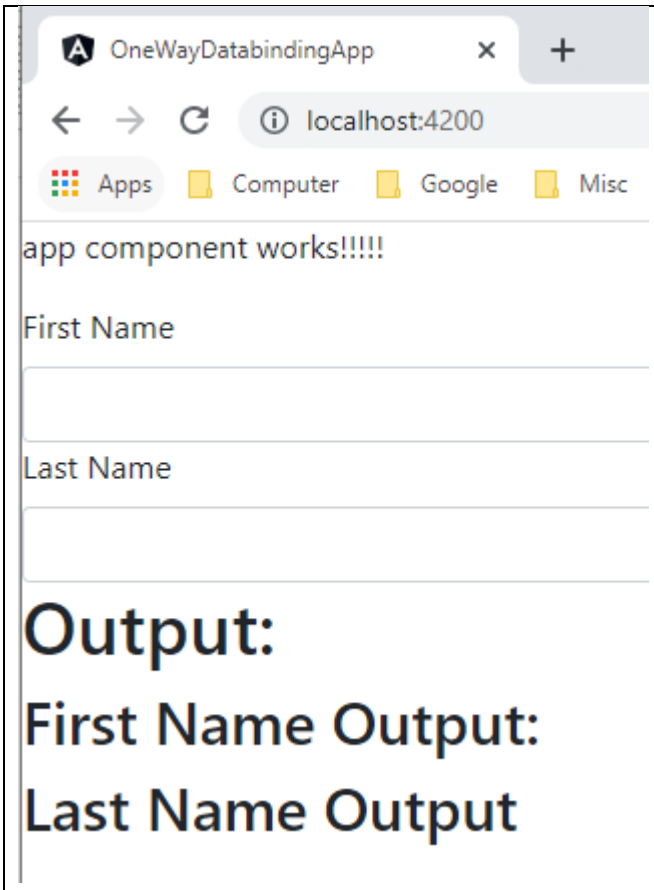
Replace the existing HTML code in form.component.html with the following.

```
<form>
    <div>
      <label for="firstname" class="form-label">First Name</label>
      <input type="text" class="form-control" id="firstname">
    </div>
    <div>
      <label for="lastname" class="form-label">Last Name</label>
      <input type="text" class="form-control" id="lastname">
    </div>
</form>
```



Add the following below the </form> tag. This is a placeholder for databinding output after more modifications to the HTML.

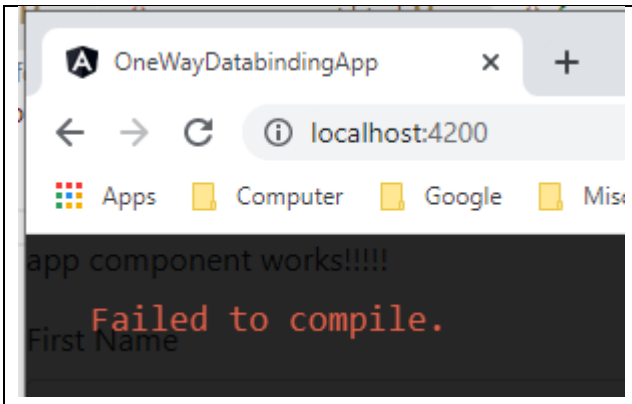
```
<h1>Output:</h1>
<div>
    <h2>First Name Output:</h2>
    <h2>Last Name Output</h2>
</div>
```



Add an event to the HTML form items:
(input)="onFirstNameInput($event)"
(input)="onLastNameInput($event)"

```
<form>
  <div>
    <label for="firstname" class="form-label">First Name</label>
    <input type="text" class="form-control" id="firstname" (input)="onFirstNameInput($event)">
  </div>
  <div>
    <label for="lastname" class="form-label">Last Name</label>
    <input type="text" class="form-control" id="lastname" (input)="onLastNameInput($event)">
  </div>
</form>
```

As you make the changes and save the file and error is produced. This is because the event handlers are not in the .ts file yet.



Initial update of form.component.ts.

Update form.component.ts with event listeners. Place the code after the ngOnInit(): void { } method.

```
  onFirstNameInput(event: Event) {
    this.firstName = (event.target as HTMLInputElement).value;
  }

  onLastNameInput(event: Event) {
    this.lastName = (event.target as HTMLInputElement).value;
  }
```

Add variable definition above the constructor(){}

```
  firstName: string = "";
  lastName: string = "";
```

Add the following below the </form> tag. This is a placeholder for databinding output after more modifications to the HTML.

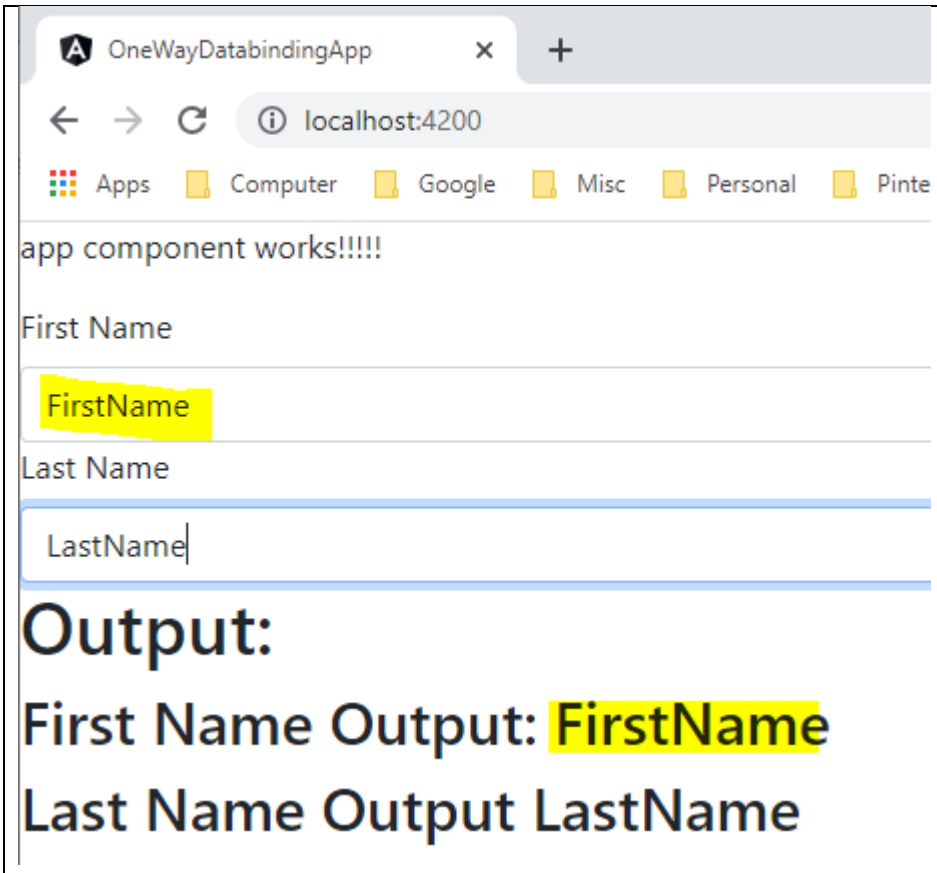
```
<h1>Output:</h1>
<div>
    <h2>First Name Output:</h2>
    <h2>Last Name Output</h2>
</div>
```

Update form.component.html with String Interpolation databinding.

Add {{ firstName }} and {{ lastName }} to the HTML output area:

```
<h1>Output:</h1>
<div>
    <h2>First Name Output: {{ firstName }}</h2>
    <h2>Last Name Output {{ lastName }}</h2>
</div>
```

As you type into the input fields on the web page, the interpolation databinding will show what is typed in the output area:



Update the output area title from Output: to:

```
<h1>Output (Using String Interpolation):</h1>
```

Output (Using String Interpolation):

Update form.component.html with an image link in the form.

Below the last name closing add the following:

```
  <div>
    <label class="form-label">Image Link</label>
    <input type="text" class="form-control" (input)="onImageLinkInput($event)">
  </div>
```

Add an output area for the image below the <h1> area for Interpolation add the following:

```
<h1>Output (Using property binding):</h1>
<div>
  <h2>Image Output:</h2>
  <img [src]="imageLink">
</div>
```

Update form.component.ts with the image event handler.

Add the image variable about the constructor(){}

```
imageLink: string = "";
```

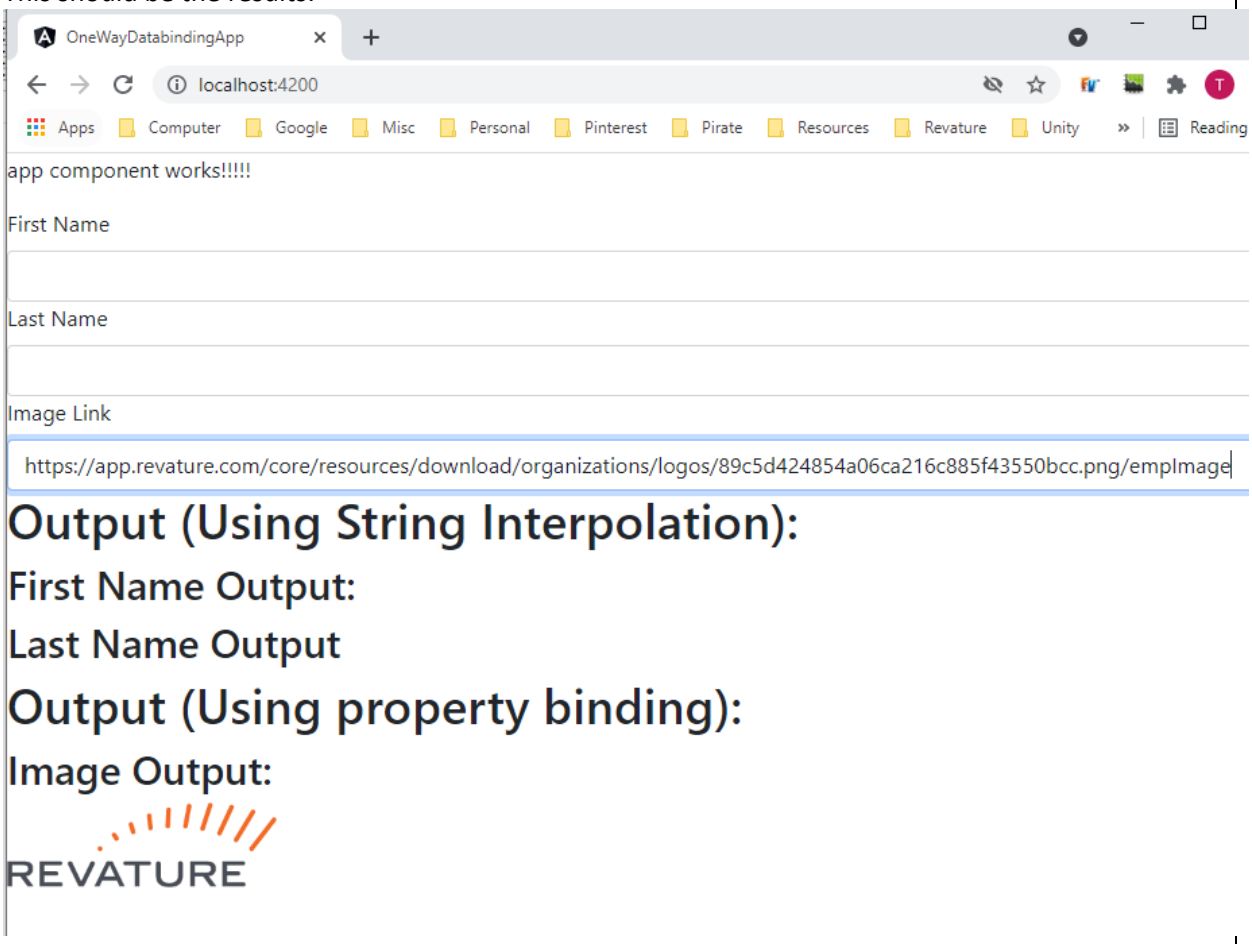
Add the image event handler after the last name handler:

```
onImageLinkInput(event: Event) {
  this.imageLink = (event.target as HTMLInputElement).value;
}
```

Test the image link.
Find an internet web image or use the following and enter into the Image Link text area:

```
https://app.revature.com/core/resources/download/organizations/logos/89c5d424854a06ca216c885
f43550bcc.png/empImage
```

This should be the results:



Update form.component.css to resize the image

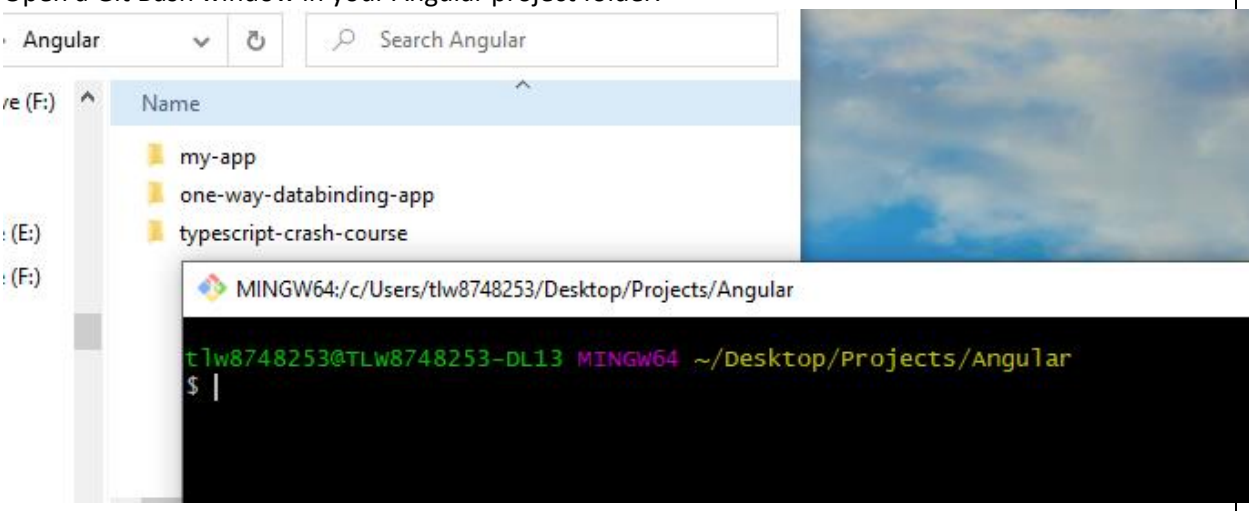
If desired you can add styling to be applied only to the form component HTML by adding the following code:

```
img {
    width: 200px;
    height: 200px;
  }
```

This will control the size of the image and as is the nature of Angular the styling is only applied to the specific component.

two-way-databinding-app

Open a Git Bash window in your Angular project folder.

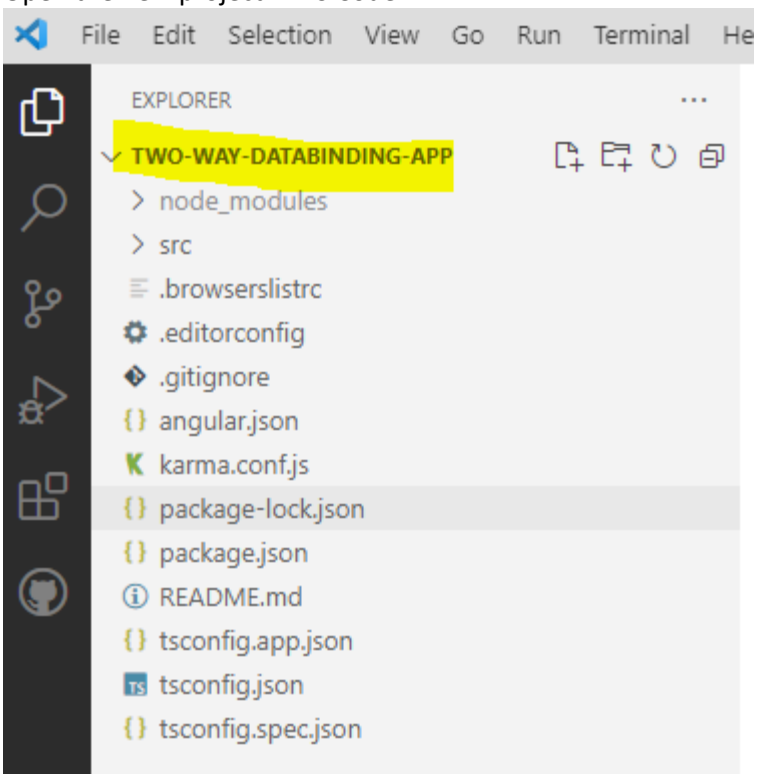


Create a new project two-way-databinding-app:

```
ng new two-way-databinding-app
```

Let installation complete.

Open the new project in VS Code.



Create new component: form

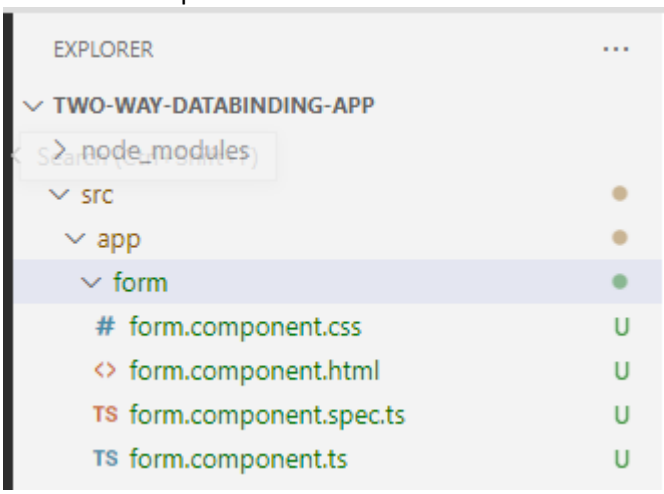
In Git Bash window cd two-way-databinding-app/src/app

```
cd two-way-databinding-app/src/app
```

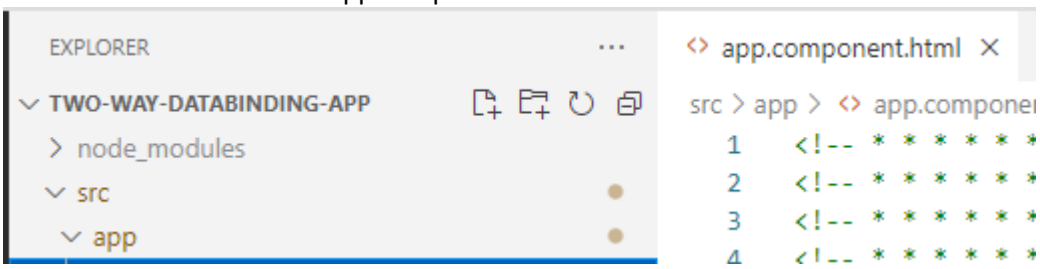
Create form component

```
ng generate component form
```

New form component is built:

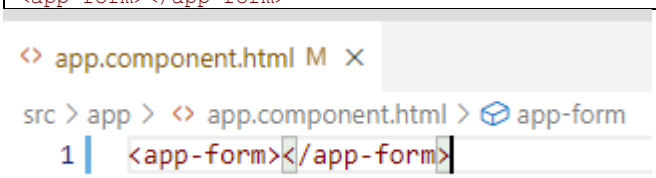


Delete all from code inside app.component.html



Add the form component to the app.component.html

```
<app-form></app-form>
```



Update form.component.html

Replace existing code with:

```
<div>
    <label class="form-label">First Name</label>
    <input type="text" class="form-control">
</div>
```

In Git Bash window:

```
npm install bootstrap
```

```
$ npm install bootstrap
```

Add bootstrap import globally to styles.css

```
@import 'bootstrap/dist/css/bootstrap.min.css';
```

app.component.html M | form.component.ts U | form.component.html U | styles.css M

src > styles.css

```
/* You can add global styles to this file, and also import other style files */
@import 'bootstrap/dist/css/bootstrap.min.css';
```

Add the following about the constructor in form.component.ts

```
firstName: string = "";
```

app.component.html M | form.component.ts U | form.con

src > app > form > form.component.ts > FormComponent

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-form',
  templateUrl: './form.component.html',
  styleUrls: ['./form.component.css']
})
export class FormComponent implements OnInit {

  firstName: string = "";

  constructor() { }

  ngOnInit(): void {
  }

}
```

Make use of default components in app.modules.ts

app.component.html M | form.component.ts U | app.module.ts M

src > app > app.module.ts > ...

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';
import { FormComponent } from './form/form.component';

@NgModule({
  declarations: [
    AppComponent,
    FormComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Add import to app.modules.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
```

```
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { FormComponent } from './form/form.component';

@NgModule({
  declarations: [
    AppComponent,
    FormComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

app.component.html M TS form.component.ts U TS app.module.ts M ×

rc > app > TS app.module.ts > AppModule

```
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3  import { FormsModule } from '@angular/forms';
4
5  import { AppComponent } from './app.component';
6  import { FormComponent } from './form/form.component';
7
8  @NgModule({
9    declarations: [
10     AppComponent,
11     FormComponent
12   ],
13   imports: [
14     BrowserModule,
15     FormsModule
16   ],
17   providers: [],
18   bootstrap: [AppComponent]
19 })
20 export class AppModule { }
21
```

AppComponent and FormComponet have access to FormsModule.
FormsModule is what allows the two way databinding.

Now bind the firstName element in form.component.html, update the following line of code:

```
From:
        <input type="text" class="form-control">
To:
        <input [(ngModel)]="firstName" type="text" class="form-control">
```

app.component.html M | form.component.ts U | app.module.ts M | form.component.html U

app > form > form.component.html > div

```
1 <div>
2     <label class="form-label">First Name</label>
3     <input [(ngModel)]="firstName" type="text" class="form-control">
4   </div>
```

Plus add the following code to the form.component.html

```
<div>
  <h1>Output:</h1>
  <h2>First Name Output: {{ firstName }}</h2>
</div>
```

app.component.html M | form.component.ts U | app.module.ts M | form.component.html U

c > app > form > form.component.html > div

```
1  <div>
2      <label class="form-label">First Name</label>
3      <input [(ngModel)]="firstName" type="text" class="form-control">
4    </div>
5
6    <div>
7      <h1>Output:</h1>
8      <h2>First Name Output: {{ firstName }}</h2>
9    </div>
```

In Git Bash window start the application

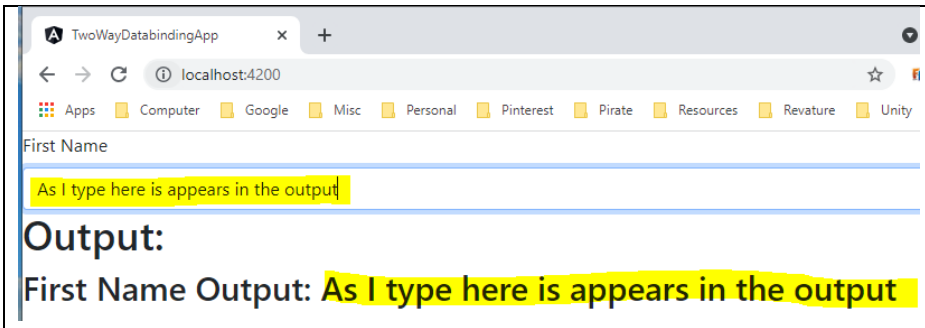
```
npm run start
```

```
$ npm run start

> two-way-databinding-app@0.0.0 start C:\Users\tlw8748253\Desktop\Projects\Angular\two-way-databin
ding-app
> ng serve

- Generating browser application bundles (phase: setup)...
Compiling @angular/core : es2015 as esm2015
Compiling @angular/common : es2015 as esm2015
Compiling @angular/platform-browser : es2015 as esm2015
Compiling @angular/forms : es2015 as esm2015
Compiling @angular/platform-browser-dynamic : es2015 as esm2015
√ Browser application bundle generation complete.

Initial Chunk Files   | Names         |      Size
vendor.js             | vendor        |   2.39 MB
styles.css, styles.js | styles        | 543.82 kB
polyfills.js          | polyfills     | 510.60 kB
main.js               | main          |  10.22 kB
runtime.js            | runtime       |   6.65 kB

| Initial Total |   3.44 MB

Build at: 2021-10-25T17:00:56.695Z - Hash: 8db56e082e3ccaf18929 - Time: 26425ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://loc
alhost:4200/ **


√ Compiled successfully.
√ Browser application bundle generation complete.

5 unchanged chunks

Build at: 2021-10-25T17:00:58.034Z - Hash: 832e2ea63f2611b6192f - Time: 702ms

√ Compiled successfully.
√ Browser application bundle generation complete.

Initial Chunk Files | Names |     Size
main.js             | main  | 11.01 kB

4 unchanged chunks

Build at: 2021-10-25T17:02:42.223Z - Hash: dfedf3659a5165442892 - Time: 354ms

√ Compiled successfully.
```

Open application in browser:

```
http://localhost:4200/
```

Unlike one way databinding, no additional code or callback function is needed to tie the elements together:



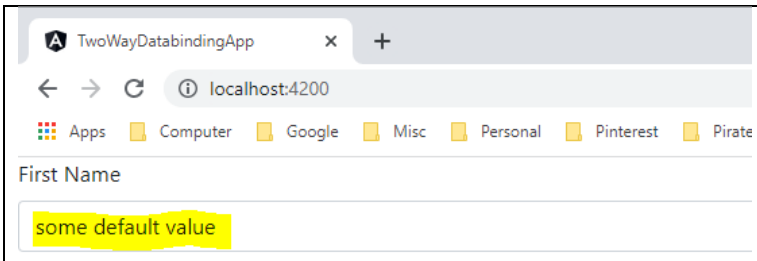
The [(ngModel)] directive replaces the code used in one way databinding to tie the elements together.

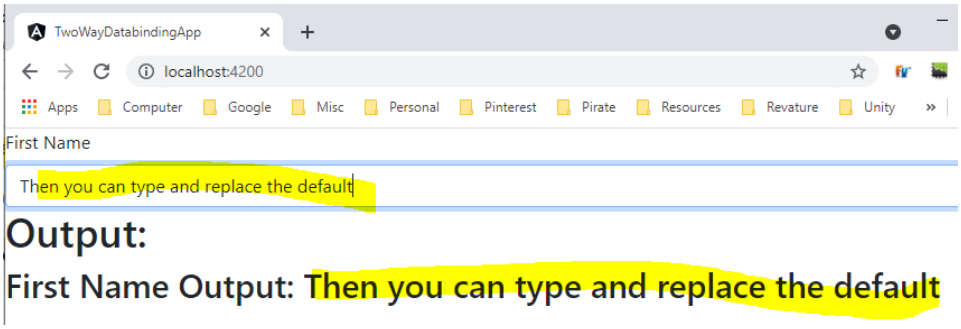
```
app.component.html M   TS form.component.ts U   TS app.module.ts M   form.component.html U
'c > app > form > form.component.html > div
1  <div>
2      <label class="form-label">First Name</label>
3      <input [(ngModel)]="firstName" type="text" class="form-control">
4    </div>
```

Demonstrate the two way nature of the binding

To demonstrate the two way databinding if you add some default value in the form.component.ts it will appear on the page.

```
app.component.html M   TS form.component.ts U   TS app.module.ts
src > app > form > TS form.component.ts > FormComponent > firstName
1   import { Component, OnInit } from '@angular/core';
2
3   @Component({
4     selector: 'app-form',
5     templateUrl: './form.component.html',
6     styleUrls: ['./form.component.css']
7   })
8   export class FormComponent implements OnInit {
9
10    firstName: string = "some default value";
11
12    constructor() { }
13
14    ngOnInit(): void {
15    }
16
17  }
```





[Appendix Aug 25th 2021: angular-two-way-databinding.md]

Appendix Aug 25th 2021: angular-one-way-databinding.md

One-way Data Binding

At this point, we have already established the structure of components. Components consist of 3 different files that specify
- The structure of that component (<name>.component.html)
- The styling of that component (<name>.component.css)
- The logic of that component (<name>.component.ts)

Data binding is the process whereby communication occurs between our component and the DOM. The way that data binding occurs is through binding data
- From the component HTML template to the component class
- From the component class to the HTML template

There are two ways of performing data binding:
1. One-way data binding
2. Two-way data binding

We will examine the 3 different types of ONE-WAY data binding:
1. String interpolation (Component class to HTML template)
2. Property binding (Component class to HTML template)
3. Event binding (HTML template to the component class)

String Interpolation
String interpolation is the process whereby data can be passed from the component class to the HTML template. This is accomplished using double curly brace syntax `{{ }}`

Example: If we have inside our our app.component.ts file a variable called `username` with the value `user12345`

```typescript
export class AppComponent {
   username = "user12345";
}
```

We can pass the value of the `username` variable over to the template to be rendered whenever that component is displayed.

```html
<p>{{ username }} is logged in.</p>
```

Property Binding
Property binding is also a process whereby data is passed from the component class to the template. Where it is different than property binding is that property binding is used to bind values to the attributes of HTML elements.

What is an HTML attribute again? It is any property that is defined for a particular HTML element that usually has a corresponding value (not always). For example, the `<img>` tag will usually have an associated `src` attribute to specify an image file to display. Other common attributes include `href`, `id`, `class`, `name`, etc.

Example: inside of the app.component.ts file, we might have a variable containing the link to an image

```typescript
export class AppComponent {
   image = "http://somewebsite.com/myimage.jpg";
}
```

Inside of the app.component.html file, we can have the `src` attribute of the img tag binded to the image variable.

```html
<h1>Property binding</h1>
<img [src]="image">
```

Event Binding
Event binding, as opposed to string interpolation and property binding, is a way to pass data from the HTML template to the component class. In this case, we bind DOM events such as keystrokes, clicks, mouseovers, etc. to some function that is defined in the component class. This function will then be executed whenever this event occurs (on some element)

For example, if we have an input element that we are typing into, we can bind the `change` event to detect when the value of the input element is changed. This will allow us to grab the value property of the input element and update a variable defined in the component class, for example.

```html
<input type="text" (change)="onChange($event)">
```
- Here we bind the change event such that the onChange function will be invoked
- The $event argument is a special argument in Angular that will pass the event object itself over to the function when it is invoked in the component class

```typescript
export class AppComponent {

   text = "";

   onChange(event) {
      this.text = event.target.value;
   }
}
```

Appendix Aug 25th 2021: angular-two-way-databinding.md

Two-way databinding

To achieve passing data from an input element to the component class using one-way databinding, we would've had to perform event binding on the `input` event, call a function defined over in the component class with the event object being passed to this function, and then getting the target element of that event and setting the value property of that target element to the variable we want to change over in our component class.

However, with two-way databinding, there are much fewer steps required in order to link input elements' values with variables defined in the component class. The way we can set this up is through importing the `FormsModule` and making that available to our components within our <module name>.module.ts file.

By default, whenever we generate components, they will belong to the AppModule itself. A module is a group of components that should be able to share functionalities from the different imports that are declared inside of the AppModule file.

```typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';
import { FormComponent } from './form/form.component';

@NgModule({
  declarations: [
    AppComponent,
    FormComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Normally the AppModule would look something like the above. In this case, we have the default AppComponent and our own user defined FormComponent that belong to this module. We have available to us all of the functionalities that are inside of the BrowserModule, which as you can see is listed in the imports property of the @NgModule decorator.

To make use of two-way databinding, which is provided through the `[(ngModel)]` directive, we need to first of all gain access to this directive. It is part of the FormsModule, which we would need to import. So, making the appropriate changes to our AppModule file would allow for the AppComponent AND FormComponent to have access to this directive.

```typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { FormComponent } from './form/form.component';

@NgModule({
  declarations: [
    AppComponent,
    FormComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
```

```
})
export class AppModule { }
```

Making use of ngModel

If we want to bind a variable in our component class with the input element's value, we just need to place the `[(ngModel)]` decorator on our input element and then set its value to the variable name

```html
<div>
  <label class="form-label">First Name</label>
  <input [(ngModel)]="firstName" type="text" class="form-control">
</div>
```

Over in our component class, we simply need to have a variable defined for that component object.

```typescript
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-form',
  templateUrl: './form.component.html',
  styleUrls: ['./form.component.css']
})
export class FormComponent implements OnInit {

  firstName: string = "";

  constructor() { }

  ngOnInit(): void {
  }

}
```