# Angular
# Lab 2
# Parent and Child Components



## 1 Lab setup

Make sure you have the following items installed

- Latest version of Node and NPM (note: labs are tested with Node v16.13 and NPM v8.1 but should work with later versions with no or minimal changes)
- Latest version of TypeScript (note: labs are tested with TypeScript v4.5.4 but should work with later versions with no or minimal changes)
- Latest version of Angular (note: labs are tested with Angular v13.1 but should work with later versions with no or minimal changes)
- Visual Studio Code (or a suitable alternative IDE for Angular/TypeScript)
- A suitable text editor (Notepad ++)
- A utility to extract zip files (7-zip)

In each of the main lab folders, there are two subfolders: `changes` and `final`.

- The `changes` subfolder holds the source code and other related files for the lab. The creation of the Angular app will proceed in a step-wise fashion in order to clearly demonstrate the various features of the framework. The various project source files in this folder are numbered in a way to reflect this gradual construction. For e.g. `xxx.v1.html, xxx.v2.html`, etc represents successive changes that are to be made to the `xxxx.html`

- The `final` subfolder holds the complete Angular project. If there was some issue in completing the step-wise construction of the app using the files from the `changes` subfolder, you can use this to run the final working version.

The project folder in `final` is however missing the crucial `node_modules` subfolder which contains all the dependencies (JavaScript libraries) that your Angular app needs in order to be built properly.

These dependencies are specified in the `package.json` file in the root project folder. They are omitted in the project commit to GitHub because the number and size of the files are prohibitively large.

In order to run the app via the Angular development server (`ng serve`), you will need to create this subfolder containing the required dependencies. There are two ways to accomplish this:

a) Copy the project root folder to a suitable location on your local drive and run `npm install` in a shell prompt in the root project folder. NPM will then create the `node_modules` subfolder and download the required dependencies as specified in `package.json` to populate it. This process can take quite a while to complete (depending on your broadband connection speed)

b) Alternatively, you can reuse the dependencies of an existing complete project as most of the projects for our workshop use the same dependencies. In this case, copy the `node_modules` subfolder from any existing project, and you should be able to immediately start the app. If you encounter any errors, this probably means that there are some missing dependencies or Angular version mismatch, in which case you will then have to resort to `npm install`.

The lab instructions here assume you are using the Chrome browser. If you are using a different browser (Edge, Firefox), do a Google search to find the equivalent functionality.

# 2 Generating a new Angular project

In either a separate command prompt (or shell terminal) or in an embedded terminal in Visual Studio Code, create a new Angular project with:

```
ng new parentchilddemo
```

Press enter to accept the default values for all the question prompts that follow regarding routing and stylesheet.

Navigate into the root project folder from the terminal with:

```
cd parentchilddemo
```

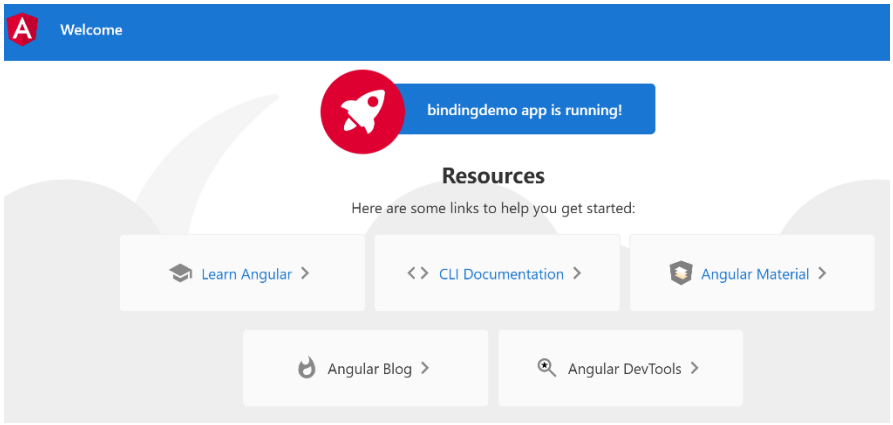
Build the app and serve it via Angular's live development server by typing:

```
ng serve
```

The final output line from this should indicate that the compilation process was successful and identify the port that the live development server is currently serving up the Angular app dynamically from (by default this will be port 4200)

Open a browser tab at:
http://localhost:4200/

You should be able to see the default landing page for all new autogenerated Angular projects.




To stop the development server, type Ctrl+C in the terminal window that it is running in. You can restart again anytime by typing `ng serve` in the project root folder.

# 3 Generating and using a component as a child

The source code for this lab can be found in the `changes` subfolder of `Parent-Child-Demo`

Open a new command prompt (separate from the one that you are running `ng serve` in) in the root folder of the project created earlier. Type this command to generate a new component:

```
ng generate component firstChild
```

By default, this command creates the following:

- A folder named after the component
- A component file, `<component-name>.component.ts`
- A template file, `<component-name>.component.html`
- A CSS file, `<component-name>.component.css`
- A testing specification file, `<component-name>.component.spec.ts`

In addition to generating all the stated files, using `ng generate` will also automatically import the new component class into the root module (AppModule) and add it to the `declarations` array.

The folder created to place the component-related files will be by default inside `src/app` (the folder holding the root component of the app and its associated files).

If camel-case naming is used for the component name in the `generate` command, then each of the component related file names will consist of the individual names of the camel case separated by a hyphen, for e.g.

```
ng generate component TryThisApple
CREATE    src/app/try-this-apple/try-this-apple.component.html    (29
bytes)
CREATE  src/app/try-this-apple/try-this-apple.component.spec.ts  (670
bytes)
…
```

…

The component class itself will have the specified name in the `generate` command suffixed with `Component`. The standard structure for an autogenerated component class includes a constructor and a ngOnInit hook.

If you would like to change the default class and file names, make sure you register the changes in the root module (AppModule) in order for the new component to accessible to Angular.

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v1.html`

Notice that placing the CSS selector for the newly generated component in the template of the root component (AppComponent) results in the template for the new component to be embedded in place of this selector when rendering the complete app view. This is functionally identical to the case of placing the CSS selector for AppComponent in the main `index.html`.

References:

https://angular.io/guide/component-overview#creating-a-component

# 4 Transferring data from parent to child via @Input and property binding

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v2.ts`

`app.component-v2.html`

Modify the following files in `src/app/first-child` with the latest update from `changes`:

`first-child.component-v2.ts`

`first-child.component-v2.html`

To transfer data from the parent to the child component, in the child component we:

- Import the @Input module from @angular/Core Library
- Declare and decorate a property with the @Input decorator. This property will receive data sent from the parent

In the parent component, we

- Bind the @Input property in the child with a parent property in the parent template within the selector tag of the child component

References:

https://angular.io/guide/inputs-outputs#sending-data-to-a-child-component

https://www.tektutorialshub.com/angular/angular-passing-data-child-component

# 5 Transferring data from child to parent via @Output and event binding

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v3.ts`

`app.component-v3.html`

Modify the following files in `src/app/first-child` with the latest update from `changes`:

`first-child.component-v3.ts`

`first-child.component-v3.html`

To transfer data from the child back to the parent component, in the child component:

- Declare a property of type EventEmitter and mark it with a @Output Decorator
- At a particular point (typically in response to an event in the child template), raise an event on this property via the emit method

In the parent component:

- Provide an event binding for the event issued from the child property. The name of the event is the same as the child property and the $event object will contain be of the type of the value emitted from this property
- Typically, the template statement for the event binding will call a method with the $event object to perform further processing.

In this example, the event emitted does not affect the property originally transferred from the parent component (`parentCounter`) to the child. However, it is possible to also change this value in response to an event in the child template, which will now be subsequently to the child property (`childCounter`) via property binding in the usual manner.

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v4.ts`

`app.component-v4.html`

Modify the following files in `src/app/first-child` with the latest update from `changes`:

`first-child.component-v4.ts`

`first-child.component-v4.html`

References:

https://angular.io/guide/inputs-outputs#sending-data-to-a-parent-component
https://www.tektutorialshub.com/angular/angular-pass-data-to-parent-component/

# 6 Accessing child component via template reference variable

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v5.html`

`app.component-v5.ts`

Modify the following files in `src/app/first-child` with the latest update from `changes`:

`first-child.component-v5.ts`

We can access properties / methods in the child component via a template variable. This provides an alternative to using event binding with @Output in certain situations.

Here we remove the event binding for the `textChanged` event that was previously emitted from the child component:
`(textChanged)="processChangeFromChild($event)"`
as we can now directly access the child property `childText` using the template variable `myChild`