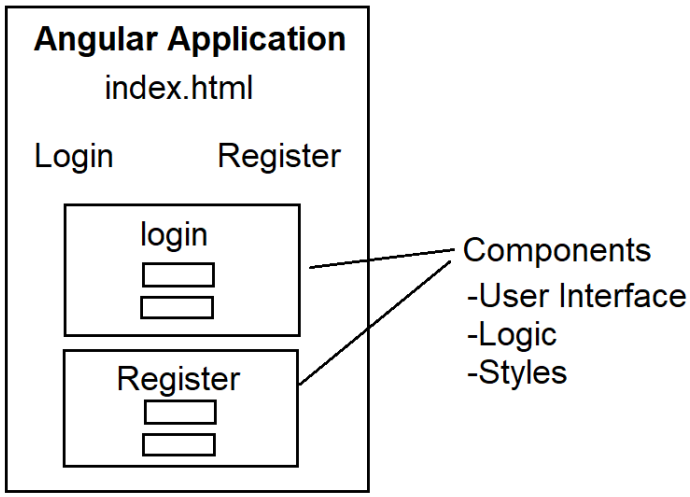
Angular Components

- Components are building-blocks for Angular Application.
- Angular Application is built by using Components at high level.



- Components are responsible for handling user interactions.
- Typically, component is like a template with
  - Presentation
  - Logic
  - Styles
- In Angular the component:
  - Presentation is defined with **HTML**.
  - Logic is defined with **TypeScript**.
  - Styles are defined with **CSS**.
- Angular components can be designed by using 2 techniques
  - Inline documentation Technique
    - Entire component is designed with **one File**, which contains presentation, logic and styles.
  - Code behind documentation Technique.

        - Component is designed with multiple files, clean separation of presentation, logic and styles.
- To create a component, you need a “TypeScript” class.
- To give component behaviour you need a component marker/decorator/directive: “@Component()”
- “@Component” is derived from “Component” base class.
- **Component** base is the member of “@angular/core” library

  Syntax:

  home.component.ts

  import { Component } from ‘@angular/core’;
  @Component()
  export class HomeComponent
  {
  }
- **@Component()** directive provides meta data, which is used by compiler to process your component.
- **Component meta data comprises of several properties and methods which includes the details about:**
    - What the markup to render when component is requested?
    - How the component needs to be accessed?
    - What is code file?
    - What animations to use?
    - What is styles file? Etc.

  Syntax:
  @Component({

```
    selector: ' ',
    template: ' ',
    templateUrl:' ',
    styles: [ ],
    stylesUrl: [ ],
    animations: [ ]
})
```

Adding a new Component to Project

- There are two techniques
  - Explicitly
    - Manually adding and configuring the dependencies.
  - Implicitly
    - Generating by using commands, which will take care about all dependencies.
    - It uses Angular CLI commands.

Angular Documentation Techniques

- There are two documentation techniques
  - Inline Documentation
  - Code Behind Documentation

Explicitly adding a component using Inline Documentation Technique:

- Inline documentation technique allows the developer to configure presentation, logic and styles all in one file.

- It is good for simple component that doesn't require regular extension.
- It is good for configuring one-time functionality.
- It improves the load time. [Only one request for component]
- ***Tightly coupled***
- ***Not easy to extend functionality regularly.***
- ***Hard to test.***

Ex:

- **Create a new folder by name "Components" in "app" folder.**
- **Add a new file into components folder**
  home.component.ts

```
import { Component } from '@angular/core';
@Component({
   selector: 'app-home',
   template: `
   <h2>{{title}}</h2>
   <p>This is our first component.</p>
   `

})
export class HomeComponent {
   title = 'Home Component';
}
```

- **Go to "app.module.ts"**
- **Set your component in bootstrap**
  bootstrap: [***HomeComponent***]

- **Go to index.html and define the component selector in <body>**
  <body>
  <app-home> </app-home>
  </body>
- Start your project

Ex: Component with Styles

```
import { Component } from '@angular/core';
@Component({
   selector: 'app-home',
   template: `
   <div>
   <h2>{{title}}</h2>
   <p>This is our first component.</p>
   </div>
   `,
   styles: ['h2{color:red; text-align:center}', 'p{color:blue}',
'div{border:2px solid darkcyan}']
})
export class HomeComponent {
   title = 'Home Component';
}
```

Component Meta Data	Description
selector	- It is a property that defines the name for component in order to access from any location. - It must use “kebab-case” - It should prefix with “**app**”. - It must have a “dash” as separator. Ex: selector: app-home selector: app-login
template	- It defines the markup to render for component. - Enclose template markup in “backtick”. - It can use data binding expression “{{ }}” to bind dynamic value. Ex: template: `<markup> {{expression}} </markup>`
styles[]	- It is a collection of style attributes defined in object format. - It is a string collection. Ex: styles:[‘selector{attribute:value}’, ‘….’]

Explicitly adding a document using Code Behind Technique:

- In this technique the code, presentation and styles are maintained in separate files.
- It is loosely couples.

- It is easy to extend the code.
- It is easy for unit test.
- It is recommended to use code behind technique for a component if you have to regularly extend your component.
- *In code behind technique we use multiple files, in any application that have multiple files to compile and process will take more time for loading and rendering.*
- Code behind technique requires 4 files
    - .html - for presentation
    - .css - for styles
    - .ts - for logic
    - .spec.ts - for testing
- Code behind technique requires the following meta data for component

Meta Data	Description
selector	- It is a property that defines the name for component in order to access from any location. **Syntax:** **selector: 'app-login'**
templateUrl	- It specifies the relative path in string format. - This path refers to HTML file that contains the presentation to render when component is requested. **Syntax:** **templateUrl:** **'login.component.html'**

styleUrls[]	- It specifies the collection of relative paths in string format. - This refers to the CSS files that contain the styles. - You can refer to multiple style sheets. **Syntax:** **styleUrls: [‘login.component.css’, ‘….’]**

Ex:

- Go to “app/components” folder
- Add a new folder “login”.
- Add following files into “login” folder
    - login.component.ts
    - login.component.html
    - login.component.css
- **login.component.ts**

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

@Component({
  selector: 'app-login',
  templateUrl: 'login.component.html',
  styleUrls: ['login.component.css']
})
export class LoginComponent {
  title = 'User Login';
}
```

- **login.component.html**

```
<div class="container-fluid">
```

```
<form>
  <h3>{{title}}</h3>
  <div class="form-group">
    <label>User Name</label>
    <div>
      <input type="text" class="form-control">
    </div>
  </div>
  <div class="form-group">
    <label>Password</label>
    <div>
      <input type="password" class="form-control">
    </div>
  </div>
  <div class="form-group">
    <button class="btn btn-primary btn-
block">Login</button>
  </div>
</form>
</div>
```

- **login.component.css**

```
.container-fluid {
  width: 300px;
  padding: 20px;
  margin: auto;
  margin-top: 50px;
  justify-content: center;
  align-items: center;
  border:2px solid darkcyan;
```

```
    border-radius: 20px;
    box-shadow: 2px 3px 4px darkcyan;
  }
```

- **Go to “app.module.ts”**
- **Import and register the component.**
- **Set login component in bootstrap**

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

  import { AppComponent } from './app.component';
  import { HomeComponent } from
  './Components/home.component';
  import { LoginComponent } from
  './Components/login/login.component';

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

- **Go to “index.html”**

```
<body class="container-fluid">
  <app-login></app-login>
</body>
```

Angular CLI Commands to Add a new Component

- Angular CLI provide a set of command and flags, which can create and configure your components implicitly.
- After adding component, you have to just customize according to requirements.
- Commands will implicitly generate everything for you. [Scaffolding - **Ruby**]
- To generate component and configure you have to run the following commands from terminal. [Your terminal must change to **app folder –** you can't generate components from workspace folder].
    - **> ng generate component componentName** --flagName
    - > ng g c componentName --flagName
- You can use several attributes to manage the component generation:

ng generate component --flag	Purpose of Flag
--dryRun=true|false	- It will not generate a component. - It is just a trail run from developer to know about the changes made by this

	command. - It contains information about what it is going to generate, where it is going to register etc. **Syntax:** ng generate component register --dryRun=true
--flat=true | false	- Generally, a folder is created for every component. - All files a generated into folder. - You can use “flat” flag to add all files without a folder. - It will directly generate all files into “components” folder or “app” folder. **Syntax:** ng generate component register --flat --dryRun=true
--inlineStyle=true | false	- It will not generate a separate “.css” file for component. - The styles are maintained in “.ts” **Syntax:** ng g c register --inlineStyle=true --dryRun=true
--inlineTemplate=true	It will not generate a separate

| false	".html" file. **Syntax:** ng g c register --inlineTemplate=true --inlineStyle=true --dryRun=true
--skipTests=true | false	It will not generate a separate "spec.ts" file. [Test] **Syntax:** ng g c register --skipTests --dryRun

Ex:

> ng g c register --skipTests

Angular Data Binding