Angular.md

Angular-cli:

Installation:

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

Create Angular starter project:

```
ng new <appname>
```

Generate new component:

```
ng generate component <componentname>
```

or

```
ng g c <componentname>
```

Files:

- **e2e**: code for end to end testing
- **node_modules**: dependencies from package.json are installed here
- **src**: main source code
  - **app**: components and modules in application
  - **index.html**: single page served by browser
  - **main.ts**: defines main module in application
  - **styles.css**: application-wide styles
  - **polyfills.ts**: browser polyfills and application imports
- **angular.json**: project config
- **package.json**:
  - **dependencies**: third party libraries that project needs to run correctly
  - **devDependencies**: required only for development

Bootstrapping the application:

1. **main.ts** calls **AppModule(app.module.ts)**
2. **app.module.ts** contains **@NgModule decorator** which contains **bootstrap array** which defines the components that Angular must know before initializing index.html
3. **bootstrap array** contains **AppComponent(app.component.ts)**
4. **app.component.ts** contains **@Component decorator** which contains **selector** which defines the tag where **template/templateUrl** will be injected

Components:

Each component has it's own HTML, styling and business logic. It helps to split a complex application into reusable parts.

Eg: Header area, Main area, Sidebar, etc can be components.

Each component must have it's own folder and component name must be equal to folder name.

Naming component files: .component.ts Eg: server.component.ts

Naming component classes: Component Eg: ServerComponent

Sample component:

```
@Component({
  selector: 'app-server',
  templateUrl: './server.component.html'
})

export class ServerComponent{

}
```

Here, **templateUrl/template is compulsory.**

Selector can also be an **attribute** on an HTML element.

Eg: `selector: '[app-servers]'`

```
<div app-servers></div>
```

Selector can also be a **class** of an HTML element.

Eg: `selector: '.app-servers'`

```
<div class="app-servers"></div>
```

Selecting by id is not supported by Angular. Pseudo selectors like hover also are not supported.

Decorators:

Decorators are used to enhance elements in our code. Eg: `@Component` is a decorator.

Modules:

Angular uses components to build webpages and modules to bundle different pieces into packages. Modules have `@NgModule` decorator which contain following properties:

- **declarations**: defines all components within the Angular application
- **imports**: imports other built-in/user-defined modules
- **providers**: ? //TODO
- **bootstrap**: which component will be used to bootstrap the application

Data binding:

Data Binding is communication between Typescript code (Business Logic) and Template (HTML).

String Interpolation:

In TS:

```
export class ServerComponent{
  serverId: number = 10;
  serverStatus: string = 'offline';

  getServerStatus(){
    this.serverStatus = 'online';
    return this.serverStatus;
  }
}
```

In HTML:

```
<p>The Server with ID {{ serverId }} is {{ getServerStatus() }}</p>
```

Here, we have binded string returned types (converted to string if not a string) of `serverId` and `getServerStatus` in TS to HTML

Property Binding:

In TS:

```
export class ServersComponent{
  allowNewServer = false;

  constructor(){
    setTimeout(() => {
      this.allowNewServer = true;
    }, 2000);
  }
}
```

In HTML:

```
<button class="btn btn-primary" [disabled]="!allowNewServer">Add server</button>
```

Here, we have binded value of `allowNewServer` property in TS to HTML attribute `disabled`.

If we want to display some value, use String Interpolation. If we want to change some property, use Property Binding. Don't mix String Interpolation and Property Binding together.

Event Binding

In TS:

```
export class ServersComponent{
  serverCreationStatus = 'No server was created.';

  onCreateServer(){
    this.serverCreationStatus = 'Server was created';
  }
  onUpdateServerName(event:Event){
    this.serverName = (<HTMLInputElement>event.target).value;
  }
}
```

Here, `<HTMLInputElement>` is used for typecasting.

In HTML:

```
<input type="text"
class="form-control"
(input)="onUpdateServerName($event)">

<p>{{ serverName }}</p>

<button class="btn btn-primary"
(click)="onCreateServer()">Add server</button>

<p>{{ serverCreationStatus }}</p>
```

Here, we have binded `onCreateServer` function to click event and passed value entered in input field to `onUpdateServerName` and display `serverName` on page. `$event` passes event related data to function (including value entered in input).