1. **Setup**

<https://angular.io/guide/setup-local>

* JavaScript
* HTML
* CSS

|  |  |
| --- | --- |
| Node js | node -v |
| npm package manager | npm -v |
| Angular installing  To allow the execution of PowerShell scripts on Windows client  Check version | npm install -g @angular/cli  Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned  ng -v |
| Create new app – Build new app  Launch server  http://localhost:4200/ | ng new my-app  cd my-app  ng serve --open |

<https://www.keycdn.com/blog/sass-vs-less>

Nesting styling from SASS & LESS will be converted back to CSS format.   
SASS is the most popular.

1. **Architecture**

<https://angular.io/guide/architecture>

**The architecture** of an Angular application relies on certain fundamental concepts. The basic building blocks of the Angular framework are *Angular components* that are organized into *NgModules*. NgModules collect related code into functional sets; an Angular application is defined by a set of NgModules. An application always has at least a *root module* that enables bootstrapping, and typically has many more *feature modules*.

**Components** *define* *views*, which are sets of **screen elements** that Angular can choose among and modify according to your program logic and data

* Components *use services*, which provide **specific functionality** not directly related to views. Service providers can be injected into components as dependencies, making your code modular, reusable, and efficient.

Modules, components and services are classes that use *decorators*. These decorators mark their type and provide metadata that tells Angular how to use them.

* The metadata for a component class associates it with a template that defines a view. A template combines HTML with Angular directives and binding markup that allow Angular to *modify the HTML before rendering it for display.*
* The metadata for a service class provides the information Angular needs to make it available to components through *dependency injection (DI)*

**Router** service to help you *define navigation paths among views*. The router provides sophisticated in-browser navigational capabilities.

**Root module**, conventionally named *AppModule*, which *provides the bootstrap mechanism* that launches the application. **An application typically contains many functional modules.**

|  |  |
| --- | --- |
| **Data bindings** |  |
| *Event Binding* | The application responds to user input in the target environment by updating your application data  **Event-Handling** |
| *Property Binding* | Interpolate values that are computed from the app data into HTML  **App Data -> HTML Template<>** |

****

1. **Component**

Components are the *main building block* for Angular applications. Each component consists of:

* A TypeScript class that defines behavior
* An HTML template
* A CSS selector

**Create a new component**

|  |
| --- |
| *Create new component* by Angular CLI  ng generate component <component-name>   * **List of files** will be generated |
| **Component file** <component-name>.component.ts |
| **Template file** <component-name>.component.html |
| **CSS file** <component-name>.component.css |
| Testing spec file <component-name>.component.spec.ts |

A Component file **top-bar.component.ts**

|  |
| --- |
| import { Component } from '@angular/core';  @Component({    selector: 'app-top-bar',    templateUrl: './top-bar.component.html', //external html file    styleUrls: ['./top-bar.component.css']  })  export class TopBarComponent { //export class  } |
| OR  template: '<h1>Hello World!</h1>', //inline html |
| OR  template: `  <h1>Hello World!</h1>  <p>Multiple lines</p>  `, //inline html |

**Component Lifecycle**

<https://angular.io/guide/glossary#lifecycle-hook>

1. **C**