How the app starts?

* How are the contents displayed in an angular app?So how does it know that the contents of *app.component.html* is rendered? Why isn’t it showing “Loading…” as depicted in the *index.html* file.
* The first file to be **served** by the server is the *index.html.* Then the relevant components are loaded. The root component is the *src/app* component. This component is loaded via the <app-root/> selector in the *index.html* file.
* The *app.component.ts* file defines a component using the @Component decorator (more on this later). Here the selector of the component is defined along with the template HTML and styling.
* Still doesn’t answer how Angular gets kick started. We can find this out by running the app and opening the Developer tools and inspecting the source. We find that there are a couple of <script/> tags added at the end of the index file. These are injected by the CLI directly while building as bundles. This helps by avoiding any manual imports from the index file.
* The first piece of code that gets **executed** is the *main.ts* file when we run the application. Here we define which build config to use (PROD or DEV) and also which module to bootstrap while starting the application (*app.module.ts* in our case).
* If we look into the *app.module.ts* file, we find that while defining the module, we also define which components should be available while bootstrapping (initializing) the module. In our case it is *AppComponent*.
* As explained earlier, through the component file, we know what template and styles are to be rendered when loading this module. That’s how the page contents are displayed.

What is a Component?

* A Component is a basic building block for all modular applications. It is more of a design pattern than a feature that is local to Angular. Other known JS frameworks like React and Vue also follow this approach.
* The *AppComponent* is the root component in an Angular app. Other components are added on top of this component to build a fully functional app.
* An application can be divided into multiple components coming together. The need of creating smaller, additional components as a part of the main application, boils down to the complexity of the existing components (more on this later).
* Each Component has its own HTML template, styling, business logic, etc. This is allows you to create reusable parts, making the maintenance of code easy and also more decoupled.
* A good practice is to have a new folder for each component. A component is essentially a class with some properties and functions relevant to the component.
* The @Component decorator is a part of Angular and it adds special meaning and additional metadata to the class. It has to be imported from @angular/core. Two important parts of it are the ‘selector’ and ‘template URL’/’template’. Once these are available, we need configure this component in the app module in order to use it.
* The ‘styleUrls’ can be replaced with ‘styles’ property, although it still remains array. We can use back-ticks (``) to enclose multi-line code.
* The ‘selector’ property basically acts like how a CSS selector works. Based on how this is written, we have to modify the HTML code as well. For example, if we have a selector as ‘[app-servers]’, then to render this, we have to write the HTML as <div app-server></div>.

What is a Module?

* Angular uses components to build parts of the application and uses modules to bundle these components into packages.
* This also, like a component, is essentially a class with its own properties and functions. In most cases, we would have only one module, unless we have a huge project.
* A module uses a different decorator named @NgModule which is also imported from @angular/core.
* There are main properties to be added to this decorator - ‘declarations’, ‘imports’, ‘providers’ and ‘bootstrap’. We already know what ‘bootstrap’ is used for.
* By default, Angular doesn’t know about any other components without registering it with the app module. This can be done using the ‘declarations’ property. Just import the component (the *.ts* file) to the module and adding the component to the list of declarations

Note:

We can add a new component via the CLI also. The command can be either ng generate component <COMPONENT\_NAME> or ng g c <COMPONENT\_NAME>. This will automatically generate the component folder with all the necessary files and also adds the component to the app module. To prevent the test files to be created, add the flag --skipTest true.

Data Binding

* Data binding is basically the communication between your TypeScript code and your template file. This could in response to some server response or some calculation.
* We can bind data in a few ways -
  + String Interpolation - {{ data }}
  + Property Binding - [property]=”data”
  + Event Binding - (event)=”expression”
  + Two-way Binding - [(ngModel)]=”data”
* Properties of a component can be available in the class of the component.
* TypeScript requires each property to be assigned a type (Number, String, etc).
* *String Interpolation* is used when we want to bind a string value to the template. We do this using {{ <PROPERTY\_NAME> }}. The binding property could either be a string, a property that contains a string or an expression that returns a string.
* *Property binding*, as the name sounds, binds to an attribute of a tag to a resultant property. For example, if we want to set the disabled attribute of a button dynamically, we can do so by using property binding. We need to add the required property to the component class and bind the property to the template in this fashion: [disabled]=”<PROPERTY\_NAME>”.
* Event binding is used to handle events in the browser. We bind it using (event)=”<HANDLER\_FUNCTION>()”. For example (click)=”onCreateServer()”. $event is an important keyword that is reserved by Angular and is used to get information about the event.
* Two-way binding helps if we want the property to change live and also react to the change simultaneously. For this, we need to import the *FormsModule* from @angular/forms and add it as an import in the app module. We then use the ngModel directive to bind the property to the tag in this way - [(ngModel)]=”<PROPERTY\_NAME>”.

Directives

* Directives are basically instructions in the DOM. Components are actually directives, but with a template. We instruct the DOM to add a component at a location where we used the selector.
* We can also have directives without a template.
* We use @Directive decorator to to create a custom directive. There are a few built-in directives that we can use.
* There are structural directives (Changes the structure of the DOM) and attribute directives (Just adds some extra styling, no DOM change). Attribute directives are always used as Property binding (Check the *server.component.html* file). Following are some of the build-in directives:

1. *\*ngIf* - This acts as a *if* statement which can be used at the template level.
2. *\*ngFor* - This a very important directive in Angular and is very commonly used as well. This allows us to loop through a array and add DOM tags for each entry in the array.
3. *ng-template* - This directive acts as a marker in the template to show conditional template. This usually has a ‘local-reference’ (more on this later). We can apply *if* condition or *for* loops on these templates to render based on condition.
4. *ngStyle* & *ngClass* - These add style to the tag based on the property bound. For *ngStyle* ,we can enter CSS properties in came-case or the regular way (have to enter as a string in this case).