Services:

* Based on what we have learned so far, We know that we can pass data between components using the @Input and @Output decorators with the help of the EventEmitter generic type.
* In large applications, following this strategy will lead to cumbersome code and difficult maintenance. There will cases where the many components can access and manipulate the data. Using this strategy, we would write the same code for every components repeatedly.
* We need a central repository for managing cross-component changes and for that, we can use Service Injection. It is just a normal class with its own properties and functions. We create a <SERVICE>.service.ts file directly (or just run ng g s <SERVICE\_NAME>) and add the functionality we would use in a lot of places.
* To make services available to each component, we need to a) add a providers array property to the @Component decorator and insert the Service class into it , and b) add a private parameter to the constructor, which in turn gets the instance of the Service class. Do not import the service class into the component directly, although it works.
* Even after doing the above, we observe that the data change is not reflected in the components. This is because Angular follows a *hierarchical injection* principle where the service instance at a particular point is also available for all its children, children’s children and so on.
* The highest point of injection can be the App module and the lowest can be a component without any children. In our app, each component is getting its own instance of the service, which is why we can’t see the data change.
* To let the instance cascade to the children, we just have to remove the service class from the providers array in the children. We still need to import of the service class and also provide an argument in the constructor. If this has to be done from the module level, we just have to import the class into the module file and add the service to the providers array in the @NgModule decorator.
* Services can be injected into other services too. This is only possible from the module level. To do this, we first need to use the @Injectable decorator imported from @angular/core.
* Lets say we have two services - Service A and Service B, and we need to inject Service B into Service A. So we use the @Injectable decorator in Service A and add Service B as a constructor argument in ServA.
* In Angular 6+, the @Injectable is added by default for all Services. This functionality still remains the same though. We can also avoid adding this into the module providers array by writing @Injectable({providedIn: ‘root’}).

Cross-Component communication:

* We can make components communicate without using @Output in each component. This, in fact, quickly becomes a nightmare in larger applications.
* Again we can define an EventEmitter in a Service and emit it from a component. The component that has to use this emitted data can use subscribe() function provided by the EventEmitter.
* This function provides a call back function, with the data as a parameter, as an argument.