Angular Development Best Practices

* It's recommended to use Angular CLI tool while working with the Angular project because it increases the productivity.
* While creating our **files**, we should pay attention to the file names. Names should be consistent with the same pattern in which we mention the file’s feature first and then the type, dot separated.

For example home.component.ts or home.component.html or auth.service.ts …

* If we want to add more descriptive names to our files we should use a **dash(-)** to separate the words in the name: menu-admin-user.component.ts…
* When we add **names to our classes**, we should use upper camel case style with the added suffix that represents the type of our file

Eg. export class DatepickerDirective

* Our files should reside in the **folders** named by the feature they represent. This is very important because we can easily identify which business logic we have implemented in those files inside our descriptive folders.
* It is very important not to create more than one component, service, directive… inside a single file. Every file should be responsible for a **single functionality**. By doing this, we are keeping our files clean, readable and maintainable.
* Using interfaces is a perfect way of describing our object literals. If our object is of an interface type, it is obligated to implement all of the interface’s properties. We shouldn’t name our interfaces with the **starting capital I letter** as we do in some programming languages.
* We can specify optional properties, by using the question mark (**?**) inside an interface as well. We don’t need to populate those properties inside an object.
* We should use the constructor method to setup Dependency Injection for our services and that is pretty much it. We shouldn’t be doing any work inside it, especially fetching the data from the server. For this type of actions, we have the lifecycle hooks in Angular.
* To be on the safe side we should always use the safe navigation operator while accessing a property from an object in a component’s template. If the object is null and we try to access a property, we are going to get an exception. But if we use the save navigation **(?)** operator**,**the template will ignore the null value and will access the property once the object is not the null anymore.
* Even though an Angular application is going to work just fine if we create just one module, the recommendation is to split our application into multi-modules. There are a lot of advantages to this approach. The project structure is better organized, it is more maintainable, readable and reusable and we are able to use the lazy-loading feature.
* The best practice for Angular application is to use a separate routing module for the router and then to register this new routing module in our app module.
* If we have a multi-modular application, implementing a lazy loading feature is recommended. The great advantage of a lazy loading approach is that we can load our resources on demand and not all at once. This helps us in decreasing the startup time. Modules that we are loading in a lazy manner will be loaded as soon as a user navigates to their routes.
* While creating our components, it is very important to separate the component, template and styling logic into the separate files. Even though we have the possibility to keep all of it inside the component file this is not a good idea.

Instead, we should separate styles and HTML in their own files and import them.

* Creating reusable components is one of the best techniques we can use while developing our project. We can reuse those types of components inside any parent component and pass the data through the @Input decorator. Those components can emit events by using the @Output decorator and EventEmmiter class.
* To keep our components readable and easy maintainable we should write a clean code inside them. We should limit the logic in the component to satisfy the template needs and nothing more. No additional complicated business logic is required. If we need additional logic, we should extract it into a service. That way our component remains clean and the extra code could be reused in other components. Furthermore, the logic inside a service can be easily tested and it hides an implementation of the component, thus making a component more readable.
* Whenever we have a situation where multiple HTML elements have the same behavior (for example: when we hover over the element it receives the blue color), we should consider using attribute directives. We shouldn’t repeat the hover logic every time we need it on some HTML element. A much better way would be to create directive and then just reuse it on the particular element.
* Services provide a great way to connect two unrelated components. To configure services for this, the best way is to register them as a singleton. Furthermore, services are a great way to extract the code from our components, thus making components more readable and maintainable. We may use a service to extract the code related to the component in which that service is provided or to share reusable code with that service between different components.
* Even though we can work with the HTTP requests inside our components, the recommended way is to do that by using a service.
* If we develop an Angular application which will be deployed in the production environment, the environment variables inside the Angular project can be very helpful. We can set up those variables to distinguish the end-points between development and production environment.
* Angular is more powerful and better than jQuery because it offers everything that you need to develop your client’s application and it’s important to choose when your intention is SPA’s. If you are planning to get the best quality website that can attract your client with this single page application then you can opt Angular.