Angular ECF Review

# Overview

This document is just a quick analysis of the existing Angular EFC.Next code base; some improvements and additions of new technologies; and considerations for best practices, where necessary. The code base is fine and I did not look at all source code to make line-by-line excruciating recommendations – I believe that is what code review or merge requests are for; and, not this exercise. That being said, there may be some anomalies that are known by our developers that are not pointed out here, but, I believe as a team we can correct them with processes that we put in place – such as code reviews or merge requests – to help us be better. This is just a few observations where I believe we could take into consideration. Again, the team can review these items and can take actions or not.

I divided this review into two parts:

* Recommendations for existing code base; and
* Suggestions for the ECF Angular application going forward to keep in line with new and arising technologies and best practices in the industry.

The later just include things that are not existing in the current code base. We can adjust and improve as we continue to code and find problems or opportunities to improve ECF.Next.

# Legend

### **Anomalies**

* **New Implementation or refactoring suggestions**

# Recommendations for Existing Code-base

### **Shared Module**

* Imports array is empty. Should include the following modules:
  + CommonModule
  + ReactiveFormsModule
  + FormsModule
  + PrimeNg Modules
* **For the above recommendation, I know we have only one component, but there could be more.**
* **Add Phone number pipe folder to shared module**

### **Services**

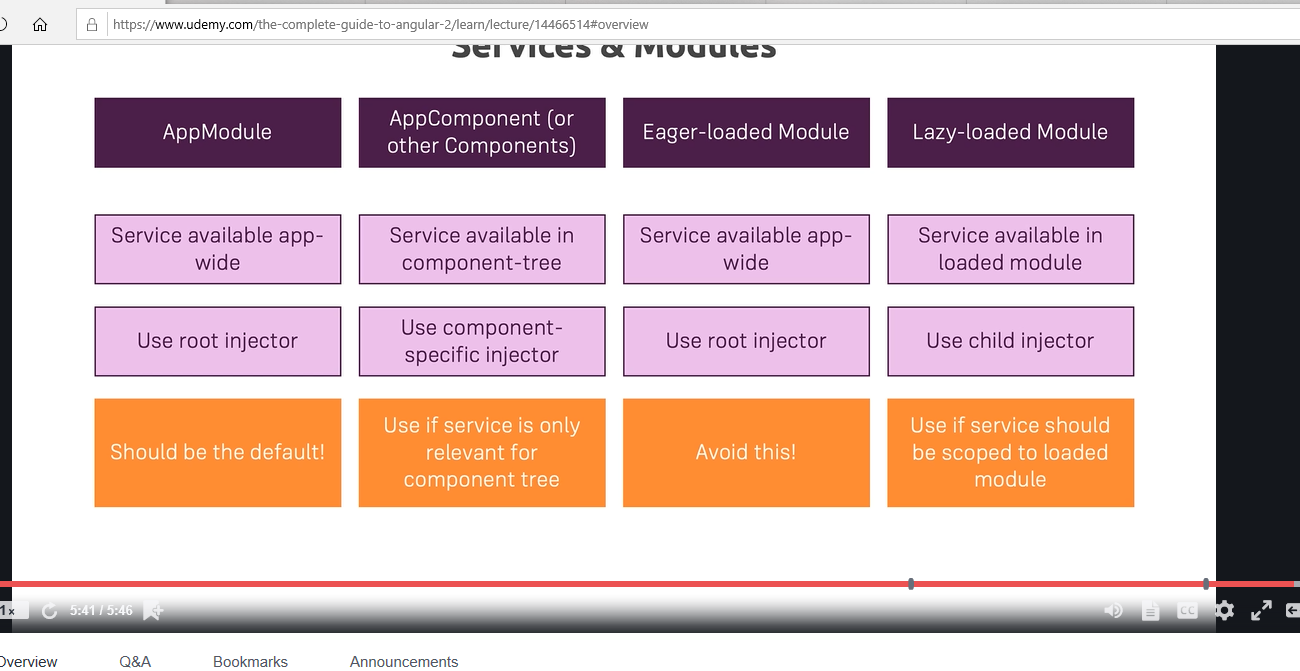
* There should be an abstract base service that handles the following:
  + All BuildHeader logic – build header function is repeated in all services. This should be in the abstract base service – follows the DRY best practice. This function should only handle content-type and access control allow headers type; as well as, all other headers that are repeated across services. Other header that are applicable to a single service should be handled accordingly in the services, itself.
  + The httpClient provider should be in the abstract base service, as well. It should use generics to return concrete types in the sub class or derived class. There should be these methods that would be called or leveraged by all implementing or sub classes:
    - getAll, if necessary
    - get
    - post
    - delete
    - patch, if necessary
    - put
  + Again, I did not look at all the services; but, Boaz could make some recommendations here because he is more up-to-date with the current code base.
  + **Add a Core Module for application wide providers or services and import in App module.**
    - **Interceptors and services should be added to the core module**

### **Container Components – Low Priority**

* All form builder properties should be declared in the current shared folder. The benefit of this is that we can reuse them, where applicable. An example would be, we can create a src/shared/form-controls/user.form-control.ts for use by login and SMI login forms. In this file, you would have the User id and password for the form builder service configured. And in the login and SMI login forms, you would pass this one user form control instance to both. This will enable us to reuse validators and form control instances, where necessary.
* NOTE: we would have to pass a deep copy of the user form control instance to both SMI and login components, because JSON objects are used as reference types by default, which may affect the application at runtime.

### **Session Storage**

* Remove user object in session storage. Password is stored along with user name in session storage. If one of our users mistakenly open the browser storage, they would see user name and password and would like to try it on other computers.
  + We could implement ngRx state management for user information, OR:
  + For ECF User JSON, we should create a global data service (a service that will be added to the app module provider array) that will be used by the whole application. It should contain something like these:
    - User info: EcfUser class
    - IsUserLoggedIn property
    - Other methods/properties, where applicable
* **Please see below diagram for services usage in Angular**



# Other suggestions for the ECF.Next application

* Upgrade PrimeNg packages from beta version to production version, where applicable.
* Implement unit testing
* Implement tools for code coverages
  + Setting percentages for code coverage and disallowing commits or pushing code whenever code coverage matrix is not met. This is a long term goal.
* Implement tools for executing unit tests before committing and pushing changes to our ECF Angular GIT repository.
* Unit testing – add unit tests for code
* Making ECF.Next Progressive Web Application (PWA) ready
  + We can add service worker and Angular Universal to our application to make ECF.Next PWA-ready. Although, this will take us to offline support; however, I must note that this will not support Android or OS support. Meaning, it will not be installed from an app store or play store. This is just in case, we want our users to have mobile support. Boaz stated that ECF.Next is responsive. And this is one quality of PWA. The next step would be offline support and startup performance.
  + For Angular Universal support, we can add the necessary code too – only, if the team feels this is necessary. Angular Universal is a way of having our application be compiled on the server and code sent to the browser will be faster. It improves startup of the application. If this is not part of our goal, then no need for this; but, this is part of (and essential) for PWA support.
    - Think of it as server-side processing. Everything will be called on the server and then sent to the browser. This is for faster load performance.
    - You can google this and read up. Like I said, these are qualities for PWA support.
    - The only downside (and this may not be a downside) is that things like window and document DOM JSON access will have to be checked before using. And, Boaz correct me if I’m wrong, I don’t see these DOM elements (window and document, confirm, alert, etc.) being used.
* Introduce ngRx state management
* **Create custom validators in shared folder. This is for future. I don’t know if custom validators have been introduced yet.**
* All services should implement an interface that specifies each service contract. This will be helpful with unit testing and dependency injection into components and other Angular artifacts, such as, pipes, services, directives, etc…
* Logging support – use splunk (Hennepin existing logging engine) and TrackJs (I’ve used this at other clients). We can deep dive into this when we identify some non-functional requirements that could be used beyond browser console logging.
* Add Memo decorator to Phone number pipe (see url for explanation of Memo decorator: <https://www.c-sharpcorner.com/blogs/memo-decorator-in-angular-pipes>).