Frontend Handbook | Angular Standards | Best Practices

By:

Ahmed Hussein

Senior Frontend Developer

Saudi Telecom Company

Angular coding standards

Impacted projects

Table of Contents

DOCUMENT HISTORY 3

CREATION LOG 3

REVIEW LOG 3

SOURCE CONTROL 4

RELEASE MANAGEMENT 4

PROJECTS STRUCTURE 4

ARCHITECTURE 4

CODING STANDARD 4

DOCUMENTATION 5

TOOLS 5

GENERAL CODING BEST PRACTICE 5

SECURITY 5

TESTING & PERFORMANCE 5

OTHERS 6

Document history

Creation log

Author name	
Author email	
Author title	
Department	
Contact	

Review log

Version	Date	Author	Reviewer	Comments

Source control

- Code branches:-
  - Branches hierarchy:- Production, Development, Release, and feature branches.
  - The production branch has production versions fully approved.
  - The development branch has Development-Quality approved versions.
  - The release branch has Development approved versions.
  - The feature branch has a single feature.
- Branches submit roles:-
  - Each feature should be separated in a single branch (feature branch).
  - After finishing the development of the feature, the developer should open a new pull request.
  - Other developers should review the structure, rules, naming convention, and logic.
  - Every pull request must have at least two approvals in order to be merged.
- Merging roles:-
  - Step 1: Merging the Feature branches to the Release branch after the reviewing approvals.
  - Step 2: Merging the Release branch to the Development branch after Quality team approval.
  - Step 3: Merging the Development branch to the Production branch after development regression.
- Peer review roles:-
  - Classes, Functions, and Variables naming conventions.
  - Performance Issues.
  - Modules Structures.
- Committing standards:-
  - [Feature Name]: Story Name.
  - [Fix]: a brief description.
  - Squashing commits in order to merge only one commit at the end.

Release management

- Pipelines and CICD
- Test coverage
- dev mode vs build mode
  - During development the application will be on dev mode and calling the backend development URL by const it on environment.ts file.
  - while build mode will handle Jenkins files by the backend team.

Projects structure

- External and Internal projects structure
- Modules/Components Structuring
  - This part is managed separately with a document display all project structure and where to find Modules, component, services, pipes, directives and any angular files
  - aslo display where our application starts and how to create new classes or services in order to follow with all team design pattern, and to be unified on all stc new front end projects

Architecture

- SPA or non-SPA
  - By default will use SPA application.
- Micro frontend

Coding standard

- Angular version
  - Will use Latest version 10
- CSS management/UI library
  - The UI team will deliver the HTML and CSS files
- Naming conventions
  - Variables
    - camelCase for let, and anonymous var
    - Uppercase for const divided by underscore
  - Functions
    - The function name is camelCase
    - body functions max 7 lines.
  - Classes, Components
    - Using the angular command line for creation. camelCase naming
    - The component folder will be lowercase divided by dash -

- Using third party components.
  Must be compatible with angular and has good documentation and support
- Business logic handling
  Component for view and services for business.

- Session handling
  - Cookies vs local storage vs session IDs
- Error management
  - Error pages (404 page)
  - Non-authorized access (design or required response for non authorized access)
  - Business exception : ex:- mon-showstopper business exception handling (with Backend)
  - JWT validation
    - handle non authenticated (expired) token with Backend.
- Observables management
  User Must be unsubscribed from all subscriptions to prevent memory leaks

  by initialization array of subscription holding all subscriptions on component or class and unsubscribing from it by iteration on ngOnDestory method.

- management
  - Lazy loading
    Based on business needs, so that we will use lazy loading to enhance performance.

  - Using promise, observables.

Based on business sometimes will need promise and the most times will use observables but developers should unsubscribe from all subscriptions on every component.

- Use of synchronous API calls
- Based on business

- Services
  - Singleton
    - Services will be singleton which only one instance exists in the app.

- Localization
  We will use Angular translation module for translation of main titles and any static titles using i18n JSON files every language with one file and the data from backend that translate the content of the application.

- Managing static contents
  1. Using Shared components and shared services with the help of observables.

Documentation

- Design
  - UX Designs will be delivered by UX/UI team lead by adobe XD url for the prototype
  - HTML and CSS files also will be delivered by UX/UI Team Lead.

Tools

- IDE (VScode preferred as it lightweight and support plugins)

General coding best practice

- Separation of concerns
  How to separate components to multi-components to reduce code and make small components shared between all application modules and components.

- Atomicity/single responsibility
  Every component will do one business logic.

- API interceptors
- Guards
  Under the core module

- Resolvers
  Will be with every module (service)

Security

- JWT token
  Managed with local storage

- Authorization
  Under the core module

Testing & performance (Using angular unit testing)

- Unit tests
- Automated testing
- Performance monitoring (Augury Extension)
- Sonarqube

Others

- Feature modules
  Based on business
- Helpers files
  Will be inside the shared module
- aliasing
  Link alias module
- Manage Http requests
  One class manage all basic Http methods
- Agreed structure for the API response
  Mange folder models for handling Http response
- Acceptance criteria
  The business will put it and clarify the feature requirement.