*angular folder structure Bash-Style Diagram (Flowchart Representation)

Thursday, August 29, 2024 9:57 PM

First level : core, shared, feature

```
project-root/
├── src/
│   ├── app/
│   │   ├── core/   # focused on global concerns (e.g singleton services, app-wide utilities) & structural foundation.
│   │   │   ├── services/
│   │   │   │   ├── logger.service.ts  # Logging service
│   │   │   │   ├── http.service.ts    # HTTP service
│   │   │   │   └── auth.service.ts    # Authentication service
│   │   │   │
│   │   │   ├── guards/
│   │   │   │   ├── auth.guard.ts      # Auth guard
│   │   │   │   ├── guest.guard.ts
│   │   │   │   └── role.guard.ts      # Role-based access guard
│   │   │   │
│   │   │   ├── interceptors/
│   │   │   │   ├── token.interceptor.ts # Token interceptor
│   │   │   │   └── error.interceptor.ts # Error response interceptor
│   │   │   │
│   │   │   ├── models/
│   │   │   │   ├── user.model.ts      # User model
│   │   │   │   └── token.model.ts     # Token model
│   │   │   │
│   │   │   ├── enums
│   │   │   │   └── core-api-enums
│   │   │   │   └── core-enums
│   │   │   │
│   │   │   ├── layout/    Layouts are not tied to any specific feature like Finance or HRMS. So keep in core.
│   │   │   │   ├── components/
│   │   │   │   │   ├── main-layout/
│   │   │   │   │   │   ├── main-layout.component.ts
│   │   │   │   │   │   ├── main-layout.component.html
│   │   │   │   │   │   └── main-layout.component.scss
│   │   │   │   │   ├── admin-layout/
│   │   │   │   │   │   ├── admin-layout.component.ts
│   │   │   │   │   │   ├── admin-layout.component.html
│   │   │   │   │   │   └── admin-layout.component.scss
│   │   │   │   │   ├── auth-layout/
│   │   │   │   │   │   ├── auth-layout.component.ts
│   │   │   │   │   │   ├── auth-layout.component.html
│   │   │   │   │   │   └── auth-layout.component.scss
│   │   │   │   │   │
│   │   │   │   │   ├── header/
│   │   │   │   │   │   ├── header.component.ts
│   │   │   │   │   │   ├── header.component.html
│   │   │   │   │   │   └── header.component.scss
│   │   │   │   │   ├── footer/
│   │   │   │   │   │   ├── footer.component.ts
│   │   │   │   │   │   ├── footer.component.html
│   │   │   │   │   │   └── footer.component.scss
│   │   │   │   │   ├── sidebar/
│   │   │   │   │   │   ├── sidebar.component.ts
│   │   │   │   │   │   ├── sidebar.component.html
│   │   │   │   │   │   └── sidebar.component.scss
│   │   │   │   ├── layout.service.ts/
│   │   │   │   └── layout.module.ts    #import in app.module as it is needed app wise.
│   │   │   │
│   │   │   │
│   │   │   └── core.module.ts         # Core module definition
│   │   │
│   │   ├── shared/                    # Shared module: Shared components, directives, pipes
│   │   │   │
│   │   │   ├── directives/
```

```
|    |    |    |    └── example.directive.ts # Example directive
|    |    |    ├── pipes/
|    |    |    |    └── example.pipe.ts      # Example pipe
|    |    |    ├── services
|    |    |    |    └── share.service.ts
|    |    |    |    └── toast.service.ts
|    |    |    |    └── http-verbs.service.ts   # all the http methods.
|    |    |    |
|    |    |    ├── enums
|    |    |    |    └── share-api-enums
|    |    |    |    └── share-enums
|    |    |    |
|    |    |    ├── utils  #Donot create util functions for feature modules. Keep that in feature logic service if any.
|    |    |    |    ├── date.util.ts       # Date utilities
|    |    |    |    ├── validation.util.ts      # Validation utilities
|    |    |    |    └── common.util.ts
|    |    |    |
|    |    |    ├── data
|    |    |    |    └── share-data.constants.ts
|    |    |    |
|    |    |    ├── models/
|    |    |    |    ├── image.model.ts      #
|    |    |    |
|    |    |    |
|    |    |    └── shared.module.ts         # Shared module definition
|    |    |
|    |    ├── features/                  # Feature modules
|    |    |    ├── auth/    The Auth a feature and should typically be placed inside the features folder.
|    |    |    |    ├── components
|    |    |    |    |    ├── login/
|    |    |    |    |    |    ├── login.component.ts
|    |    |    |    |    |    ├── login.component.html
|    |    |    |    |    |    └── login.component.scss
|    |    |    |    |    |
|    |    |    |    |    |─── sign-up
|    |    |    |    |    |    ├── sign-up.component.ts
|    |    |    |    |    |    ├── sign-up.component.html
|    |    |    |    |    |    └── sign-up.component.scss
|    |    |    |    |    |
|    |    |    |    |    └─── forgot-password
|    |    |    |    |         ├── forgot-password.component.ts
|    |    |    |    |         ├── forgot-password.component.html
|    |    |    |    |         └── forgot-password.component.scss
|    |    |    |    |
|    |    |    |    |    └── auth.service.ts   # Authentication service logic
|    |    |    |    |    ├── auth.module.ts
|    |    |    |    |
|    |    |    ├── dashboard              # Dashboard feature module, if one dashboard then no need for module. But keep in feature
|    |    |    |    ├── user/
|    |    |    |    |    ├── user-dashboard/
|    |    |    |    |    |    ├── user-dashboard.component.ts
|    |    |    |    |    |    ├── user-dashboard.component.html
|    |    |    |    |    |    └── user-dashboard.component.scss
|    |    |    |    |    └── user-dashboard-module    #lazily load user dashboard when user logins.
|    |    |    |    |
|    |    |    |    ├── admin/
|    |    |    |    |    ├── admin-dashboard/
|    |    |    |    |    |    ├── admin-dashboard.component.ts
|    |    |    |    |    |    ├── admin-dashboard.component.html
|    |    |    |    |    |    └── admin-dashboard.component.scss
|    |    |    |    |    └── admin-dashboard-module    #lazily load user dashboard when user logins.
|    |    |    |
|    |    |    ├── static-pages/
|    |    |    |    ├── home/
|    |    |    |    |    ├── home.component.ts
|    |    |    |    |    ├── home.component.html
|    |    |    |    |    ├── home.component.scss
```

```
│   │   │   │   │   └── home.component.spec.ts
│   │   │   │   ├── about/
│   │   │   │   │   ├── about.component.ts
│   │   │   │   │   ├── about.component.html
│   │   │   │   │   ├── about.component.scss
│   │   │   │   │   └── about.component.spec.ts
│   │   │   │   │   pages-routing.module.ts
│   │   │   │   └── pages.module.ts
│   │   │   │
│   │   │   ├── user/                       # User feature module
│   │   │   │   ├── components
│   │   │   │   │   ├── user-profile/
│   │   │   │   │   │   ├── user-profile.component.ts
│   │   │   │   │   │   ├── user-profile.component.html
│   │   │   │   │   │   └── user-profile.component.scss
│   │   │   │   │   │
│   │   │   │   │   │───user-settings
│   │   │   │   │   │   ├── user-settings.component.ts
│   │   │   │   │   │   ├── user-settings.component.html
│   │   │   │   │   │   └── user-settings.component.scss
│   │   │   │   │   │
│   │   │   │   │   └───user-dashboard   # in case the feature have its own dashboard.
│   │   │   │   │       ├── user-settings.component.ts
│   │   │   │   │       ├── user-settings.component.html
│   │   │   │   │       └── user-settings.component.scss
│   │   │   │   │
│   │   │   │   │
│   │   │   │   ├─────── nested-feature/          # Nested feature module inside the User module
│   │   │   │   │        ├── components/
│   │   │   │   │        │   ├── nested-list.component.ts
│   │   │   │   │        │   └── nested-detail.component.ts
│   │   │   │   │        ├── services/
│   │   │   │   │        ├── nested-feature.service.ts
│   │   │   │   │        ├── nested-feature-routing.module.ts
│   │   │   │   │        ├── nested-feature.module.ts
│   │   │   │   │        └── index.ts
│   │   │   │   │
│   │   │   │   │
│   │   │   │   ├─────── services
│   │   │   │   │       ├── user.service OR user-logic.service.ts      # state and stateless methods for user logic
│   │   │   │   │       ├── user-state.service  # State Management Service contain subject  | behaviour subject | EFI
│   │   │   │   │       └── user-data.service  # Service for calling APIs if user api enums is not used.
│   │   │   │   │
│   │   │   │   ├────── models
│   │   │   │   │       └── user-models.ts
│   │   │   │   │
│   │   │   │   │
│   │   │   │   ├────── data      # if one file then donot need to create data folder.
│   │   │   │   │       └── user.constants.ts  export const USER_ROLES = ['Admin', 'User', 'Guest'];
│   │   │   │   │
│   │   │   │   ├────── enums
│   │   │   │   │       └── user-api-enums
│   │   │   │   │       └── user-controller-enums # all the api
│   │   │   │   │
│   │   │   │   ├─────── user-routing.module.ts   # Routing for user feature
│   │   │   │   ├─────── user.module.ts           # User feature module definition
│   │   │   │   └──────  index.ts
│   │   │
│   │   ├── app-routing.module.ts          # Global routing module
│   │   ├── app.component.ts               # Root application component
│   │   ├── app.component.html
│   │   ├── app.component.scss
│   │   ├── app.module.ts                  # Root application module
│   │   └── main.ts                        # Main entry point for Angular application
│   ├── assets/                            # Static assets (images, fonts, etc.)
│   │   ├── images/
│   │   ├── fonts/
│   └── environments/                   # Environment-specific configuration
│       ├── environment.ts              # Development environment settings
│       └── environment.prod.ts         # Production environment settings
└── angular.json                        # Angular CLI configuration
```

Common service

It may be used to share functionality among some feature modules or a smaller part of the application. While Shared Service Available across the entire application

Feature module

Feature modules should contain business-specific components, not foundational layouts.

Core module (for all application)

The core folder is designed for application-wide services (guards, interceptors), utilities, and **App-level Constants** Such as API endpoints or global app settings, that are used throughout the entire application.
Import the CoreModule only in the AppModule to ensure singleton behavior. Avoid importing the CoreModule in feature or shared modules.

- **Purpose**:
  - Centralized functionality for the whole app.
  - Contains **singleton services** and configurations that should only be instantiated once.
  - No feature-specific or reusable components/directives here.

Layout

- These components define the structural foundation of your application and are reused across multiple features. Therefore, the y play a crucial role in the app's architecture but are not feature-specific.

Dashboard.

If there is only one static dashboard or two static different roles dashboards keep in a separate feature under dashboard fol der
Until and unless your every feature have a unique dashboard, then keep in that specific feature. Like hrms-dashboard

Shared folder (for feature modules)

The **shared** folder is for **reusable components, directives, pipes, and modules** that are shared across different feature modules.

Purpose:

- Provides common, reusable building blocks for features.
- Meant to be imported into feature modules as needed.
- To Avoids code duplication.

Best Practices:

- Shared modules can be imported into any **feature module** but **never directly into the AppModule** to avoid bloat.
- Keep the SharedModule lightweight and focused on **UI and utilities**.

Core Module vs shared module Key Differences

Aspect	Core	Shared
Purpose	App-wide services and utilities	Reusable UI and utilities
Scope	Singleton, globally available	Feature-specific or module-wide
Examples	Auth services, guards, interceptors	UI components, pipes, directives
Import	Only in AppModule	In feature modules as needed
Reusability	Not modular or reusable	Highly modular and reusable

How They Work Together

- **Core** provides the backbone (services, configuration, guards) for the app.
- **Shared** provides building blocks (UI components, directives, pipes) for feature modules.

For example:

- A **LoggerService** in the core module logs errors globally.
- A **LoaderComponent** in the shared module is used in multiple feature modules.

This separation ensures clean architecture and scalability.