Software Development Guideline & Standard

MS SQL Server | Asp.Net core | Angular 13+

There must be some set of rules, techniques, conventions and guidelines for software development so that the code is clean, readable and efficient with minimum error. The software engineers can easily build sophisticated and highly functional code.

Database Design & Scripting Standard

For the database back-end part, we are using the latest version of mySQL, MS SQL Server and Oracle. In every case we have a common technique to design a database.

Database design Technique

1. Identify records from business analysis documents.
2. Perform normalization.
3. Create a data dictionary with full of description and logic considering "Technical Analysis documents (Business & check constraints))"
4. Create a database considering naming conventions.

Naming Convention for creating database objects

SL	Object	Description	Example
1	Database	UPPER_CASE. (_) is a separator of each word.	RIS_ERP
2	Schema	Single word.	Auth
3	Table	Snack_Case	User_Info
4	Field	snack_case	user_info_id
5	PK Field	<table name> _id	person.person_id
6	View	snack_case→ "view" + (_) + <table name>	view_user_info
7	Store Procedure	For Insert Update and Delete→	sp_iud_user_info
8		For Process	sp_process_attendance
9		For reports	sp_report_attendance
10	Function	Snack case→ based on the return type table or value the convention will be	fn_lst_string_to_list fn_value_list_to_string

11	Parameter variable	To easily understand when a variable is declared coming from a parameter we can set "param" as a suffix.	@param_person_id
12	Private variable	When a variable is declared to the Store procedure or function or any script, set "pv" as suffix	@pv_product_quantity
13	Unique Key		UC_Column_Name
14	Index		IDX_Column_Name

Database scripting

1. Write a short description at the beginning of each sql file like the following.

   -- ==

   -- Author: <Author,,Name>

   -- Create date: <Create Date>

   -- Description: <Description>

   -- ==

   -- ==

   -- Modified by : <Modifier Name>

   -- Modify Date: <Modify Date>

   -- Description: <Description>

   -- ==
2. Always write a query on UPPERCASE. Ex: "SELECT * FROM user_info;"
3. In case of insert, update and delete, please start with "Begin Tran" and end with "Commit Tran".
4. In Other cases like reports there is no need to use Begin Tran and Commit tran. Start with "Begin" and end with "End".
5. Use Rollback transactions to maintain atomicity and integrity.
6. Try to avoid cursors that can be managed by temp tables.
7. Use (NOLOCK) in the reporting script.
8. Create indexes where necessary.

Backend Project Structure, Guideline & Standard (C#, .net core)

A project solution may have multiple projects based on business modules and database schema so that each module can act individually. As a basic framework the solution should have the following projects.

1. Auth: The purpose of this project is to keep all types of security issues like
   a. Menus
   b. User management
   c. User authentication
   d. User authorization
   e. User Password management
   f. User Login
   g. Audit Log

h. Etc.

2. Admin: The purpose of this project is to keep all the common structured (Master) models so that all modules can use this model. The model may be the following
    a. Settings (Email, Notification, Approval Matrix etc.)
    b. Basic Information (MoU, currency, Charts of accounts, Product, etc)
    c. Organizational Structure
    d. Calendar
    e. Store

Naming Convention

SL	Item	Description	Example
1	Class,Struct, Enum	PascalCasing	public class **AttendancePolicy** {}
2	Interface	Prefix "I" with PascalCasing	Public interface **IAttendancePolicy**{}
3	Method	PascalCasing	public double GetBonus()
4	Method Argument	CamelCasing	public double GetBonus(int employeeId)
5	Public Member (property)(ViewModel)	PascalCasing	Public int EmployeeId {get;set;}
6	Private Member (property)(ViewModel)	CamelCasing	Private double employeeSalary {get;set;}
	Member (Domain Model)	snake-case (DB object)	employee_id
7	Typed Variable	CamelCasing with type	numeric : int nTotalNumber, double nBonusAmount. string : String sName Boolean: bool isActive, bool isSync DateTime: DateTime dtStartDate
8	Non Typed variable	CamelCashing	Var firstName="Kamal";
9	Constant Variable	UPPER_CASE with underscore separator	Public const String IP_ADDRESS="192.168.0.1"
10	Single Instance	PascalCasing with prefix "o"	oEmployee = new Employee();
11	List Object	PascalCasing with prefix "o" and suffix s	oEmployees = List<Employee>[].
12	Global TypedVariable	CamelCasing with prefix "_" and type	_sName, _oEmployee, _nBonusAmount
13	LINQ query	Lower case	from customer in customers where customer.City == "Seattle" select customer.Name;

Front End Project Structure , Naming Convention and Standard (Angular)

Project Structure

Based on the elasticity, scalability of the project, the structure is classified into the following modules.

1. Root Module
2. Feature Module
3. Shared Module
4. Core Module
5. Style
6. Asset

Root Module

Angular requires one module to be loaded as the application starts. We call this the root module. The *root module* loads the root component and all other modules. The *root module* is conventionally called as `AppModule` and created under the `/src/app`.folder

Feature Module

The Features module implements a specific feature of the Application. All the components, pipes & directives which implement the feature become part of the module.

Shared Module

There are many components, directives & pipes, which we may like to share across various modules. All these components should go into the shared module.

The shared module and must declare the components, pipes, and directives using the `declarations` metadata and export it using the `exports` metadata

Other Modules can import the shared modules and use the *exported* components, directives & pipes

The Services must not be defined here. Since the shared modules are imported everywhere, it may create a new instance of the service if it is imported in the *lazy loaded modules*.

The Shared module must be created under the folder `/src/app/shared` folder.

The Shared module should not have any dependency on any of the other modules in the application.

The commonly required angular modules like (CommonModule, FormsModule, etc) or third party modules can be imported here and re-exported. The other module importing the shared module does not have to import those modules.

Core Module

The Services shared across the application must become part of the `CoreModule`. The user authentication services, services that fetch data are examples of such services.

The Services usually need to be Singleton, Only one instance of the Service must exist. Providing it in CoreModule ensures that the services remain singleton

The core module must be imported only in the root module. Other modules must not import the core modules. You can use the following code to stop the other modules from importing the core module.

Source:

1. https://medium.com/@shijin_nath/angular-right-file-structure-and-best-practices-that-help-to-scale-2020-52ce8d967df5
2. https://www.tektutorialshub.com/angular/angular-folder-structure-best-practices/
3. https://github.com/mathisGarberg/angular-folder-structure/tree/master/src/app
4. https://stackoverflow.com/questions/52933476/angular-project-structure-best-practice

Naming Convention

SL	Item	Description	Example
	folders, component selectors, files	Kebab-case. All file name must be in lower case with the item name like component, service, module.	attendance-policy.component.ts/css/html
1	Class	PascalCasing	class **AttendancePolicyComponent** {}
2	Interface	Prefix "I" with PascalCasing	Public interface **IAttendancePolicy**{}
3	Method	camelCasing	getBonus()
4	Method Argument	camelCasing	getBonus(int employeeId)
5	Member (property)	PascalCasing	Public int EmployeeId {get;set;}
6			
7	Typed Variable	CamelCasing with type	numeric : int nTotalNumber, double nBonusAmount. string : String sName Boolean: bool isActive, bool isSync DateTime: DateTime dtStartDate
8	Non Typed variable	CamelCashing	Var firstName="Kamal";
9	Constant Variable	UPPER_CASE with underscore separator	Public const String IP_ADDRESS="192.168.0.1"
10	Single Instance	PascalCasing with prefix "o"	oEmployee = new Employee();
11	List Object	PascalCasing with prefix "o" and suffix s	oEmployees = List<Employee>[].
12	Global TypedVariable	CamelCasing with prefix "_" and type	_sName, _oEmployee, _nBonusAmount