**code architecture**

OA / Base

* **Common**
  + Date Time Base.
    - GetCurrentDateTime(int hour) function: return current data time.
    - CustomDateTimeFormat(int hour) function: return custom format date time.
  + Generate Randoms Base.
    - RandomNumDigit(int num) function: return generate random code
    - Random4Digit() function: return generate random code from 4digit
* **Dictionary**
  + Roles Dictionary
    - AddRoles(EnumTypeRole type) function: add system roles to dictionary.
    - GetRoles() function: return all roles.
    - GetRoleKey() function: return single role by value.
    - GetRoleValue() function: return single role by key.
* **Enums**
  + Enums.
  + Feed Back.
* **Helpers**
  + Json Helper
    - Serialize();function: serialize object to json
    - Deserialize<TObject>(string json); function: deserialize from json to object
* **Messages**
  + System Messages : Contains all feedback messages.

OA / Data

* **Base Entity**: content info of (user CreatedBy, user UpdatedBy)
* **Models**
  + Asp Net User: Contains all users info
  + Department: Contains all department info
  + Department Employee: Contains all department employee info
  + Evaluation: Contains all evaluation info

OA / Dtos

* **Service View Model**
  + Asp Net User
  + Department Employee View Model
  + Department View Model
  + Evaluation View Model
  + Roles View Model
  + User Manage View Model
  + User View Model
* Mailer Domin Model
* Exception Handling Model
* Fire Base View Model

OA / Repo

* App Db Context: data base Shema
* Start Migration: creating database schema

OA / Service

* **Interfaces**
  + ICrud:
    - Task<TEntity> FindAsync(short id);
      * Param: id
      * Return: T model
      * References: 4 refs
    - Task<TEntity> FindAsync(int id);
      * Param: id
      * Return: T model
      * References: 4 refs
    - Task<TEntity> FindAsync(long id);
      * Param: id
      * Return: T model
      * References: 4 refs
    - Task<TEntity> FirstOrDefaultAsync(Expression<Func<TEntity, bool>> filter = null, params Expression<Func<TEntity, object>>[] includes);
      * Param: none
      * Return: T model
      * References: 3 refs
    - Task<IEnumerable<TEntity>> ListAsync(params Expression<Func<TEntity, object>>[] includes);
      * Param: list of params
      * Return: T model
      * References: 3 refs
    - Task<IEnumerable<TEntity>> ListAsync(Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>> orderBy = null, params Expression<Func<TEntity, object>>[] includes);
      * Param: list of params
      * Return: T model
      * References: 3 refs
    - Task<IEnumerable<TEntity>> ListAsync(Expression<Func<TEntity, bool>> filter = null, Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>> orderBy = null, params Expression<Func<TEntity, object>>[] includes);
      * Param: list of params
      * Return: T model
      * References: 10 refs
    - Task<IEnumerable<TEntity>> ListAsync(int skip, int take, Expression<Func<TEntity, bool>> filter = null, Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>> orderBy = null, params Expression<Func<TEntity, object>>[] includes);
      * Param: skip and take and list of params
      * Return: T model
      * References: 10 refs
    - Task<(FeedBack, TEntity)> PostAsync(TEntity model);
      * Param: T model
      * Return: feedback and T of model
      * References: 6 refs
    - Task<(FeedBack, IEnumerable<TEntity>)> PostAsync(IEnumerable<TEntity> model);
      * Param: T model
      * Return: feedback and T of model
      * References: 3 refs
    - Task<(FeedBack, TEntity)> UpdateAsync(TEntity model);
      * Param: T model
      * Return: feedback and T of model
      * References: 7 refs
    - Task<FeedBack> DeleteAsync(TEntity model);
      * Param: T model
      * Return: feedback
      * References: 6 refs
    - Task<FeedBack> DeleteAsync(IEnumerable<TEntity> model);
      * Param: T model
      * Return: feedback
      * References: 3 refs
  + IRole:
    - Task<IdentityRole> FindByNameAsync(string name);
      * Param: user name
      * Return: role
      * References: 3 refs
    - Task<bool> RoleExistsAsync(string name);
      * Param: user name
      * Return: is exist or not
      * References: 4 refs
    - Task<FeedBack> PostAsync(IdentityRole role);
      * Param: role model
      * Return feedback
      * References: 2 refs
    - Task<FeedBack> DelAsync(IdentityRole role);
      * Param: role model
      * Return: feedback
      * References: 2 refs
  + IUserAuth:
    - string GetUserId(ClaimsPrincipal currentUser);
      * Param: current login user
      * Return: user id
      * References: 2 refs
    - Task<AspNetUser> FindAsync(string userId);
      * Param: userId
      * Return: user model
      * References: 17 refs
    - Task<AspNetUser> FindByNameAsync(string name);
      * Param: user name
      * Return: user model
      * References: 4 refs
    - Task<AspNetUser> FindByEmailAsync(string email);
      * Param: user email
      * Return: user model
      * References: 3 refs
    - Task<AspNetUser> FindByPhoneNumberAsync(string phoneNumber);
      * Param: user phone
      * Return: user model
      * References: 4 refs
    - string PasswordHasher(AspNetUser model);
      * Param: user model
      * Return: hash password
      * References: 3
    - bool AnyByUserName(string userName);
      * Param: user name
      * Return: is exist or not
      * References: 1 ref
    - bool AnyByUserIdAndUserName(string userId, string userName);
      * Param: user Id and name
      * Return: is exist or not
      * References: 1 ref
    - Task<bool> AnyPasswordAsync(AspNetUser model, string password);
      * Param: user model and password
      * Return: is exist or not
      * References: 2 refs
    - bool AnyByPhoneNumberAndAccountStatus(string phoneNumber, bool accountStatus);
      * Param: phone number and account status
      * Return: is exist or not
      * References: 2 refs
    - bool AnyByEmailAndAccountStatus(string email, bool accountStatus);
      * Param: user email and account status
      * Return: is exist or not
      * References: 1 ref
    - bool AnyByPhoneNumberAndOTPCode(string phoneNumber, int otpCode);
      * Param: user phone and otp code
      * Return: is exist or not
      * References: 2 refs
    - bool AnyByPhoneNumberAndOTPCodeAndOTPUsed(string phoneNumber, int otpCode, bool otpUsed);
      * Param: user phone and otp code and otp user id
      * Return: is exist or not
      * References: 2 refs
    - bool AnyByUserNameAndEmailAndPhoneNumber(string userName, string email, string phoneNumber);
      * Param: user name email and phone
      * Return: is exist or not
      * References: 2 refs
    - bool AnyByUserNameOrEmailOrPhoneNumber(string userName, string email, string phoneNumber);
      * Param: user name and email or phone number
      * Return: is exist or not
      * References: 1 ref
    - bool AnyByUserIdAndUserNameAndEmailAndPhoneNumber(string UserId, string userName, string email, string phoneNumber);
      * Param: userId and user name and email or phone number
      * Return: user model
      * References: 5 refs
    - Task<string> GeneratePasswordResetTokenAsync(AspNetUser model);
      * Param: userId
      * Return: user model
      * References: 17 refs
    - Task<IdentityResult> ResetPasswordAsync(AspNetUser model, string code, string password);
      * Param: user model new password
      * Return: password
      * References: 2 refs
    - Task<(FeedBack, AspNetUser)> PostAsync(AspNetUser model);
      * Param: user model
      * Return: feedback and user model
      * References: 3 refs
    - Task<FeedBack> PutAsync(AspNetUser model);
      * Param: user model
      * Return: feedback
      * References: 10 refs
    - Task<FeedBack> DelAsync(AspNetUser model);
      * Param: user model
      * Return: feedback
      * References: 2 refs
  + IUserManage
    - Task<List<AspNetUser>> GetAsync(string role);
      * Param: role
      * Return: list of user model
      * References: 2 refs
    - ask<List<AspNetUser>> GetAsync(string[] roles);
      * Param: roles array
      * Return: list of user model.
      * References: 2 refs
    - Task<List<AspNetUser>> GetAsync(string role, int skip, int take);
      * Param: role and skip and take
      * Return: list of user model
      * References: 2 refs
    - Task<List<AspNetUser>> GetAsync(string[] roles, int skip, int take);
      * Param: roles array and skip and take
      * Return: feedback
      * References: 2 refs
    - Task<List<AspNetUser>> GetAsync(EnumUserSearchBy searchBy, string searchValue, int skip, int take);
      * Param: searchBy parm and searchValue and skip and take
      * Return: feedback
      * References: 2 refs
    - Task<(FeedBack, AspNetUser)> PostAsync(AspNetUser model);
      * Param: user model
      * Return: feedback and user model
      * References: 2 refs
  + IUserRole
    - Task<IEnumerable<string>> GetAsync(AspNetUser model);
      * Param: user model
      * Return: list of user roles
      * References: 5 refs
    - Task<bool> AnyUserInRoleAsync(AspNetUser model, string role);
      * Param: user model and role
      * Return: is exist or not
      * References: 2 refs
    - Task<FeedBack> PostAsync(AspNetUser model, string role);
      * Param: user model and role
      * Return: feedback
      * References: 2 refs
    - Task<FeedBack> PostAsync(AspNetUser model, IEnumerable<string> roles);
      * Param: user model and roles list
      * Return: feedback
      * References: 2 refs
    - Task<FeedBack> DelAsync(AspNetUser model, string role);
      * Param: user model and role
      * Return: feedback
      * References: 2 refs
    - Task<FeedBack> DelAsync(AspNetUser model, IEnumerable<string> roles);
      * Param: user model and roles list
      * Return: feedback
      * References: 2 refs

OA Api

* **Configure** **Services**
  + Authentication Service
  + Custom Services
  + Db Service
  + Health Checks Service
  + Installer
  + Installer Extensions
  + Other Services
  + Endpoints
* **Background Job**
  + Background Job Service
* **Filters**
  + Api Key
* **Profiles**
  + User Manage Profile
  + User Profile
* **Services**
  + App Settings
    - For creating default’s data in database
  + Feature Flags
  + Http Clients
    - public bool GoogleRecaptcha(string Response);
    - public FeedBack PushNotification(string DeviceId, string Title, string Body, object Data = null, string AndroidChannelId = null);
    - public async Task<string> SendSMSCode(string phone, string lable, string body);
    - public FeedBack Mailer(MailerDominModel data, EnumMailer MailerType);
* **Unit Of Work**
  + Unit Of Work Classes