Technical documentation

Contents

1- Overview
2- Objectives
3- Framework
4- Global usings
5- Database
6- Localization
7- Areas
8- Services
    a. AwardService
    b. NewsService
    c. CategoryService
    d. FeedbackService
    e. PhotoService
    f. VideoService
    g. ImageServices
    h. ProjectService
    i. SelectedItemService
    j. SuccessStoryService
    k. DashboardService
9- Helper
10- Controllers
    a. Backend Controllers
        i. DashboardController
        ii. CategoryController
        iii. FeedbacksController
        iv. AwardsController
        v. NewsController
        vi. ProjectsController
        vii. PhotoGallery
        viii. VideoGalleriesController
        ix. SuccessStoriesController
        x. SelectedItemsController
    b. User Controllers
        i. HomeController

ii. AwardsController
iii. CategoriesController
iv. FeedbackController
v. NewsController
vi. ProjectsController
vii. PhotoGalleriesController
viii. SuccessStoriesController
ix. VideoGalleriesController
x. LanguagesController

11- JavaScript files

a. ConfirmDelete.js

Overview

Provide a brief summary covering what your product/feature is, the context behind its development and release and what it does. Touch on what documentation is available in your template to help your reader familiarize themselves with it.

Objectives

What does your product do? How should it be used? How will your technical documentation help people use your product and achieve those goals?

Framework

This project is create by using .NET 8 and MS SQL Server 2019 to handle backend requests and HTML 5, CSS 3, Bootstrap 5, JQuery, JS and Ajax to handle client side requests.

And there are required packages that are used in the application

```
<TargetFramework>net8.0</TargetFramework>
<ItemGroup>
  <PackageReference Include="EntityFramework" Version="6.4.4" />
  <PackageReference Include="EPPlus" Version="7.0.9" />
  <PackageReference Include="Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore" Version="7.0.13" />
  <PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="7.0.13" />
  <PackageReference Include="Microsoft.AspNetCore.Identity.UI" Version="7.0.13" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite" Version="7.0.12" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="7.0.13" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="7.0.13" />
  <PackageReference Include="Microsoft.VisualStudio.Azure.Containers.Tools.Targets" Version="1.19.5" />
  <PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Design" Version="7.0.11" />
  <PackageReference Include="Newtonsoft.Json" Version="13.0.3" />
  <PackageReference Include="QRCoder" Version="1.4.1" />
  <PackageReference Include="X.PagedList" Version="9.0.0-prerelease" />
  <PackageReference Include="X.PagedList.Mvc.Core" Version="9.0.0-prerelease" />
  <PackageReference Include="X.PagedList.Web.Common" Version="8.0.7" />
  <PackageReference Include="ZXing.Net" Version="0.16.9" />
  <PackageReference Include="ZXing.Net.Mobile" Version="2.4.1" /> </ItemGroup>
```

Global usings

This is a class that provides used namespaces for all project. It is public for all class and can use packages and namesapces in any file class without needing to use it in that class.

These all namespaces and packages that are used in all files:

```
global using ICT_Fund.Data;
global using ICT_Fund.Models;
global using Microsoft.AspNetCore.Identity;
global using Microsoft.EntityFrameworkCore;
global using Microsoft.AspNetCore.Mvc.Razor;
global using System.ComponentModel.DataAnnotations.Schema;
global using System.ComponentModel.DataAnnotations;
global using ICT_Fund.Helper;
global using ICT_Fund.Services;
global using ICT_Fund.ViewModels;
global using Microsoft.AspNetCore.Mvc;
global using System.Web;
global using System.Net.NetworkInformation;
global using X.PagedList;
global using X.PagedList.Mvc.Core;
global using X.PagedList.Web.Common;
global using X.PagedList;
global using LazZiya.ImageResize;
global using LazZiya.ImageResize;
global using ICT_Fund.Interfaces;
global using System.Drawing;
global using System.Drawing.Imaging;
global using Microsoft.AspNetCore.Identity;
global using Microsoft.AspNetCore.Mvc.Localization;
global using Microsoft.Extensions.Localization;
```

Database

This application uses MS SQL 2019. To use it, you should configure MS SQL in appsettings.json and Program.cs.

In appsettings.json:

```
"ConnectionStrings": {
            "DefaultConnection":"Server=(localdb)\\mssqllocaldb;Database=IctFund;Trusted_Connection=True;MultipleActiveResultSets=true;User ID=admin;Password=Asd123_+" }
```

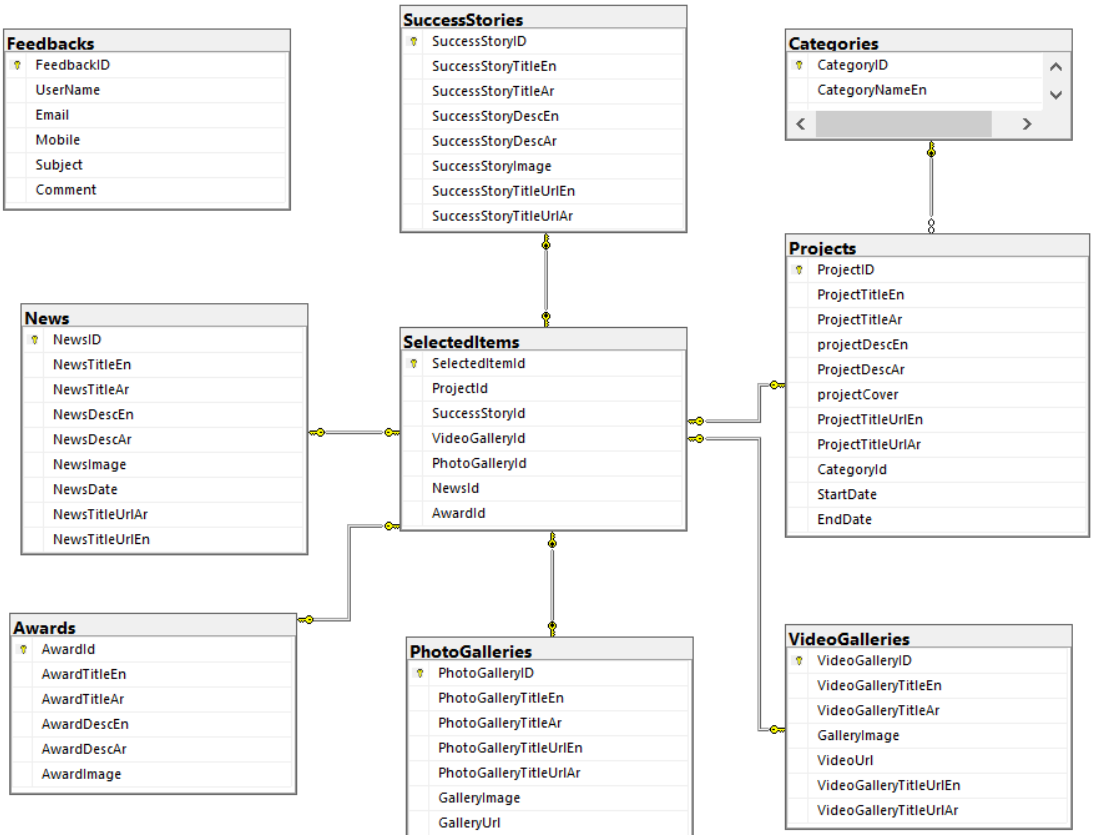
In Program.cs:

```
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");

builder.Services.AddDbContext<ApplicationDbContext>(options =>

   options.UseSqlServer(connectionString));

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

builder.Services.AddIdentity<ApplicationUser, IdentityRole>(options => options.SignIn.RequireConfirmedAccount = true)

   .AddEntityFrameworkStores<ApplicationDbContext>()

   .AddDefaultUI()

   .AddDefaultTokenProviders();
```

This project contains 12 main tables:

1- Users:
   a. Stores data of admins.
2- Roles:
   a. Stores roles of users.
3- UserRoles:
   a. Represents the relationship between Users and Roles tables.
4- Feedbacks:
   a. Stores feedbacks that are added by users.
5- SuccessStories:
   a. Stores success stories that are added by admins, and showcase the achievements of the department.
6- Projects:
   a. Stores projects that are managed by the department. Admins who are allowed to add projects
7- Categories:
   a. Stores categories of projects. Each project has category (if exists).

8- News:
   a. Stores all data of news that are added by admins.
9- Awards:
   a. Stores the awards obtained by the department to showcase to users.
10- PhotoGalleries:
   a. Stores photo galleries that are added by admins.
11- VideoGalleries:
   a. Stores video galleries that are added by admins.
12- SelectedItems:
   a. This table represents selected project, news, award, photo gallery and video gallery that are added by admin.



Using code-first approach to create these tables, generate classes represent each table, Project and SelectedItem classes have a configuration class that inherits from `IEntityTypeConfiguration<ClassName> `to configure relationships between tables. In **ApplicationDbContext.cs** that inherits from `IdentityDbContext<ApplicationUser> ` (note: ApplicationUser is a class inherits from `IdentityUser` class) to use Identity for using authentication and authorization and creating tables in database. To generate a table use:

```
public DbSet<SelectedItem> SelectedItems { get; set; }
public DbSet<Project> Projects { get; set; }
public DbSet<Category> Categories { get; set; }
public DbSet<Award> Awards{ get; set; }
public DbSet<News> News { get; set; }
public DbSet<SuccessStory> SuccessStories { get; set; }
public DbSet<PhotoGallery> PhotoGalleries { get; set; }
public DbSet<VideoGallery> VideoGalleries { get; set; }
public DbSet<Feedback> Feedbacks { get; set; }
```

To use Identity classes and configuration classes use:

```
protected override void OnModelCreating(ModelBuilder builder) {
    base.OnModelCreating(builder);
    builder.Entity<ApplicationUser>().ToTable("Users", "Security");
    builder.Entity<IdentityRole>().ToTable("Roles", "Security");
    builder.Entity<IdentityUserRole<string>>().ToTable("UserRoles", "Security");
    builder.Entity<IdentityUserClaim<string>>().ToTable("UserClaims", "Security");
    builder.Entity<IdentityUserLogin<string>>().ToTable("UserLogins", "Security");
    builder.Entity<IdentityRoleClaim<string>>().ToTable("RoleClaims", "Security");
    builder.Entity<IdentityUserToken<string>>().ToTable("UserTokens", "Security");
    builder.ApplyConfiguration(new ApplicationUserConfiguration());
    builder.ApplyConfiguration(new CertificateConfiguration());
    builder.ApplyConfiguration(new NationalConfiguration()); }
```

Localization

The web application provides English and Arabic languages. For using localization, you should configure it in Program.cs:

```
var resourcePath = "Resources";

builder.Services.AddLocalization(op => op.ResourcesPath = resourcePath);

var supportedCultures = new[] { Language.EN, Language.AR };

builder.Services.Configure<RequestLocalizationOptions>(op => {

   op.SetDefaultCulture(supportedCultures[0])

    .AddSupportedCultures(supportedCultures)

    .AddSupportedUICultures(supportedCultures);  });
```

To add a middleware to use localization, add this code in Program.cs after ‘app.UseAuthorization()’

```
var localizationOptions = new RequestLocalizationOptions()

                .SetDefaultCulture(supportedCultures[0])

                .AddSupportedCultures(supportedCultures)

                .AddSupportedUICultures(supportedCultures);

app.UseRequestLocalization();
```

Note: ‘Resources’ is a file that contains all resource files in project. In each view must have 2 resource file(English and Arabic) and follow that naming convention for Arabic ‘viewName.ar-EG.resx’ and for English ‘viewName.en-US.resx’.

Areas

This project uses areas provided by ASP.NET. to provide a way to partition an ASP.NET Core Web app into smaller functional groups. This area for admins only. To configure Areas in the project, you should configure the routing of an area in Program.cs and this all routes in the app:

```
app.MapControllerRoute(
   name: "default",
   pattern: "/{controller=Home}/{action=Index}/{id?}",
   defaults:new { area= "" });


app.MapControllerRoute(
  name: "backend",
  pattern: "{area:exists}/{controller=Dashboard}/{action=Index}/{id?}",
  defaults: new { area = "Backend" });
```

Services

To separate low level modules from high level modules to achieve Dependency Inversion principle. You must use an interface for each class in a service class that implements that interface to manage all CRUD operations on database.

- **‘AwardService’** is service class that implements ‘IAwardService’ interface (client class) to manage Awards in database.
- **‘NewsService’** is service class that implements ‘ICategoryService’ interface (client class) to manage News in database.
- **‘FeedbackService’** is service class that implements ‘IFeedbackService’ interface (client class) to manage Feedback in database.
- **‘PhotoService’** is service class that implements ‘PhotoService’ interface (client class) to manage photos in database.
- **‘ImageServices’** is service class that implements ‘IImages’ interface (client class) to manage check the size, extensions and resizing of uploaded photos.
- **‘ProjectService’** is service class that implements ‘IProjectService’ interface (client class) to manage projects in database.
- **‘SelectedItemService’** is service class that implements ‘ISelectedItemService’ interface (client class) to manage selected items in database.
- **‘SuccessStoryService’** is service class that implements ‘ISuccessStoryService’ interface (client class) to manage successStories in database.

- **‘VideoService’** is service class that implements ‘IVideoService’ interface (client class) to manage videos in database.
- **‘DashboardService’** is service class that implements ‘IDashboardService’ interface (client class) to manage Users table in database.
- **‘CategoryService’** is service class that implements ‘ICategoryService’ interface (client class) to manage categories in database.

To inject these services and use them in Controllers use this code in program.cs:

```
builder.Services.AddScoped<IAwardService,AwardService>();
builder.Services.AddScoped<INewsService,NewsService>();
builder.Services.AddScoped<ICategoryService,CategoryService>();
builder.Services.AddScoped<IFeedbackService,FeedbackService>();
builder.Services.AddScoped<IPhotoService,PhotoService>();
builder.Services.AddScoped<IImages,ImageServices>();
builder.Services.AddScoped<IProjectService,ProjectService>();
builder.Services.AddScoped<ISelectedItemService,SelectedItemService>();
builder.Services.AddScoped<ISuccessStoryService,SuccessStoryService>();
builder.Services.AddScoped<IVideoService,VideoService>();
builder.Services.AddScoped<IDashboardService,DashboardService>();
```

AwardService

This class inherits from `IAwardService` that responsible for getting, creating, searching and deleting an award.

'AwardService' includes:

- **'AddAsync(Award model)':**
  - Is a sync method to add a new award. It has an instance of Award model class as a parameter and add it in Awards table.
- **'DeleteAsync(int id)':**
  - Is a sync method to delete an award, it has integer parameter 'id' to check if there is an award in Awards table has that id to delete.
- **'GetAllAsync()':**
  - Is a sync method that retrieves all rows in Awards table.
- **'GetFilteredAwardsAsync(string titleSearch, int? page)':**
  - Is a sync method to retrieve filtered rows in Awards table according to parameter 'titleSearch'.
- **'GetByIdAsync(int id)':**
  - Is a sync method that retrieves an award that has ID that equals 'id' parameter.
- **'UpdateAsync(int id, Award model)':**
  - Is a sync method to update an award. 'id' parameter to get an awards with that ID. 'model' is an instance of Awards model class that has new data that will be stored

NewsService

This class inherits from `INewsService` that responsible for getting, creating, searching and deleting news. It includes:

- **'AddAsync(News model)':**
  - Is a sync method to add a new News. It has an instance of News model class as a parameter and add it in News table.
- **'DeleteAsync(int id)':**
  - Is a sync method to delete News, it has integer parameter 'id' to check if there is News in News table has that id to delete.
- **'GetAllAsync()':**
  - Is a sync method that retrieves all rows in News table.
- **'GetFilteredNewsAsync(string titleSearch, int? page)':**
  - Is a sync method to retrieve filtered rows in News table according to parameter 'titleSearch'.
- **'GetByIdAsync(int id)':**
  - Is a sync method that retrieves a News that has ID that equals 'id' parameter.

- **‘UpdateAsync(int id, News model)’:**
    - Is a sync method to update News. ‘id’ parameter to get News with that ID. ‘model’ is an instance of News model class that has new data that will be stored
- **‘GetAllWithPaginationAsync(int? page)’:**
    - Retrieve 6 news in each page.

CategoryService

This class inherits from `ICategoryService` that responsible for getting, creating, searching and deleting a category:

- **‘AddAsync(Category model)’:**
    - Is a sync method to add a new Category. It has an instance of Category model class as a parameter and add it in Category table.
- **‘DeleteAsync(int id)’:**
    - Is a sync method to delete Category, it has integer parameter ‘id’ to check if there is Category in Category table has that id to delete.
- **‘GetAllAsync()’:**
    - Is a sync method that retrieves all rows in Category table.
- **‘GetFilteredCategoriesAsync(string titleSearch, int? page)’:**
    - Is a sync method to retrieve filtered rows in Category table according to parameter ‘titleSearch’.
- **‘GetByIdAsync(int id)’:**
    - Is a sync method that retrieves a Category that has ID that equals ‘id’ parameter.
- **‘UpdateAsync(int id, Category model)’:**
    - Is a sync method to update Category. ‘id’ parameter to get News with that ID. ‘model’ is an instance of Category model class that has new data that will be stored.

FeedbackService

This class inherits from `IFeedbackService` that responsible for getting, creating, searching and deleting a feedback.

- **‘AddAsync(Feedbackmodel)’:**
  - Is a sync method to add a new Feedback. It has an instance of Feedbackmodel class as a parameter and add it in Feedbacks table.
- **‘DeleteAsync(int id)’:**
  - Is a sync method to delete Feedback, it has integer parameter ‘id’ to check if there is Feedbackin Feedbacks table has that id to delete.
- **‘GetAllAsync()’:**
  - Is a sync method that retrieves all rows in Feedbacks table.
- **‘GetFilteredFeedbacksAsync(string titleSearch, int? page)’:**
  - Is a sync method to retrieve filtered rows in Feedbacks table according to parameter ‘titleSearch’.
- **‘GetByIdAsync(int id)’:**
  - Is a sync method that retrieves a Feedbacks that has ID that equals ‘id’ parameter.

PhotoService

This class inherits from `IPhotoService` that responsible for getting, creating, searching and deleting photos. It includes:

- **‘AddAsync(PhotoGallery model)’:**
  - Is a sync method to add a new Photo. It has an instance of PhotoGallery model class as a parameter and add it in PhotoGalleries table.
- **‘DeleteAsync(int id)’:**
  - Is a sync method to delete PhotoGallery, it has integer parameter ‘id’ to check if there is PhotoGallery in PhotoGalleries table has that id to delete.
- **‘GetAllAsync()’:**
  - Is a sync method that retrieves all rows in PhotoGalleries table.
- **‘GetFilteredPhotoGalleriesAsync(string titleSearch, int? page)’:**
  - Is a sync method to retrieve filtered rows in PhotoGalleries table according to parameter ‘titleSearch’.
- **‘GetByIdAsync(int id)’:**
  - Is a sync method that retrieves a PhotoGalleries that has ID that equals ‘id’ parameter.

- **‘UpdateAsync(int id, PhotoGallery model)’:**
    - Is a sync method to update PhotoGallery. ‘id’ parameter to get PhotoGalleries with that ID. ‘model’ is an instance of PhotoGallery model class that has new data that will be stored
- **‘GetAllWithPaginationAsync(int? page)’:**
    - Retrieve 6 PhotoGalleries in each page.

VideoService

This class inherits from `IVideoService` that responsible for getting, creating, searching and deleting videos. It includes:

- **‘AddAsync(VideoGallery model)’:**
    - Is a sync method to add a new video. It has an instance of VideoGallery model class as a parameter and add it in VideoGalleries table.
- **‘DeleteAsync(int id)’:**
    - Is a sync method to delete VideoGallery, it has integer parameter ‘id’ to check if there is VideoGallery in VideoGalleries table has that id to delete.
- **‘GetAllAsync()’:**
    - Is a sync method that retrieves all rows in VideoGalleries table.
- **‘GetFilteredVideoGalleriesAsync(string titleSearch, int? page)’:**
    - Is a sync method to retrieve filtered rows in VideoGalleries table according to parameter ‘titleSearch’.
- **‘GetByIdAsync(int id)’:**
    - Is a sync method that retrieves a VideoGalleries that has ID that equals ‘id’ parameter.
- **‘UpdateAsync(int id, VideoGallery model)’:**
    - Is a sync method to update VideoGallery. ‘id’ parameter to get VideoGalleries with that ID. ‘model’ is an instance of VideoGallery model class that has new data that will be stored
- **‘GetAllWithPaginationAsync(int? page)’:**
    - Retrieve 6 VideoGalleries in each page.

ImageServices

This class inherits from `IImage` that responsible for uploading, resizing and deleting images. It includes:

- **‘UploadFile(IFormFile img, string root)’:**
  - A sync method to upload a file in wwwroot directory and retrieves image name. ‘img’ parameter is an instance of IFormFile that contains image. ‘root’ is a string to store an image in specific root in wwwroot folder.
- **‘ResizeImg(string fileName, string root, int width, int height)’:**
  - Void method to resize an image after uploading. ‘fileName’ parameter represents image name in its directory, ‘root’ the directory of image name, width and height represent the new revolution of image.
- **‘DeleteFile(string img, string root)’**
  - Void method to delete an image. ‘Img’ represents image name, ‘root’ the directory of image name.

ProjectService

This class inherits from `IProjectService` that responsible for getting, creating, searching and deleting project. It includes:

- **‘AddAsync(Project model)’:**
  - Is a sync method to add a new Project. It has an instance of Project model class as a parameter and add it in Projects table.
- **‘DeleteAsync(int id)’:**
  - Is a sync method to delete Project, it has integer parameter ‘id’ to check if there is Project in Projects table has that id to delete.
- **‘GetAllAsync()’:**
  - Is a sync method that retrieves all rows in Projects table.
- **‘GetFilteredProjectsAsync(string titleSearch, int? page)’:**
  - Is a sync method to retrieve filtered rows in Projects table according to parameter ‘titleSearch’.
- **‘GetByIdAsync(int id)’:**
  - Is a sync method that retrieves a Project that has ID that equals ‘id’ parameter.
- **‘UpdateAsync(int id, Project model)’:**
  - Is a sync method to update Project. ‘id’ parameter to get Project with that ID. ‘model’ is an instance of Project model class that has new data that will be stored
- **‘GetAllWithPaginationAsync(int? page)’:**
  - Retrieve 6 news in each page.
- **‘GetAllByCategoryIdAsync(int CategoryId,int? page)’:**
  - A sync method to retrieve projects that are related to specific category. ‘CategoryId’ is a parameter that represents ID of a category. ‘page’ is a parameter represents number of page.

- **‘GetEffectedCategoryAsync(int? page)’:**
    - A sync method to retrieve projects that are related to effected categories. ‘page’ is a parameter represents number of page.

SelectedItemService

This class inherits from `ISelectedItemService` that responsible for getting and updating selected item. It includes:

- **‘GetAllAsync()’:**
    - Is a sync method that retrieves all rows in SelectedItems table.
- **‘GetByIdAsync(int id)’:**
    - Is a sync method that retrieves a Selected Item that has ID that equals ‘id’ parameter.
- **‘UpdateAsync(int id, SelectedItem model)’:**
    - Is a sync method to update SelectedItem. ‘id’ parameter to get SelectedItem with that ID. ‘model’ is an instance of SelectedItem model class that has new data that will be stored.

SuccessStoryService

This class inherits from `ISuccessStoryService` that responsible for getting, creating, searching and deleting SuccessStory. It includes:

- **‘AddAsync(SuccessStorymodel)’:**
    - Is a sync method to add a new SuccessStory. It has an instance of SuccessStory model class as a parameter and add it in SuccessStories table.
- **‘DeleteAsync(int id)’:**
    - Is a sync method to delete SuccessStory, it has integer parameter ‘id’ to check if there is SuccessStory in SuccessStories table has that id to delete.
- **‘GetAllAsync()’:**
    - Is a sync method that retrieves all rows in SuccessStories table.
- ‘**GetFilteredSuccessStoriesAsync(string titleSearch, int? page)’:**
    - Is a sync method to retrieve filtered rows in SuccessStories table according to parameter ‘titleSearch’.
- **‘GetByIdAsync(int id)’:**
    - Is a sync method that retrieves a SuccessStory that has ID that equals ‘id’ parameter.

- **‘UpdateAsync(int id, SuccessStory model)’:**
    - Is a sync method to update SuccessStory. ‘id’ parameter to get News with that ID. ‘model’ is an instance of SuccessStory model class that has new data that will be stored
- **‘GetAllWithPaginationAsync(int? page)’:**
    - Retrieve 4 news in each page.

DashboardService

This class inherits from `IDashboardService` that responsible for getting and filtering admins in Users table. It includes:

- **‘GetAllAsync()’:**
    - Is a sync method that retrieves all rows in Users table.
- **‘GetFilteredUsersAsync(string titleSearch, int? page)’:**
    - Is a sync method to retrieve filtered rows in Users table according to parameter ‘titleSearch’.

Helper

Is a folder that contains 4 classes used for general purpose:

- **‘FileValidation’:**
    - This class for checking the size and extension of uploaded photos.
- **‘Language’:**
    - A class contains 2 constants fields that stores supported languages.
- **‘Role’:**
    - Class has constants fields that represent roles of system.
- **‘ValidationOnUrl’**
    - Class for check validation of URL.

Controllers

In the Backend Area of the application, several controllers handle different aspects of functionality. These controllers serve as intermediaries between the user interface and the backend logic, ensuring seamless interaction and data processing.

1- Backend Controllers

a. DashboardController

The DashboardController is responsible for managing and displaying user data within the backend of the application. It supports filtering and pagination of users. This controller interacts with the IDashboardService to retrieve the necessary data and uses localization resources to provide multilingual support. The controller is secured to ensure that only users with the ADMIN role can access its functionality.

- **'Index(string titleSearch, string sortOrder, int? page)':**
  - The action method fetches filtered user data based on the search query and sorts the results according to the specified order.
  - It supports both English and Arabic sorting through the ViewBag.TitleSortParmEn and ViewBag.TitleSortParamAr.
  - If no users match the search criteria, a localized message indicating that no records were found is displayed.
  - The data is then paginated and returned to the view.
  - 'titleSearch' a search string to filter users by title. 'sortOrder' determines the sorting order of the results and int? page is the page number for pagination.

b. CategoryController

The CategoriesController manages the categories within the application. It provides functionalities for viewing, creating, editing, and deleting categories. The controller relies on ICategoryService for data operations and uses IStringLocalizer for localization support. The controller is secured to ensure that only users with the ADMIN role can access its functionality.

- **'Index(string titleSearch, string sortOrder, int? page)':**
  - Retrieves a filtered and sorted list of categories.
  - Handles localization of messages if no records are found.
  - The results are paginated and displayed in the view.
  - string titleSearch: Filter categories by title.
  - string sortOrder: Sort the categories.

- int? page: Page number for pagination.

- **‘Create()’**
  - A get method that renders the view for creating a new category.
- **‘Create([Bind("CategoryNameEn,CategoryNameAr")] CategoryViewModel model)’:**
  - Validates and creates a new category based on the provided model.

- ‘**Edit(int id)’:**
  - A get method that retrieves the category by ID and displays it in an editable form.
- **‘Edit(int id, CategoryEditViewModel category)’:**
  - Updates the category details if valid and redirects to the category index.
- **‘Delete(int id)’:**
  - Deletes the category if the ID is valid and the category exists.
  - int id: The ID of the category to be deleted.

c. FeedbacksController

The FeedbacksController handles the management and display of feedback records within the backend of the application. It supports filtering, sorting, and pagination of feedback data. The controller utilizes the IFeedbackService for data operations and IStringLocalizer for localized messages. The controller is secured to ensure that only users with the ADMIN role can access its functionality.

- **‘Index(string titleSearch, string sortOrder, int? page)’:**
  - Retrieves and filters feedback records based on the search query.
  - Supports sorting of feedback data by title in both English and Arabic.
  - If no feedback records match the search criteria, a localized message indicating that no records were found is displayed.
  - The results are paginated and returned to the view.
  - string titleSearch: A search string to filter feedback by title.
  - string sortOrder: Determines the sorting order of the results.
  - int? page: The page number for pagination.

d. AwardsController

The AwardsController manages the operations related to awards within the backend of the application. It handles the creation, display, editing, and deletion of awards, along with file uploads for award images. The controller integrates with the IAwardService for data operations, IImages for image handling, and IStringLocalizer for localization of text. The controller is secured to ensure that only users with the ADMIN role can access its functionality.

- **‘Index(string titleSearch, string sortOrder, int? page)’:**
  - Retrieves and filters award records based on the search query.
  - Supports sorting of awards by title in both English and Arabic.
  - If no awards match the search criteria, a localized message indicating that no records were found is displayed.
  - The results are paginated and returned to the view.
  - string titleSearch: A search string to filter awards by title.
  - string sortOrder: Determines the sorting order of the results.
  - int? page: The page number for pagination.
- **‘Create()’:**
  - A get method that renders the view for creating a new category.
- **‘Create([Bind("AwardTitleEn,AwardTitleAr,AwardDescEn,AwardDescAr, AwardImage")] AwardViewModel model, IFormFile uploadPhoto)’:**
  - Validates the uploaded image for size and extension before saving.
  - If the image passes validation, it is uploaded and resized.
  - A new award is created with the provided data and image, then saved to the database.
- **‘Edit()’:**
  - Displays the form for editing an existing award.
- **‘Edit(int id, AwardEditViewModel Awards, IFormFile? uploadPhoto)’:**
  - Displays the form for editing an existing award.
  - Validates and updates the award details, including the image if a new one is uploaded.
  - If an image is uploaded, the existing image is deleted, and the new one is saved.
- **‘Delete(int id)’:**
  - Deletes an award by its ID.
  - Before deletion, the associated image file is removed from the server.

e. NewsController

The NewsController is responsible for managing news articles within the backend of the application. It handles creating, displaying, editing, and deleting news, along with managing image uploads. The controller interacts with the INewsService for data operations, IImages for image handling, and IStringLocalizer for localization. The controller is secured to ensure that only users with the ADMIN role can access its functionality.

- **‘Index(string titleSearch, string sortOrder, int? page)’:**
  - Retrieves and filters news records based on the search query.
  - Supports sorting of news by title in both English and Arabic.
  - If no news articles match the search criteria, a localized message indicating that no records were found is displayed.
  - The results are paginated and returned to the view.
  - string titleSearch: A search string to filter news articles by title.
  - string sortOrder: Determines the sorting order of the results.
  - int? page: The page number for pagination.
- **‘Create()’:**
  - Displays the form for creating a new news article.
- **‘Create([Bind("NewsTitleEn,NewsTitleAr,NewsDescEn,NewsDescAr,NewsImage,NewsDate,NewsTitleUrlEn,NewsTitleUrlAr")] NewsViewModel model, IFormFile uploadPhoto)’:**
  - Validates the uploaded image for size and extension before saving.
  - If the image passes validation, it is uploaded and resized.
  - A new news article is created with the provided data and image, then saved to the database.
- **‘Edit(int id)’:**
  - Displays the form for editing an existing news article.
- **‘Edit(int id, NewsEditViewModel news, IFormFile? uploadPhoto)’:**
  - Validates and updates the news article details, including the image if a new one is uploaded.
  - If an image is uploaded, the existing image is deleted, and the new one is saved.
- **‘Delete(int id)’:**
  - Deletes a news article by its ID.
  - Before deletion, the associated image file is removed from the server.
  - If successful, the user is redirected to the index page.

f. ProjectsController

The ProjectsController manages the CRUD operations for the "Projects" section in the Backend area of the application. It utilizes multiple services such as IProjectService, ICategoryService, and IImages for handling projects, categories, and image-related operations. Localization is handled using IStringLocalizer. The controller is secured to ensure that only users with the ADMIN role can access its functionality.

- **‘Index(string titleSearch, string sortOrder, int? page)’:**
  - Displays a paginated list of projects with search and sort functionality based on project titles (English and Arabic).
- **‘Create()’:**
  - A get method that renders the view for creating a new project.
- **‘Create([Bind("ProjectTitleEn,ProjectTitleAr,projectDescEn,ProjectDescAr,projectCover,CategoryId,StartDate,EndDate")] ProjectViewModel model, IFormFile uploadPhoto)’:**
  - Validates the uploaded image for size and extension before saving.
  - If the image passes validation, it is uploaded and resized.
  - A new project is created with the provided data and image, then saved to the database.
- **‘Edit(int id)’:**
  - Displays the form for editing an existing project.
- **‘Edit(int id, [Bind("CategoryId,ProjectId,ProjectDescAr,projectDescEn,ProjectTitleAr,ProjectTitleEn,StartDate,EndDate")] ProjectEditViewModel Projects, IFormFile? uploadPhoto)’:**
  - Validates and updates the project details, including the image if a new one is uploaded.
  - If an image is uploaded, the existing image is deleted, and the new one is saved.
- **‘Delete(int id)’:**
  - Deletes a project by its ID.
  - Before deletion, the associated image file is removed from the server.
  - If successful, the user is redirected to the index page.

g. PhotoGallery

The PhotoGalleriesController is part of the Backend area in the .NET 8 web application. It is responsible for managing photo galleries, including creating, editing, and deleting photo galleries, as well as displaying them with pagination and sorting capabilities. The controller is secured to ensure that only users with the ADMIN role can access its functionality.

- **'Index(string titleSearch, string sortOrder, int? page)':**
  - Displays a paginated list of photos with search and sort functionality based on photo titles (English and Arabic).
- **'Create()':**
  - Displays the form for creating a photo gallery.
- **'Create([Bind("PhotoGalleryTitleEn,PhotoGalleryTitleAr,GalleryUrl")] PhotoGalleryViewModel model, IFormFile uploadPhoto)':**
  - Validates the uploaded image for size and extension before saving.
  - If the image passes validation, it is uploaded and resized.
  - A new photo gallery is created with the provided data and image, then saved to the database.
- **'Edit(int id)':**
  - Displays the form for editing an existing photo gallery.
- **'Edit(int id, PhotoGalleryEditViewModel photoGalleries, IFormFile? uploadPhoto)':**
  - Validates and updates the photo gallery details, including the image if a new one is uploaded.
  - If an image is uploaded, the existing image is deleted, and the new one is saved.
- **'Delete(int id)':**
  - Deletes a photo gallery by its ID.
  - Before deletion, the associated image file is removed from the server.
  - If successful, the user is redirected to the index page.

h. VideoGalleriesController

The VideoGalleriesControlleris part of the Backend area in the .NET 8 web application. It is responsible for managing video galleries, including creating, editing, and deleting photo galleries, as well as displaying them with pagination and sorting capabilities. The controller is secured to ensure that only users with the ADMIN role can access its functionality.

- **'Index(string titleSearch, string sortOrder, int? page)':**
  - Displays a paginated list of videos with search and sort functionality based on photo titles (English and Arabic).
- **'Create()':**
  - Displays the form for creating a new video gallery.
- **'Create([Bind("VideoGalleryTitleEn,VideoGalleryTitleAr,VideoUrl")] VideoGalleryViewModel model)':**
  - Validates the uploaded image for size and extension before saving.
  - If the image passes validation, it is uploaded and resized.
  - A new video gallery is created with the provided data and image, then saved to the database.
- **'Edit(int id)':**
  - Displays the form for editing an existing video gallery.
- **'Edit(int id, VideoGalleryEditViewModel VideoGalleries)':**
  - Validates and updates the video gallery details.
- **'Delete(int id)':**
  - Deletes a video gallery by its ID.
  - If successful, the user is redirected to the index page.

i. SuccessStoriesController

The SuccessStoriesController is part of the Backend area in the .NET 8 web application, and it manages operations related to success stories. It allows administrators to create, edit, delete, and list success stories, providing support for pagination, sorting, and image handling.

- **'Create()':**
  - Displays the form for creating a new success story.
- **'Create([Bind("SuccessStoryTitleEn,SuccessStoryTitleAr,SuccessStoryDescEn,SuccessStoryDescAr,SuccessStoryCover")] SuccessStoryViewModel model, IFormFile uploadPhoto)':**
  - Validates the uploaded image for size and extension before saving.
  - If the image passes validation, it is uploaded and resized.
  - A new success story is created with the provided data and image, then saved to the database.

- **‘Edit(int id)’:**
  - Displays the form for editing an existing success story.
- **‘Edit(int id, SuccessStoryEditViewModel successStories, IFormFile? uploadPhoto)’:**
  - Validates and updates the success story details, including the image if a new one is uploaded.
  - If an image is uploaded, the existing image is deleted, and the new one is saved.
- **‘Delete(int id)’:**
  - Deletes a success story by its ID.
  - Before deletion, the associated image file is removed from the server.
  - If successful, the user is redirected to the index page.

j. SelectedItemsController

The SelectedItemsController is located in the Backend area of the .NET 8 web application and is responsible for managing selected items. It interacts with various services to handle operations related to awards, news, photos, videos, projects, and success stories. This controller allows administrators to view and edit selected items. The controller is secured to ensure that only users with the ADMIN role can access its functionality.

- **‘Edit(int id)’:**
  - Displays a form for editing a selected item.
- **‘Edit(int id, [Bind("ProjectID,AwardID,NewsID,SuccessStoryID,PhotoGalleryID,VideoGalleryID")] SelectedItemEditViewModel model)’:**
  - Updates the selected item with the new data from the form.
  - Retrieves and assigns entities from various services based on the IDs submitted.
  - Handles database updates

2- User Controllers

a. HomeController

The HomeController is responsible for managing the homepage. It interacts with three services: INewsService, IAwardService, and ISelectedItemService, which provide data for the homepage view.

- **'Index()':**
  - This action asynchronously retrieves collections of news, awards, and selected items from the respective services. The data is encapsulated in a CollectionViewModel and passed to the view for rendering the homepage.

b. AwardsController

The AwardsController handles the retrieval and presentation of award details in the application.

- **'AwardsDetails(int id)':**
  - This action retrieves the details of a specific award based on the provided ID

c. CategoriesController

The CategoriesController is responsible for managing the display of projects based on categories and highlighting specific categories.

- '**Index(int id, int? page)':**
  - This action retrieves and displays all projects associated with a specific category ID. It also supports pagination.
- **'EffectedCategories(int? page)':**
  - This action retrieves and displays projects that belong to "effected" categories (When 'CategoryId' is NULL), also with pagination support.

d. FeedbackController

The FeedbacksController handles the creation and submission of user feedback.

- **'Create()':**
  - This action renders the form for submitting feedback. It returns the view where users can enter their feedback.
- **'Create(FeedbackViewModel model)':**
  - This action processes the submitted feedback form.

e. NewsController

The NewsController is responsible for managing and displaying news articles in the application.

- **'Index(int? page)':**
  - This action retrieves a paginated list of news articles using the INewsService. The page number is optional and defaults to the first page if not provided. The data is passed to the view for rendering the news list.
- **'NewsDetails(int id)':**
  - This action retrieves and displays the details of a specific news article based on the provided ID.

f. ProjectsController

The ProjectsController manages the display of projects and their details in the application.

- **'ProjectsDetails(int id, int? page)':**
  - This action retrieves and displays the details of a specific project based on the provided ID, along with related projects.

g. PhotoGalleriesController

The PhotoGalleriesController manages the display and details of photo galleries in the application.

- **'Index(int? page)':**
  - This action retrieves a paginated list of photo galleries.
- **'GetDetails(int id)':**
  - This action retrieves and displays the details of a specific photo gallery based on the provided ID.

h. SuccessStoriesController

The SuccessStoriesController is responsible for managing and displaying success stories in the application.

- 'Index(int? page)':
  - This action retrieves a paginated list of success stories.
- 'GetDetails(int id)':
  - This action retrieves and displays the details of a specific success story based on the provided ID.

i. VideoGalleriesController

The VideoGalleriesController manages the display and details of video galleries in the application.

- 'Index(int? page)':
  - This action retrieves a paginated list of video galleries.
- 'GetDetails(int id)':
  - This action retrieves and displays the details of a specific video gallery based on the provided ID.

j. LanguagesController

The LanguagesController manages setting the preferred language for the user.

- **'SetLanguage(string culture)':**
  - This action enables users to change the application's language by setting a cookie with the selected culture. The user's preference is stored for a year, and they are redirected back to the page they were viewing.

JavaScript files

JavaScript code within these files can manipulate the HTML DOM (Document Object Model), add visual effects, respond to user input (clicks, form submissions), make asynchronous requests (fetching data), and more.

ConfirmDelete.js

The confirmDelete.js file contains a JavaScript function used to handle the deletion of items with user confirmation. It leverages the SweetAlert2 library for displaying a confirmation dialog and performs an AJAX request to delete the specified item from the server.

- **ID**: The unique identifier of the item to be deleted.
- **Name**: The name of the item to be displayed in the confirmation dialog.
- **ControllerName**: The name of the controller handling the deletion action.
- **ConfirmDelete**: The message prefix for the confirmation prompt.
- **Yes**: The text for the confirm button.
- **Cancel**: The text for the cancel button.