Project Development Roadmap

Note: In order to open the links, you must download and open the pdf file with a browser.

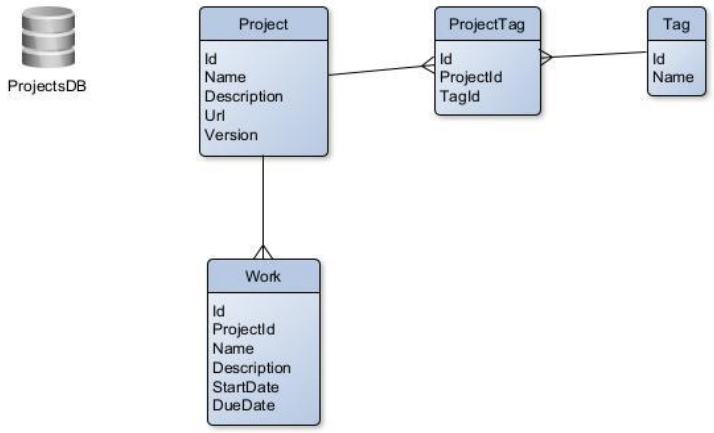
Note: For database, you can either use SQL Server LocalDB if you use Windows and Visual Studio Community, or SQLite (https://www.sqlite.org/) if you use an operating system other than Windows, or SQL Server with Docker if you use an operating system other than Windows.

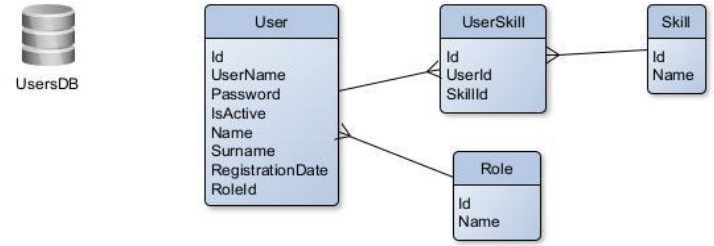
1. Visual Studio Community installation for Windows: https://need4code.com/DotNet/Home/Index?path=.NET\00_Files\Visual%20Studio%20Community\Installation.pdf

2. Rider for MAC: https://www.jetbrains.com/rider

3. Docker Desktop installation for MAC: https://need4code.com/DotNet?path=.NET%5C00_Files%5CDocker%5CDocker%20Microsoft%20SQL%20Server.pdf

4. The E-R Diagram of the project:





5. Create a .NET Aspire Empty App project.

6. Give Solution name as your project name and if you want change the solution folder in Location, Create in new folder option must be checked.

7. Select .NET 8.0 as the Framework, check Configure for HTTPS and select .NET Aspire version as 8.2.

8. In Solution Explorer create a new project called CORE under your solution, right click the Solution then Add -> New Project, then search for Class Library and select it, give the project name CORE then select .NET 8.0 as the Framework.

9. Create the folders and classes under the CORE project as in:
   https://github.com/cagilalsac/PMSCTIS/tree/master/CORE

   You can right click on CORE project then Add -> New Folder to create a folder,
   you can right click on the folder then Add -> Class to create a new C# Class File with the extension .cs.

   The folder and file structure must be as below:
   APP\Domain\Entity.cs
   APP\Features\Handler.cs
   APP\Features\Request.cs
   APP\Features\Response.cs

10. Create a new project under your solution as Class Library and give the name APP.Projects.

11. Create the Tag entity class under the Domain folder as in:
    https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Projects/Domain/Tag.cs

12. Create the ProjectsDb DbContext class under the Domain folder as in:
    https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Projects/Domain/ProjectsDb.cs

13. Right click on your APP.Projects project then Manage NuGet Packages, then from Browse tab search for Microsoft.EntityFrameworkCore.SqlServer package, select the latest version starting with 8 and install.

14. If you want to use SQLite database instead of SQL Server LocalDB, right click on your APP.Projects project then Manage NuGet Packages, then from Browse tab search for System.Data.SQLite.Core package, select the latest version and install, then from the Browse tab search for Microsoft.EntityFrameworkCore.Sqlite package, select the latest version starting with 8 and install.

For data management with SQLite, you can click Extensions -> Manage Extensions from the Visual Studio menu, then search for “sqlite”, select the extension “SQLite and SQL Server Compact Toolbox” and install. You need to close Visual Studio for applying changes.

After you run Visual Studio, from the Tools menu select “SQLite/SQL Server Compact Toolbox”. From the opened tab, select “Add SQLite Connection” from the top menu. From the opened window click “Browse” and select the SQLite database file “PMSCTISProjectsDB” which is under your API.Projects folder. After you close this window, you should see the database in the SQLite/SQL Server Compact Toolbox tab.

15. Create a new project under the solution named API.Projects, search for web api and select ASP.NET Core Web API template, select .NET 8 as the Framework, Authentication type as None, check Configure for HTTPS, check Enable OpenAPI support, Use controllers and Enlist in .NET Aspire orchestration.

16. Right click on API.Projects then Manage NuGet Packages, then from Browse tab search for Microsoft.EntityFrameworkCore.Tools package, select the latest version starting with 8 and install. Right click API.Projects in Solution Explorer then click Set as Startup Project.

17. To create your database under SQL Server (localdb)\MSSQLLocalDB or SQLite database file, from Visual Studio menu, click Tools -> NuGet Package Manager -> Package Manager Console, select the project that has the DbContext, which is APP.Projects as the Default project, run “add-migration v1” (v1 must be a unique name) in the console then after the operation is completed run “update-database”.

    For Rider you can use the UI for these operations as explained in:
    https://www.jetbrains.com/help/rider/Visual_interface_for_EF_Core_commands.html

    You can follow the sub topics such as Entity Framework Core: Add Migration and Entity Framework Core: Update Database. You must select the project which has the DbContext as explained in the Visual Studio database create operations above.

18. You can see your created database under Visual Studio menu -> View -> SQL Server Object Explorer. Expand SQL Server then (localdb)\MSSQLLocalDB. Under Databases our database called PMSCTISProjectsDB can be found. When necessarry, data can be viewed by right clicking the table under the database PMSCTISProjectsDB then clicking View Data. Data can also be inserted, updated or deleted from this window.

    If you use SQLite database, you can find your database file under API.Projects.

19. Right click on APP.Projects then Manage NuGet Packages, then from Browse tab search for MediatR package, select the latest version and install.

20. In APP.Projects, create a new folder named Features. Under Features create a new class called ProjectsDbHandler as in below:
https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Projects/Features/ProjectsDbHandler.cs

21. Under Features folder in APP.Projects, create a new folder named Tags and create the class TagQueryHandler under this folder as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Projects/Features/Tags/TagQueryHandler.cs

22. Then create the file containing TagCreateHandler and TagCreateRequest classes under APP.Projects -> Features -> Tags folder as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Projects/Features/Tags/TagCreateHandler.cs

23. Then create the file containing TagUpdateHandler and TagUpdateRequest classes under APP.Projects -> Features -> Tags folder as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Projects/Features/Tags/TagUpdateHandler.cs

24. Then create the file containing TagDeleteHandler and TagDeleteRequest classes under APP.Projects -> Features -> Tags folder as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Projects/Features/Tags/TagDeleteHandler.cs

25. In API.Projects, open Program.cs to configure the Inversion of Control for the ProjectsDb and IMediator dependency injections as below in the IoC section:
https://github.com/cagilalsac/PMSCTIS/blob/master/API.Projects/Program.cs

26. Instead of using the connection string we defined in ProjectsDb class, we will use the connection string from our project's configuration file which is appsettings.json. So open the ProjectsDb class and comment or delete the OnConfiguring method we overrode before as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Projects/Domain/ProjectsDb.cs
Db classes inherited from the base DbContext class must have a constructor with a DbContextOptions instance parameter in order the use the connection string defined in appsettings.json.

    Then add the connection string to appsettings.json as below:
    https://github.com/cagilalsac/PMSCTIS/blob/master/API.Projects/appsettings.json

27. In API.Projects, create an empty API Controller called TagsController then modify it as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/API.Projects/Controllers/TagsController.cs

28. In APP.Projects, create the Project and ProjectTag entity classes under the Domain folder as in:
    https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Projects/Domain/Project.cs
    https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Projects/Domain/ProjectTag.cs

    Also add the navigation property ProjectTags in the Tag entity:
    ```
    public List<ProjectTag> ProjectTags { get; set; } = new List<ProjectTag>();
    ```

29. In APP.Projects, add the Projects and ProjectTags DbSets in the ProjectsDb class in the Domain folder:
    ```
    public DbSet<Project> Projects { get; set; }
    public DbSet<ProjectTag> ProjectTags { get; set; }
    ```

30. In Package Manager Console, select the project that has the DbContext (ProjectsDb), which is APP.Projects as the Default project, run “add-migration v2” in the console then after the operation is completed run “update-database” to add the Projects and ProjectTags tables to the database.

31. In APP.Projects add ProjectQueryHandler class under the Features\Projects folder with ProjectQueryRequest and ProjectQueryResponse classes as below:
    https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Projects/Features/Projects/ProjectQueryHandler.cs

32. In APP.Projects add ProjectCreateHandler class under the Features\Projects folder with ProjectCreateRequest class as below:
    https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Projects/Features/Projects/ProjectCreateHandler.cs

33. In APP.Projects add ProjectUpdateHandler class under the Features\Projects folder with ProjectUpdateRequest class as below:
    https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Projects/Features/Projects/ProjectUpdateHandler.cs

34. In APP.Projects add ProjectDeleteHandler class under the Features\Projects folder with ProjectDeleteRequest class as below:
    https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Projects/Features/Projects/ProjectDeleteHandler.cs

35. From now on we will use the Scaffolding Templates that the Scaffolding Mechanism of Visual Studio uses, which generates code for our API Controllers. In order to use these templates, go to https://need4code.com/DotNet/Home/Index?path=.NET\00_Files\Scaffolding%20Templates and extract the Templates folder inside the compressed Templates.7z file under your API.Projects folder.

36. Right click the Controllers folder in your API.Projects, Add -> Controller. Then select API from the left menu, then API Controller with actions, using Entity Framework. Select Project entity as the Model class, ProjectsDb for the DbContext class and give the Controller name ProjectsController. After the Scaffolding operation, your controller will be created as below: https://github.com/cagilalsac/PMSCTIS/blob/master/API.Projects/Controllers/ProjectsController.cs

37. In APP.Projects add WorkQueryHandler class under the Features\Works folder with WorkQueryRequest and WorkQueryResponse classes as below: https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Projects/Features/Works/WorkQueryHandler.cs

38. In APP.Projects add WorkCreateHandler class under the Features\Works folder with WorkCreateRequest class as below: https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Projects/Features/Works/WorkCreateHandler.cs

39. In APP.Projects add WorkUpdateHandler class under the Features\Works folder with WorkUpdateRequest class as below: https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Projects/Features/Works/WorkUpdateHandler.cs

40. In APP.Projects add WorkDeleteHandler class under the Features\Works folder with WorkDeleteRequest class as below: https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Projects/Features/Works/WorkDeleteHandler.cs

41. Right click the Controllers folder in your API.Projects, Add -> Controller. Then select API from the left menu, then API Controller with actions, using Entity Framework. Select Work entity as the Model class, ProjectsDb for the DbContext class and give the Controller name WorksController. After the Scaffolding operation, your controller will be created as below: https://github.com/cagilalsac/PMSCTIS/blob/master/API.Projects/Controllers/WorksController.cs

42. **The steps above should be applied to create APP.Users and API.Users projects for entities User, Role, Skill and UserSkill.**

43. In API.Users open the appsettings.json file and add the AppSettings section as below: https://github.com/cagilalsac/PMSCTIS/blob/master/API.Users/appsettings.json

44. In APP.Users create a class called AppSettings as below: https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Users/AppSettings.cs

45. Open the UsersDbHandler class in APP.Users\Features folder, then add CreateAccessToken and GetClaims methos as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Users/Features/UsersDbHandler.cs

46. In APP.Users, install the Microsoft.AspNetCore.Authentication.JwtBearer with the latest version starting with 8 NuGet Package.

47. Create the class TokenHandler in APP.Users\Features\Users folder with TokenRequest and TokenResponse classes as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Users/Features/Users/TokenHandler.cs

48. Open the Program.cs in API.Users and add the parts commented as APP SETTINGS, AUTHENTICATION and SWAGGER as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/API.Users/Program.cs

49. In API.Users, open the UsersController in Controllers folder and add the Token and Authorize actions below:
https://github.com/cagilalsac/PMSCTIS/blob/master/API.Users/Controllers/UsersController.cs

    Now you can get the JWT from the Token action and send the JWT starting with "Bearer" within any request's Authorization header to the actions of the controllers.

50. In the same controller, add **[Authorize]** attribute on top of the **UsersController** class therefore authorization, which will validate if the JWT exists within the request's Authorization header, will be effective for all controller actions.

    Add **[Authorize(Roles = "Admin")]** attribute to the **Post**, **Put** and **Delete** actions therefore authorization for Admin role, which will validate if the JWT exists and the containment of Role claim type with value Admin within the request's Authorization header, will be effective only for these actions.

    Add **[AllowAnonymous]** attribute to **Token** and **Authorize** actions therefore the **[Authorize]** attribute of the **UsersController** class will be ineffective, and any request can be sent to these actions without applying authorization.

51. Apply the Authorize attributes to the RolesController of API.Users as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/API.Users/Controllers/RolesController.cs

52. Apply the Authorize attributes to the SkillsController of API.Users as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/API.Users/Controllers/SkillsController.cs

53. Apply the Authorize attributes to the ProjectsController of API.Projects as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/API.Projects/Controllers/ProjectsController.cs

54. Apply the Authorize attributes to the TagsController of API.Projects as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/API.Projects/Controllers/TagsController.cs

55. Apply the Authorize attributes to the WorksController of API.Projects as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/API.Projects/Controllers/WorksController.cs

56. In APP.Users, open the User entity under the Domain folder, and add the RefreshToken and RefreshTokenExpiration properties as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Users/Domain/User.cs

    Do not forget to add a new migration and update the database.

57. In APP.Users, open the UsersDbHandler class under the Features folder and add the CreateRefreshToken method for random refresh token generation, and GetPrincipal method to reach the user claims principal including claims from a JWT as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Users/Features/UsersDbHandler.cs

58. In APP.Users, open the TokenHandler class under the Features\Users folder and modify the response and handler classes as commented with **REFRESH TOKEN** as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Users/Features/Users/TokenHandler.cs

    Now you can get the JWT with refresh token in the response when you send a request to the Token action of the Users controller.

59. In APP.Users, create the classes RefreshTokenRequest, RefreshTokenResponse and RefreshTokenHandler in the RefreshTokenHandler.cs file under the Features\Users folder as below for the generation of a new JWT and new refresh token when the existing JWT expires:
https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Users/Features/Users/RefreshTokenHandler.cs

60. In APP.Users, add the property RefreshTokenExpirationInDays to the AppSettings class under the APP.Users project as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/APP.Users/AppSettings.cs

61. In API.Users, add the property RefreshTokenExpirationInDays to the appsettings.json file's AppSettings section under the API.Users project as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/API.Users/appsettings.json

62. In API.Users, open the UsersContoller under the Controllers folder and add the RefreshToken action as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/API.Users/Controllers/UsersController.cs

    Now you can get a new JWT with a new refresh token from this action when you send the existing JWT and existing refresh token request.

63. Create a new project under the solution named API.Gateway, search for web api and select ASP.NET Core Web API template, select .NET 8 as the Framework, Authentication type as None, check Configure for HTTPS, do not check Enable OpenAPI support, do not check Use controllers and check Enlist in .NET Aspire orchestration.

64. Right click on the API.Gateway project then click Manage NuGet Packages. Under the Browse tab search for Ocelot and install the latest version.

65. After installation open the Program.cs file of the API.Gateway and modify as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/API.Gateway/Program.cs

66. Add the file ocelot.json under the API.Gateway project and modify as below:
https://github.com/cagilalsac/PMSCTIS/blob/master/API.Gateway/ocelot.json

    During modification, you can get the Host and Port of DownstreamHostAndPorts section from the launchSettings.json files which are under the Properties folder of API.Projects and API.Users.
    If you are running your solution with https, look for the https profile and use the first url defined at applicationUrl section which are for our solution https://localhost:7244 for API.Projects, and https://localhost:7279 for API.Users.
    If you are running your solution with http, look for the http profile and use the url defined at applicationUrl section which are for our solution http://localhost:5231 for API.Projects, and http://localhost:5292 for API.Users.

    Now you can reach all of the project and user endpoints defined in the ocelot.json file through this gateway.

67. You can get the url of API.Gateway from the launchSettings.json file under the Properties folder which is for our solution,

    https: https://localhost:7212 (first url defined at applicationUrl of the https profile),
    http: http://localhost:5260 (url defined at applicationUrl of the http profile).

    Now you can send requests from your client application or Postman to the gateway url above. Unfortunaltey, you can't use Swagger to send requests to the gateway with this project configuration.