Project Development Roadmap

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

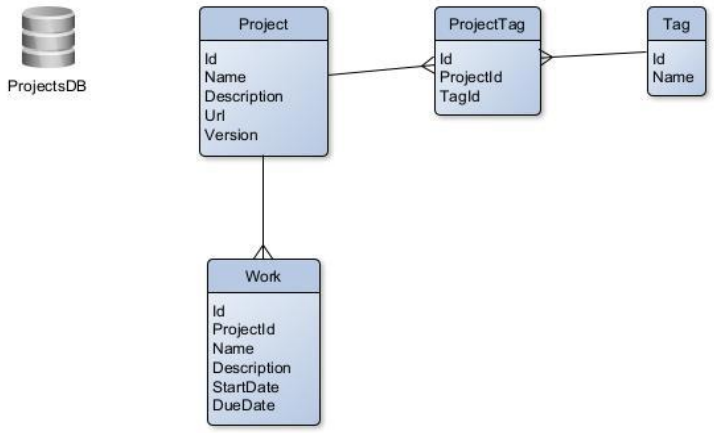
Note: For database, you can either use SQL Server LocalDB if you use Windows and Visual Studio Community, or SQLite (https://www.sqlite.org/) if you use an operating system other than Windows, or SQL Server with Docker if you use an operating system other than Windows.

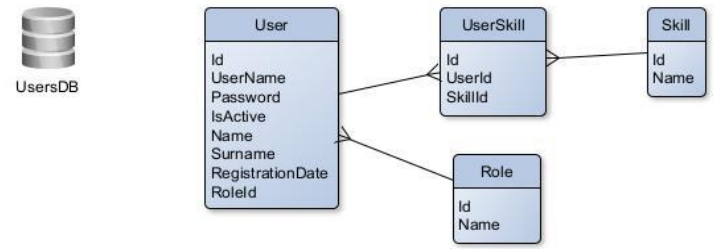
1. Visual Studio Community installation for Windows: https://need4code.com/DotNet/Home/Index?path=.NET\00_Files\Visual%20Studio%20Community\Installation.pdf

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. Right click on the CORE project, then click Properties. Select Build, then change Nullable to Disable. This can also be changed by left clicking the CORE project, then changing the Nullable value to disable in the XML file. This should be done for all class library projects.

10. Create the folders and classes under the CORE project as in:
    https://github.com/cagilalsac/PMS/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\Services\ServiceBase.cs
    APP\Models\Request.cs
    APP\Models\Response.cs
    APP\Models\CommandResponse.cs

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

12. Right click on the Projects.APP project, then click Properties. Select Build, then change Nullable to Disable. This can also be changed by left clicking the Projects.APP project, then changing the Nullable value to disable in the XML file.

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

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

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

16. If you want to use SQLite database instead of SQL Server LocalDB, right click on your Projects.APP 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 "PMSProjectsDB" which is under your Projects.API folder. After you close this window, you should see the database in the SQLite/SQL Server Compact Toolbox tab.

17. Create a new project under the solution named Projects.API, 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.

18. Right click on Projects.API 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 Projects.API in Solution Explorer then click Set as Startup Project.

19. 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 Projects.APP 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.

20. 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 PMSProjectsDB can be found. When necessarry, data can be viewed by right clicking the table under the database PMSProjectsDB 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 Projects.API.

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

22. In Projects.APP, create a new folder named Services. Under Services create a new class called ProjectsDbService as in below:
    https://github.com/cagilalsac/PMS/blob/master/Projects.APP/Services/ProjectsDbService.cs

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

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

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

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

27. In Projects.API, 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/PMS/blob/master/Projects.API/Program.cs

28. 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/PMS/blob/master/Projects.APP/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/PMS/blob/master/Projects.API/appsettings.json

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

30. In Projects.APP, create the Project, ProjectTag and Work entity classes under the Domain folder:
https://github.com/cagilalsac/PMS/blob/master/Projects.APP/Domain/Project.cs
https://github.com/cagilalsac/PMS/blob/master/Projects.APP/Domain/ProjectTag.cs
https://github.com/cagilalsac/PMS/blob/master/Projects.APP/Domain/Work.cs

    Also add the navigation property ProjectTags in the Tag entity:
    ```
    public List<ProjectTag> ProjectTags { get; set; } = new List<ProjectTag>();
    ```
    https://github.com/cagilalsac/PMS/blob/master/Projects.APP/Domain/Tag.cs

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

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

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

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

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

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

37. 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 Projects.API folder.

38. Right click the Controllers folder in your Projects.API, 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/PMS/blob/master/Projects.API/Controllers/ProjectsController.cs

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

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

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

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

43. Right click the Controllers folder in your Projects.API, 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/PMS/blob/master/Projects.API/Controllers/WorksController.cs

44. **Now we are going to implement a base generic service class and use it to manage CRUD operations. Therefore, we can inherit this base generic service class to our applications' service classes and implement the logic needed for our applications' business operations.**

45. install the database independent Microsoft.EntityFrameworkCore NuGet package to the CORE project, therefore DbContext class can be used in the base generic service class.

46. Add the Service class under the APP\Services folder of the CORE project as below:
https://github.com/cagilalsac/PMS/blob/master/CORE/APP/Services/Service.cs

47. Create the UserService class under the Services folder of the Users.APP project as below:
https://github.com/cagilalsac/PMS/blob/master/Users.APP/Services/UserService.cs

48. Create the following handler classes under the Features\Users folder of the Users.APP project as below:
https://github.com/cagilalsac/PMS/blob/master/Users.APP/Features/Users/UserCreateHandler.cs
https://github.com/cagilalsac/PMS/blob/master/Users.APP/Features/Users/UserDeleteHandler.cs
https://github.com/cagilalsac/PMS/blob/master/Users.APP/Features/Users/UserQueryHandler.cs
https://github.com/cagilalsac/PMS/blob/master/Users.APP/Features/Users/UserUpdateHandler.cs

49. Create the following handler classes under the Features\Roles folder of the Users.APP project as below:
https://github.com/cagilalsac/PMS/blob/master/Users.APP/Features/Roles/RoleCreateHandler.cs
https://github.com/cagilalsac/PMS/blob/master/Users.APP/Features/Roles/RoleDeleteHandler.cs
https://github.com/cagilalsac/PMS/blob/master/Users.APP/Features/Roles/RoleQueryHandler.cs
https://github.com/cagilalsac/PMS/blob/master/Users.APP/Features/Roles/RoleUpdateHandler.cs

50. Create the following handler classes under the Features\Skills folder of the Users.APP project as below:
https://github.com/cagilalsac/PMS/blob/master/Users.APP/Features/Skills/SkillCreateHandler.cs
https://github.com/cagilalsac/PMS/blob/master/Users.APP/Features/Skills/SkillDeleteHandler.cs
https://github.com/cagilalsac/PMS/blob/master/Users.APP/Features/Skills/SkillQueryHandler.cs
https://github.com/cagilalsac/PMS/blob/master/Users.APP/Features/Skill/SkillUpdateHandler.cs

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

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

53. Open the UserService class in Users.APP\Services folder, then add CreateAccessToken and GetClaims methos as below:
https://github.com/cagilalsac/PMS/blob/master/Users.APP/Services/UserService.cs

54. In Users.APP, install the Microsoft.AspNetCore.Authentication.JwtBearer with the latest version starting with 8, and Microsoft.IdentityModel.Tokens with version 7.1.2 NuGet Packages.

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

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

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

58. 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.

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

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

61. In Projects.APP, install the Microsoft.IdentityModel.Tokens with version 7.1.2 NuGet Package then add AppSettings.cs file in the project folder as below:
https://github.com/cagilalsac/PMS/blob/master/Projects.APP/AppSettings.cs

62. In Projects.API, add AppSettings section to the appsettings.json file as below:
https://github.com/cagilalsac/PMS/blob/master/Projects.API/appsettings.json

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

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

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

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

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

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

68. In Users.APP, open the UserService class under the Services 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/PMS/blob/master/Users.APP/Services/UserService.cs

69. In Users.APP, 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/PMS/blob/master/Users.APP/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.

70. In Users.APP, 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/PMS/blob/master/Users.APP/Features/Users/RefreshTokenHandler.cs

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

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

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

74. Create a new project under the solution named Gateway.API, 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.

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

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

77. Add the file ocelot.json under the Gateway.API project and modify as below: https://github.com/cagilalsac/PMS/blob/master/Gateway.API/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 Projects.API and Users.API. 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 Projects.API, and https://localhost:7279 for Users.API.
    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 Projects.API, and http://localhost:5292 for Users.API.

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

78. You can get the url of Gateway.API 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.