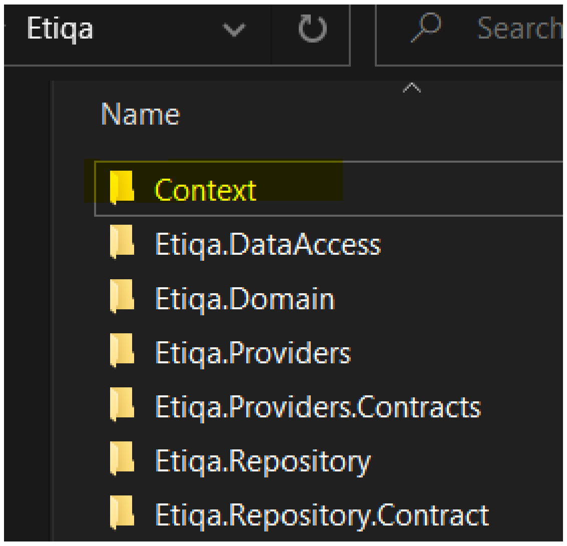
Etiqa

How to Execute.

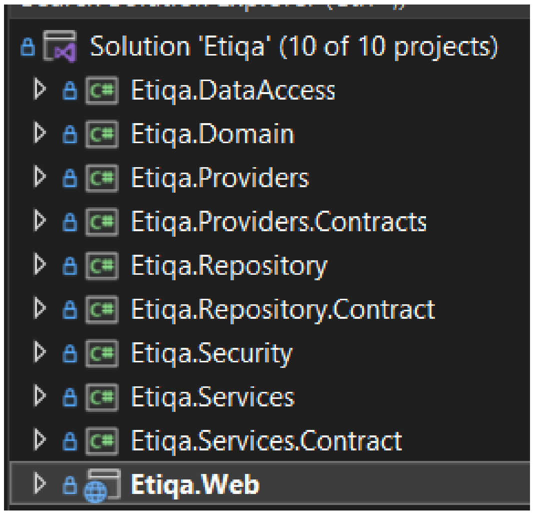
- Here you can find the project. I have pushed to my GitHub and commit a pull request to master branch. You can check the full code in develop branch. PR from develop to master, is there to check the commits and changes which is pending.
    - Git - https://github.com/shehanks/Etiqa
    - PR - https://github.com/shehanks/Etiqa/pull/1
- Execute the MS SQL server "DBScript.sql" file inside the "Context" that exsist in the solution root.



- Add the correct connection string of the DB in "appsettings.json" file.
- Then run the project. Make sure to set up the web project as startup project.
- Swagger Open API will be opened where you can test the endpoints.
    - There are 5 endpoints implemented.
        - Create a user. (Authorization required)
        - Get a use by id.
        - Get user list using the load options like page, page index and search term which supports paging and extendable.
        - Update an existing user. (Authentication required)
        - Delete user. (Authorization required)
- For authorization API key is required and it can be found in "appsettings.json" file.

Architecture and Technologies

- This is a ASP.NET Core 6 Web API application. I have used .NET 6 standard as it has long term support.
- I have used DB first approach with EF core. I prefer code first approach but here I decided to go with DB first as I can write some SQL too.
- I have architected the project which is a clean and layered architecture.
- Repository pattern was used with unit of work.

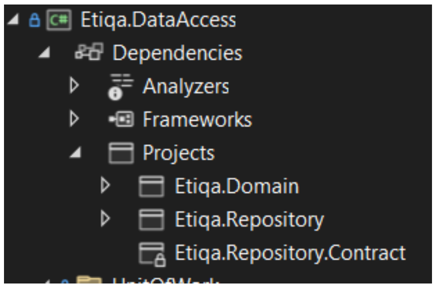


```
Solution 'Etiqa' (10 of 10 projects)
  Etiqa.DataAccess
  Etiqa.Domain
  Etiqa.Providers
  Etiqa.Providers.Contracts
  Etiqa.Repository
  Etiqa.Repository.Contract
  Etiqa.Security
  Etiqa.Services
  Etiqa.Services.Contract
  Etiqa.Web
```

- Below is the architecture of the project.

  APIs are fully asynchronously implemented. Project was designed with interfaces segregation and SOLID principles were applied wherever possible.

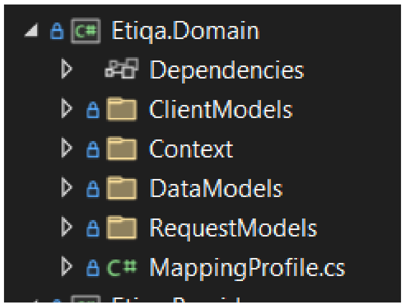
  - Etiqa.DataAccess – Data access unit of work pattern. This layer relates to repository layer.

    ```
    Etiqa.DataAccess
      Dependencies
        Analyzers
        Frameworks
        Projects
          Etiqa.Domain
          Etiqa.Repository
          Etiqa.Repository.Contract
    ```

  - Etiqa.Domain – All the entity models are included here with proper breakdown.
    - DB context is also included in this project.

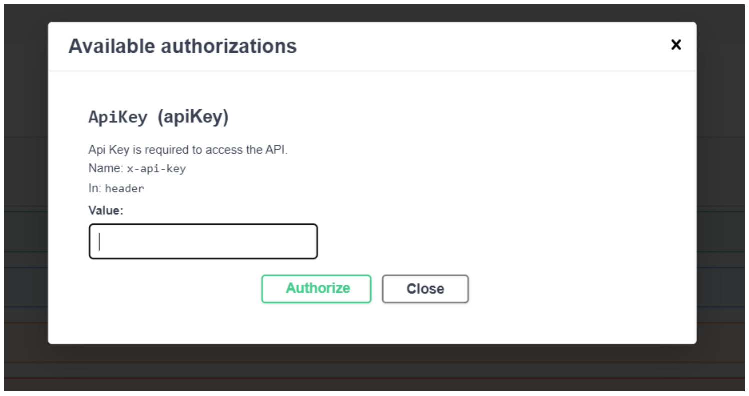
        - Auto-mapper was used to map the entities everywhere in the project. Check the mapping profile. We can use auto-mapper projection with IQueryable to enhance performance that we can only fetch related fields from DB.



    - Etiqa.Providers – This is the layer which connects with entity repositories.
    - Etiqa.Providers.Contracts – Interface contracts of above layer.
    - Etiqa.Repository – Repositories comply to repository pattern implemented with SQL server and EF core.
    - Etiqa.Repository.Contract – Interface contracts of above layer.
    - Etiqa.Security – Authentication was handled here.
        - I have used “x-api-key” authentication to protect the API. This is a static key approach.
        - I would prefer to implement JWT authentication with refresh token with identity, but with the time I had I implemented this for now.
        - Security can be handled with both a middleware and an authorization filter.
        - Both was implemented and I have commented the middleware approach (Program.cs file) because authorization filter has more control over API endpoints.
    - Etiqa.Services – This is the layer of business logic, cache handling service error handling.
        - For caching I have used object cache which is inbuilt in .NET. We can also use “IMemoryCache” for this.
        - When it comes to large scale application we can go for distributed caching, which is more scalable.
        - This needs to be decided on the application requirement and size.
        - For the application separate cache service was implemented.
        - Service errors was handled using “ErrorOr” nuget library as these errors are interacts with API (Controller) layer.
        - Using this library, we can send properly formatted description to the client.
    - Etiqa.Services.Contract - Interface contracts of above layer.
    - Etiqa.Web – This is the web API project.
        - I have implemented a global error controller to handle all the errors, which was configured in the Program.cs file.
        - There we can implement logging or any other you required.
        - I have already implemented them.
        - So, errors have been handled in both service layer and globally.

Other

- I have experience with ReactJS and Redux, that I can implement a client app to demonstrate this. But with the time I have fully configured the swagger open API to test this. We can go ahead with postman too.
- This is a sample react application implemented by me using redux and axios, if you may have a look.
  https://github.com/shehanks/shopping-cart
- Add the API key in swagger to grant access as I have mentioned earlier. If you try to access without authorization a friendly message will be sent with the response.

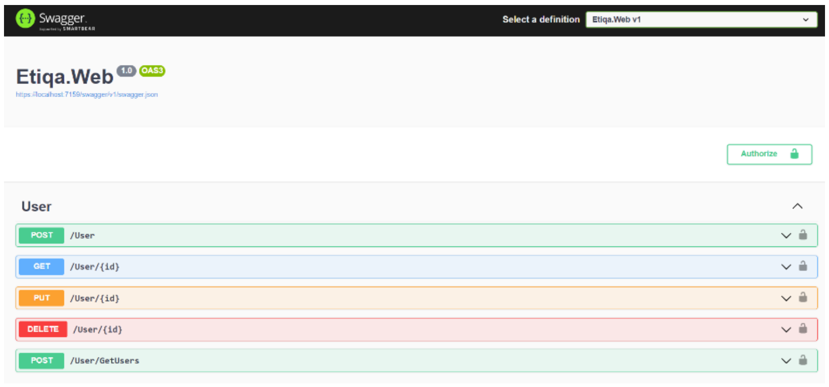


Available authorizations

ApiKey (apiKey)

Api Key is required to access the API.
Name: x-api-key
In: header
Value:

- This is how it looks like. Here you can fully test the API adding, deleting, and obtaining data.



Select a definition: Etiqa.Web v1

Etiqa.Web 1.0 OAS3
https://localhost:7159/swagger/v1/swagger.json

User

```
POST    /User
GET     /User/{id}
PUT     /User/{id}
DELETE  /User/{id}
POST    /User/GetUsers
```

- It is possible to implement unit testing using MsTest, xUnit or NUnit project using a faking library like MockIt or FakeItEasy. With the time I didn’t go for that.
- Also, I can deploy it to azure for testing. To do that I have to create another account and try out since my free credits have been expired. I don’t have experience in AWS for now, but I am willing to learn.