The app is called "The Quotes". The idea is to handle quotes like:

"That’s one small step for a man, one giant leap for mankind." - *Neil Armstrong*

"The way to get started is to quit talking and begin doing." - *Walt Disney*

You will need to create a web API with Entity Framework to handle the backend work. The backend is implemented in src/QuoteApi project. The API provides basic CRUD (Create, Read, Update and Delete) operations for the quotes data. Use Entity Framework to persist the quotes in a SQLite database. You will need to design the datamodel as instructed. Entity classes are implemented in QuoteApi project and they are not usable in the frontend. Use a class library in src/Shared project to implement classes that both the backend and the frontend uses. This app separates model classes that are trasmitted between the backend and the frontend from the data entity classes. This way the frontend is not tightly coupled with the database implementation.

Next you will need to create a user interface with Blazor WASM to handle different operations on the quotes. The frontend application is created in src/Frontend project.

The project templates are only modified to enable them to be tested. No other help for the task is given.

The projects are created with following dotnet CLI commands:

* Frontend: dotnet new blazorwasm
* QuoteApi: dotnet new webapi
* SharedLib: dotnet new classlib

The app requirements:

1. Entity Framework Context

Create a Entity Framework DbContext for the quotes. Create the required classes to the QuoteApi project. Use QuoteContext as the class' name and model the class in QuoteApi.Data namespace. Create the classes in Data folder. The QuoteContext class has a DbSet property for the quotes named Quotes. Model the quote class in the same namespace and name the quote class as Quote. The Quote class has following properties:

* + Id, int, unique id for the quote
  + TheQuote, string, the quote text
  + WhoSaid, string, who said the quote
  + WhenWasSaid, DateTime, when the quote was said
  + QuoteCreator, string, who saved the quote to this app
  + QuoteCreatorNormalized, string, uppercase, the QuoteCreator value in all CAPS
  + QuoteCreateDate, DateTime, when the quote was saved to the app

The QuoteContext has only one constructor. It must be possible to provide the confguration to the context via the constructor.

1. The Web API

Create the web api to handle CRUD operations to the quotes. The api has an ApiController named QuotesController which is accessible from url [https://localhost](https://localhost/):/quotes. The quote api has the following endpoints:

* + HTTP GET [https://localhost](https://localhost/):/quotes -> returns top 5 latest saved quotes
  + HTTP GET [https://localhost](https://localhost/):/quotes/michael -> returns all quotes which are saved by a user name michael or Michael (the username must be case insensitive)
  + HTTP GET [https://localhost](https://localhost/):/quotes/jane/3 -> return a quote with id 3 from user named jane
  + HTTP POST [https://localhost](https://localhost/):/quotes/lisa -> saves a new quote for user name lisa
  + HTTP PUT [https://localhost](https://localhost/):/quotes/berry/4 -> edits the quote with id 4 from user named berry
  + HTTP DELETE [https://localhost](https://localhost/):/quotes/dennis/7 -> removes the quote with id 7 from user named dennis
  + HTTP GET [https://localhost](https://localhost/):/quotes/berry -> returns all quotes which are saved by a user name berry
  + HTTP GET [https://localhost](https://localhost/):/quotes/dennis/3 -> must not work because the quote with id 3 belongs to user named jane. The api must return a 404 Not Found in this case.
  + HTTP PUT [https://localhost](https://localhost/):/quotes/michael/4 -> must not work because the quote with id 4 belongs to user named berry. The api must return a 404 Not Found in this case.

The quotes controller has different endpoints that supports the username (which is case insensitive in all cases) and the quote id. The api must not allow quotes from a different user to be read, edited or deleted.

The GET endpoints that returns a quote or a list of quotes must return the quote data as following json (the comments can be omitted) and also the PUT and POST endpoints must take the same json as request payload:

{

"id": 2, // the quote id

"quote": "quote-text-here",

"saidBy": "who said the quote",

"when": "2022-05-23" // when the quote was said, must use ISO8601 format

}

When creating a new quote the POST endpoint must ignore the possible id value in the request payload. The server (or database) must ensure that the new quote will have an unique id value. The POST endpoint must return properly configured 201 Created response with Location header pointing to the created resourse. When editing and existing quote in the PUT endpoint the api must ignore the possible id value in the request payload. The edited quote id is read from the url.

Implement the required model class in the src/SharedLib project. Use name QuoteDTO as the class name and use namespace SharedLib. Reference the SharedLib project in the QuoteApi project.

Configure the web api to use the QuoteContext with SQLite database. Read the database connection string from appsettings section named ConnectionStrings with key QuoteDb. The connection string must point to a file named quotes.db. The QuotesController gets the QuoteContext via constructor injection. All endpoints uses the QuoteContext to access the database. Use async methods.

Configure the web api to allow cross-origin request from any url with any HTTP verb and any header.

1. The Frontend (i.e. the Blazor WASM app)

Create the frontend app to allow CRUD operations on the quotes. The only dependencies that can be used are HttpClient, ILogger and NavigationManager.

Use the QuoteDTO model class for the data from SharedLib project.

Create a component QuoteItem.razor in Frontend/Shared folder (like the SurveyPrompt.razor file). The QuoteItem component takes a QuoteDTO object as a parameter named Quote and renders the quote's data in a div element with css classes: well quote. Set some nice styles for the .quote css class in Frontend/wwwroot/css/app.css file to make the app shine.

The component also has optional parameters for edit and delete links named EditLink and DeleteLink. The values contains the urls to edit and delete pages. If values for the links are empty or null then nothing is rendered for the links. Use NavLink component to render the links.

Use the QuoteItem component to render the quotes on every page where the quotes are shown.

The app's main page (Index.razor) renders quotes from the QuoteApi's /quotes endpoint (the top 5 latest additions).

The app has a page (Login.razor) in url /login to allow the user to login (i.e. provide the user's name). The Login page allows the user to provide a username which is used with the QuotesApi. No actual logging in checks are needed. Only a input for the username. The login page probably should navigate to a url within the app with the provided user's name so that the urls has the name.

Use state within the frontend app so that the username is persisted in the URL (i.e. the username is part of the frontend app url like /quotes/michael to get all quotes saved by a user named Michael). It is only necessary that the pages in Quotes folder are able to handle the state (i.e. the user's name in the URL). User parameter named User for the user data and parameter named Id for the possible id data in the pages.

Create pages for the quotes CRUD operations. Group the pages in folder Pages/Quotes (i.e. create a Quotes folder inside Pages folder so that the quotes pages namespace will be Frontend.Pages.Quotes).

In the Quotes folder:

* + Create QuotesList.razor page (/quotes)
  + Create EditQuote.razor page (/quotes/edit)
  + Create AddQuote.razor page (/quotes/add)
  + Create DeleteQuote.razor page (/quotes/delete)

Remember to use the user's name and possible quote id with the URLs (and map the values to parameters User and Id). If username is not provided with the pages in Quotes folder then the user is redirected to the Login.razor page.

Use EditForm component with EditQuote and AddQuote pages with proper Input components for the three (3) data fields. Do not write input component for the id field. The pages has only one submit button to save the changes.

The DeleteQuote page shows the quote's information and has only one submit button to confirm the deletion.

In every page in Quotes folder print the user's name in h2 element.