Thank you for sending this application, it was very interesting to solve.

I will be happy to discuss the code in greater details however I thought of sharing some design

decision for the application and along with some additional notes.

For running the application, readme page ([Link](https://github.com/shantanusrivastv/Calculator/blob/master/README.md)) has the required instructions.

# Backend

## Calculator:

1. Since Non-negative integer inputs are not in scope hence decimal values in input expression are not considered.
2. Operator Precedence are included for future version since all our supported operators works fine using **LeftToRight** Precedence hence this is default setup.
3. Support for non-single digit number is implemented and covered in Unit Test Cases.
4. Spaces is allowed between values for example: 4 + 5 \* 2.
5. Thorough Validation is performed, some validation is using Regex.

Instead of creating one Regx to validate all rules individual regx is used for easier understanding and enhancements. Comparison is done in optimised way.

1. Rounding is not done from calculator side as we do not know how precise users want their calculation to be. Hence double type is used, and client can round it according to their need, like Unit Test Cases.

## Web API:

1. Postman collection is included as part of Git repository (Postman\_Collection.json).
2. The layering of application is inspired by **Onion Architecture**.
3. The dependencies in UI layer is kept at minimum, service dependencies are handled in Services Project part of Infrastructure.

It helps in Keeping UI light and loosely coupled with infrastructure and other non-UI related dependencies.

1. HTTPs are not implemented to avoid any certificate issues while running.
2. POST is used for accepting input expression for better security and to overcome limitation of query string
3. **Swagger**/ Open API specification is included for documentation.
4. A sample **PremiumCalculator** Endpoint is created for future use.
5. API versioning, Global Error handling and Logging will be part of future release with other requirements once finalised.

## Persistence /Database:

1. Although the required exercise does not require any backend, persistence layer is created using EF Core, code first approach .
2. One possible future use case is to limit no of request per day depending upon their subscription plan. The necessary setup has been created to support Premium, Standard and Free user.
3. Fully abstracted generic repository is implemented with basic database operation.

Currently only FindBy is used for validating credentials.

1. Instead of FindByID a very generic FindBy is implemented where we can inject our predicated for more complex scenarios than just finding by Primary Key.
2. EF Core fluent API is used and encouraged to avoid tying the Persistence specific information in Entity Classes.
3. Very generous db schema is used like nvarchar max, it needs to be more restrictive according to business requirements.
4. modelBuilder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly())

is used so that developer does not need to exmplcily register new Entity Configurations.

1. Authentication was not in scope however basic but very extensible JWT authentication is implemented backed by database.

(It still usage some very bad practise like storing password in plain text etc for simplicity.

1. Data is seeded for Simon and Stephanie seed with fictitious email address.
2. Registration module will be supported in future release.

# Frontend / React UI

1. Although optional a very basic client application is created using React.
2. Material-UI is used for more modern look and feel.
3. The application is using ReactHooks like useState, useEffect and useReducer

Hence no class-based component is used its all functional components as per recommended practise by React Team.

1. Client-side routing as well as state is implemented.
2. To ease use default credentials are populate for Premium user.

(password is shown in plain text)

1. No validation implemented at client side however the representation of success and failure is implemented.

Kind Regards,

Kumar Shantanu