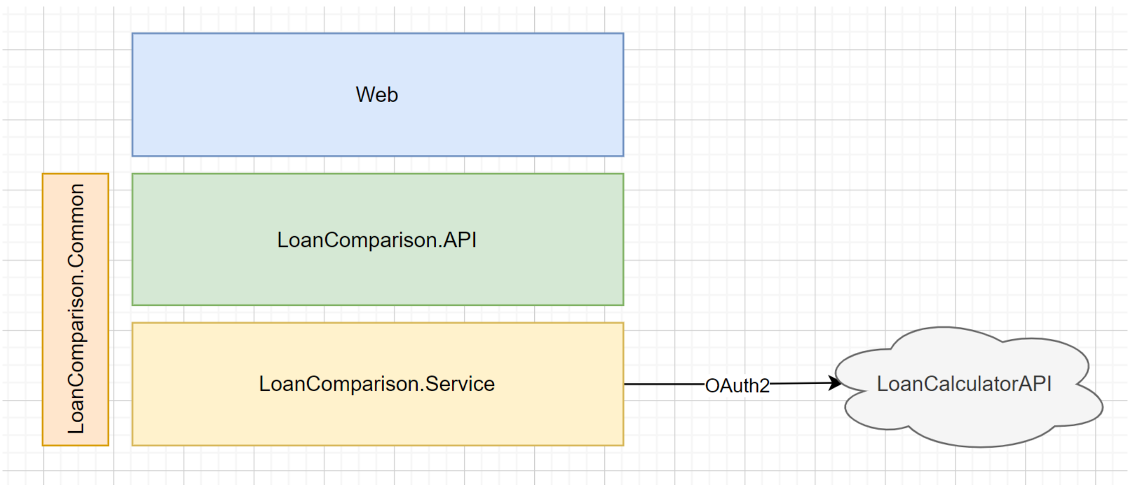
Contents

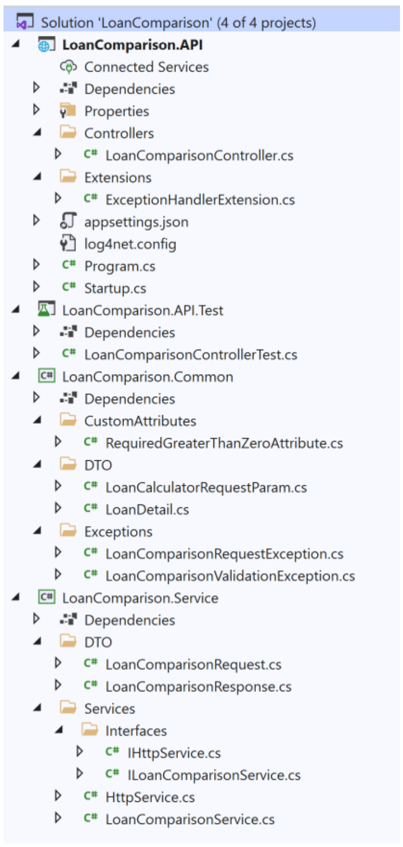
High Level Architecture



Backend (API)

A .NET Core (.NET 5) Web API application has been implemented as the back end. The application consists of 3 layers.

1. **LoanComparison.API**- Web API Layer. Contains the API endpoint that will be accessed by the front end.
2. **LoanComparison.Service**-Service Layer responsible for business logic. This layer will invoke the LoanCalculator external API (https://api.stage.tictoc.ai/product/v1.0/calculator/getloancomparison) to retrieve the Saving amount.
3. **LoanComparison.Common**- Common layer that contains the relevant classes shared by both Service and API layers.



```
Solution 'LoanComparison' (4 of 4 projects)
  LoanComparison.API
    Connected Services
    Dependencies
    Properties
    Controllers
      LoanComparisonController.cs
    Extensions
      ExceptionHandlerExtension.cs
    appsettings.json
    log4net.config
    Program.cs
    Startup.cs
  LoanComparison.API.Test
    Dependencies
    LoanComparisonControllerTest.cs
  LoanComparison.Common
    Dependencies
    CustomAttributes
      RequiredGreaterThanZeroAttribute.cs
    DTO
      LoanCalculatorRequestParam.cs
      LoanDetail.cs
    Exceptions
      LoanComparisonRequestException.cs
      LoanComparisonValidationException.cs
  LoanComparison.Service
    Dependencies
    DTO
      LoanComparisonRequest.cs
      LoanComparisonResponse.cs
    Services
      Interfaces
        IHttpService.cs
        ILoanComparisonService.cs
      HttpService.cs
      LoanComparisonService.cs
```

1. LoanComparison.API

- **Startup.cs :**
  In addition to the default configurations, following has been configured in the startup class.
  1. Register IdentityModel token service with token endpoint details for OAuth2 request, and HTTP client that uses the IdentityModel token management.
  2. CORS enabled.
  3. ILoanComparisonService dependency injection configured.
  4. Register LoanCalculatorRequestParam with the configuration which will be used in LoanComparisonRequest instance creation.
  5. Log4Net enabled for logging.
  6. Custom exception handler added to handle exceptions at the global level.

- **LoanComparisonController.cs :**
  Contains the GetSavingAmount Web API action method to retrieve savings amount for a given borrowing amount and customer current rate.

- **ExceptionHandlerExtension.cs :**
  1. Contains the extension method to IApplicationBuilder to configure exception handling at global level of the application.
  2. The method will log the exception details and sets relevant HTTP Status Code and message in the response.

- **appsettings.json :**
  1. OAuth section contains configuration details required for OAuth2 authentication which is required to access external loan calculator API to get the savings amount.
  2. LoanCalculatorApiBaseUrl is the base URL of external API.
  3. LoanCalculatorRequestParam section contains the predefined values to be set in the request object of external API request.

2. LoanComparison.Service

- **DTO folder:**
  Contains the DTO classes representing external loan calculator API request (LoanComparisonRequest) and response (LoanComparisonResponse).

- **Services folder:**
  Contains the service interfaces (within Interfaces folder) and the service classes.

- **LoanComparisonService.cs :**
  1. Contains the implementation of service method (GetSavingAmountAsync) which will invoke the external loan calculator API (using HttpSerivce class) with given borrowed

amount and customer current rate and returns the savings amount retrieved from the external API response.
2. If the response status code of the external API is not success, throws exception LoanComparisonRequestException with error details. It will be handled in by the custom exception handler configured in the Web API.

- **HttpService.cs:**
  1. Class that contains the methods to invoke http requests. Only a method to post a request (PostAsync) implemented for now. Other methods shoul be implemented when required.
  2. Obtaining an OAuth2 access token and setting the bearer token authorization header in http request are handled by the IdentityModel assembly that was registered in the Startup.cs of the Web API project.

3. LoanComparison.Common

- **DTO folder:**
  Contains the DTO classes shared by both API layer and Service layer.

- **CustomAttributes folder:**
  1. Contains custom validation attribute implementation that will validate whether the value is greater than 0 (RequiredGreaterThanZeroAttribute.cs).
  2. Used in LoanDetail.cs DTO class.

- **Exceptions folder:**
  1. Contains the custom exception classes used by the application.
  2. LoanComparisonRequestException- Exception that will be thrown when the External API returns an error code.
  3. LoanComparisonValidationException.cs- Exception that will be thrown when model validations fail.

4. LoanComparison.API.Test

1. Unit test project.
2. Contains unit test for GetSavingAmount action method which will mock ILoanComparisonService using Moq package.

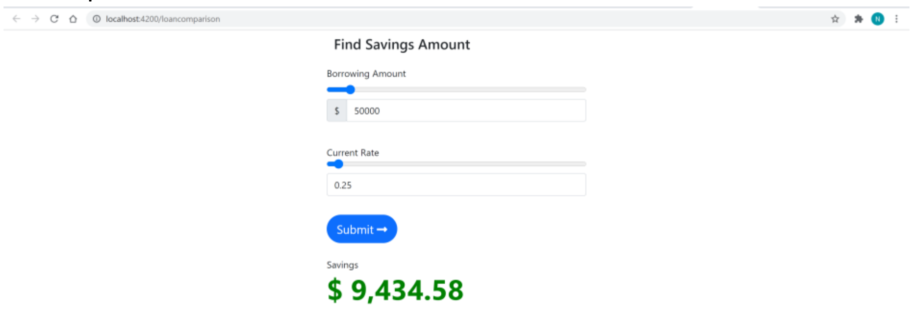
5. Handling OAuth2 flow for external API authentication

To handle the OAuth2 authentication flow to access external loan calculator API, (retrieve access token, persist until token expiry, retrieve new token if expired, and setting Bearer token on request Authorization header) application used IdentityModel library that was readily available.

Reference: https://identitymodel.readthedocs.io/en/latest/index.html

Front-end (Angular app)- LoanComparison.Web.Angular folder

1. Since I am more familiar with Angular than ReactJS, and it has stated that the UI could be in other framework, front end has been implemented using Angular because the timeline was limited to refresh ReactJS knowledge.
2. A responsive design is implemented using bootstrap grid framework.
   Desktop:



localhost:4200/loancomparison

Find Savings Amount

Borrowing Amount

$ 50000

Current Rate

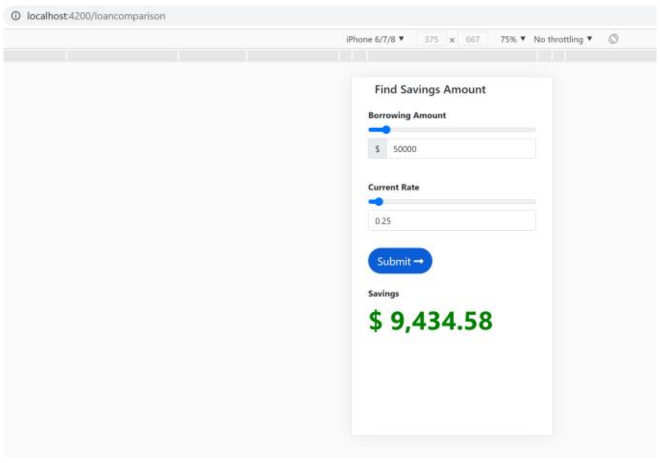
0.25

Submit ➞

Savings

$ 9,434.58

   iPhone view (captured from chrome developer tool device toggle option)



localhost:4200/loancomparison

iPhone 6/7/8 ▾ 375 × 667 75% ▾ No throttling ▾

Find Savings Amount

Borrowing Amount

$ 50000

Current Rate

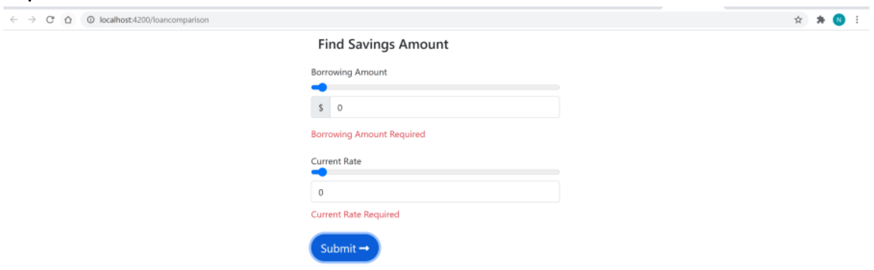
0.25

Submit ➞

Savings

$ 9,434.58

3. Inputs are validated on form submission.



localhost:4200/loancomparison

Find Savings Amount

Borrowing Amount

$ 0

Borrowing Amount Required

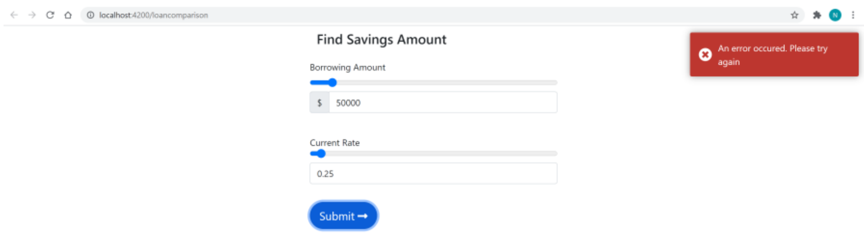
Current Rate

0

Current Rate Required

Submit ➞

4. Error toaster will be shown if the API response is unsuccessful and unable to retrieve saving amount.



localhost:4200/loancomparison

Find Savings Amount

An error occured. Please try again

Borrowing Amount

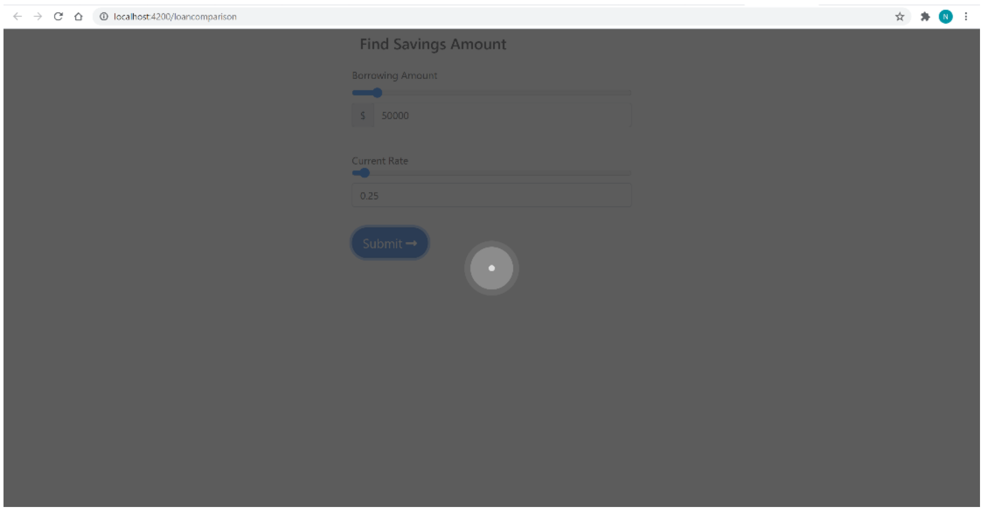
$ 50000

Current Rate

0.25

Submit ➞

5. A spinner is added to improve the user experience on the waiting time after submit button is clicked and receiving a response.



- **LoanComparisonComponent**

1. This is the component which contains html template to input borrowing amount and customer current rate and display savings amount.
2. The component will invoke backend API on submit if the form is valid and retrieve savings amount.

- **http-service.ts**

  HttpService class contains common service to invoke backend API using httpClient. Only a method to POST is implemented for now, GET and other methods need to implement here when required.

- **loan-detail-model.ts**

  LoanDetail class contains model definition which will be used for model binding of borrowing amount and customer rate in the component and passed to the back end API.

Front-end 2 (ReactJS app)- LoanComparison.Web.React folder

After completing the task (backend and Angular front-end), since I had some time remaining, I followed a ReactJS tutorial and implemented a ReactJS front end as well since it has stated ReactJS preferable for UI.

Assumptions

1. The API that the client application is accessing to get the saving amount is a public API which does not require any authentication to access since no authentication requirement was stated in the task document.