.NET Assignment- 2

Assignment: Minimal Chat Application (Backend)

Introduction

In this assignment, you will be building a minimal chat application backend using ASP.NET Core and Entity Framework. The goal is to create a set of APIs that allow users to register, authenticate, initiate conversations, send messages, retrieve message history, and apply sorting and paging mechanisms.

Tech Stack

- ASP.NET Core 6+
- EF Core 6+ (with code first approach)
- PostgreSQL or another preferred database (e.g. SQL Server Express/MySQL)

Requirements

Your task is to implement the following functionalities:

N-tier Architecture

- The idea of this section is to implement N-tier architecture within your .NET application. To do so, we need to ensure the following conditions are met:
    - The application should be built using an N-tier architecture, with separate layers for presentation, business logic, and data access
    - Presentation Layer:
        - Responsible for handling all user interface and browser communication logic.
        - Collects data from users and presents data to users.
        - Passes data to the business logic layer for further processing.
        - Should not be concerned about data access from database or modifying the data in any way.
    - Business Logic Layer:
        - Responsible for executing specific business rules associated with the request.
        - Works as a bridge between the presentation layer and the data access layer.
        - It contains all the business logic of the program.
        - Performs data transformation before passing data to the data access layer.
        - Should not be concerned about how data is stored or retrieved.
    - Data Access Layer:
        - Responsible for managing the physical storage and retrieval of data.
        - Separates data-access logic from business objects.
        - Receives requests from the business logic layer and builds queries to the database to handle these requests.
        - Should not be concerned about how data is presented or how data is processed.
        - The data access layer should be implemented using Entity Framework Core
        - Create a Generic Data Repository class that contains all the methods for data access

        - Inject this Data Repository class with the required data type wherever you need database access

User Registration

- The application should use ASP.NET Core Identity to manage users, passwords, profile data, tokens, etc.
- **API Endpoint**: **POST /api/register**
- **Request Parameters**:
    - **email** (string): User's email address (required) (unique)
    - **name** (string): User's full name (required)
    - **password** (string): User's password (required)
- **Response**:
    - **200 OK** - Registration successful
    - **400 Bad Request** - Registration failed due to validation errors
    - **409 Conflict** - Registration failed because the email is already registered
- **Response Body** (in case of success):
    - **userId** (int/guid): User's unique identifier
    - **name** (string): User's full name
    - **email** (string): User's email address
- **Response Body** (in case of failure):
    - **error** (string): Error message indicating the cause of the failure

Note: Password must not be stored in plaintext in database.

User Login

- **API Endpoint**: **POST /api/login**
- **Request Parameters**:
    - **email** (string): User's email address (required)
    - **password** (string): User's password (required)
- **Response**:
    - **200 OK** - Login successful
    - **400 Bad Request** - Login failed due to validation errors
    - **401 Unauthorized** - Login failed due to incorrect credentials
- **Response Body** (in case of success):
    - **token** (string): JWT token for authentication
    - **profile** (object): User profile details (e.g., **id**, **name**, **email**)
- **Response Body** (in case of failure):
    - **error** (string): Error message indicating the cause of the failure

Social Login

- The .NET application should allow users to log in using their Google account.
- The Social Login provider (Google) should be configured within ASP.NET Core Identity

Retrieve User List

- **API Endpoint**: **GET /api/users**
- **Response**:
  - **200 OK** - User list retrieved successfully
  - **401 Unauthorized** - Unauthorized access
- **Response Body** (in case of success):
  - **users** (array of objects):
    - **id** (int/guid): User's unique identifier
    - **name** (string): User's full name
    - **email** (string): User's email address
- **Response Body** (in case of failure):
  - **error** (string): Error message indicating the cause of the failure

Note: Retrieved list shouldn't contain the user who is calling the API.

Send Message

- **API Endpoint**: **POST /api/messages**
- **Request Parameters**:
  - **receiverId** (int/guid): ID of the receiver user (required)
  - **content** (string): Message content (required)
- **Request Headers**:
  - **Authorization** (string): Bearer token obtained from user login
- **Response**:
  - **200 OK** - Message sent successfully
  - **400 Bad Request** - Message sending failed due to validation errors
  - **401 Unauthorized** - Unauthorized access
- **Response Body** (in case of success):
  - **messageId** (int/guid): Message's unique identifier
  - **senderId** (int/guid): ID of the sender user
  - **receiverId** (int/guid): ID of the receiver user
  - **content** (string): Message content
  - **timestamp** (timestamp): Message timestamp
- **Response Body** (in case of failure):
  - **error** (string): Error message indicating the cause of the failure

Edit Message

- **API Endpoint**: **PUT /api/messages/{messageId}**
- **Request Parameters**:
  - **messageId** (int/guid): ID of the message to edit
  - **content** (string): Updated message content
- **Request Headers**:
  - **Authorization** (string): Bearer token obtained from user login
- **Response**:
  - **200 OK** - Message edited successfully
  - **400 Bad Request** - Message editing failed due to validation errors

    - **401 Unauthorized** - Unauthorized access
    - **404 Not Found** - Message not found
- **Response Body** (in case of failure):
    - **error** (string): Error message indicating the cause of the failure

Note: User should only be able to edit message sent by him and not of other users. If user attempts to do so, API should return 401.

Delete Message

- **API Endpoint**: **DELETE /api/messages/{messageId}**
- **Request Parameters**:
    - **messageId** (int/guid): ID of the message to delete (required)
- **Request Headers**:
    - **Authorization** (string): Bearer token obtained from user login
- **Response**:
    - **200 OK** - Message deleted successfully
    - **401 Unauthorized** - Unauthorized access
    - **404 Not Found** - Message not found
- **Response Body** (in case of failure):
    - **error** (string): Error message indicating the cause of the failure

Note: User should only be able to delete messages sent by him and not of other users. If user attempts to do so, API should return 401.

Retrieve Conversation History

- **API Endpoint**: **GET /api/messages**
- **Request Parameters**:
    - **userId** (int/guid): ID of the user to retrieve the conversation with (required)
    - **before** (timestamp): All messages before this timestamp should be returned from API (optional) (default: Current Timestamp)
    - **count** (number): number of messages to be retrieved (optional) (default: 20)
    - **sort** (string): Sorting mechanism ("asc" or "desc") based on timestamp (optional, default order: asc)
- **Request Headers**:
    - **Authorization** (string): Bearer token obtained from user login
- **Response**:
    - **200 OK** - Conversation history retrieved successfully
    - **400 Bad Request** - Invalid request parameters
    - **401 Unauthorized** - Unauthorized access
    - **404 Not Found** - User or conversation not found
- **Response Body** (in case of success):
    - **messages** (array of objects):
        - **id** (int/guid): Message's unique identifier
        - **senderId** (int/guid): ID of the sender user
        - **receiverId** (int/guid): ID of the receiver user

        - **content** (string): Message content
        - **timestamp** (timestamp): Message timestamp
- **Response Body** (in case of failure):
    - **error** (string): Error message indicating the cause of the failure

Request-Logging Middleware

- Create a custom middleware that logs all the API requests with details like, IP of caller, request body, time of call, username
    - Fetch username from auth token
    - Keep blank if no auth token
- Create an API to fetch logs
    - **Endpoint: GET /api/log**
    - **Request Parameters:**
        - EndTime (timestamp): Logs before this timestamp will be returned. (optional) (default: Current Timestamp)
        - StartTime (timestamp): Logs after this timestamp will be returned. (optional) (default: Current Timestamp – 5 minutes)
    - **Request Headers**:
        - **Authorization** (string): Bearer token obtained from user login
    - **Response**:
        - **200 OK** – Log list received successfully
        - **400 Bad Request** - Invalid request parameters
        - **401 Unauthorized** - Unauthorized access
        - **404 Not Found** – No logs found
    - **Response Body** (in case of success):
        - **Logs** (array of objects)
    - **Response Body** (in case of failure):
        - **error** (string): Error message indicating the cause of the failure

Real-time messaging

- Create a SignalR hub to allow real time communication via messages with the frontend
- Whenever the Send Message API is used, the message sent, should be pushed to the receiver via WebSockets in real time
- The system should be able to handle multiple connections from different users at the same time (User 1 chatting with User 2, User 3 chatting with User 4)
- Messages should be received by the intended recipient only (Messages from User 1 should be received by User 2 only based on the above)
- Only Authorized Users may send and receive messages

Search Conversations

- User should be able to search within his conversations (where he is sender/receiver of the message) to find the messages that contains provided keywords in API "query" parameter
- **API Endpoint:** GET api/conversation/search
    - **Query String Parameters:**

        - query (string): The string that will be used to search in the database of conversations for a message
    - **Request Headers**:
        - **Authorization** (string): Bearer token obtained from user login
    - **Response**:
        - **200 OK** – Log list received successfully
        - **400 Bad Request** - Invalid request parameters
        - **401 Unauthorized** - Unauthorized access
    - **Response Body** (in case of success):
        - **messages** (array of objects):
            - **id** (int/guid): Message's unique identifier
            - **senderId** (int/guid): ID of the sender user
            - **receiverId** (int/guid): ID of the receiver user
            - **content** (string): Message content
            - **timestamp** (timestamp): Message timestamp
    - **Response Body** (in case of failure):
        - **error** (string): Error message indicating the cause of the failure

Scoring

- EF models, PK, FK, Relationship - 5
- N-tier Architecture – 10
- User Registration - 10
- User Login – 5
- Social Login – 5
- Retrieve User List - 5
- Send Message - 5
- Edit Message – 5
- Delete Message – 5
- Retrieve Conversation History – 10
- Request Logging Middleware – 10
- Real-time Messaging – 15
- Search Conversations – 10

Timeline

You have a timeline of **3 days** to complete this.

Ground Rules

- Code must be pushed to GitHub repository before leaving for the day.
- EF models must have proper primary key, foreign key and relationships defined.
- EF must use async methods wherever possible.
- Make sure to follow all points mentioned in the Requirements section. Scoring will be based on adherence of all conditions mentioned in requirements.

Submission Guidelines

- Share link of public GitHub Repository with Reporting Person

- Create a short screen recording of the features as mentioned in the document and share it with the Reporting Person

Good Luck!