C# API Client using RestSharp

Market.io uses a popular API Client library called *RestSharp*, it brings in the traditional C# HTTP Library, but adds many features and simplifications.

Setup:

**ApiClientService*.cs can be found in the '*Common*' C# class library

To be able to access inside of the Blazor application, the following must be added to the Builder:

```
// Add services to the container.
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddSingleton(new RestClient(new HttpClient())); // added
builder.Services.AddSingleton<ApiClientService>(); // added
```

ApiClientService Class:

The constant variable '**DEFAULT_ENDPOINT'** is the only value that should be changed when configuring your Web API's URL. (At the time of the image, I am using my Web API that is hosted on IIS without HTTPS)

The default constructor is the only one being used inside of the Blazor app, but if there is a case where you need to reach another address, the overloaded constructor can be used.

```
private RestClient _client;
private const string DEFAULT_ENDPOINT = "http://172.18.31.108:7244/";

#region constructors

// default constructor
7 references
public ApiClientService()
{
    var httpClient = new HttpClient();
    httpClient.BaseAddress = new Uri(DEFAULT_ENDPOINT);

    _client = new RestClient(httpClient);
}

// overload that takes endpoint url as parameter
0 references
public ApiClientService(string endpointUrl)
{
    var httpClient = new HttpClient();
    httpClient.BaseAddress = new Uri(endpointUrl);

    _client = new RestClient(httpClient);
}

#endregion
```

As you can see in the image, a regular C# HTTP class is being initialized then being passed into the 'RestClient' (which is RestSharp's HTTP Client).

This is because RestSharp is acting as a superset for the traditional C# HTTP Client

Login / Register HTTP Handlers:

The **ApiClientService** class has 2 methods that access the Login and Register API endpoints inside of the Web API's **AuthController.cs** class (more can be this auth controller in the 'jwt-auth-system' documentation)

Log User In

```
/// <summary>
///
/// If the user successfully logs in,
/// the HTTP response will return a UserSessionDTO object
/// as JSON that includes a valid JWT token.
///
/// </summary>
2 references
public async Task<RestResponse> LogUserInAsync(LoginDTO u)
{
    // hash password
    var loginDTO = new LoginDTO
    {
        Email = u.Email,
        PasswordHash = SHA256.Hash(u.PasswordHash)
    };

    // send request to api
    var request = new RestRequest("api/Auth/login").AddJsonBody(loginDTO);

    var response = await _client.PostAsync(request);

    return response;
}
```

This method reaches the Login POST method in the API controller then will return a session token if the user provides valid credentials.

An example of where this is used is in 'LoginForm.razor'.

It has a form that takes an email and password then converts to a loginDTO.

Then the loginDTO is serialized as JSON then sent to the API.

Register User

This method will create a RegisterDTO and post it to the 'Auth/register' endpoint.

A new RegisterDTO object is being created because referencing the same object used in the Blazor form can cause conflicts, so it is isolated.

An example of where it is used is in 'RegisterForm.razor' inside of the Blazor application.

When the user is added to the database, this method returns a 200 status code.

```
/// <summary>
///
/// Send 'RegisterDTO' object to api,
/// if successful, the api will return add user to database
///
/// </summary>
1 reference
public async Task<RestResponse> RegisterUserAsync(RegisterDTO regDTO)
{
    var newUser = new RegisterDTO()
    {
        Email = regDTO.Email,
        FirstName = regDTO.FirstName,
        LastName = regDTO.LastName,
        PasswordHash = SHA256.Hash(regDTO.PasswordHash),
        ConfirmPasswordHash = SHA256.Hash(regDTO.ConfirmPasswordHash),
        RegisterDate = DateTime.Now,
    };

    var request = new RestRequest("api/Auth/register").AddJsonBody(newUser);

    var response = await _client.PostAsync(request);

    return response;
}
```

The status code notifies the client app what to do from there, if successful it will automatically log the user in using the credentials of the newly registered user. (Using the 'LogUserIn' method).

Generic POST

In the ApiClientService class, there is a generic method that handles all requests that are inheriting the IDbModel interface.

The IDbModel interface is used by all models inside the Common/Models folder. It simply implements 'Id' property.

```
7 references
public async Task<RestResponse> PostAsync<T>(IDbModel model, string? controllerName = null) where T : IDbModel
{
    var modelAsJson = JsonConvert.SerializeObject(model);
    RestResponse response;
    string endpoint = controllerName ?? typeof(T).Name + "s";

    var request = new RestRequest($"api/{endpoint}").AddStringBody(modelAsJson, ContentType.Json);
    //var request = new RestRequest($"api/{endpoint}").AddJsonBody(model);

    try
    {
        response = await _client.PostAsync(request);
    }
    catch (HttpRequestException)
    {
        throw;
    }

    return response;
}
```

This method can take in any model that inherits from 'IDbModel' (User, Listing, Transaction, ListingImage…)

Optionally, you can pass in the controller's name, otherwise that is handled for you by taking the model's name and pluralizing it (ex. Model: User, Controller: UsersController)

If your controller's name can't be handled by just appending an 's' to the end (ex. Category -> Categories),

That puts you in the situation where you would have to enter the 'controllerName' parameter in the method.

This also applies to the GET, PUT, and DELETE methods below.

Examples of use cases for this method:

```
if (editContext != null && editContext.Validate())
{
    ApiClient = new ApiClientService();
    var response = await ApiClient.PostAsync<ListingCategory>(_category, "ListingCategories");

    if (response.IsSuccessStatusCode)
    {
        toastService.ShowSuccess("Category created successfully!");
    }
```

```
var message = new Message
{
    Content = _messageText.Trim(),
    TransactionId = _transaction.Id,
    Transaction = _transaction,
    SenderId = _userSession.UserId,
    Sender = currentUser,
    ReceiverId = _transaction.SellerId,
    Receiver = receiver,
    DateSent = DateTime.UtcNow,
    ReadByReceiver = false,
};

await ApiClient.PostAsync<Message>(message);
```

Generic GET

To use this method, you must enter the type of model you are trying to get.

For example, GetAsync<User> will return a list of all users in the database.

This method returns the list of the model you passed in as the generic type instead of an HTTP response.

```
/// <summary>
///
/// This method will get all items from a table
///
/// </summary>
9 references
public async Task<List<T>> GetAsync<T>(string? controllerName = null) where T : IDbModel
{
    List<T>? ret = new List<T>();

    string endpoint = controllerName ?? typeof(T).Name + "s";

    var request = new RestRequest($"api/{endpoint}");

    var response = await _client.GetAsync(request);
    if (response.StatusCode == HttpStatusCode.OK)
    {
        ret = JsonConvert.DeserializeObject<List<T>>(response.Content);
    }

    return ret;
}
```

Generic GET by ID

Similar to the method above, but this one is return a single Model instead of the list.

The only parameters it needs is the generic type and ID of the record in the database.

```
14 references
public async Task<T> GetByIdAsync<T>(int id) where T : IDbModel
{
    var request = new RestRequest($"api/{typeof(T).Name}s/{id}");

    var response = await _client.GetAsync(request);
    if (response.StatusCode == HttpStatusCode.OK)
    {
        return JsonConvert.DeserializeObject<T>(response.Content);
    }

    return default;
}
```

Get Messages by Transaction ID

```
2 references
public async Task<List<Message>> GetMessagesByTransactionId(int transactionId)
{
    var request = new RestRequest($"api/Messages");
    var response = await _client.GetAsync(request);

    List<Message> messages = JsonConvert.DeserializeObject<List<Message>>(response.Content)

    return messages.Where(m => m.TransactionId == transactionId).ToList();
}
```

This method was made because each message relies on the TransactionID (FK) field and not the Id (PK) field when trying to show the messages that belong to the client user.

Get User Ratings by User ID

```
2 references
public async Task<List<UserRating>> GetUserRatingByUserId(int userId)
{
    var request = new RestRequest($"api/UserRatings");
    var response = await _client.GetAsync(request);
    List<UserRating> userRatings = JsonConvert.DeserializeObject<List<UserRating>>(response.Content)
    return userRatings.Where(m => m.UserId == userId).ToList();
}
```

Similar to the method above, this one also relies on the UserId to show the ratings for a specific user.

Generic PUT

```
/// <summary>
///
/// Generic method that is able to update and IDbModel object
///
/// </summary>
3 references
public async Task<RestResponse> PutAsync<T>(T model, string? controllerName = null) where T : IDbModel
{
    var modelAsJson = JsonConvert.SerializeObject(model);


    string endpoint = controllerName ?? typeof(T).Name + "s";
    var request = new RestRequest("api/"+endpoint+$"/{model.Id}").AddStringBody(modelAsJson, ContentType.Json);
    RestResponse response;
    try
    {
        response = await _client.ExecutePutAsync(request);
    }
    catch (Exception)
    {
        throw;
    }

    return response;
}
```

This method can update any object that inherits from 'IDBModel'.

If the update is successful, the API will return the updated object from the database.

Here is an example of where it is used with the Blazor application:

```
var t = Transaction;
t.Completed = true;
t.DateCompleted = DateTime.UtcNow;
var res = await ApiClient.PutAsync<Transaction>(t);

if (res.IsSuccessStatusCode)
{
    ShowModal();
    toastService.ShowInfo($"You marked '{Transaction.Listing.Title}' as completed.");
}
```

Marking the Transaction as ‘Complete’ when the user clicks the ‘Mark as Complete’ button.

Generic DELETE

```
/// <summary>
///
/// Deletes the item with the given id
///
/// </summary>
0 references
public async Task<RestResponse> DeleteAsync<T>(int id, string? controllerName = null) where T : IDbModel
{
    string endpoint = controllerName ?? typeof(T).Name + "s";
    var request = new RestRequest($"api/{endpoint}/{id}");
    return await _client.DeleteAsync(request);
}
```

This method will delete the object that is stored in the database using the ‘ID’ parameter that is passed in.