LABORATORY ASSIGNMENT №6

Development of the client-side applications using Blazor WebAssembly

Goal: develop practical skills of building and integrating a client-side application with a RESTful Web API using Blazor WebAssembly.

Workflow:

Task 1. Create a Blazor WebAssembly project in your solution that will interact with WebAPI. Implement pages and necessary components for performing CRUD-operations with existing entities. Add validation for created forms.

Task 2. Add authentication and authorization to your client application. Add pages and forms for registration and login. Implement JWT-based authentication.

I performed two tasks in parallel, it was more convenient for me.

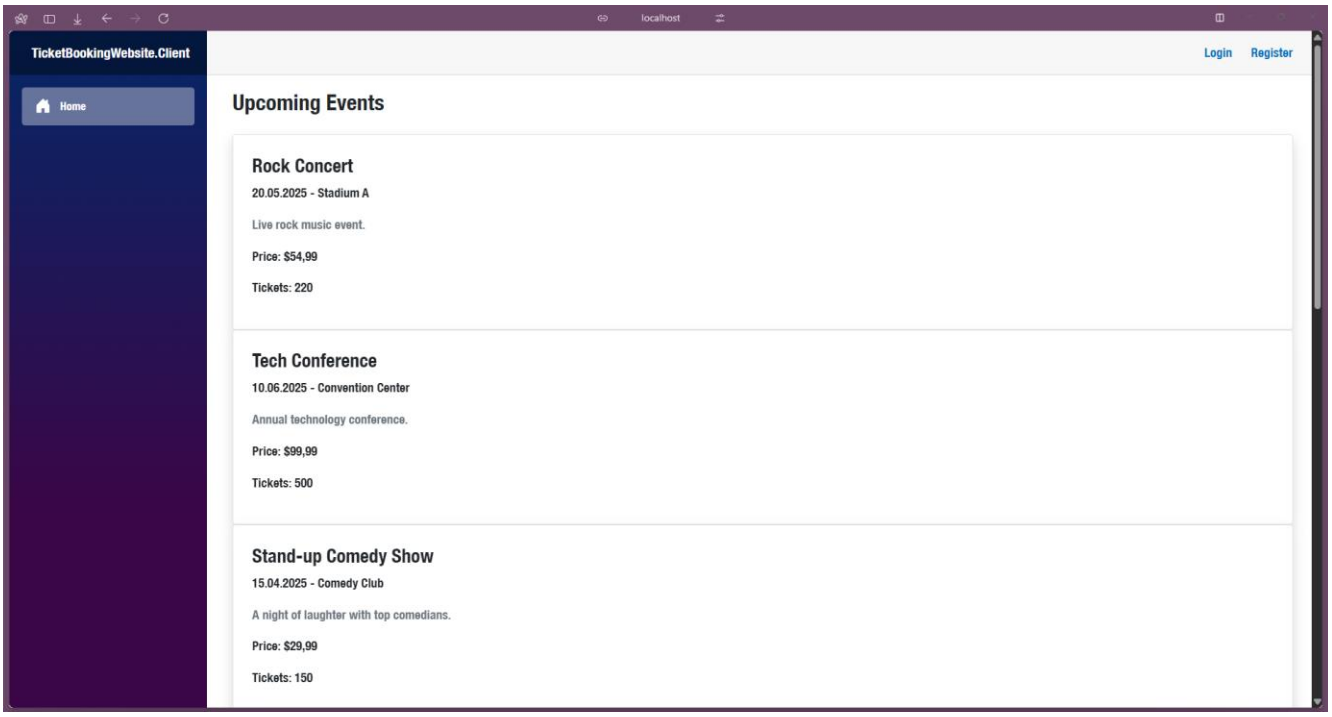
First, I've created homepage so unauthorized user can see events. Implemented EventService, which uses API:

Homepage:

```
@page "/"
@inject EventService EventService
@using TicketBookingWebsite.Models
@using TicketBookingWebsite.Client.Services

<h3 class="text-xl font-semibold mb-4">Upcoming Events</h3>

@if (events == null)
{
    <p>Loading events...</p>
}
else if (!events.Any())
{
    <p>No events found.</p>
}
else
{
    <div class="grid gap-4 md:grid-cols-2 lg:grid-cols-3">
        @foreach (var e in events)
        {
            <div class="card p-4 border rounded shadow">
                <h4 class="font-bold">@e.Name</h4>
                <p>@e.Date.ToShortDateString() - @e.Location</p>
                <p class="text-sm text-muted">@e.Description</p>
                <p><strong>Price:</strong> $@e.Price</p>
                <p><strong>Tickets:</strong> @e.AvailableTickets</p>
            </div>
        }
    </div>
}

@code {
    private List<Event>? events;

    protected override async Task OnInitializedAsync()
    {
        events = await EventService.GetAllAsync();
    }
}
```

EventService:

```
public class EventService
{
    private readonly HttpClient _http;
    private readonly ILocalStorageService _localStorage;

    public EventService(HttpClient http, ILocalStorageService localStorage)
    {
        _http = http;
        _localStorage = localStorage;
    }

    public async Task<List<Event>> GetAllAsync()
    {
        var token = await _localStorage.GetItemAsync<string>("authToken");
        if (!string.IsNullOrWhiteSpace(token))
        {
            _http.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", token);
        }

        var response = await _http.GetFromJsonAsync<JsonElement>("api/events");

        if (response.TryGetProperty("$values", out var values))
        {
            try
            {
                var eventsList =
JsonSerializer.Deserialize<List<Event>>(values.GetRawText(), new JsonSerializerOptions
                {
                    PropertyNameCaseInsensitive = true
                });

                return eventsList ?? new List<Event>();
            }
            catch (JsonException ex)
            {
                Console.WriteLine($"Deserialization error: {ex.Message}");
            }
        }

        return new List<Event>();
    }

    public async Task<Event?> GetByIdAsync(int id)
    {
        return await _http.GetFromJsonAsync<Event>($"api/events/{id}");
    }

    public async Task<bool> CreateAsync(Event ev)
    {
        var response = await _http.PostAsJsonAsync("api/events", ev);
        return response.IsSuccessStatusCode;
    }

    public async Task<bool> UpdateAsync(Event ev)
    {
        var response = await _http.PutAsJsonAsync($"api/events/{ev.Id}", ev);
        return response.IsSuccessStatusCode;
    }

    public async Task<bool> DeleteAsync(int id)
    {
        var response = await _http.DeleteAsync($"api/events/{id}");
        return response.IsSuccessStatusCode;
    }
}
```



Upcoming Events

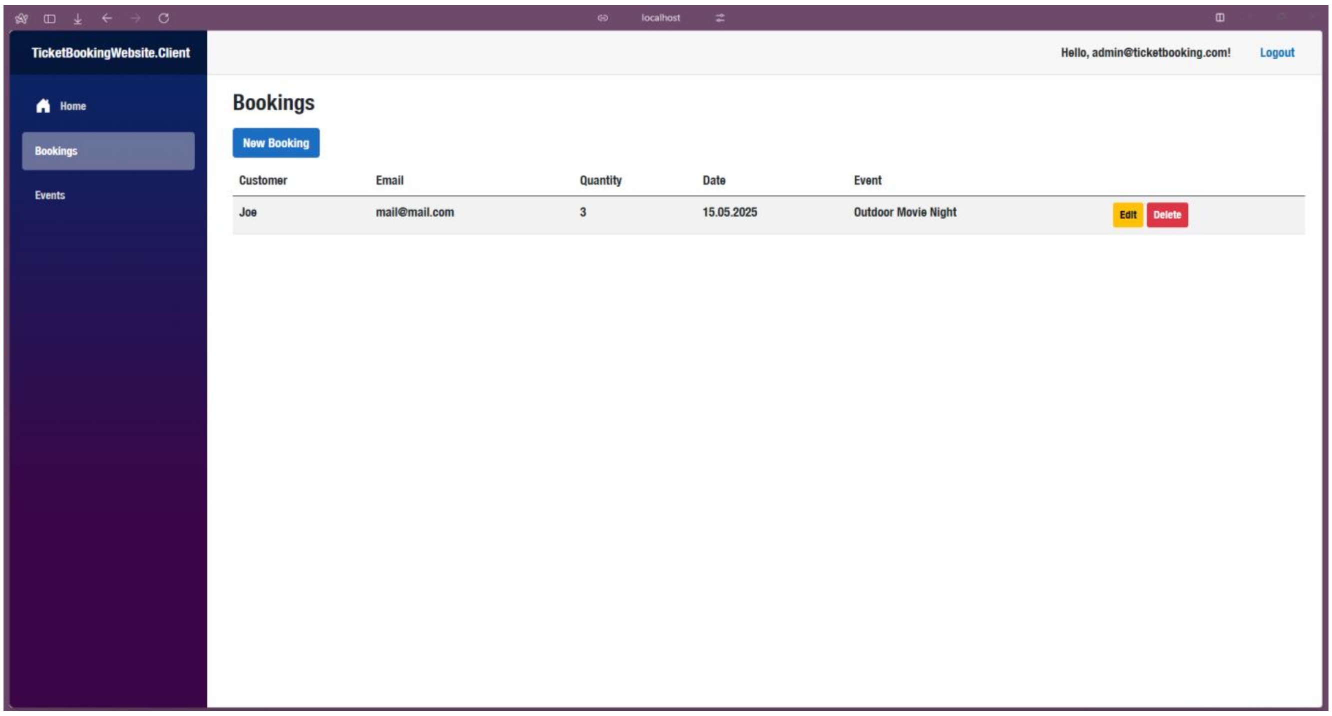
Rock Concert
20.05.2025 - Stadium A
Live rock music event.
Price: $54,99
Tickets: 220

Tech Conference
10.06.2025 - Convention Center
Annual technology conference.
Price: $99,99
Tickets: 500

Stand-up Comedy Show
15.04.2025 - Comedy Club
A night of laughter with top comedians.
Price: $29,99
Tickets: 150

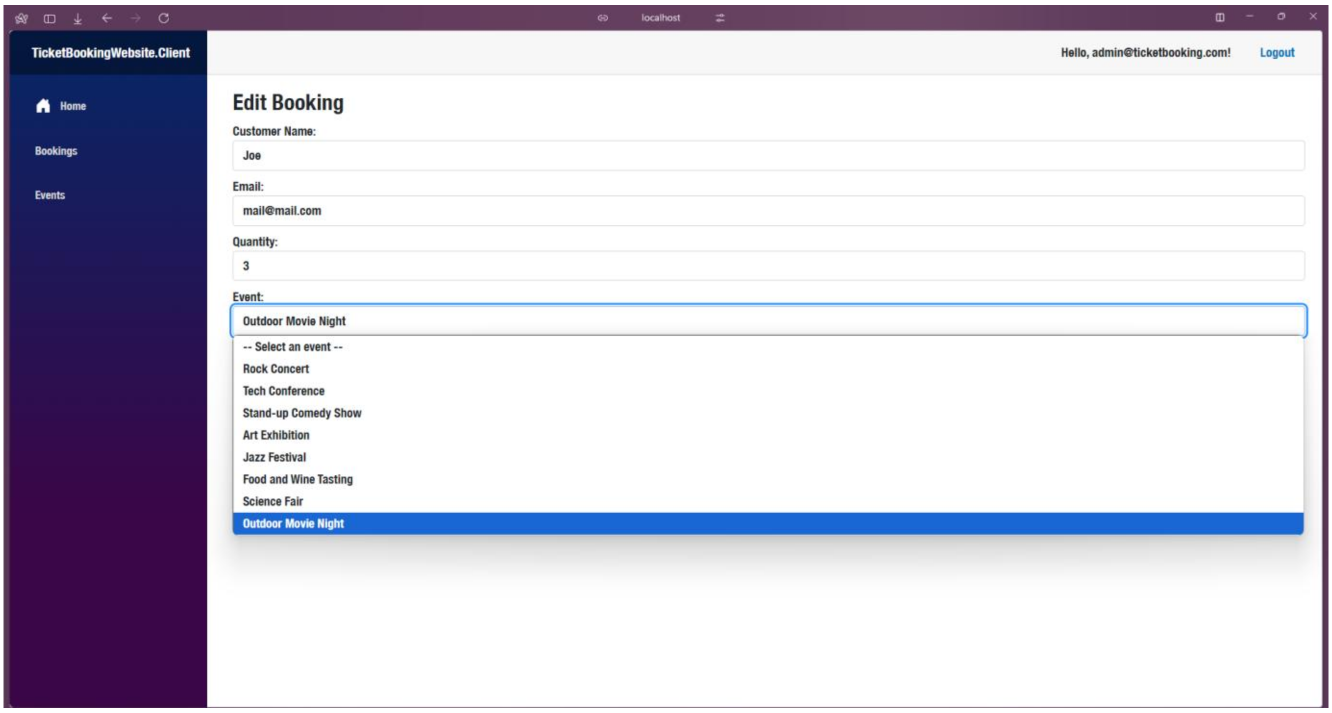
		Бейлах С.В.			*ДУ «Житомирська політехніка».23.121.01.000 – Лр6*	*Арк.*
		Українець М.О.				3
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		

BookingService is mostly the same, code is available on GitHub. Then I've proceed to making CRUD operations with authorization. CRUD was mostly copy and pasting operations to fetch enpoints and displaying it so I don't think that it makes sense to paste all code, it is available on GitHub. Also I want to mention that CRUD operations is only available for authorized users(I haven't implemented rights by roles cause it was not mentioned in the task). Input validation was automatically made with DataAnnotations.



Bookings

Customer	Email	Quantity	Date	Event
Joe	mail@mail.com	3	15.05.2025	Outdoor Movie Night



Edit Booking

Customer Name: Joe

Email: mail@mail.com

Quantity: 3

Event: Outdoor Movie Night

- -- Select an event --
- Rock Concert
- Tech Conference
- Stand-up Comedy Show
- Art Exhibition
- Jazz Festival
- Food and Wine Tasting
- Science Fair
- Outdoor Movie Night

Let’s proceed to authorization, here is how it works:

I have AuthStateProvider, which determines whether user is logged into account or not:

```
public class JwtAuthenticationStateProvider : AuthenticationStateProvider
{
    private readonly ILocalStorageService _localStorage;
    private readonly HttpClient _http;

    public JwtAuthenticationStateProvider(ILocalStorageService localStorage, HttpClient 
http)
    {
        _localStorage = localStorage;
        _http = http;
    }

    public override async Task<AuthenticationState> GetAuthenticationStateAsync()
    {
        var token = await _localStorage.GetItemAsync<string>("authToken");

        var identity = string.IsNullOrWhiteSpace(token)
            ? new ClaimsIdentity()
            : new ClaimsIdentity(ParseClaimsFromJwt(token), "jwt");

        if (!string.IsNullOrWhiteSpace(token))
            _http.DefaultRequestHeaders.Authorization = new 
AuthenticationHeaderValue("Bearer", token);

        var user = new ClaimsPrincipal(identity);
        return new AuthenticationState(user);
    }

    public void NotifyUserAuthentication(string token)
    {
        var identity = new ClaimsIdentity(ParseClaimsFromJwt(token), "jwt");
        var user = new ClaimsPrincipal(identity);
        NotifyAuthenticationStateChanged(Task.FromResult(new AuthenticationState(user)));
    }

    public void NotifyUserLogout()
    {
        var anonymousUser = new ClaimsPrincipal(new ClaimsIdentity());
        NotifyAuthenticationStateChanged(Task.FromResult(new 
AuthenticationState(anonymousUser)));
    }

    private IEnumerable<Claim> ParseClaimsFromJwt(string jwt)
    {
        var claims = new List<Claim>();
        var payload = jwt.Split('.')[1];
        var jsonBytes = Convert.FromBase64String(PadBase64(payload));
        var keyValuePairs = JsonSerializer.Deserialize<Dictionary<string, 
object>>(jsonBytes);

        if (keyValuePairs != null)
        {
            claims.AddRange(keyValuePairs.Select(kvp => new Claim(kvp.Key, 
kvp.Value.ToString())));
```

```
        }

        return claims;
    }

    private string PadBase64(string base64)
    {
        return base64.PadRight(base64.Length + (4 - base64.Length % 4) % 4, '=');
    }
}
```

Logging is implemented by fetching the api and saving JWT to localstorage:

```
@page "/login"
@inject HttpClient Http
@inject NavigationManager Nav
@inject Blazored.LocalStorage.ILocalStorageService LocalStorage
@inject JwtAuthenticationStateProvider AuthStateProvider

<h3>Login</h3>

<EditForm Model="loginModel" OnValidSubmit="HandleLogin">
    <InputText @bind-Value="loginModel.Username" placeholder="Username" />
    <InputText @bind-Value="loginModel.Password" type="password" placeholder="Password"
/>
    <button type="submit">Login</button>
</EditForm>

@code {
    public class LoginModel
    {
        public string Username { get; set; } = string.Empty;
        public string Password { get; set; } = string.Empty;
    }

    private LoginModel loginModel = new();

    private async Task HandleLogin()
    {
        var response = await Http.PostAsJsonAsync("api/auth/login", loginModel);
        if (response.IsSuccessStatusCode)
        {
            var result = await response.Content.ReadFromJsonAsync<TokenResponse>();
            await LocalStorage.SetItemAsync("authToken", $"{result!.Token}");
            Http.DefaultRequestHeaders.Authorization = new
System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", result.Token);
            AuthStateProvider.NotifyUserAuthentication(result.Token);
            Nav.NavigateTo("/");

        }
    }

    public class TokenResponse
    {
        public string Token { get; set; }
    }
}
```

Other things just use this token to get data from api endpoints protected by authorization.

Also I’ve implemented redirect to login page if user is not logged in and wants to perform only logged users operations:

```
<AuthorizeRouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)">
     <NotAuthorized>
         <RedirectToLogin />
     </NotAuthorized>
 </AuthorizeRouteView>
```

Conslusion: I’ve developed practical skills of building and integrating a client-side application with a RESTful Web API using Blazor WebAssembly.