

**BlazorHotelBooking/Client/Auth/APIAuthProvider.cs**

```
using Blazored.LocalStorage;
using Microsoft.AspNetCore.Components.Authorization;
using System.Net.Http.Headers;
using System.Security.Claims;
using System.Text.Json;

namespace BlazorHotelBooking.Client.Auth
{
    public class APIAuthProvider : AuthenticationStateProvider
    {
        private readonly HttpClient _httpClient;
        private readonly ILocalStorageService _localStorage;

        public APIAuthProvider(HttpClient httpClient,
            ILocalStorageService localStorage)
        {
            _httpClient = httpClient;
            _localStorage = localStorage;
        }

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

            if (string.IsNullOrEmpty(userToken))
            {
                return new AuthenticationState(new ClaimsPrincipal(new
                    ClaimsIdentity()));
            }

            _httpClient.DefaultRequestHeaders.Authorization = new
                AuthenticationHeaderValue("bearer", userToken);

            return new AuthenticationState(new ClaimsPrincipal(new
                ClaimsIdentity(ParseClaimsFromJwt(userToken), "jwt")));
        }

        public void MarkUserAsAuthenticated(string token)
        {
            ClaimsPrincipal authenticatedUser = new ClaimsPrincipal(new
                ClaimsIdentity(ParseClaimsFromJwt(token), "jwt"));
            Task<AuthenticationState> authState = Task.FromResult(new
                AuthenticationState(authenticatedUser));
            NotifyAuthenticationStateChanged(authState);
        }

        public void MarkUserAsLoggedOut()
        {
        }
    }
}
```

```
{
    ClaimsPrincipal anonymousUser = new ClaimsPrincipal(new
ClaimsIdentity());
    Task<AuthenticationState> authState = Task.FromResult(new
AuthenticationState(anonymousUser));
    NotifyAuthenticationStateChanged(authState);
}

private IEnumerable<Claim> ParseClaimsFromJwt(string jwt)
{
    List<Claim> claims = new List<Claim>();
    string payload = jwt.Split('.')[1];
    byte[] jsonBytes = ParseBase64WithoutPadding(payload);
    Dictionary<string, object>? keyValuePairs =
JsonSerializer.Deserialize<Dictionary<string, object>>(jsonBytes);

    keyValuePairs!.TryGetValue(ClaimTypes.Role, out object roles);

    if (roles != null)
    {
        if (roles.ToString().Trim().StartsWith("[")
        {
            string[]? parsedRoles =
JsonSerializer.Deserialize<string[]>(roles.ToString());

            foreach (string parsedRole in parsedRoles!)
            {
                claims.Add(new Claim(ClaimTypes.Role, parsedRole));
            }
        }
        else
        {
            claims.Add(new Claim(ClaimTypes.Role,
roles.ToString(!)));
        }

        keyValuePairs.Remove(ClaimTypes.Role);
    }

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

    return claims;
}

private byte[] ParseBase64WithoutPadding(string base64)
{
    switch (base64.Length % 4)
    {
        case 2: base64 += "=="; break;
        case 3: base64 += "="; break;
    }

    return Convert.FromBase64String(base64);
}
```

```
    }  
  }  
}
```

### BlazorHotelBooking/Client/Pages/Admin.razor

```
@page "/admin"  
@using BlazorHotelBooking.Shared  
@using Microsoft.AspNetCore.Authorization  
@inject HttpClient http  
@inject NavigationManager NavigationManager  
@attribute [Authorize(Roles = "Admin")]  
  
<h3>Admin Page</h3>  
  
<button @onclick="ViewPayments" class="btn btn-success">View All  
Payments</button>  
<button @onclick="ViewBookings" class="btn btn-success">View All  
Bookings</button>  
  
<br />  
<br />  
  
<button @onclick="AddHotel" class="btn btn-primary">Add Hotel</button>  
<br />  
  
@if (hotels is null)  
{  
    <span>Loading Hotels...</span>  
}  
else  
{  
    <h5>Hotels</h5>  
  
    <table class="table">  
        <thead>  
            <tr>  
                <th>ID</th>  
                <th>Name</th>  
                <th>Single Bed</th>  
                <th>Double Bed</th>  
                <th>Family Room</th>  
            </tr>  
  
        </thead>  
        <tbody>  
            @foreach (var h in hotels)  
            {  
                <tr>  
                    <td width="5%">@h.Id</td>  
                    <td width="5%">@h.Name</td>
```

```

        <td width="5%">@h.SBPrice</td>
        <td width="5%">@h.DBPrice</td>
        <td width="5%">@h.FamPrice</td>
        <td width="5%">
            <button @onclick="(() => EditHotel(h.Id))"
class="btn btn-primary">Edit</button>
        </td>
    </tr>
</tbody>
</table>
}

<br/>
<button @onclick="AddTour" class="btn btn-primary">Add Tour</button>
<br />
@if (tours is null)
{
    <span>Loading Tours...</span>
}
else
{
    <h5>Tours</h5>
    <table class="table">
        <thead>
            <tr>
                <th>ID</th>
                <th>Name</th>
                <th>Price</th>
                <th>Duration</th>
                <th>Max Num Of Guests</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var t in tours)
            {
                <tr>
                    <td width="5%">@t.Id</td>
                    <td width="5%">@t.Name</td>
                    <td width="5%">@t.Cost</td>
                    <td width="5%">@t.DurationInDays</td>
                    <td width="5%">@t.MaxNumberOfGuests</td>
                    <td width="5%">
                        <button @onclick="(() => EditTour(t.Id))"
class="btn btn-primary">Edit</button>
                    </td>
                </tr>
            }
        </tbody>
    </table>
}

```

```
@code {
    List<Hotel>? hotels;
    List<Tour>? tours;

    protected override async Task OnInitializedAsync()
    {
        var result = await http.GetFromJsonAsync<List<Hotel>>
("/api/hotel");
        if (result != null)
        {
            hotels = result;
        }

        var result2 = await http.GetFromJsonAsync<List<Tour>>("/api/tour");
        if (result2 != null)
        {
            tours = result2;
        }
    }

    private void AddHotel()
    {
        NavigationManager.NavigateTo("/hoteledit");
    }

    private void EditHotel(int id)
    {
        NavigationManager.NavigateTo($"hoteledit/{id}");
    }

    private void AddTour()
    {
        NavigationManager.NavigateTo("/touredit");
    }

    private void EditTour(int id)
    {
        NavigationManager.NavigateTo($"touredit/{id}");
    }

    private void ViewPayments()
    {
        NavigationManager.NavigateTo("/admin/payments");
    }

    private void ViewBookings()
    {
        NavigationManager.NavigateTo("/admin/bookings");
    }
}
```

```
}  
}
```

### BlazorHotelBooking/Client/Pages/AdminBookings.razor

```
@page "/admin/bookings"  
@using BlazorHotelBooking.Shared  
@using Microsoft.AspNetCore.Authorization  
@using System.Security.Claims  
@inject HttpClient http  
@inject NavigationManager NavigationManager  
@attribute [Authorize(Roles = "Admin")]  
  
@if (hotel is null)  
{  
    <span>Loading hotels...</span>  
}  
else  
{  
    <h5>Hotel Bookings</h5>  
  
    <table class="table">  
        <thead>  
            <tr>  
                <th>User ID</th>  
                <th>Booking ID</th>  
                <th>Hotel ID</th>  
                <th>Room Type</th>  
                <th>Check in</th>  
                <th>Check Out</th>  
                <th>Nights</th>  
                <th>Deposit</th>  
                <th>Total Price</th>  
                <th>Paid In Full</th>  
                <th>Cancelled</th>  
            </tr>  
  
        </thead>  
        <tbody>  
            @foreach (var h in hotel)  
            {  
                <tr>  
                    <td width="5%">@h.UserId</td>  
                    <td width="5%">@h.Id</td>  
                    <td width="5%">@h.HotelId</td>  
                    <td width="5%">@h.RoomType</td>  
                    <td width="5%">@h.CheckIn</td>  
                    <td width="5%">@h.CheckOut</td>  
                    <td width="5%">@h.NumberOfNights</td>  
                    <td width="5%">@h.DepositAmountPaid</td>
```

```
        <td width="5%">@h.TotalPrice</td>
        <td width="5%">@h.PaidInfull</td>
        <td width="5%">@h.IsCancelled</td>
    </tr>
}
</tbody>
</table>
}

@if (tour is null)
{
    <span>Loading tours...</span>
}
else
{
    <h5>Tour Bookings</h5>

    <table class="table">
        <thead>
            <tr>
                <th>User ID</th>
                <th>Booking ID</th>
                <th>Tour ID</th>
                <th>Start Date</th>
                <th>End Date</th>
                <th>Num Of People</th>
                <th>Deposit</th>
                <th>Total Price</th>
                <th>Paid In Full</th>
                <th>Cancelled</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var t in tour)
            {
                <tr>
                    <td width="5%">@t.UserId</td>
                    <td width="5%">@t.Id</td>
                    <td width="5%">@t.TourId</td>
                    <td width="5%">@t.CommencementDate</td>
                    <td width="5%">@t.EndDate</td>
                    <td width="5%">@t.NumberOfPeople</td>
                    <td width="5%">@t.DepositAmountPaid</td>
                    <td width="5%">@t.TotalPrice</td>
                    <td width="5%">@t.PaidInfull</td>
                    <td width="5%">@t.IsCancelled</td>
                </tr>
            }
        </tbody>
    </table>
}

@if (package is null)
```

```
{
    <span>Loading packages...</span>
}
else
{
    <h5>Package Bookings</h5>

    <table class="table">
        <thead>
            <tr>
                <th>User ID</th>
                <th>Booking ID</th>
                <th>Tour ID</th>
                <th>Start Date</th>
                <th>End Date</th>
                <th>Num Of People</th>
                <th>Hotel ID</th>
                <th>Room Type</th>
                <th>Check in</th>
                <th>Check Out</th>
                <th>Nights</th>
                <th>Deposit</th>
                <th>Total Price</th>
                <th>Paid In Full</th>
                <th>Cancelled</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var p in package)
            {
                <tr>
                    <td width="5%">@p.UserId</td>
                    <td width="5%">@p.Id</td>
                    <td width="5%">@p.TourId</td>
                    <td width="5%">@p.TourStartDate</td>
                    <td width="5%">@p.TourEndDate</td>
                    <td width="5%">@p.NumberOfPeopleOnTour</td>
                    <td width="5%">@p.HotelId</td>
                    <td width="5%">@p.RoomType</td>
                    <td width="5%">@p.HotelCheckIn</td>
                    <td width="5%">@p.HotelCheckOut</td>
                    <td width="5%">@p.NumberOfNights</td>
                    <td width="5%">@p.DepositAmountPaid</td>
                    <td width="5%">@p.TotalPrice</td>
                    <td width="5%">@p.PaidInfull</td>
                    <td width="5%">@p.IsCancelled</td>
                </tr>
            }
        </tbody>
    </table>
}
```



```
@code {
    List<HotelBooking> hotel = new List<HotelBooking>();
    List<TourBooking> tour = new List<TourBooking>();
    List<PackageBooking> package = new List<PackageBooking>();

    [CascadingParameter]
    private Task<AuthenticationState>? authenticationState { get; set; }

    protected override async Task OnInitializedAsync()
    {
        var result = await http.GetFromJsonAsync<List<HotelBooking>>
($"/api/Admin/hotelbooking");
        if (result != null)
        {
            hotel = result;
        }

        var result2 = await http.GetFromJsonAsync<List<TourBooking>>
($"/api/Admin/tourbooking");
        if (result2 != null)
        {
            tour = result2;
        }

        var result3 = await http.GetFromJsonAsync<List<PackageBooking>>
($"/api/Admin/packagebooking");
        if (result3 != null)
        {
            package = result3;
        }
    }
}
```

### BlazorHotelBooking/Client/Pages/AdminPayments.razor

```
@page "/admin/payments"
@using BlazorHotelBooking.Shared
@using Microsoft.AspNetCore.Authorization
@using System.Security.Claims
@inject HttpClient http
@inject NavigationManager NavigationManager
@attribute [Authorize(Roles = "Admin")]

@if (payments is null)
{
    <span>Loading Payments...</span>
}
else
{

```

```
<h5>Payments</h5>

<table class="table">
  <thead>
    <tr>
      <th>User ID</th>
      <th>Booking ID</th>
      <th>Booking Type</th>
      <th>Payment Type</th>
      <th>Payment Date</th>
      <th>Amount Paid</th>
    </tr>
  </thead>
  <tbody>
    @foreach (var p in payments)
    {
      <tr>
        <td width="5%">@p.UserId</td>
        <td width="5%">@p.bookingId</td>
        <td width="5%">@p.bookingType</td>
        <td width="5%">@p.paymentType</td>
        <td width="5%">@p.PaymentDate</td>
        <td width="5%">@p.AmountPaid</td>
      </tr>
    }
  </tbody>
</table>
}

@code {
  List<Payments> payments = new List<Payments>();

  [CascadingParameter]
  private Task<AuthenticationState>? authenticationState { get; set; }

  protected override async Task OnInitializedAsync()
  {
    var result = await http.GetFromJsonAsync<List<Payments>>
($"/api/Payment");
    if (result != null)
    {
      payments = result;
    }
  }
}
```

```
@page "/editbooking/hotel/{id}"
@using BlazorHotelBooking.Shared
@using Microsoft.AspNetCore.Authorization
@inject HttpClient http
@inject NavigationManager NavigationManager
@attribute [Authorize]

<PageTitle> Edit Booking </PageTitle>
<h4>Editing booking for @hotel.Name on
@selectedBooking.CheckIn.ToString("dd/MM/yyyy")</h4>

<EditForm Model="selectedBooking" OnSubmit="HandleSubmit">
    <div>
        <label for="HotelName">Hotel</label>
        <InputText id="HotelName" @bind-Value="hotel.Name" class="form-
control" disabled="true" />
    </div>
    <div>
        <label for="RoomType">Room Type</label>
        <InputSelect id="roomType" @bind-Value="selectedBooking.RoomType"
placeholder="Room Type">
            <option value="">---</option>
            @foreach (var room in roomType)
            {
                <option value="@room">@room</option>
            }
        </InputSelect>
    </div>
    <div>
        <label for="NumOfNights">Number Of Nights</label>
        <InputNumber id="NumOfNights" @bind-
Value="selectedBooking.NumberOfNights" class="form-control" />
    </div>
    @{
        var currentDeposit = selectedBooking.DepositAmountPaid;
        decimal currentTotal = selectedBooking.TotalPrice;
        var currentCheckOut = selectedBooking.CheckOut;

        switch (selectedBooking.RoomType)
        {
            case "Single":
                newTotal = hotel.SBPrice * selectedBooking.NumberOfNights;
                newDeposit = newTotal / 5;
                newCheckOut =
selectedBooking.CheckIn.AddDays(selectedBooking.NumberOfNights);
                break;
            case "Double":
                newTotal = hotel.DBPrice * selectedBooking.NumberOfNights;
                newDeposit = newTotal / 5;
                newCheckOut =
```

```
selectedBooking.CheckIn.AddDays(selectedBooking.NumberOfNights);
    break;
    case "Family":
        newTotal = hotel.FamPrice * selectedBooking.NumberOfNights;
        newDeposit = newTotal / 5;
        newCheckOut =
selectedBooking.CheckIn.AddDays(selectedBooking.NumberOfNights);
        break;
    }

    if(selectedBooking.PaidInfull)
    {
        if (newTotal > currentTotal)
        {
            difference = newTotal - currentTotal;
            <div>
                <p>The new total for this booking is £@newTotal</p>
                <p>You will need to pay £@difference more to complete
this booking</p>
            </div>
        }
        else if (newTotal < currentTotal)
        {
            var refund = currentTotal - newTotal;
            difference = refund * -1;
            <div>
                <p>The new total for this booking is £@newTotal</p>
                <p>You will be refunded £@refund</p>
            </div>
        }
    }
    else
    {
        if (newTotal > currentTotal)
        {
            difference = newDeposit - currentDeposit;
            <div>
                <p>The new total for this booking is £@newTotal</p>
                <p>The new deposit for this booking is £@newDeposit</p>
                <p>You will need to pay £@difference more to complete
this booking</p>
            </div>
        }
        else if (newTotal < currentTotal)
        {
            var refund = currentDeposit - newDeposit;
            difference = refund * -1;
            <div>
                <p>The new total for this booking is £@newTotal</p>
                <p>The new deposit for this booking is £@newDeposit</p>
                <p>You will be refunded £@refund</p>
            </div>
        }
    }
}
```

```

        }
    }

    surcharge = newTotal * (decimal)0.05;

    <p>Modifying this booking will also incur a 5% surcharge
    (£@surcharge)</p>

    if (showOverlap)
    {
        <div class="alert alert-danger" role="alert">
            <p>There are no more spaces for this room on these dates
            please select new dates.</p>
        </div>
    }
}
<button type="submit" class="btn btn-primary">Save</button>
</EditForm>

```

```

@code {
    [Parameter]
    public string? Id { get; set; }

    private HotelBooking selectedBooking = new HotelBooking();
    private Hotel hotel = new Hotel();
    List<string> roomType = new List<string>() { "Single", "Double",
    "Family" };
    private int currentNights;
    private decimal newTotal, newDeposit;
    private DateTime newCheckOut;
    private bool showOverlap = false;
    private int numOfOverlap;
    private decimal difference;
    private decimal surcharge;

    protected override async Task OnParametersSetAsync()
    {
        if (Id is not null)
        {
            var result = await http.GetFromJsonAsync<HotelBooking>
            ($"api/bookings/hotel/{Id}");
            if (result is not null)
            {
                selectedBooking = result;
            }
        }

        currentNights = selectedBooking.NumberOfNights;

        hotel = await http.GetFromJsonAsync<Hotel>
        ($"api/hotel/{selectedBooking.HotelId}");
    }
}

```

```

async Task HandleSubmit()
{
    if (selectedBooking.PaidInfull)
    {
        selectedBooking.DepositAmountPaid = newDeposit;
        selectedBooking.TotalPrice = newTotal;
    }
    else
    {
        selectedBooking.DepositAmountPaid = newDeposit;
        selectedBooking.TotalPrice = newTotal;
    }

    selectedBooking.CheckOut = newCheckOut;

    numOfOverlap = await http.GetFromJsonAsync<int>
($"/api/bookings/hotel/overlap?checkIn={selectedBooking.CheckIn}&checkOut=
{selectedBooking.CheckOut}&hotelId={selectedBooking.HotelId}&roomType=
{selectedBooking.RoomType}");

    if (numOfOverlap > 20)
    {
        showOverlap = true;
    }
    else
    {
        await http.PutAsJsonAsync($"api/bookings/hotel/{Id}?
paymentRemainder={difference}&surcharge={surcharge}", selectedBooking);
        NavigationManager.NavigateTo("/mybookings");
    }
}
}

```

### BlazorHotelBooking/Client/Pages/EditPackageBooking.razor

```

@page "/editbooking/package/{id}"
@using BlazorHotelBooking.Shared
@using Microsoft.AspNetCore.Authorization
@inject HttpClient http
@inject NavigationManager NavigationManager
@attribute [Authorize]

<PageTitle> Edit Booking </PageTitle>
<h7>Editing booking for @hotel.Name on
@selectedBooking.HotelCheckIn.ToString("dd/MM/yyyy") & @tour.Name on
@selectedBooking.TourStartDate</h7>

```

```

<EditForm Model="selectedBooking" OnSubmit="HandleSubmit">
    <div>
        <label for="HotelName">Hotel</label>
        <InputText id="HotelName" @bind-Value="hotel.Name" class="form-
control" disabled="true" />
    </div>
    <div>
        <label for="RoomType">Room Type</label>
        <InputSelect id="roomType" @bind-Value="selectedBooking.RoomType"
placeholder="Room Type">
            <option value="">---</option>
            @foreach (var room in roomType)
            {
                <option value="@room">@room</option>
            }
        </InputSelect>
    </div>
    <div>
        <label for="NumOfNights">Number Of Nights</label>
        <InputNumber id="NumOfNights" @bind-
Value="selectedBooking.NumberOfNights" class="form-control" />
    </div>
    <br/>
    <br/>
    <div>
        <label for="TourName">Tour</label>
        <InputText id="TourName" @bind-Value="tour.Name" class="form-
control" disabled="true" />
    </div>
    <div>
        <label for="NumOfPeople">Number Of Guests</label>
        <InputNumber id="NumOfPeople" @bind-
Value="selectedBooking.NumberOfPeopleOnTour" class="form-control" />
    </div>

    @{
        var currentDeposit = selectedBooking.DepositAmountPaid;
        decimal currentTotal = selectedBooking.TotalPrice;
        var currentCheckOut = selectedBooking.HotelCheckOut;

        switch (selectedBooking.RoomType)
        {
            case "Single":
                newTotal = (hotel.SBPrice * selectedBooking.NumberOfNights)
+ tour.Cost;
                newTotal = newTotal * (decimal)0.9;
                newDeposit = newTotal / 5;
                newCheckOut =
selectedBooking.HotelCheckIn.AddDays(selectedBooking.NumberOfNights);
                break;

```

```

        case "Double":
            newTotal = (hotel.DBPrice * selectedBooking.NumberOfNights)
+ tour.Cost;
            newTotal = newTotal * (decimal)0.8;
            newDeposit = newTotal / 5;
            newCheckOut =
selectedBooking.HotelCheckIn.AddDays(selectedBooking.NumberOfNights);
            break;
        case "Family":
            newTotal = (hotel.FamPrice *
selectedBooking.NumberOfNights) + tour.Cost;
            newTotal = newTotal * (decimal)0.6;
            newDeposit = newTotal / 5;
            newCheckOut =
selectedBooking.HotelCheckIn.AddDays(selectedBooking.NumberOfNights);
            break;
    }

    if(selectedBooking.PaidInfull)
    {
        if (newTotal > currentTotal)
        {
            difference = newTotal - currentTotal;
            <div>
                <p>The new total for this booking is £@newTotal</p>
                <p>You will need to pay £@difference more to complete
this booking</p>
            </div>
        }
        else if (newTotal < currentTotal)
        {
            var refund = currentTotal - newTotal;
            difference = refund * -1;
            <div>
                <p>The new total for this booking is £@newTotal</p>
                <p>You will be refunded £@refund</p>
            </div>
        }
    }
    else
    {
        if (newTotal > currentTotal)
        {
            difference = newDeposit - currentDeposit;
            <div>
                <p>The new total for this booking is £@newTotal</p>
                <p>The new deposit for this booking is £@newDeposit</p>
                <p>You will need to pay £@difference more to complete
this booking</p>
            </div>
        }
        else if (newTotal < currentTotal)

```



```

        {
            var refund = currentDeposit - newDeposit;
            difference = refund * -1;
            <div>
                <p>The new total for this booking is £@newTotal</p>
                <p>The new deposit for this booking is £@newDeposit</p>
                <p>You will be refunded £@refund</p>
            </div>

        }
    }

    surcharge = newTotal * (decimal)0.05;

    <p>Modifying this booking will also incur a 5% surcharge
    (£@surcharge)</p>

    if (showHotelOverlap)
    {
        <div class="alert alert-danger" role="alert">
            <p>There are no more spaces for this room on these dates
            please select new dates.</p>
        </div>
    }

    if (showTourOverlap)
    {
        <div class="alert alert-danger" role="alert">
            <p>There are no more spaces left in this tour for that many
            people on these dates please select new dates or less people.</p>
        </div>
    }
}
<button type="submit" class="btn btn-primary">Save</button>
</EditForm>

@code {
    [Parameter]
    public string? Id { get; set; }

    private PackageBooking selectedBooking = new PackageBooking();
    private Hotel hotel = new Hotel();
    private Tour tour = new Tour();
    List<string> roomType = new List<string>() { "Single", "Double",
    "Family" };
    private int currentNights;
    private decimal newTotal, newDeposit;
    private DateTime newCheckOut;
    private int numOfTourOverlap, numOfHotelOverlap;
    private bool showHotelOverlap, showTourOverlap = false;
    private decimal difference;
    private decimal surcharge;

```

```
protected override async Task OnParametersSetAsync()
{
    if (Id is not null)
    {
        var result = await http.GetFromJsonAsync<PackageBooking>
($"api/bookings/package/{Id}");
        if (result is not null)
        {
            selectedBooking = result;
        }
    }

    currentNights = selectedBooking.NumberOfNights;

    hotel = await http.GetFromJsonAsync<Hotel>
($"api/hotel/{selectedBooking.HotelId}");
    tour = await http.GetFromJsonAsync<Tour>
($"api/tour/{selectedBooking.TourId}");
}

async Task HandleSubmit()
{
    if (selectedBooking.PaidInfull)
    {
        selectedBooking.DepositAmountPaid = newTotal;
        selectedBooking.TotalPrice = newTotal;
    }
    else
    {
        selectedBooking.DepositAmountPaid = newDeposit;
        selectedBooking.TotalPrice = newTotal;
    }

    selectedBooking.HotelCheckOut = newCheckOut;

    numOfHotelOverlap = await http.GetFromJsonAsync<int>
("/api/bookings/hotel/overlap?checkIn=
{selectedBooking.HotelCheckIn}&checkOut=
{selectedBooking.HotelCheckOut}&hotelId={hotel.Id}&roomType=
{selectedBooking.RoomType}");

    numOfTourOverlap = await http.GetFromJsonAsync<int>
("/api/bookings/tour/overlap?start={selectedBooking.TourStartDate}&end=
{selectedBooking.TourEndDate}&tourId={tour.Id}");

    if ((tour.MaxNumberOfGuests < numOfTourOverlap +
selectedBooking.NumberOfPeopleOnTour) & (numOfHotelOverlap > 20))
    {
        showHotelOverlap = true;
        showTourOverlap = true;
    }
    else
    {

```

```

        await http.PutAsJsonAsync($"api/bookings/package/{Id}?
paymentRemainder={difference}&surcharge={surcharge}", selectedBooking);
        NavigationManager.NavigateTo("/mybookings");
    }
}
}

```

### BlazorHotelBooking/Client/Pages/EditTourBooking.razor

```

@page "/editbooking/tour/{id}"
@using BlazorHotelBooking.Shared
@using Microsoft.AspNetCore.Authorization
@inject HttpClient http
@inject NavigationManager NavigationManager
@attribute [Authorize]

<PageTitle> Edit Booking </PageTitle>
<h4>Editing booking for @tour.Name on
@selectedBooking.CommencementDate.ToString("dd/MM/yyyy")</h4>

<EditForm Model="selectedBooking" OnSubmit="HandleSubmit">
    <div>
        <label for="TourName">Tour</label>
        <InputText id="TourName" @bind-Value="tour.Name" class="form-
control" disabled="true" />
    </div>
    <div>
        <label for="NumOfPeople">Number Of Guests</label>
        <InputNumber id="NumOfPeople" @bind-
Value="selectedBooking.NumberOfPeople" class="form-control" />
    </div>

    @{
        var total = selectedBooking.TotalPrice;
        surcharge = (double)total * 0.05;

        <p>Modifying this booking will also incur a 5% surcharge
(£@surcharge)</p>

        if (showOverlap)
        {
            <div class="alert alert-danger" role="alert">
                <p>There are no more spaces for this tour for that many
people.</p>
            </div>
        }
    }
    <button type="submit" class="btn btn-primary">Save</button>
</EditForm>

```

```
@code {
    [Parameter]
    public string? Id { get; set; }
    private TourBooking selectedBooking = new TourBooking();
    private Tour tour = new Tour();
    private int currentNights;
    private decimal newTotal, newDeposit;
    private DateTime newCheckOut;
    private bool showOverlap = false;
    private int numOfOverlap;
    private int currentPeople;
    private double surcharge;

    protected override async Task OnParametersSetAsync()
    {
        if (Id is not null)
        {
            var result = await http.GetFromJsonAsync<TourBooking>
($"api/bookings/tour/{Id}");
            if (result is not null)
            {
                selectedBooking = result;
            }

            currentPeople = selectedBooking.NumberOfPeople;

            tour = await http.GetFromJsonAsync<Tour>
($"api/tour/{selectedBooking.TourId}");
        }

        async Task HandleSubmit()
        {
            numOfOverlap = await http.GetFromJsonAsync<int>
($"api/bookings/tour/overlap?start={selectedBooking.CommencementDate}&end=
{selectedBooking.EndDate}&tourId={tour.Id}");

            if (tour.MaxNumberOfGuests < numOfOverlap +
(selectedBooking.NumberOfPeople - currentPeople))
            {
                showOverlap = true;
            }
            else
            {
                await http.PutAsJsonAsync($"api/bookings/tour/{Id}?surcharge=
{surcharge}", selectedBooking);
                NavigationManager.NavigateTo("/mybookings");
            }
        }
    }
}
```

## BlazorHotelBooking/Client/Pages/HotelDetails.razor

```
@page "/hotels/{id}"
@using BlazorHotelBooking.Shared;
@using Microsoft.AspNetCore.Authorization;
@using Microsoft.AspNetCore.Http;
@using Syncfusion.Blazor.DropDowns;
@using System.ComponentModel.DataAnnotations;
@using System.Security.Claims
@Inject IHttpContextAccessor httpContextAccessor
@Inject HttpClient http
@Inject NavigationManager NavigationManager
[Attribute [Authorize]

<div class="media">
    <div class="media-body">
        <h2 class="mb-0">@selectedHotel.Name</h2>
        <p>@selectedHotel.Description</p>
        <h4 class="price">
            Single Room: £@selectedHotel.SBPrice (per night)
        </h4>
        <h4 class="price">
            Double Room: £@selectedHotel.DBPrice (per night)
        </h4>
        <h4 class="price">
            Family Room: £@selectedHotel.FamPrice (per night)
        </h4>
        <button @onclick="ShowBookingForm" class="btn btn-primary"><i
class="oi oi-cart">&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~&nbsp;&nbsp;&nbsp;&nbsp;& Book Hotel</i></button>
    </div>
</div>

@if(showBookingForm)
{
    <div class="backgroundPopupBox">
        <div class="popupCreate">
            <EditForm Model=@newBooking OnValidSubmit=BookHotel>
                <DataAnnotationsValidator />
                *@ <ValidationSummary /> *@

                <div class="form-group">
                    <label class="control-label">Check-In Date</label>
                    <InputDate @bind-Value="newBooking.CheckIn" min="@min"
max="@max" Placeholder="Enter Date" />
                    <ValidationMessage For="@(( ) => newBooking.CheckIn)" />
                </div>
                <div class="form-group mt-3 col-sm-10">
                    <label class="control-label">Room Type</label>
                    <InputSelect id="roomType" @bind-
Value="newBooking.RoomType" placeholder="Room Type">
```

```

        <option value="">---</option>
        @foreach (var room in roomType)
        {
            <option value="@room">@room</option>
        }
    </InputSelect>
    <ValidationMessage For="@(( ) => newBooking.RoomType)"
/>

</div>
<div class="form-group mt-3">
    <label class="control-label">Number of nights</label>
    <InputNumber id="NumOfNights" @bind-
Value="newBooking.NumberOfNights" class="form-control" />
    <ValidationMessage For="@(( ) =>
newBooking.NumberOfNights)" />
</div>

@{
    switch (newBooking.RoomType)
    {
        case "Single":
            newBooking.TotalPrice =
newBooking.NumberOfNights * selectedHotel.SBPrice;
            break;
        case "Double":
            newBooking.TotalPrice =
newBooking.NumberOfNights * selectedHotel.DBPrice;
            break;
        case "Family":
            newBooking.TotalPrice =
newBooking.NumberOfNights * selectedHotel.FamPrice;
            break;
    }
    newBooking.DepositAmountPaid = newBooking.TotalPrice /
5 ;

    newBooking.CheckOut =
newBooking.CheckIn.AddDays(newBooking.NumberOfNights);
}

<div>Check In Date:
@newBooking.CheckIn.ToString("dd/MM/yyyy")</div>
<div>Check Out Date:
@newBooking.CheckOut.ToString("dd/MM/yyyy")</div>

<div>Total Price is £@newBooking.TotalPrice</div>
<div>Total to pay today (20%)
£@newBooking.DepositAmountPaid</div>

@if (showOverlap)
{

```

```

        <div class="alert alert-danger" role="alert">
            <p>There are no more spaces for this room on these
dates please select new dates.</p>
        </div>
    }

    <button type="button" class="btn btn-success"
@onclick="BookHotel">Book</button>

    </EditForm>

    <div class="form-group">
        <button class="btn btn-danger"
@onclick="ClosePopup">Cancel</button>
    </div>
</div>
</div>

}

@code {
    private Hotel selectedHotel = new Hotel();
    private HotelBooking newBooking = new HotelBooking();
    private bool showBookingForm = false;
    private string min;
    private string max;
    private string userId;
    private int numOfOverlap;
    private bool showOverlap = false;

    [CascadingParameter]
    private Task<AuthenticationState>? authenticationState { get; set; }

    [Parameter]
    public string? Id { get; set; }

    List<Hotel> hotels = new List<Hotel>();
    List<string> roomType = new List<string>() { "Single", "Double",
"Family" };

    protected override async Task OnInitializedAsync()
    {
        min =
DateOnly.FromDateTime(DateTime.Now.Date.AddMonths(2)).ToString("yyyy-MM-
dd");
        max =
DateOnly.FromDateTime(DateTime.Now.Date.AddYears(5)).ToString("yyyy-MM-
dd");

        var authState = await authenticationState;
        var user = authState?.User;
        userId = user?.FindFirst(c => c.Type ==
ClaimTypes.NameIdentifier)?.Value;

```

```
        var result = await http.GetFromJsonAsync<List<Hotel>>
("/api/hotel");
        if (result != null)
        {
            hotels = result;
        }

        selectedHotel = hotels.FirstOrDefault(h => h.Id ==
Int32.Parse(Id));
    }

    private void ShowBookingForm()
    {
        showBookingForm = true;
    }

    async Task BookHotel()
    {
        //code to check if the newbooking overlaps with existing bookings
        numOfOverlap = await http.GetFromJsonAsync<int>
($" /api/bookings/hotel/overlap?checkIn={newBooking.CheckIn}&checkOut=
{newBooking.CheckOut}&hotelId={selectedHotel.Id}&roomType=
{newBooking.RoomType}");

        if (numOfOverlap > 20)
        {
            showOverlap = true;
        }
        else
        {
            newBooking.HotelId = selectedHotel.Id;
            newBooking.UserId = userId;
            newBooking.PaymentDueDate = newBooking.CheckIn.AddDays(-28);

            await http.PostAsJsonAsync("/api/bookings/hotel/book",
newBooking);

            ClosePopup();
            NavigationManager.NavigateTo("/mybookings");
        }
    }

    private void ClosePopup()
    {
        showBookingForm = false;
    }
}
```



```
@page "/hoteledit"
@page "/hoteledit/{id:int}"
@using BlazorHotelBooking.Shared
@using Microsoft.AspNetCore.Authorization
@inject HttpClient http
@inject NavigationManager NavigationManager
@attribute [Authorize(Roles="Admin")]

@if (Id is null)
{
    <PageTitle>Add New Hotel</PageTitle>
    <h3>Add New Hotel</h3>
}
else
{
    <PageTitle> Edit @selectedHotel.Name </PageTitle>
    <h3>@selectedHotel.Name</h3>
}

<EditForm Model="selectedHotel" OnSubmit="HandleSubmit">

    <div>
        <label for="Id">Id</label>
        <InputNumber id="Id" @bind-Value="selectedHotel.Id" class="form-control" disabled="true" />
    </div>
    <div>
        <label for="Name">Name</label>
        <InputText id="Name" @bind-Value="selectedHotel.Name" class="form-control" />
    </div>
    <div>
        <label for="SBPrice">Single Room Price</label>
        <InputNumber id="SBPrice" @bind-Value="selectedHotel.SBPrice" class="form-control" />
    </div>
    <div>
        <label for="DBPrice">Double Room Price</label>
        <InputNumber id="DBPrice" @bind-Value="selectedHotel.DBPrice" class="form-control" />
    </div>
    <div>
        <label for="FamPrice">Family Room Price</label>
        <InputNumber id="FamPrice" @bind-Value="selectedHotel.FamPrice" class="form-control" />
    </div>
    <div>
        <label for="Description">Description</label>
        <InputText id="Description" @bind-Value="selectedHotel.Description" class="form-control" />
    </div>
    <button type="submit" class="btn btn-primary">Save</button>
    @if (Id is not null)
```

```
{
    <button type="button" class="btn btn-danger"
@onclick="DeleteHotel">Delete</button>
}
</EditForm>

@code {
    [Parameter]
    public int? Id { get; set; }

    private Hotel selectedHotel = new Hotel { Name = "New Hotel" };

    protected override async Task OnParametersSetAsync()
    {
        if (Id is not null)
        {
            var result = await http.GetFromJsonAsync<Hotel>
($"api/hotel/{Id}");
            if (result is not null)
            {
                selectedHotel = result;
            }
        }
    }

    async Task DeleteHotel()
    {
        await http.DeleteAsync($"api/Admin/hotel/{Id}");
        NavigationManager.NavigateTo("/admin");
    }

    async Task HandleSubmit()
    {
        if(Id is null)
        {
            await http.PostAsJsonAsync("/api/Admin/hotel", selectedHotel);
        }
        else
        {
            await http.PutAsJsonAsync($"api/Admin/hotel/{Id}",
selectedHotel);
        }
        NavigationManager.NavigateTo("/admin");
    }
}
```

### BlazorHotelBooking/Client/Pages/Hotels.razor

```
@page "/hotels"
@using BlazorHotelBooking.Shared
```

```
@using Microsoft.AspNetCore.Authorization
@Inject HttpClient http
[Authorize]

<h3>Hotels </h3>

<input @oninput="Search" placeholder="Search..." class="p-3" />

<br />
<br />
<EditForm Model="checkin" OnSubmit="DateSearch">
    <div class="form-group">
        <label class="control-label">Check-In Date</label>
        <InputDate @bind-Value="checkin" min="@min" max="@max"
Placeholder="Enter Date" />
        <label class="control-label">Number Of Nights</label>
        <InputNumber @bind-Value="numofnights" Placeholder="Number of
Nights" />
    </div>
    <button type="button" class="btn btn-success"
@onclick="DateSearch">Search</button>
</EditForm>

@if (hotels.Count <= 0)
{
    <span> Loading Hotels...</span>
}
else
{
    @foreach (var hotel in hotels)
    {
        <li class="media my-3">
            <div class="media-body">
                <a href="/hotels/@hotel.Id">
                    <h4 class="mb-0">@hotel.Name</h4>
                </a>
                <p>@hotel.Description</p>
                <h6 class="price">
                    Single bed: £@hotel.SBPrice
                </h6>
                <h6 class="price">
                    Double bed: £@hotel.DBPrice
                </h6>
                <h6 class="price">
                    Family bed: £@hotel.FamPrice
                </h6>
            </div>
        </li>
    }
}
```

```
@code {
    List<Hotel> hotels = new List<Hotel>();
    DateTime checkin = DateTime.Now.Date.AddMonths(2);
    DateTime checkout;
    private int numofnights;
    private string min;
    private string max;
    private int numSingleOver, numDoubleOver, numFamilyOver;

    protected override async Task OnInitializedAsync()
    {

        min =
        DateOnly.FromDateTime(DateTime.Now.Date.AddMonths(2)).ToString("yyyy-MM-
dd");
        max =
        DateOnly.FromDateTime(DateTime.Now.Date.AddYears(5)).ToString("yyyy-MM-
dd");

        var result = await http.GetFromJsonAsync<List<Hotel>>
("/api/hotel");
        if (result != null)
        {
            hotels = result;
        }
    }

    private async void Search(ChangeEventArgs args)
    {
        var searchTerm = (string)args.Value;

        var result = await http.GetFromJsonAsync<List<Hotel>>
("/api/hotel");
        // make it case insensitive to search hotels

        hotels = result.Where(x => x.Name.IndexOf(searchTerm,
StringComparison.OrdinalIgnoreCase) >= 0)
            .OrderByDescending(x => x.Id)
            .ToList();

        StateHasChanged();
    }

    private async void DateSearch()
    {
        var result = await http.GetFromJsonAsync<List<Hotel>>
("/api/hotel");
        var tempList = new List<Hotel>();
        checkout = checkin.AddDays(numofnights);

        if (result != null)
        {
```

```

        foreach (var hotel in result)
        {
            numSingleOver = await http.GetFromJsonAsync<int>
($"/api/bookings/hotel/overlap?checkIn={checkin}&checkOut=
{checkout}&hotelId={hotel.Id}&roomType=Single");
            numDoubleOver = await http.GetFromJsonAsync<int>
($"/api/bookings/hotel/overlap?checkIn={checkin}&checkOut=
{checkout}&hotelId={hotel.Id}&roomType=Double");
            numFamilyOver = await http.GetFromJsonAsync<int>
($"/api/bookings/hotel/overlap?checkIn={checkin}&checkOut=
{checkout}&hotelId={hotel.Id}&roomType=Family");

            if (!(numSingleOver > 20 && numDoubleOver > 20 &&
numFamilyOver > 20))
            {
                tempList.Add(hotel);
            }
        }

        hotels = tempList;
    }
    StateHasChanged();
}
}

```

### BlazorHotelBooking/Client/Pages/Index.razor

```

@page "/"

<PageTitle>Blazing Hotels</PageTitle>

<AuthorizeView>
    <Authorized>
        <h1>Welcome to Blazing Hotel @context.User.Identity!.Name!</h1>
        <p>Click on the Hotels link to see the list of hotels and book your
desired hotel.</p>
    </Authorized>
    <NotAuthorized>
        <h1>Welcome to Blazing Hotel!</h1>
        <p>You are not logged in. Please log in to access the website.</p>
    </NotAuthorized>
</AuthorizeView>
<AuthorizeView Roles="Admin">
    <p>You are logged in as an Admin.</p>
</AuthorizeView>

```

### BlazorHotelBooking/Client/Pages/Login.razor

```
@page "/login"
@using BlazorHotelBooking.Client.Service;
@using BlazorHotelBooking.Shared.Models;
@inject IAuthService AuthService
@inject NavigationManager NavigationManager

<h1>Login</h1>

@if (ShowErrors)
{
    <div class="alert alert-danger" role="alert">
        <p>@Error</p>
    </div>
}

<div class="card">
    <div class="card-body">
        <h5 class="card-title">Enter Your Login Details</h5>
        <EditForm Model="loginModel" OnValidSubmit="HandleLogin">
            <DataAnnotationsValidator />
            <ValidationSummary />

            <div class="form-group mt-2">
                <label for="email">Email address</label>
                <InputText Id="email" Class="form-control" @bind-
Value="loginModel.Email" />
                <ValidationMessage For="@(() => loginModel.Email)" />
            </div>
            <div class="form-group mt-2">
                <label for="password">Password</label>
                <InputText Id="password" type="password" Class="form-
control" @bind-Value="loginModel.Password" />
                <ValidationMessage For="@(() => loginModel.Password)" />
            </div>
            <button type="submit" class="btn btn-primary mt-
2">Login</button>
        </EditForm>
    </div>
</div>

@code {
    private LoginModel loginModel = new LoginModel();
    private bool ShowErrors;
    private string Error = "";

    private async Task HandleLogin()
    {
        ShowErrors = false;
        var result = await AuthService.Login(loginModel);
        if (result.Successful)
        {
            NavigationManager.NavigateTo("/");
        }
    }
}
```

```
        else
        {
            ShowErrors = true;
            Error = result.Error!;
        }
    }
}
```

### BlazorHotelBooking/Client/Pages/LoginDisplay.razor

```
@using Microsoft.AspNetCore.Components.Authorization

<AuthorizeView>
<Authorized>
    Hi, @context.User.Identity!.Name!
    <a href="logout">Logout</a>
</Authorized>
<NotAuthorized>
    <a href="register">Register</a>
    <a href="login">Login</a>
</NotAuthorized>
</AuthorizeView>
```

### BlazorHotelBooking/Client/Pages/Logout.razor

```
@page "/logout"
@using BlazorHotelBooking.Client.Service;
@inject IAuthService AuthService
@inject NavigationManager NavigationManager

@code {

    protected override async Task OnInitializedAsync()
    {
        await AuthService.Logout();
        NavigationManager.NavigateTo("/login");
    }
}
```

### BlazorHotelBooking/Client/Pages/MyBookings.razor

```
@page "/mybookings"
@using BlazorHotelBooking.Shared.Models
@using Microsoft.AspNetCore.Authorization
@using System.Security.Claims
@inject HttpClient http
```

```
@inject NavigationManager NavigationManager
@inject IJSRuntime JsRuntime
@attribute [Authorize]

@if (hotelBookings is null)
{
    <h3>Loading Hotels...</h3>
}
else
{
    if(hotelBookings.Count == 0)
    {
        <h3>You have no hotel bookings</h3>
    }
    else
    {
        <h2>Hotels</h2>
        <table class="table">
            <thead>
                <tr>
                    <th>Hotel</th>
                    <th>Room Type</th>
                    <th>Check In Date</th>
                    <th>Check Out Date</th>
                    <th>Number of Nights</th>
                    <th>Deposit</th>
                    <th>Total Cost</th>
                    <th>Payment Due Date</th>
                    <th>Paid in full?</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var h in hotelBookings)
                {
                    if (!h.IsCancelled)
                    {
                        disableCancel = false;
                        disableModify = false;

                        var today = DateTime.Now;
                        var datediff = h.CheckIn - today;
                        if (datediff.Days < 5)
                        {
                            disableCancel = true;
                        }

                        if (datediff.Days < 14)
                        {

```



```

        disableModify = true;
    }

    <tr>
        <td width="5%">@h.hotelName</td>
        <td width="5%">@h.RoomType Room</td>
        <td
width="5%">@h.CheckIn.ToString("dd/MM/yyyy")</td>
        <td
width="5%">@h.CheckOut.ToString("dd/MM/yyyy")</td>
        <td width="5%">@h.NumberOfNights Nights</td>
        <td width="5%">£@h.DepositAmountPaid</td>
        <td width="5%">£@h.TotalPrice</td>
        <td width="5%">@h.PaymentDueDate</td>
        <td width="5%">@h.paidInfull</td>
        <td width="5%">
            <button class="btn btn-primary"
disabled="@disableModify" @onclick="() =>
ModifyHotelBooking(h.bookingId)">Modify</button>
        </td>
        <td width="5%">
            <button class="btn btn-danger"
disabled="@disableCancel" @onclick="() =>
CancelHotelBooking(h.bookingId)">Cancel</button>
        </td>
        <td width="5%">
            <button class="btn btn-success"
disabled="@h.paidInfull" @onclick="() => ShowHotelPayPopup(h.bookingId,
h.DepositAmountPaid, h.TotalPrice)">Pay Remainder</button>
        </td>
    </tr>
    }
}
</tbody>
</table>

```

```

if (hasCancelledBookingsHotel)
{
    <h3>Cancelled Hotels for late payment</h3>
    <table class="table">
        <thead>
            <tr>
                <th>Hotel</th>
                <th>Room Type</th>
                <th>Check In Date</th>
                <th>Check Out Date</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var h in hotelBookings)

```

```

        {
            if (h.IsCancelled)
            {
                <tr>
                    <td width="5%">@h.hotelName</td>
                    <td width="5%">@h.RoomType Room</td>
                    <td
width="5%">@h.CheckIn.ToString("dd/MM/yyyy")</td>
                    <td
width="5%">@h.CheckOut.ToString("dd/MM/yyyy")</td>
                </tr>
            }
        }
    </tbody>
</table>

    <br />
    <br />
}
}
}

@{
    if (showHotelPayPopup)
    {
        var remainder = totalToPay - currentlyPaid;

        <div class="backgroundPopupBox">
            <div class="popupCreate">
                <h3>Pay Remainder</h3>
                <p>Are you sure you want to pay the remainder of this hotel
booking? It will cost you £@remainder </p>
                <button class="btn btn-success" @onclick="() =>
PayHotelRemainder(bookingId, remainder)">Yes</button>
                <button class="btn btn-danger"
@onclick="ClosePopup">No</button>
            </div>
        </div>
    }
}

@if (tourBookings is null)
{
    <h3>Loading Tours...</h3>
}
else
{
    if (tourBookings.Count == 0)
    {
        <h3>You have no tour bookings</h3>
    }
    else

```

```
{
    <h2>Tours</h2>
    <table class="table">
        <thead>
            <tr>
                <th>Tour</th>
                <th>Start Date</th>
                <th>End Date</th>
                <th>Number of Guests</th>
                <th>Amount Paid so Far</th>
                <th>Total Cost</th>
                <th>Payment Due Date</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var t in tourBookings)
            {
                if (!t.IsCancelled)
                {
                    disableCancel = false;
                    disableModify = false;

                    var today = DateTime.Now;
                    var datediff = t.CommencementDate - today;
                    if (datediff.Days < 5)
                    {
                        disableCancel = true;
                    }

                    if (datediff.Days < 14)
                    {
                        disableModify = true;
                    }

                    <tr>
                        <td width="5%">@t.TourName</td>
                        <td
width="5%">@t.CommencementDate.ToString("dd/MM/yyyy")</td>
                        <td
width="5%">@t.EndDate.ToString("dd/MM/yyyy")</td>
                        <td width="5%">@t.NumberOfGuests people</td>
                        <td width="5%">£@t.DepositAmountPaid</td>
                        <td width="5%">£@t.TotalPrice</td>
                        <td
width="5%">@t.PaymentDueDate.ToString("dd/MM/yyyy")</td>
                        <td width="5%">
                            <button class="btn btn-primary"
disabled="@disableModify" @onclick="() =>
ModifyTourBooking(t.bookingId)">Modify</button>
                        </td>
                    </tr>
                }
            }
        </tbody>
    </table>
}
```

```

                <td width="5%">
                    <button class="btn btn-danger"
disabled="@disableCancel" @onclick="() =>
CancelTourBooking(t.bookingId)">Cancel</button>
                </td>
                <td width="5%">
                    <button class="btn btn-success"
disabled="@t.paidInfull" @onclick="() => ShowTourPayPopup(t.bookingId,
t.DepositAmountPaid, t.TotalPrice)">Pay Remainder</button>
                </td>
            </tr>
        }
    }
</tbody>
</table>

if (hasCancelledBookingsTour)
{
    <h3>Cancelled Tours for late payment</h3>
    <table class="table">
        <thead>
            <tr>
                <th>Tour</th>
                <th>Number of People</th>
                <th>Start Date</th>
                <th>Check Out Date</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var t in tourBookings)
            {
                if (t.IsCancelled)
                {
                    <tr>
                        <td width="5%">@t.TourName</td>
                        <td width="5%">@t.NumberOfGuests</td>
                        <td
width="5%">@t.CommencementDate.ToString("dd/MM/yyyy")</td>
                        <td>
                    </tr>
                }
            }
        </tbody>
    </table>

    <br />
    <br />
}
}
}
@{

```

```

    if (showTourPayPopup)
    {
        var remainder = totalToPay - currentlyPaid;

        <div class="backgroundPopupBox">
            <div class="popupCreate">
                <h3>Pay Remainder</h3>
                <p>Are you sure you want to pay the remainder of this tour
booking? It will cost you £@remainder </p>
                <button class="btn btn-success" @onclick="() =>
PayTourRemainder(bookingId)">Yes</button>
                <button class="btn btn-danger"
@onclick="ClosePopup">No</button>
            </div>
        </div>
    }
}

@if (packageBookings is null)
{
    <h3>Loading Packages...</h3>
}
else
{
    if (packageBookings.Count == 0)
    {
        <h3>You have no Package bookings</h3>
    }
    else
    {
        <h2>Packages</h2>
        <table class="table">
            <thead>
                <tr>
                    <th>Tour</th>
                    <th>Start Date</th>
                    <th>End Date</th>
                    <th>Number of Guests</th>
                    <th>Hotel</th>
                    <th>Room Type</th>
                    <th>Check In Date</th>
                    <th>Check Out Date</th>
                    <th>Number of Nights</th>
                    <th>Amount Paid so Far</th>
                    <th>Total Cost</th>
                    <th>Payment Due Date</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var p in packageBookings)
                {

```

```

        if (!p.IsCancelled)
        {
            disableCancel = false;
            disableModify = false;

            var today = DateTime.Now;

            TimeSpan datediff = new TimeSpan();

            if(p.CheckIn > p.CommencementDate)
            {
                datediff = p.CommencementDate - today;
            }
            else
            {
                datediff = p.CheckIn - today;
            }

            if (datediff.Days < 5)
            {
                disableCancel = true;
            }

            if (datediff.Days < 14)
            {
                disableModify = true;
            }

            <tr>
                <td width="5%">@p.TourName</td>
                <td
width="5%">@p.CommencementDate.ToString("dd/MM/yyyy")</td>
                <td
width="5%">@p.EndDate.ToString("dd/MM/yyyy")</td>
                <td width="5%">@p.NumberOfGuests people</td>
                <td width="5%">@p.hotelName</td>
                <td width="5%">@p.RoomType Room</td>
                <td
width="5%">@p.CheckIn.ToString("dd/MM/yyyy")</td>
                <td
width="5%">@p.CheckOut.ToString("dd/MM/yyyy")</td>
                <td width="5%">@p.NumberOfNights Nights</td>
                <td width="5%">£@p.DepositAmountPaid</td>
                <td width="5%">£@p.TotalPrice</td>
                <td
width="5%">@p.PaymentDueDate.ToString("dd/MM/yyyy")</td>
                <td width="5%">
                    <button class="btn btn-primary"
disabled="@disableModify" @onclick="() =>
ModifyPackageBooking(p.bookingId)">Modify</button>
                </td>
                <td width="5%">

```

```

                <button class="btn btn-danger"
disabled="@disableCancel" @onclick="() =>
CancelPackageBooking(p.bookingId)">Cancel</button>
            </td>
            <td width="5%">
                <button class="btn btn-success"
disabled="@p.paidInfull" @onclick="() => ShowPackagePayPopup(p.bookingId,
p.DepositAmountPaid, p.TotalPrice)">Pay Remainder</button>
            </td>
        </tr>
    }
}
</tbody>
</table>

if (hasCancelledPackage)
{
    <h3>Cancelled Packages for late payment</h3>
    <table class="table">
        <thead>
            <tr>
                <th>Tour</th>
                <th>Number of People</th>
                <th>Start Date</th>
                <th>Check Out Date</th>
                <th>Hotel</th>
                <th>Room Type</th>
                <th>Check In Date</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var p in packageBookings)
            {
                if (p.IsCancelled)
                {
                    <tr>
                        <td width="5%">@p.TourName</td>
                        <td width="5%">@p.NumberOfGuests</td>
                        <td
width="5%">@p.CommencementDate.ToString("dd/MM/yyyy")</td>
                        <td width="5%">@p.hotelName</td>
                        <td width="5%">@p.RoomType Room</td>
                        <td
width="5%">@p.CheckIn.ToString("dd/MM/yyyy")</td>
                    </tr>
                }
            }
        </tbody>
    </table>

    <br />
    <br />

```

```

    }
  }
}

@{
  if (showPackagePayPopup)
  {
    var remainder = totalToPay - currentlyPaid;

    <div class="backgroundPopupBox">
      <div class="popupCreate">
        <h3>Pay Remainder</h3>
        <p>Are you sure you want to pay the remainder of this tour
booking? It will cost you £@remainder </p>
        <button class="btn btn-success" @onclick="() =>
PayPackageRemainder(bookingId)">Yes</button>
        <button class="btn btn-danger"
@onclick="ClosePopup">No</button>
      </div>
    </div>
  }
}

```

```

@code {

  [CascadingParameter]
  private Task<AuthenticationState>? authenticationState { get; set; }
  private List<HotelBookingViewModel>? hotelBookings;
  private List<TourBookingViewModel>? tourBookings;
  private List<PackageBookingViewModel>? packageBookings;
  private string userId;
  private bool disableCancel = false;
  private bool disableModify = false;
  private bool showHotelPayPopup = false;
  private bool showTourPayPopup = false;
  private bool showPackagePayPopup = false;
  private string bookingId;
  private decimal currentlyPaid, totalToPay;
  private string bookingType;
  private bool hasCancelledBookingsHotel, hasCancelledBookingsTour,
hasCancelledPackage = false;

  protected override async Task OnInitializedAsync()
  {
    var authState = await authenticationState;
    var user = authState?.User;
    userId = user?.FindFirst(c => c.Type ==
ClaimTypes.NameIdentifier)?.Value;

```



```
        var result = await
http.GetFromJsonAsync<List<HotelBookingViewModel>>
($"api/bookings/hotel/userbooking?userId={userId}");
        if (result != null)
        {
            hotelBookings = result;
        }

        var result2 = await
http.GetFromJsonAsync<List<TourBookingViewModel>>
($"api/bookings/tour/userbooking?userId={userId}");
        if (result2 != null)
        {
            tourBookings = result2;
        }

        var result3 = await
http.GetFromJsonAsync<List<PackageBookingViewModel>>
($"api/bookings/package/userbooking?userId={userId}");
        if (result3 != null)
        {
            packageBookings = result3;
        }

        if (hotelBookings is not null && hotelBookings.Count > 0)
        {
            foreach (var h in hotelBookings)
            {
                if (h.IsCancelled)
                {
                    hasCancelledBookingsHotel = true;
                    break;
                }
            }
        }

        if (tourBookings is not null && tourBookings.Count > 0)
        {
            foreach (var t in tourBookings)
            {
                if (t.IsCancelled)
                {
                    hasCancelledBookingsTour = true;
                    break;
                }
            }
        }

        if (packageBookings is not null && packageBookings.Count > 0)
        {
            foreach (var p in packageBookings)
            {
                if (p.IsCancelled)
```

```
        {
            hasCancelledPackage = true;
            break;
        }
    }
}

async Task CancelHotelBooking(string id)
{
    var result = await http.DeleteAsync($"api/bookings/hotel/{id}");
    NavigationManager.NavigateTo("/mybookings", true);
}

private void ModifyHotelBooking(string id)
{
    NavigationManager.NavigateTo($"/editbooking/hotel/{id}");
}

private void ShowHotelPayPopup(string id, decimal depoist, decimal
total)
{
    showHotelPayPopup = true;
    bookingId = id;
    currentlyPaid = depoist;
    totalToPay = total;
}

private void PayHotelRemainder(string id, decimal remainder)
{
    var result = http.PutAsync($"api/bookings/hotel/payment/{id}?
paymentRemainder={remainder}", null);
    NavigationManager.NavigateTo("/mybookings", true);
}

private void ClosePopup()
{
    showHotelPayPopup = false;
    showTourPayPopup = false;
}

async Task CancelTourBooking(string id)
{
    var result = await http.DeleteAsync($"api/bookings/tour/{id}");
    NavigationManager.NavigateTo("/mybookings", true);
}

private void ModifyTourBooking(string id)
{
    NavigationManager.NavigateTo($"/editbooking/tour/{id}");
}

private void ShowTourPayPopup(string id, decimal depoist, decimal
total)
```

```

    {
        showTourPayPopup = true;
        bookingId = id;
        currentlyPaid = depoist;
        totalToPay = total;
    }

    private void PayTourRemainder(string id)
    {
        var result = http.PutAsync($"api/bookings/tour/payment/{id}",
null);
        NavigationManager.NavigateTo("/mybookings", true);
    }

    async Task CancelPackageBooking(string id)
    {
        var result = await http.DeleteAsync($"api/bookings/package/{id}");
        NavigationManager.NavigateTo("/mybookings", true);
    }

    private void ModifyPackageBooking(string id)
    {
        NavigationManager.NavigateTo($"/editbooking/package/{id}");
    }

    private void ShowPackagePayPopup(string id, decimal depoist, decimal
total)
    {
        showPackagePayPopup = true;
        bookingId = id;
        currentlyPaid = depoist;
        totalToPay = total;
    }

    private void PayPackageRemainder(string id)
    {
        var result = http.PutAsync($"api/bookings/package/payment/{id}",
null);
        NavigationManager.NavigateTo("/mybookings", true);
    }
}

```

### BlazorHotelBooking/Client/Pages/Packages.razor

```

@page "/packages"
@using BlazorHotelBooking.Shared
@using Microsoft.AspNetCore.Authorization
@using System.Security.Claims
@Inject HttpClient http
[Authorize]

```

```

@Inject NavigationManager NavigationManager

<h3>Packages </h3>

@if (hotels.Count <= 0 || tours.Count <= 0)
{
    <span> Loading...</span>
}
else
{
    <EditForm Model=@newBooking OnValidSubmit=ShowPackagePopup>
        <DataAnnotationsValidator />
        @* <ValidationSummary /> *@

        <div class="form-group">
            <label class="control-label">Choose Hotel</label>
            <InputSelect id="Hotel" @bind-Value="newBooking.HotelId"
placeholder="Hotel">

                @foreach (var hotel in hotels)
                {
                    <option value="@hotel.Id">@hotel.Name</option>
                }
            </InputSelect>
        </div>
        <div class="form-group">
            <label class="control-label">Choose Tour</label>
            <InputSelect id="Tour" @bind-Value="newBooking.TourId"
placeholder="Hotel">
                @foreach (var tour in tours)
                {
                    <option value="@tour.Id">@tour.Name</option>
                }
            </InputSelect>
        </div>
        <button type="button" class="btn btn-success"
@onclick="ShowPackagePopup">Book</button>
    </EditForm>

    @if (showBookingForm)
    {

        <div class="backgroundPopupBox">
            <div class="popupCreate">
                <EditForm Model=@newBooking OnValidSubmit=BookPackage>
                    <DataAnnotationsValidator />
                    @* <ValidationSummary /> *@

                    <div class="form-group">

```

```

        <label class="control-label">Check-In Date</label>
        <InputDate @bind-Value="newBooking.HotelCheckIn"
min="@min" max="@max" Placeholder="Enter Date" />
        <ValidationMessage For="@((() =>
newBooking.HotelCheckIn)" />
    </div>
    <div class="form-group mt-3 col-sm-10">
        <label class="control-label">Room Type</label>
        <InputSelect id="roomType" @bind-
Value="newBooking.RoomType" placeholder="Room Type">
            <option value="">---</option>
            @foreach (var room in roomType)
            {
                <option value="@room">@room</option>
            }
        </InputSelect>
        <ValidationMessage For="@((() => newBooking.RoomType)"
/>
    </div>
    <div class="form-group mt-3">
        <label class="control-label">Number of nights</label>
        <InputNumber id="NumOfNights" @bind-
Value="newBooking.NumberOfNights" class="form-control" />
        <ValidationMessage For="@((() =>
newBooking.NumberOfNights)" />
    </div>

    @{
        switch (newBooking.RoomType)
        {
            case "Single":
                newBooking.TotalPrice =
newBooking.NumberOfNights * selectedHotel.SBPrice;
                break;
            case "Double":
                newBooking.TotalPrice =
newBooking.NumberOfNights * selectedHotel.DBPrice;
                break;
            case "Family":
                newBooking.TotalPrice =
newBooking.NumberOfNights * selectedHotel.FamPrice;
                break;
        }
        newBooking.HotelCheckOut =
newBooking.HotelCheckIn.AddDays(newBooking.NumberOfNights);
    }

    <div>Check In Date:
    @newBooking.HotelCheckIn.ToString("dd/MM/yyyy")</div>
    <div>Check Out Date:
    @newBooking.HotelCheckOut.ToString("dd/MM/yyyy")</div>

```

```

        <div class="form-group">
            <label class="control-label">Tour Start Date</label>
            <InputDate @bind-Value="newBooking.TourStartDate"
min="@min" max="@max" Placeholder="Enter Date" />
            <ValidationMessage For="@(() =>
newBooking.TourStartDate)" />
        </div>
        <div class="form-group mt-3">
            <label class="control-label">Number of guests</label>
            <InputNumber id="NumOfGuests" @bind-
Value="newBooking.NumberOfPeopleOnTour" class="form-control" />
            <ValidationMessage For="@(() =>
newBooking.NumberOfPeopleOnTour)" />
        </div>

        <div>Tour start Date:
@newBooking.TourStartDate.ToString("dd/MM/yyyy")</div>
        <div>Tour End Date:
@newBooking.TourEndDate.ToString("dd/MM/yyyy")</div>

        @{
            newBooking.TourEndDate =
newBooking.TourStartDate.AddDays(selectedTour.DurationInDays - 1);
            newBooking.TotalPrice += selectedTour.Cost;

            switch(newBooking.RoomType)
            {
                case "Single":
                    newBooking.TotalPrice =
newBooking.TotalPrice * (decimal)0.9;
                    <p>(Single room includes 10% discount)</p>
                    break;
                case "Double":
                    newBooking.TotalPrice =
newBooking.TotalPrice * (decimal)0.8;
                    <p>(Double room includes 20% discount)</p>
                    break;
                case "Family":
                    newBooking.TotalPrice =
newBooking.TotalPrice * (decimal)0.6;
                    <p>(Family room includes 40% discount)</p>
                    break;
            }

            newBooking.DepositAmountPaid =
newBooking.TotalPrice / 5;
        }

        <div>Total Price is £@newBooking.TotalPrice</div>
        <div>Total to pay today (20%)
£@newBooking.DepositAmountPaid</div>

```

```

        @if(showHotelOverlap)
        {
            <div class="alert alert-danger" role="alert">
                <p>There are no more spaces for this room on these
dates please select new dates.</p>
            </div>
        }

        @if(showTourOverlap)
        {
            <div class="alert alert-danger" role="alert">
                <p>There are no more spaces left in this tour for
that many people on these dates please select new dates or less people.</p>
            </div>
        }

        <button type="button" class="btn btn-success"
@onclick="BookPackage">Book</button>

    </EditForm>

    <div class="form-group">
        <button class="btn btn-danger"
@onclick="ClosePopup">Cancel</button>
    </div>
</div>
</div>

}
}

@code {
    List<Hotel> hotels = new List<Hotel>();
    List<Tour> tours = new List<Tour>();
    Hotel selectedHotel = new Hotel();
    Tour selectedTour = new Tour();
    private PackageBooking newBooking = new PackageBooking();
    private bool showBookingForm = false;
    List<string> roomType = new List<string>() { "Single", "Double",
"Family" };
    private string min;
    private string max;
    private string userId;
    private int numOfTourOverlap, numOfHotelOverlap;
    private bool showHotelOverlap, showTourOverlap = false;

    [CascadingParameter]
    private Task<AuthenticationState>? authenticationState { get; set; }

    protected override async Task OnInitializedAsync()
    {

```

```
        min =
DateOnly.FromDateTime(DateTime.Now.Date.AddMonths(2)).ToString("yyyy-MM-
dd");
        max =
DateOnly.FromDateTime(DateTime.Now.Date.AddYears(5)).ToString("yyyy-MM-
dd");

        var authState = await authenticationState;
        var user = authState?.User;
        userId = user?.FindFirst(c => c.Type ==
ClaimTypes.NameIdentifier)?.Value;

        var result = await http.GetFromJsonAsync<List<Hotel>>
("/api/hotel");
        if (result != null)
        {
            hotels = result;
        }

        var result2 = await http.GetFromJsonAsync<List<Tour>>("/api/tour");
        if (result2 != null)
        {
            tours = result2;
        }
    }

    private async void Search(ChangeEventArgs args)
    {
        var searchTerm = (string)args.Value;

        var result = await http.GetFromJsonAsync<List<Hotel>>
("/api/hotel");
        // make it case insensitive to search hotels

        hotels = result.Where(x => x.Name.IndexOf(searchTerm,
StringComparison.OrdinalIgnoreCase) >= 0)
            .OrderByDescending(x => x.Id)
            .ToList();

        StateHasChanged();
    }

    private void ShowPackagePopup()
    {
        showBookingForm = true;

        selectedHotel = hotels.FirstOrDefault(h => h.Id ==
newBooking.HotelId);
        selectedTour = tours.FirstOrDefault(h => h.Id ==
newBooking.TourId);
    }
}
```



```
private void ClosePopup()
{
    showBookingForm = false;
}

async Task BookPackage()
{
    numOfHotelOverlap = await http.GetFromJsonAsync<int>
($" /api/bookings/hotel/overlap?checkIn={newBooking.HotelCheckIn}&checkOut=
{newBooking.HotelCheckOut}&hotelId={selectedHotel.Id}&roomType=
{newBooking.RoomType}");
    numOfTourOverlap = await http.GetFromJsonAsync<int>
($" /api/bookings/tour/overlap?start={newBooking.TourStartDate}&end=
{newBooking.TourEndDate}&tourId={selectedTour.Id}");

    if ((selectedTour.MaxNumberOfGuests < numOfTourOverlap +
newBooking.NumberOfPeopleOnTour) && (numOfHotelOverlap > 20))
    {
        showHotelOverlap = true;
        showTourOverlap = true;
    }
    else
    {
        newBooking.UserId = userId;

        if (newBooking.TourStartDate < newBooking.HotelCheckIn)
        {
            newBooking.PaymentDueDate =
newBooking.TourStartDate.AddDays(-28);
        }
        else
        {
            newBooking.PaymentDueDate =
newBooking.HotelCheckIn.AddDays(-28);
        }

        await http.PostAsJsonAsync("/api/bookings/package/book",
newBooking);

        ClosePopup();
        NavigationManager.NavigateTo("/mybookings");
    }
}
}
```

### BlazorHotelBooking/Client/Pages/PaymentPage.razor

```
@page "/payments"
@using BlazorHotelBooking.Shared
```

```
@using Microsoft.AspNetCore.Authorization
@using System.Security.Claims
@inject HttpClient http
@inject NavigationManager NavigationManager
@attribute [Authorize]

@if (payments is null)
{
    <span>Loading Payments...</span>
}
else
{
    <h5>Payments</h5>

    <table class="table">
        <thead>
            <tr>
                <th>Booking ID</th>
                <th>Booking Type</th>
                <th>Payment Type</th>
                <th>Payment Date</th>
                <th>Amount Paid</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var p in payments)
            {
                <tr>
                    <td width="5%">@p.bookingId</td>
                    <td width="5%">@p.bookingType</td>
                    <td width="5%">@p.paymentType</td>
                    <td width="5%">@p.PaymentDate</td>
                    <td width="5%">@p.AmountPaid</td>
                </tr>
            }
        </tbody>
    </table>
}

@code {
    List<Payments> payments = new List<Payments>();

    [CascadingParameter]
    private Task<AuthenticationState>? authenticationState { get; set; }
    private string userId;

    protected override async Task OnInitializedAsync()
    {
        var authState = await authenticationState;
        var user = authState?.User;
        userId = user?.FindFirst(c => c.Type ==
```

```
ClaimTypes.NameIdentifier)?.Value;

        var result = await http.GetFromJsonAsync<List<Payments>>
($" /api/Payment/{userId}");
        if (result != null)
        {
            payments = result;
        }
    }
}
```

### BlazorHotelBooking/Client/Pages/Register.razor

```
@page "/register"
@using BlazorHotelBooking.Client.Service;
@using BlazorHotelBooking.Shared.Models;
@inject IAuthService AuthService
@inject NavigationManager NavigationManager

<h1>Register</h1>

@if (ShowErrors)
{
    <div class="alert alert-danger" role="alert">
        <ul>
            @foreach (var error in Errors)
            {
                <li>@error</li>
            }
        </ul>
    </div>
}

<div class="card">
    <div class="card-body">
        <h5 class="card-title">Please Enter Registration Details</h5>
        <EditForm Model="registerModel" OnValidSubmit="HandleRegistration">
            <DataAnnotationsValidator />
            <ValidationSummary />

            <div class="form-group mt-2">
                <label for="email">Email address</label>
                <InputText Id="email" class="form-control" @bind-
Value="registerModel.Email" />
                <ValidationMessage For="@(() => registerModel.Email)" />
            </div>
            <div class="form-group mt-2">
                <label for="passportnum">Passport Number</label>
                <InputText Id="passportnum" class="form-control" @bind-
Value="registerModel.PassportNumber" />
            </div>
        </EditForm>
    </div>
</div>
```

```

        <ValidationMessage For="@(() =>
registerModel.PassportNumber)" />
    </div>
    <div class="form-group mt-2">
        <label for="phonenum">Contact Number</label>
        <InputText Id="passportnum" class="form-control" @bind-
Value="registerModel.PhoneNumber" />
        <ValidationMessage For="@(() => registerModel.PhoneNumber)"
/>
    </div>
    <div class="form-group mt-2">
        <label for="password">Password</label>
        <InputText Id="password" type="password" class="form-
control" @bind-Value="registerModel.Password" />
        <ValidationMessage For="@(() => registerModel.Password)" />
    </div>
    <div class="form-group mt-2">
        <label for="password">Confirm Password</label>
        <InputText Id="password" type="password" class="form-
control" @bind-Value="registerModel.ConfirmPassword" />
        <ValidationMessage For="@(() =>
registerModel.ConfirmPassword)" />
    </div>
    <button type="submit" class="btn btn-primary mt-
2">Register</button>
</EditForm>
</div>
</div>

```

```
@code {
```

```

    private RegisterModel registerModel = new RegisterModel();
    private bool ShowErrors;
    private IEnumerable<string>? Errors;

    private async Task HandleRegistration()
    {
        ShowErrors = false;

        var result = await AuthService.Register(registerModel);
        if (result.Successful)
        {
            NavigationManager.NavigateTo("/login");
        }
        else
        {
            ShowErrors = true;
            Errors = result.Errors;
        }
    }
}

```

## BlazorHotelBooking/Client/Pages/TourDetails.razor

```
@page "/tours/{id}"
@using BlazorHotelBooking.Shared;
@using Microsoft.AspNetCore.Authorization;
@using Microsoft.AspNetCore.Http;
@using Syncfusion.Blazor.DropDowns;
@using System.ComponentModel.DataAnnotations;
@using System.Security.Claims
@Inject IHttpContextAccessor httpContextAccessor
@Inject HttpClient http
@Inject NavigationManager NavigationManager
[Attribute(Authorize)]

<div class="media">
    <div class="media-body">
        <h2 class="mb-0">@selectedTour.Name</h2>
        <p>@selectedTour.Description</p>
        <p>Max Number of Guests: @selectedTour.MaxNumberOfGuests</p>
        <h4 class="price">
            @selectedTour.Cost
        </h4>
        <button onclick="ShowBookingForm" class="btn btn-primary"><i
class="oi oi-cart">&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;& Book Tour</i></button>
    </div>
</div>

@if(showBookingForm)
{
    <div class="backgroundPopupBox">
        <div class="popupCreate">
            <EditForm Model=@newBooking OnValidSubmit=BookTour>
                <DataAnnotationsValidator />
                *@ <ValidationSummary /> *@

                <div class="form-group">
                    <label class="control-label">Commencement Date</label>
                    <InputDate @bind-Value="newBooking.CommencementDate"
min="@min" max="@max" Placeholder="Enter Date" />
                    <ValidationMessage For="@(( ) =>
newBooking.CommencementDate)" />
                </div>
                <div class="form-group mt-3">
                    <label class="control-label">Number of guests</label>
                    <InputNumber id="NumOfGuests" @bind-
Value="newBooking.NumberOfPeople" class="form-control" />
                    <ValidationMessage For="@(( ) =>
newBooking.NumberOfPeople)" />
                </div>
            </EditForm>
        </div>
    </div>
}
```

```

        newBooking.TotalPrice = selectedTour.Cost;
        newBooking.DepositAmountPaid = newBooking.TotalPrice /
5;

        newBooking.EndDate =
newBooking.CommencementDate.AddDays(selectedTour.DurationInDays - 1);
    }

    <div>Tour start Date:
    @newBooking.CommencementDate.ToString("dd/MM/yyyy")</div>
    <div>Tour End Date:
    @newBooking.EndDate.ToString("dd/MM/yyyy")</div>

    <div>Total Price is £@newBooking.TotalPrice</div>
    <div>Total to pay today (20%)
    £@newBooking.DepositAmountPaid</div>

    @if(showOverlap)
    {
        <div class="alert alert-danger" role="alert">
            <p>There are no more spaces left in this tour for
that many people on these dates please select new dates or less people.</p>
        </div>
    }

    <button type="button" class="btn btn-success"
    @onclick="BookTour">Book</button>

    </EditForm>

    <div class="form-group">
        <button class="btn btn-danger"
    @onclick="ClosePopup">Cancel</button>
    </div>
</div>

}

@code {
    private Tour selectedTour = new Tour();
    private TourBooking newBooking = new TourBooking();
    private bool showBookingForm = false;
    private string min;
    private string max;
    private string userId;
    private int numOfOverlap;
    private bool showOverlap = false;

    [CascadingParameter]

```

```
private Task<AuthenticationState>? authenticationState { get; set; }

[Parameter]
public string? Id { get; set; }

List<Tour> tours = new List<Tour>();

protected override async Task OnInitializedAsync()
{
    min =
DateOnly.FromDateTime(DateTime.Now.Date.AddMonths(2)).ToString("yyyy-MM-
dd");
    max =
DateOnly.FromDateTime(DateTime.Now.Date.AddYears(5)).ToString("yyyy-MM-
dd");

    var authState = await authenticationState;
    var user = authState?.User;
    userId = user?.FindFirst(c => c.Type ==
ClaimTypes.NameIdentifier)?.Value;

    var result = await http.GetFromJsonAsync<List<Tour>>("/api/tour");
    if (result != null)
    {
        tours = result;
    }

    selectedTour = tours.FirstOrDefault(h => h.Id == Int32.Parse(Id));
}

private void ShowBookingForm()
{
    showBookingForm = true;
}

async Task BookTour()
{
    // Need to change for tours

    // code to check if the newbooking overlaps with existing bookings
    numOfOverlap = await http.GetFromJsonAsync<int>
($"api/bookings/tour/overlap?start={newBooking.CommencementDate}&end=
{newBooking.EndDate}&tourId={selectedTour.Id}");

    if (selectedTour.MaxNumberOfGuests < numOfOverlap +
newBooking.NumberOfPeople)
    {
        showOverlap = true;
    }
    else
    {
        newBooking.TourId = selectedTour.Id;
        newBooking.UserId = userId;
    }
}
```

```

        newBooking.PaymentDueDate =
newBooking.CommencementDate.AddDays(-28);

        await http.PostAsJsonAsync("/api/bookings/tour/book",
newBooking);

        ClosePopup();
        NavigationManager.NavigateTo("/mybookings");
    }
}

private void ClosePopup()
{
    showBookingForm = false;
}
}

```

### BlazorHotelBooking/Client/Pages/TourEdit.razor

```

@page "/touredit"
@page "/touredit/{id:int}"
@using BlazorHotelBooking.Shared
@using Microsoft.AspNetCore.Authorization
@inject HttpClient http
@inject NavigationManager NavigationManager
@attribute [Authorize(Roles="Admin")]

@if (Id is null)
{
    <PageTitle>Add New Tour</PageTitle>
    <h3>Add New Tour</h3>
}
else
{
    <PageTitle> Edit @selectedTour.Name </PageTitle>
    <h3>@selectedTour.Name</h3>
}

<EditForm Model="selectedTour" OnSubmit="HandleSubmit">

    <div>
        <label for="Id">Id</label>
        <InputNumber id="Id" @bind-Value="selectedTour.Id" class="form-
control" disabled="true" />
    </div>
    <div>
        <label for="Name">Name</label>
        <InputText id="Name" @bind-Value="selectedTour.Name" class="form-
control" />

```



```

</div>
<div>
    <label for="Cost">Cost</label>
    <InputNumber id="Cost" @bind-Value="selectedTour.Cost" class="form-
control" />
</div>
<div>
    <label for="Duration">Duration</label>
    <InputNumber id="Duration" @bind-
Value="selectedTour.DurationInDays" class="form-control" />
</div>
<div>
    <label for="MaxGuests">Max Number of Guests</label>
    <InputNumber id="MaxGuests" @bind-
Value="selectedTour.MaxNumberOfGuests" class="form-control" />
</div>
<div>
    <label for="Description">Description</label>
    <InputText id="Description" @bind-Value="selectedTour.Description"
class="form-control" />
</div>
<button type="submit" class="btn btn-primary">Save</button>
@if(Id is not null)
{
    <button type="button" class="btn btn-danger"
@onclick="DeleteTour">Delete</button>
}
</EditForm>

```

```

@code {
    [Parameter]
    public int? Id { get; set; }

    private Tour selectedTour = new Tour { Name = "New Tour" };

    protected override async Task OnParametersSetAsync()
    {
        if (Id is not null)
        {
            var result = await http.GetFromJsonAsync<Tour>
($"api/tour/{Id}");
            if (result is not null)
            {
                selectedTour = result;
            }
        }
    }

    async Task DeleteTour()
    {
        await http.DeleteAsync($"api/Admin/tour/{Id}");
        NavigationManager.NavigateTo("/admin");
    }
}

```

```
async Task HandleSubmit()
{
    if(Id is null)
    {
        await http.PostAsJsonAsync("/api/Admin/tour", selectedTour);
    }
    else
    {
        await http.PutAsJsonAsync($"api/Admin/tour/{Id}",
selectedTour);
    }
    NavigationManager.NavigateTo("/admin");
}
}
```

### BlazorHotelBooking/Client/Pages/Tours.razor

```
@page "/tours"
@using BlazorHotelBooking.Shared
@using Microsoft.AspNetCore.Authorization
@inject HttpClient http
@attribute [Authorize]

<h3>Tours</h3>

<input @oninput="Search" placeholder="Search..." class="p-3" />

<br />
<br />
<EditForm Model="startDate" OnSubmit="DateSearch">
    <div class="form-group">
        <label class="control-label">Start Date</label>
        <InputDate @bind-Value="startDate" min="@min" max="@max"
Placeholder="Enter Date" />
        <label class="control-label">Number Of People</label>
        <InputNumber @bind-Value="numOfPeople" Placeholder="Number of
People" />
    </div>
    <button type="button" class="btn btn-success"
@onclick="DateSearch">Search</button>
</EditForm>

@if (tours.Count <= 0)
{
    <span> Loading tours...</span>
}
else
{
    @foreach (var tour in tours)
```

```

    {
        <li class="media my-3">
            <div class="media-body">
                <a href="/tours/@tour.Id">
                    <h4 class="mb-0">@tour.Name</h4>
                </a>
                <p>@tour.Description</p>
                <p>Max Number of Guests: @tour.MaxNumberOfGuests</p>
                <h6 class="price">
                    £@tour.Cost
                </h6>
            </div>
        </li>
    }
}

@code {
    List<Tour> tours = new List<Tour>();
    private DateTime startDate = DateTime.Now.Date.AddMonths(2);
    private DateTime endDate;
    private int numOfOverlap;
    private int numOfPeople;
    private string min;
    private string max;

    protected override async Task OnInitializedAsync()
    {
        min =
        DateOnly.FromDateTime(DateTime.Now.Date.AddMonths(2)).ToString("yyyy-MM-
dd");
        max =
        DateOnly.FromDateTime(DateTime.Now.Date.AddYears(5)).ToString("yyyy-MM-
dd");

        var result = await http.GetFromJsonAsync<List<Tour>>("/api/tour");
        if (result != null)
        {
            tours = result;
        }
    }

    private async void Search(ChangeEventArgs args)
    {
        var searchTerm = (string)args.Value;

        var result = await http.GetFromJsonAsync<List<Tour>>("/api/tour");
        // make it case insensitive to search hotels

        tours = result.Where(x => x.Name.IndexOf(searchTerm,
StringComparison.OrdinalIgnoreCase) >= 0)
            .OrderByDescending(x => x.Id)
            .ToList();
    }
}

```

```

        StateHasChanged();
    }

    private async void DateSearch()
    {
        var result = await http.GetFromJsonAsync<List<Tour>>("/api/tour");
        var tempList = new List<Tour>();

        if (result != null)
        {
            foreach (var tour in result.ToList())
            {
                endDate = startDate.AddDays(tour.DurationInDays);
                numOfOverlap = await http.GetFromJsonAsync<int>
($"/api/bookings/tour/overlap?start={startDate}&end={endDate}&tourId=
{tour.Id}");

                if (!(tour.MaxNumberOfGuests < numOfOverlap + numOfPeople))
                {
                    tempList.Add(tour);
                }
            }

            tours = tempList;
        }

        StateHasChanged();
    }
}

```

### BlazorHotelBooking/Client/Service/AuthService.cs

```

using Blazored.LocalStorage;
using BlazorHotelBooking.Client.Auth;
using BlazorHotelBooking.Shared.Models;
using Microsoft.AspNetCore.Components.Authorization;
using System.Net.Http.Headers;
using System.Net.Http.Json;
using System.Text;
using System.Text.Json;

namespace BlazorHotelBooking.Client.Service
{
    public class AuthService : IAuthService
    {
        private readonly HttpClient _httpClient;
        private readonly AuthenticationStateProvider _authStateProvider;
        private readonly ILocalStorageService _localStorage;

        public AuthService(HttpClient httpClient,

```

```
AuthenticationStateProvider authStateProvider, ILocalStorageService
localStorage)
{
    _httpClient = httpClient;
    _authStateProvider = authStateProvider;
    _localStorage = localStorage;
}

public async Task<RegisterResult> Register(RegisterModel
registerModel)
{
    HttpResponseMessage result = await
_httpClient.PostAsJsonAsync("api/register", registerModel);
    if (!result.IsSuccessStatusCode)
        return new RegisterResult { Successful = false, Errors =
new List<string> { "Error occurred" } };
    return new RegisterResult { Successful = true, Errors = new
List<string> { "Account Created successfully" } };
}

public async Task<LoginResult> Login(LoginModel loginModel)
{
    string JsonLogin = JsonSerializer.Serialize(loginModel);
    HttpResponseMessage response = await
_httpClient.PostAsync("api/login",
        new StringContent(JsonLogin, Encoding.UTF8,
"application/json"));

    LoginResult? loginResult =
JsonSerializer.Deserialize<LoginResult>(
        await response.Content.ReadAsStringAsync(),
        new JsonSerializerOptions { PropertyNameCaseInsensitive =
true });

    if (!response.IsSuccessStatusCode) return loginResult!;

    await _localStorage.SetItemAsync("authToken",
loginResult!.Token);

    ((APIAuthStateProvider)_authStateProvider).MarkUserAsAuthenticated(loginRes
ult.Token!);
    _httpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("bearer", loginResult.Token);

    return loginResult;
}

public async Task Logout()
{
    await _localStorage.RemoveItemAsync("authToken");

    ((APIAuthStateProvider)_authStateProvider).MarkUserAsLoggedOut();
    _httpClient.DefaultRequestHeaders.Authorization = null;
}
```

```
}  
}
```

### BlazorHotelBooking/blob/master/BlazorHotelBooking/Client/Service/IAuthService.cs

```
using BlazorHotelBooking.Shared.Models;  
  
namespace BlazorHotelBooking.Client.Service  
{  
    public interface IAuthService  
    {  
        Task<RegisterResult> Register(RegisterModel registerModel);  
        Task<LoginResult> Login(LoginModel loginModel);  
        Task Logout();  
    }  
}
```

### BlazorHotelBooking/Client/Program.cs

```
using Blazored.LocalStorage;  
using BlazorHotelBooking.Client;  
using BlazorHotelBooking.Client.Auth;  
using BlazorHotelBooking.Client.Service;  
using Microsoft.AspNetCore.Components.Authorization;  
using Microsoft.AspNetCore.Components.Web;  
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;  
using Syncfusion.Blazor;  
  
WebAssemblyHostBuilder builder =  
    WebAssemblyHostBuilder.CreateDefault(args);  
builder.RootComponents.Add<App>("#app");  
builder.RootComponents.Add<HeadOutlet>("head::after");  
builder.Services.AddBlazoredLocalStorage();  
builder.Services.AddAuthorizationCore();  
builder.Services.AddScoped<AuthenticationStateProvider,  
    APIAuthStateProvider>();  
builder.Services.AddScoped<IAAuthService, AuthService>();  
builder.Services.AddSyncfusionBlazor();  
  
builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new  
    Uri(builder.HostEnvironment.BaseAddress) });  
  
await builder.Build().RunAsync();
```

### BlazorHotelBooking/Server/Controllers/AdminController.cs

```
using BlazorHotelBooking.Server.Data;
using BlazorHotelBooking.Shared;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace BlazorHotelBooking.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [Authorize(Roles = "Admin")]

    public class AdminController : ControllerBase
    {
        private readonly DataContext _context;

        public AdminController(DataContext context)
        {
            _context = context;
        }

        // Admin for Hotels

        [HttpGet("hotel")]
        public async Task<ActionResult<List<Hotel>>> GetAllHotels()
        {
            List<Hotel> list = await _context.Hotels.ToListAsync();

            return Ok(list);
        }

        [HttpGet("hotel/{id}")]
        public async Task<ActionResult<Hotel>> GetHotel(int id)
        {
            Hotel? dbhotel = await _context.Hotels.FindAsync(id);

            if (dbhotel == null)
            {
                return NotFound("This hotel does not exist");
            }

            return Ok(dbhotel);
        }

        [HttpPost("hotel")]
        public async Task<ActionResult<List<Hotel>>> AddHotel(Hotel hotl)
        {
            _context.Hotels.Add(hotl);
            await _context.SaveChangesAsync();
        }
    }
}
```

```
        return await GetAllHotels();
    }

    [HttpPut("hotel/{id}")]
    public async Task<ActionResult<List<Hotel>>> UpdateHotel(int id,
Hotel hotl)
    {
        Hotel? dbHotel = await _context.Hotels.FindAsync(id);

        if (dbHotel == null)
        {
            return NotFound("This hotel does not exist");
        }

        dbHotel.Id = hotl.Id;
        dbHotel.Name = hotl.Name;
        dbHotel.SBPrice = hotl.SBPrice;
        dbHotel.DBPrice = hotl.DBPrice;
        dbHotel.FamPrice = hotl.FamPrice;
        dbHotel.Description = hotl.Description;

        _context.Hotels.Update(dbHotel);
        await _context.SaveChangesAsync();

        return await GetAllHotels();
    }

    [HttpDelete("hotel/{id}")]
    public async Task<ActionResult<List<Hotel>>> DeleteHotel(int id)
    {
        Hotel? dbHotel = await _context.Hotels.FindAsync(id);

        if (dbHotel == null)
        {
            return NotFound("This hotel does not exist");
        }

        _context.Hotels.Remove(dbHotel);
        await _context.SaveChangesAsync();

        return await GetAllHotels();
    }

    //Admin For tours
    [HttpGet("tour")]
    public async Task<ActionResult<List<Tour>>> GetAllTours()
    {
        List<Tour> list = await _context.Tours.ToListAsync();

        return Ok(list);
    }
}
```



```
}

[HttpGet("tour/{id}")]
public async Task<ActionResult<Tour>> GetTour(int id)
{
    Tour? dbtour = await _context.Tours.FindAsync(id);

    if (dbtour == null)
    {
        return NotFound("This tour does not exist");
    }

    return Ok(dbtour);
}

[HttpPost("tour")]
public async Task<ActionResult<List<Tour>>> AddTour(Tour tour)
{
    _context.Tours.Add(tour);
    await _context.SaveChangesAsync();

    return await GetAllTours();
}

[HttpPut("tour/{id}")]
public async Task<ActionResult<List<Tour>>> UpdateTour(int id, Tour
tour)
{
    Tour? dbtour = await _context.Tours.FindAsync(id);

    if (dbtour == null)
    {
        return NotFound("This tour does not exist");
    }

    dbtour.Id = tour.Id;
    dbtour.Name = tour.Name;
    dbtour.Cost = tour.Cost;
    dbtour.MaxNumberOfGuests = tour.MaxNumberOfGuests;
    dbtour.DurationInDays = tour.DurationInDays;
    dbtour.Description = tour.Description;

    _context.Tours.Update(dbtour);
    await _context.SaveChangesAsync();

    return await GetAllTours();
}

[HttpDelete("tour/{id}")]
public async Task<ActionResult<List<Tour>>> DeleteTour(int id)
{
    /
```

```

        Tour? dbtour = await _context.Tours.FindAsync(id);

        if (dbtour == null)
        {
            return NotFound("This tour does not exist");
        }

        _context.Tours.Remove(dbtour);
        await _context.SaveChangesAsync();

        return await GetAllTours();
    }

    // create 3 methods to get all bookings for each type of booking

    [HttpGet("hotelbooking")]
    public async Task<ActionResult<List<HotelBooking>>>
    GetAllHotelBookings()
    {
        List<HotelBooking> list = await
        _context.HotelBookings.ToListAsync();

        return Ok(list);
    }

    [HttpGet("tourbooking")]
    public async Task<ActionResult<List<TourBooking>>>
    GetAllTourBookings()
    {
        List<TourBooking> list = await
        _context.TourBookings.ToListAsync();

        return Ok(list);
    }

    [HttpGet("packagebooking")]
    public async Task<ActionResult<List<PackageBooking>>>
    GetAllPackageBookings()
    {
        List<PackageBooking> list = await
        _context.PackageBookings.ToListAsync();

        return Ok(list);
    }
}

```

### BlazorHotelBooking/Server/Controllers/BookingsController.cs

```

using BlazorHotelBooking.Server.Data;
using BlazorHotelBooking.Shared;
using BlazorHotelBooking.Shared.Models;
using Microsoft.AspNetCore.Authorization;

```

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace BlazorHotelBooking.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [Authorize(Roles = "User")]
    public class BookingsController : ControllerBase
    {
        private readonly DataContext _context;

        public BookingsController(DataContext context)
        {
            _context = context;
        }

        //Booking for Hotels
        [HttpGet("hotel/userbooking")]
        public async Task<ActionResult<List<HotelBookingViewModel>>>
        GetAllHotelBookingsWithUserId(string userId)
        {
            var query = from hotel in _context.Hotels
                        join booking in _context.HotelBookings on hotel.Id
                        equals booking.HotelId
                        where booking.UserId == userId
                        select new HotelBookingViewModel
                        {
                            bookingId = booking.Id,
                            hotelName = hotel.Name,
                            RoomType = booking.RoomType,
                            CheckIn = booking.CheckIn,
                            CheckOut = booking.CheckOut,
                            NumberOfNights = booking.NumberOfNights,
                            TotalPrice = booking.TotalPrice,
                            DepositAmountPaid = booking.DepositAmountPaid,
                            BookingDate = booking.BookingDate,
                            paidInfull = booking.PaidInfull,
                            IsCancelled = booking.IsCancelled,
                            PaymentDueDate = booking.PaymentDueDate
                        };
            var result = await query.ToListAsync();

            return Ok(result);
        }

        [HttpPut("hotel/payment/{id}")]
        public async Task<ActionResult<string>> PayHotelRemainder(string
        id)
```

```
{
    var dbHotel = await _context.HotelBookings.FindAsync(id);
    Payments payment = new Payments();

    if (dbHotel == null)
    {
        return NotFound("This hotel does not exist");
    }

    if (dbHotel.PaidInfull == true)
    {
        return BadRequest("You have already paid in full");
    }

    //dbHotel.DepositAmountPaid = dbHotel.TotalPrice;
    dbHotel.PaidInfull = true;

    payment.UserId = dbHotel.UserId;
    payment.bookingId = dbHotel.Id;
    payment.bookingType = "Hotel";
    payment.AmountPaid = dbHotel.TotalPrice -
dbHotel.DepositAmountPaid;
    payment.paymentType = "PayRemainder";

    _context.Payments.Add(payment);

    _context.HotelBookings.Update(dbHotel);
    await _context.SaveChangesAsync();

    return Ok("Payment Successful");
}

[HttpPut("hotel/{id}")]
public async Task<ActionResult<string>> UpdateHotel(string id,
HotelBooking hotl, decimal paymentRemainder, decimal surcharge)
{
    var dbHotel = await _context.HotelBookings.FindAsync(id);
    Payments payment = new Payments();
    Payments payment2 = new Payments();

    if (dbHotel == null)
    {
        return NotFound("This hotel does not exist");
    }

    dbHotel.Id = hotl.Id;
    dbHotel.HotelId = hotl.HotelId;
    dbHotel.RoomType = hotl.RoomType;
    dbHotel.CheckIn = hotl.CheckIn;
    dbHotel.CheckOut = hotl.CheckOut;
    dbHotel.NumberOfNights = hotl.NumberOfNights;
    dbHotel.TotalPrice = hotl.TotalPrice;
    dbHotel.DepositAmountPaid = hotl.DepositAmountPaid;
}
```

```
        dbHotel.BookingDate = hotl.BookingDate;
        dbHotel.UserId = hotl.UserId;
        dbHotel.PaidInfull = hotl.PaidInfull;
        dbHotel.IsCancelled = hotl.IsCancelled;
        dbHotel.PaymentDueDate = hotl.PaymentDueDate;

        payment.UserId = dbHotel.UserId;
        payment.bookingId = dbHotel.Id;
        payment.bookingType = "Hotel";
        payment.AmountPaid = paymentRemainder;
        payment.paymentType = "ModifyBooking";

        payment2.UserId = dbHotel.UserId;
        payment2.bookingId = dbHotel.Id;
        payment2.bookingType = "Hotel";
        payment2.AmountPaid = surcharge;
        payment2.paymentType = "Surcharge";

        _context.Payments.Add(payment2);
        _context.Payments.Add(payment);

        _context.HotelBookings.Update(dbHotel);
        await _context.SaveChangesAsync();

        return Ok("Booking Updated Successfully");
    }

    [HttpGet("hotel/{id}")]
    public async Task<ActionResult<HotelBooking>>
    GetHotelBookingById(string id)
    {
        var dbhotel = await _context.HotelBookings.FindAsync(id);

        if (dbhotel == null)
        {
            return NotFound("This hotel does not exist");
        }

        return Ok(dbhotel);
    }

    [HttpGet("hotel/overlap")]
    public ActionResult<int> CheckIfBookingOverlaps(DateTime checkIn,
    DateTime checkOut, int hotelId, string roomType)
    {
        var hotelOverlap = _context.HotelBookings.Where(x =>
            x.CheckIn <= checkOut &&
            x.CheckOut >= checkIn &&
            x.RoomType == roomType &&
            x.HotelId == hotelId
        ).ToList().Count();

        var packageOverlap = _context.PackageBookings.Where(x =>
```

```
        x.HotelCheckIn <= checkOut &&
        x.HotelCheckOut >= checkIn &&
        x.RoomType == roomType &&
        x.HotelId == hotelId
    ).ToList().Count();

    var overlap = hotelOverlap + packageOverlap;

    return Ok(overlap);
}

[HttpDelete("hotel/{id}")]
public async Task<ActionResult<string>> DeleteHotelBooking(string
id)
{
    var dbHotel = await _context.HotelBookings.FindAsync(id);
    Payments payment = new Payments();
    var today = DateTime.Now;
    var datediff = dbHotel.CheckIn - today;

    if (dbHotel == null)
    {
        return NotFound("This hotel does not exist");
    }

    if (datediff.Days < 5)
    {
        return BadRequest("You cannot cancel this booking");
    }

    payment.UserId = dbHotel.UserId;
    payment.bookingId = dbHotel.Id;
    payment.bookingType = "Hotel";

    if (dbHotel.PaidInfull == true)
    {
        payment.AmountPaid = dbHotel.TotalPrice * -1;
    }
    else
    {
        payment.AmountPaid = dbHotel.DepositAmountPaid * -1;
    }

    payment.paymentType = "Refund";

    _context.Payments.Add(payment);

    _context.HotelBookings.Remove(dbHotel);
    await _context.SaveChangesAsync();

    return Ok("Booking Deleted");
}

[HttpPost("hotel/book")]
```

```
        public async Task<ActionResult<string>>
AddHotelBooking(HotelBooking hotl)
    {

        Payments payment = new Payments();
        payment.UserId = hotl.UserId;
        payment.bookingId = hotl.Id;
        payment.bookingType = "Hotel";
        payment.AmountPaid = hotl.DepositAmountPaid;
        payment.paymentType = "Deposit";

        _context.Payments.Add(payment);

        _context.HotelBookings.Add(hotl);
        await _context.SaveChangesAsync();

        return Ok("Booking Successful");
    }

    [HttpGet("tour/userbooking")]
    public async Task<ActionResult<List<TourBookingViewModel>>>
GetAllTourBookingsWithUserId(string userId)
    {
        var query = from tour in _context.Tours
                    join booking in _context.TourBookings on tour.Id
equals booking.TourId
                    where booking.UserId == userId
                    select new TourBookingViewModel
                    {
                        bookingId = booking.Id,
                        TourName = tour.Name,
                        CommencementDate = booking.CommencementDate,
                        EndDate = booking.EndDate,
                        NumberOfGuests = booking.NumberOfPeople,
                        TotalPrice = booking.TotalPrice,
                        DepositAmountPaid = booking.DepositAmountPaid,
                        BookingDate = booking.BookingDate,
                        paidInfull = booking.PaidInfull,
                        IsCancelled = booking.IsCancelled,
                        PaymentDueDate = booking.PaymentDueDate
                    };
        var result = await query.ToListAsync();

        return Ok(result);
    }

    [HttpPut("tour/payment/{id}")]
    public async Task<ActionResult<string>> PayTourRemainder(string id)
    {
        var dbTour = await _context.TourBookings.FindAsync(id);
```

```
Payments payment = new Payments();
if (dbTour == null)
{
    return NotFound("This tour does not exist");
}

if (dbTour.PaidInfull == true)
{
    return BadRequest("You have already paid in full");
}

// dbTour.DepositAmountPaid = dbTour.TotalPrice;
dbTour.PaidInfull = true;

payment.UserId = dbTour.UserId;
payment.bookingId = dbTour.Id;
payment.bookingType = "Tour";
payment.AmountPaid = dbTour.TotalPrice -
dbTour.DepositAmountPaid;
payment.paymentType = "PayRemainder";

_context.Payments.Add(payment);

_context.TourBookings.Update(dbTour);
await _context.SaveChangesAsync();

return Ok("Payment Successful");
}

[HttpPut("tour/{id}")]
public async Task<ActionResult<string>> UpdateTour(string id,
TourBooking tour, decimal surcharge)
{
    var dbTour = await _context.TourBookings.FindAsync(id);
    Payments payment2 = new Payments();

    if (dbTour == null)
    {
        return NotFound("This tour does not exist");
    }

    dbTour.Id = tour.Id;
    dbTour.TourId = tour.TourId;
    dbTour.CommencementDate = tour.CommencementDate;
    dbTour.EndDate = tour.EndDate;
    dbTour.NumberOfPeople = tour.NumberOfPeople;
    dbTour.TotalPrice = tour.TotalPrice;
    dbTour.DepositAmountPaid = tour.DepositAmountPaid;
    dbTour.BookingDate = tour.BookingDate;
    dbTour.UserId = tour.UserId;
    dbTour.PaidInfull = tour.PaidInfull;
    dbTour.IsCancelled = tour.IsCancelled;
```



```
        dbTour.PaymentDueDate = tour.PaymentDueDate;

        payment2.UserId = dbTour.UserId;
        payment2.bookingId = dbTour.Id;
        payment2.bookingType = "Tour";
        payment2.AmountPaid = surcharge;
        payment2.paymentType = "Surcharge";

        _context.Payments.Add(payment2);

        _context.TourBookings.Update(dbTour);
        await _context.SaveChangesAsync();

        return Ok("Booking Updated Successfully");
    }

    [HttpGet("tour/{id}")]
    public async Task<ActionResult<TourBooking>>
GetTourBookingById(string id)
    {
        var dbTour = await _context.TourBookings.FindAsync(id);

        if (dbTour == null)
        {
            return NotFound("This Tour does not exist");
        }

        return Ok(dbTour);
    }

    //check if bookings overlap
    [HttpGet("tour/overlap")]
    public ActionResult<int> CheckIfTourBookingOverlaps(DateTime start,
DateTime end, int tourId)
    {
        var guestCount = 0;
        List<TourBooking> tourOverlap = _context.TourBookings.Where(x
=>
            x.CommencementDate <= end &&
            x.EndDate >= start &&
            x.TourId == tourId
        ).ToList();

        List<PackageBooking> packageOverlap =
_context.PackageBookings.Where(x =>
            x.TourStartDate <= end &&
            x.TourEndDate >= start &&
            x.TourId == tourId
        ).ToList();

        foreach (var item in tourOverlap)
```

```
        {
            guestCount += item.NumberOfPeople;
        }

        foreach (var item in packageOverlap)
        {
            guestCount += item.NumberOfPeopleOnTour;
        }

        return Ok(guestCount);
    }

    [HttpDelete("tour/{id}")]
    public async Task<ActionResult<string>> DeleteTourBooking(string
id)
    {
        var dbTour = await _context.TourBookings.FindAsync(id);
        var today = DateTime.Now;
        var datediff = dbTour.CommencementDate - today;
        Payments payment = new Payments();

        if (dbTour == null)
        {
            return NotFound("This tour booking does not exist");
        }

        if (datediff.Days < 5)
        {
            return BadRequest("You cannot cancel this booking");
        }

        payment.UserId = dbTour.UserId;
        payment.bookingId = dbTour.Id;
        payment.bookingType = "Package";

        if (dbTour.PaidInfull == true)
        {
            payment.AmountPaid = dbTour.TotalPrice * -1;
        }
        else
        {
            payment.AmountPaid = dbTour.DepositAmountPaid * -1;
        }

        payment.paymentType = "Refund";

        _context.Payments.Add(payment);

        _context.TourBookings.Remove(dbTour);
        await _context.SaveChangesAsync();

        return Ok("Booking Deleted");
    }
}
```

```
}

[HttpPost("tour/book")]
public async Task<ActionResult<string>> AddTourBooking(TourBooking
tour)
{
    Payments payment = new Payments();
    payment.UserId = tour.UserId;
    payment.bookingId = tour.Id;
    payment.bookingType = "Tour";
    payment.AmountPaid = tour.DepositAmountPaid;
    payment.paymentType = "Deposit";

    _context.Payments.Add(payment);

    _context.TourBookings.Add(tour);
    await _context.SaveChangesAsync();

    return Ok("Booking Successful");
}

// Bookings for Packages

[HttpPost("package/book")]
public async Task<ActionResult<string>>
AddPackageBooking(PackageBooking pkg)
{
    Payments payment = new Payments();
    payment.UserId = pkg.UserId;
    payment.bookingId = pkg.Id;
    payment.bookingType = "Package";
    payment.AmountPaid = pkg.DepositAmountPaid;
    payment.paymentType = "Deposit";

    _context.Payments.Add(payment);

    _context.PackageBookings.Add(pkg);
    await _context.SaveChangesAsync();

    return Ok("Booking Successful");
}

[HttpGet("package/userbooking")]
public async Task<ActionResult<List<TourBookingViewModel>>>
GetAllPackageBookingsWithUserId(string userId)
{
    var query =
        from booking in _context.PackageBookings
        join tour in _context.Tours on booking.TourId equals
```

```
tour.Id
hotel.Id

        join hotel in _context.Hotels on booking.HotelId equals
        where booking.UserId == userId
        select new PackageBookingViewModel
        {
            bookingId = booking.Id,
            TourName = tour.Name,
            CommencementDate = booking.TourStartDate,
            EndDate = booking.TourEndDate,
            NumberOfGuests = booking.NumberOfPeopleOnTour,
            hotelName = hotel.Name,
            RoomType = booking.RoomType,
            CheckIn = booking.HotelCheckIn,
            CheckOut = booking.HotelCheckOut,
            NumberOfNights = booking.NumberOfNights,
            TotalPrice = booking.TotalPrice,
            DepositAmountPaid = booking.DepositAmountPaid,
            BookingDate = booking.BookingDate,
            paidInfull = booking.PaidInfull,
            IsCancelled = booking.IsCancelled,
            PaymentDueDate = booking.PaymentDueDate
        };
        var result = await query.ToListAsync();

        return Ok(result);
    }

    [HttpPut("package/payment/{id}")]
    public async Task<ActionResult<string>> PayPackageRemainder(string
id)
    {
        var dbPackage = await _context.PackageBookings.FindAsync(id);
        Payments payment = new Payments();

        if (dbPackage == null)
        {
            return NotFound("This bookings does not exist");
        }

        if (dbPackage.PaidInfull == true)
        {
            return BadRequest("You have already paid in full");
        }

        //dbPackage.DepositAmountPaid = dbPackage.TotalPrice;

        payment.UserId = dbPackage.UserId;
        payment.bookingId = dbPackage.Id;
        payment.bookingType = "Package";
        payment.AmountPaid = dbPackage.TotalPrice -
dbPackage.DepositAmountPaid;
        payment.paymentType = "PayRemainder";
    }
}
```

```
        _context.Payments.Add(payment);

        dbPackage.PaidInfull = true;

        _context.PackageBookings.Update(dbPackage);
        await _context.SaveChangesAsync();

        return Ok("Payment Successful");
    }

    [HttpPut("package/{id}")]
    public async Task<ActionResult<string>> UpdatePackage(string id,
        PackageBooking pkg, decimal paymentRemainder, decimal surcharge)
    {
        var dbPackage = await _context.PackageBookings.FindAsync(id);
        Payments payment = new Payments();
        Payments payment2 = new Payments();

        if (dbPackage == null)
        {
            return NotFound("This Package Booking does not exist");
        }

        dbPackage.Id = pkg.Id;
        dbPackage.TourId = pkg.TourId;
        dbPackage.TourStartDate = pkg.TourStartDate;
        dbPackage.TourEndDate = pkg.TourEndDate;
        dbPackage.NumberOfPeopleOnTour = pkg.NumberOfPeopleOnTour;
        dbPackage.HotelId = pkg.HotelId;
        dbPackage.RoomType = pkg.RoomType;
        dbPackage.HotelCheckIn = pkg.HotelCheckIn;
        dbPackage.HotelCheckOut = pkg.HotelCheckOut;
        dbPackage.NumberOfNights = pkg.NumberOfNights;
        dbPackage.TotalPrice = pkg.TotalPrice;
        dbPackage.DepositAmountPaid = pkg.DepositAmountPaid;
        dbPackage.BookingDate = pkg.BookingDate;
        dbPackage.UserId = pkg.UserId;
        dbPackage.PaidInfull = pkg.PaidInfull;
        dbPackage.IsCancelled = pkg.IsCancelled;
        dbPackage.PaymentDueDate = pkg.PaymentDueDate;

        if (paymentRemainder != 0)
        {
            payment.UserId = dbPackage.UserId;
            payment.bookingId = dbPackage.Id;
            payment.bookingType = "Package";
            payment.AmountPaid = paymentRemainder;
            payment.paymentType = "ModifyBooking";
            _context.Payments.Add(payment);
        }

        payment2.UserId = dbPackage.UserId;
```

```
        payment2.bookingId = dbPackage.Id;
        payment2.bookingType = "Package";
        payment2.AmountPaid = surcharge;
        payment2.paymentType = "Surcharge";

        _context.Payments.Add(payment2);

        _context.PackageBookings.Update(dbPackage);
        await _context.SaveChangesAsync();

        return Ok("Booking Updated Successfully");
    }

    [HttpGet("package/{id}")]
    public async Task<ActionResult<PackageBooking>>
GetPackageBookingById(string id)
    {
        var dbPackage = await _context.PackageBookings.FindAsync(id);

        if (dbPackage == null)
        {
            return NotFound("This booking does not exist");
        }

        return Ok(dbPackage);
    }

    [HttpDelete("package/{id}")]
    public async Task<ActionResult<string>> DeletePackageBooking(string
id)
    {
        var dbPackage = await _context.PackageBookings.FindAsync(id);
        var today = DateTime.Now;
        var tourdatediff = dbPackage.TourStartDate - today;
        var hoteldatediff = dbPackage.HotelCheckIn - today;
        Payments payment = new Payments();

        if (dbPackage == null)
        {
            return NotFound("This tour booking does not exist");
        }

        if (tourdatediff.Days < 5 || hoteldatediff.Days < 5)
        {
            return BadRequest("You cannot cancel this booking");
        }

        payment.UserId = dbPackage.UserId;
        payment.bookingId = dbPackage.Id;
        payment.bookingType = "Package";
    }
}
```

```
        if (dbPackage.PaidInfull == true)
        {
            payment.AmountPaid = dbPackage.TotalPrice * -1;
        }
        else
        {
            payment.AmountPaid = dbPackage.DepositAmountPaid * -1;
        }

        payment.paymentType = "Refund";

        _context.Payments.Add(payment);

        _context.PackageBookings.Remove(dbPackage);
        await _context.SaveChangesAsync();

        return Ok("Booking Deleted");
    }
}
```

### BlazorHotelBooking/Server/Controllers/HotelController.cs

```
using BlazorHotelBooking.Server.Data;
using BlazorHotelBooking.Shared;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace BlazorHotelBooking.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [Authorize(Roles = "User")]

    public class HotelController : ControllerBase
    {
        private readonly DataContext _context;

        public HotelController(DataContext context)
        {
            _context = context;
        }

        [HttpGet]
        public async Task<ActionResult<List<Hotel>>> GetAllHotels()
        {
            /
```

```
        List<Hotel> list = await _context.Hotels.ToListAsync();

        return Ok(list);
    }

    [HttpGet("{id}")]
    public async Task<ActionResult<Hotel>> GetHotel(int id)
    {
        Hotel? dbhotel = await _context.Hotels.FindAsync(id);

        if (dbhotel == null)
        {
            return NotFound("This hotel does not exist");
        }

        return Ok(dbhotel);
    }
}
```

#### BlazorHotelBooking/Server/Controllers/LoginController.cs

```
using BlazorHotelBooking.Server.Models;
using BlazorHotelBooking.Shared.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

namespace BlazorHotelBooking.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class LoginController : ControllerBase
    {
        private readonly IConfiguration _config;
        private readonly SignInManager<ApplicationUser> _signInManager;

        public LoginController(IConfiguration config,
            SignInManager<ApplicationUser> signInManager)
        {
            _config = config;
            _signInManager = signInManager;
        }

        [HttpPost]
        public async Task<ActionResult> Login([FromBody] LoginModel login)
        {
            Microsoft.AspNetCore.Identity.SignInResult result = await
```



```

_signInManager.PasswordSignInAsync(login.Email, login.Password, false,
false);

        if (!result.Succeeded)
        {
            return BadRequest(new LoginResult { Successful = false,
Error = "Username and password are invalid." });
        }

        ApplicationUser? user = await
_signInManager.UserManager.FindByEmailAsync(login.Email);
        IList<string> roles = await
_signInManager.UserManager.GetRolesAsync(user!);
        List<Claim> claims = new List<Claim>
        {
            new Claim(ClaimTypes.Name, login.Email)
        };

        foreach (string role in roles)
        {
            claims.Add(new Claim(ClaimTypes.Role, role));
        }

        claims.Add(new Claim(ClaimTypes.NameIdentifier, user.Id));

        SymmetricSecurityKey key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["JwtKey"]));
        SigningCredentials creds = new SigningCredentials(key,
SecurityAlgorithms.HmacSha256);
        DateTime expiry =
DateTime.Now.AddDays(Convert.ToInt32(_config["JwtExpiryInDays"]));

        JwtSecurityToken token = new JwtSecurityToken(
            _config["JwtIssuer"],
            _config["JwtAudience"],
            claims,
            expires: expiry,
            signingCredentials: creds
        );

        return Ok(new LoginResult { Successful = true, Token = new
JwtSecurityTokenHandler().WriteToken(token) });
    }
}

```

### BlazorHotelBooking/Server/Controllers/PaymentController.cs

```

using BlazorHotelBooking.Server.Data;
using BlazorHotelBooking.Shared;
using Microsoft.AspNetCore.Authorization;

```

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace BlazorHotelBooking.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class PaymentController : ControllerBase
    {
        private readonly DataContext _context;

        public PaymentController(DataContext context)
        {
            _context = context;
        }

        [HttpGet]
        [Authorize(Roles = "Admin")]
        public async Task<ActionResult<List<Payments>>> GetAllPayments()
        {
            List<Payments> list = await
            _context.Payments.OrderByDescending(x => x.PaymentDate).ToListAsync();

            return Ok(list);
        }

        [HttpGet("{id}")]
        [Authorize]
        public async Task<ActionResult<List<Payments>>>
        GetAllPaymentsByUserId(string id)
        {
            List<Payments> list = await _context.Payments.Where(x =>
            x.UserId == id).OrderByDescending(x => x.PaymentDate).ToListAsync();

            return Ok(list);
        }
    }
}
```

### BlazorHotelBooking/Server/Controllers/RegisterController.cs

```
using BlazorHotelBooking.Server.Models;
using BlazorHotelBooking.Shared.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace BlazorHotelBooking.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
```

```

public class RegisterController : ControllerBase
{
    private readonly UserManager<ApplicationUser> _userManager;

    public RegisterController(UserManager<ApplicationUser> userManager)
    {
        _userManager = userManager;
    }

    [HttpPost]
    public async Task<IActionResult> Register([FromBody] RegisterModel
model)
    {
        ApplicationUser user = new ApplicationUser
        {
            UserName = model.Email,
            Email = model.Email,
            PassportNumber = model.PassportNumber,
            PhoneNumber = model.PhoneNumber
        };

        IdentityResult result = await _userManager.CreateAsync(user,
model.Password);

        if (!result.Succeeded)
        {
            IEnumerable<string> errors = result.Errors.Select(e =>
e.Description);
            return Ok(new RegisterResult { Successful = false, Errors =
errors });
        }

        await _userManager.AddToRoleAsync(user, "User");
        //if (user.Email == "admin@localhost")
        //{
        //    await _userManager.AddToRoleAsync(user, "Admin");
        //    return Ok(new RegisterResult { Successful = true });
        //}

        return Ok(new RegisterResult { Successful = true });
    }
}

```

### BlazorHotelBooking/Server/Controllers/TourController.cs

```

using BlazorHotelBooking.Server.Data;
using BlazorHotelBooking.Shared;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

```

```
namespace BlazorHotelBooking.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [Authorize(Roles = "User")]

    public class TourController : ControllerBase
    {
        private readonly DataContext _context;

        public TourController(DataContext context)
        {
            _context = context;
        }

        [HttpGet]
        public async Task<ActionResult<List<Tour>>> GetAllTours()
        {
            List<Tour> list = await _context.Tours.ToListAsync();

            return Ok(list);
        }

        [HttpGet("{id}")]
        public async Task<ActionResult<Tour>> GetTour(int id)
        {
            Tour? dbtour = await _context.Tours.FindAsync(id);

            if (dbtour == null)
            {
                return NotFound("This tour does not exist");
            }

            return Ok(dbtour);
        }
    }
}
```

### BlazorHotelBooking/Server/Data/DataContext.cs

```
using BlazorHotelBooking.Server.Models;
using BlazorHotelBooking.Shared;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace BlazorHotelBooking.Server.Data
{
    public class DataContext : IdentityDbContext<ApplicationUser>
```

```
{
    public DataContext(DbContextOptions<DataContext> options) :
base(options)
    {
    }
    public DbSet<Hotel> Hotels { get; set; }
    public DbSet<HotelBooking> HotelBookings => Set<HotelBooking>();
    public DbSet<TourBooking> TourBookings => Set<TourBooking>();
    public DbSet<PackageBooking> PackageBookings => Set<PackageBooking>
());

    public DbSet<Tour> Tours { get; set; }
    public DbSet<Payments> Payments => Set<Payments>();

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.Entity<IdentityRole>().HasData(new IdentityRole
        {
            Name = "User",
            NormalizedName = "USER",
            Id = "1",
            ConcurrencyStamp = Guid.NewGuid().ToString()
        });

        modelBuilder.Entity<IdentityRole>().HasData(new IdentityRole
        {
            Name = "Admin",
            NormalizedName = "ADMIN",
            Id = "2",
            ConcurrencyStamp = Guid.NewGuid().ToString()
        });

        modelBuilder.Entity<Hotel>().HasData(
            new Hotel
            {
                Id = 1,
                Name = "Hilton London Hotel",
                SBPrice = 375,
                DBPrice = 775,
                FamPrice = 950,
                Description = "Experience luxury in the heart of the
city, with sophisticated rooms and a stone's throw from London's major
attractions and shopping districts."
            },
            new Hotel
            {
                Id = 2,
                Name = "London Marriott Hotel",
                SBPrice = 300,
                DBPrice = 500,
                FamPrice = 900,
```

```
        Description = "Indulge in elegance and comfort at this centrally located hotel, featuring top-notch amenities and easy access to London's historical landmarks"
    },

    new Hotel
    {
        Id = 3,
        Name = "Travelodge Brighton Seafront",
        SBPrice = 80,
        DBPrice = 120,
        FamPrice = 150,
        Description = " Enjoy affordable comfort with stunning seafront views, ideally situated for exploring Brighton's vibrant beach and pier attractions."
    },

    new Hotel
    {
        Id = 4,
        Name = "Kings Hotel Brighton",
        SBPrice = 180,
        DBPrice = 400,
        FamPrice = 520,
        Description = "A charming, budget-friendly hotel on Brighton's seafront, offering cozy accommodations with easy access to the city's lively nightlife and cultural sites"
    },

    new Hotel
    {
        Id = 5,
        Name = "Leonardo Hotel Brighton",
        SBPrice = 180,
        DBPrice = 400,
        FamPrice = 520,
        Description = "Modern and stylish, this hotel provides a comfortable base to discover Brighton, conveniently close to the beach and the buzzing city center."
    },

    new Hotel
    {
        Id = 6,
        Name = "Nevis Bank Inn, Fort William",
        SBPrice = 90,
        DBPrice = 100,
        FamPrice = 155,
        Description = "Nestled in the Scottish Highlands, this inn offers a serene getaway with scenic views, perfect for outdoor enthusiasts and nature lovers"
    }
);
```

```

        modelBuilder.Entity<Tour>().HasData(
            new Tour
            {
                Id = 1,
                Name = "Real Britain",
                Cost = 1200,
                DurationInDays = 6,
                MaxNumberOfGuests = 30,
                Description = "Dive into charming villages, rolling
hills, and iconic castles in this 6-day escape to authentic Britain"
            },
            new Tour
            {
                Id = 2,
                Name = "Britain and Ireland Explorer",
                Cost = 2000,
                DurationInDays = 16,
                MaxNumberOfGuests = 40,
                Description = "Journey through 16 days of cityscapes,
dramatic coasts, and Celtic charm. Uncover the best of Britain and
Ireland."
            },
            new Tour
            {
                Id = 3,
                Name = "Best of Britain",
                Cost = 2900,
                DurationInDays = 12,
                MaxNumberOfGuests = 30,
                Description = "Indulge in 12 days of luxury. Explore
stately homes, savor Michelin stars, and discover hidden gems of Britain's
finest"
            }
        );
    }
}

```

### BlazorHotelBooking/Server/Models/ApplicationUser.cs

```

using Microsoft.AspNetCore.Identity;

namespace BlazorHotelBooking.Server.Models
{
    public class ApplicationUser : IdentityUser
    {
        public string PassportNumber { get; set; }
    }
}

```

**BlazorHotelBooking/Server/Services/CheckLatePaymentService.cs**

```
using BlazorHotelBooking.Server.Data;
using Sgbj.Cron;

namespace BlazorHotelBooking.Server.Services
{
    // This service is used to check if a booking has been paid in full or
    // not by a due date and cancel it if it has not been paid in full
    public class CheckLatePaymentService : BackgroundService
    {
        private readonly DataContext _context;
        private const int generalDelay = 1 * 10 * 1000;

        public CheckLatePaymentService(IServiceScopeFactory factory)
        {
            _context =
                factory.CreateScope().ServiceProvider.GetRequiredService<DataContext>();
        }

        protected override async Task ExecuteAsync(CancellationToken
            stoppingToken)
        {
            // Run every day at midnight
            using CronTimer timer = new CronTimer("0 0 * * *",
                TimeZoneInfo.Local);

            while (await timer.WaitForNextTickAsync(stoppingToken))
            {
                List<Shared.HotelBooking> hotelbookings =
                    _context.HotelBookings.Where(x => x.PaidInfull == false & x.IsCancelled ==
                        false & x.PaymentDueDate < DateTime.Now).ToList();
                List<Shared.TourBooking> tourbookings =
                    _context.TourBookings.Where(x => x.PaidInfull == false & x.IsCancelled ==
                        false & x.PaymentDueDate < DateTime.Now).ToList();
                List<Shared.PackageBooking> packageBookings =
                    _context.PackageBookings.Where(x => x.PaidInfull == false & x.IsCancelled
                        == false & x.PaymentDueDate < DateTime.Now).ToList();

                foreach (Shared.HotelBooking? booking in hotelbookings)
                {
                    Console.WriteLine($"{booking.Id} has been cancelled");
                    booking.IsCancelled = true;
                    _context.HotelBookings.Update(booking);
                }

                foreach (Shared.TourBooking? booking in tourbookings)
                {
                    Console.WriteLine($"{booking.Id} has been cancelled");
                    booking.IsCancelled = true;
                    _context.TourBookings.Update(booking);
                }
            }
        }
    }
}
```



```

        }

        foreach (Shared.PackageBooking? booking in packageBookings)
        {
            Console.WriteLine($"{booking.Id} has been cancelled");
            booking.IsCancelled = true;
            _context.PackageBookings.Update(booking);
        }

        await _context.SaveChangesAsync();
    }
}
}
}

```

### BlazorHotelBooking/Server/Program.cs

```

using BlazorHotelBooking.Server.Data;
using BlazorHotelBooking.Server.Models;
using BlazorHotelBooking.Server.Services;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using System.Text;

WebApplicationBuilder builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews();
builder.Services.AddHostedService<CheckLatePaymentService>();
builder.Services.AddRazorPages();
builder.Services.AddSwaggerGen();
builder.Services.AddDbContext<DataContext>(options =>
{
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConn
ection"));
});

builder.Services.AddDefaultIdentity<ApplicationUser>()
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<DataContext>();

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new
TokenValidationParameters
    {

```

```

        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        ValidIssuer = builder.Configuration["JwtIssuer"],
        ValidAudience = builder.Configuration["JwtAudience"],
        IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["JwtKey"]
!))
    });

});

WebApplication app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseWebAssemblyDebugging();
}
else
{
    app.UseExceptionHandler("/Error");
    // The default HSTS value is 30 days. You may want to change this for
production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();

app.UseBlazorFrameworkFiles();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();
app.UseAuthorization();

app.UseSwagger();
app.UseSwaggerUI();

app.MapRazorPages();
app.MapControllers();
app.MapFallbackToFile("index.html");

app.Run();

```

### BlazorHotelBooking/Shared/Models/HotelBookingViewModel.cs

```

namespace BlazorHotelBooking.Shared.Models
{

```

```
public class HotelBookingViewModel
{
    public string bookingId { get; set; }
    public string hotelName { get; set; }
    public string RoomType { get; set; }
    public DateTime CheckIn { get; set; }
    public DateTime CheckOut { get; set; }
    public int NumberOfNights { get; set; }
    public decimal TotalPrice { get; set; }
    public decimal DepositAmountPaid { get; set; }
    public DateTime BookingDate { get; set; }
    public bool paidInfull { get; set; }
    public bool IsCancelled { get; set; }
    public DateTime PaymentDueDate { get; set; }
}
}
```

#### BlazorHotelBooking/Shared/Models/LoginModel.cs

```
using System.ComponentModel.DataAnnotations;

namespace BlazorHotelBooking.Shared.Models
{
    public class LoginModel
    {
        [Required, EmailAddress, Display(Name = "Email")]
        public string? Email { get; set; }
        [Required, DataType(DataType.Password), Display(Name = "Password")]
        public string? Password { get; set; }
    }
}
```

#### BlazorHotelBooking/Shared/Models/LoginResult.cs

```
namespace BlazorHotelBooking.Shared.Models
{
    public class LoginResult
    {
        public bool Successful { get; set; }
        public string? Error { get; set; }
        public string? Token { get; set; }
    }
}
```

#### BlazorHotelBooking/Shared/Models/PackageBookingViewModel.cs

```
namespace BlazorHotelBooking.Shared.Models
{
    public class PackageBookingViewModel
    {
        public string bookingId { get; set; }
        public string TourName { get; set; }
        public DateTime CommencementDate { get; set; }
        public DateTime EndDate { get; set; }
        public int NumberOfGuests { get; set; }
        public string hotelName { get; set; }
        public string RoomType { get; set; }
        public DateTime CheckIn { get; set; }
        public DateTime CheckOut { get; set; }
        public int NumberOfNights { get; set; }
        public decimal TotalPrice { get; set; }
        public decimal DepositAmountPaid { get; set; }
        public DateTime BookingDate { get; set; }
        public bool paidInfull { get; set; }
        public bool IsCancelled { get; set; }
        public DateTime PaymentDueDate { get; set; }
    }
}
```

### BlazorHotelBooking/Shared/Models/RegisterModel.cs

```
using System.ComponentModel.DataAnnotations;

namespace BlazorHotelBooking.Shared.Models
{
    public class RegisterModel
    {
        [Required, EmailAddress, Display(Name = "Email")]
        public string? Email { get; set; }

        [Required, DataType(DataType.Password), Display(Name = "Password"),
        StringLength(100, ErrorMessage = "The {0} must be at least {2}
characters long.", MinimumLength = 6)]
        public string? Password { get; set; }

        [DataType(DataType.Password), Display(Name = "Confirm Password"),
        Compare("Password", ErrorMessage = "Passwords Do Not Match")]
        public string? ConfirmPassword { get; set; }

        [Required, Display(Name = "Passport Number")]
        public string? PassportNumber { get; set; }
        [Required, Display(Name = "Phone Number")]
        public string? PhoneNumber { get; set; }
    }
}
```

**BlazorHotelBooking/Shared/Models/RegisterResult.cs**

```
namespace BlazorHotelBooking.Shared.Models
{
    public class RegisterResult
    {
        public bool Successful { get; set; }
        public IEnumerable<string>? Errors { get; set; }
    }
}
```

**BlazorHotelBooking/Shared/Models/TourBookingViewModel.cs**

```
namespace BlazorHotelBooking.Shared.Models
{
    public class TourBookingViewModel
    {
        public string bookingId { get; set; }
        public string TourName { get; set; }
        public DateTime CommencementDate { get; set; }
        public DateTime EndDate { get; set; }
        public int NumberOfGuests { get; set; }
        public decimal TotalPrice { get; set; }
        public decimal DepositAmountPaid { get; set; }
        public DateTime BookingDate { get; set; }
        public bool paidInfull { get; set; }
        public bool IsCancelled { get; set; }
        public DateTime PaymentDueDate { get; set; }
    }
}
```

**BlazorHotelBooking/Shared/Hotel.cs**

```
using System.ComponentModel.DataAnnotations.Schema;

namespace BlazorHotelBooking.Shared
{
    public class Hotel
    {
        public int Id { get; set; }
        public string Name { get; set; } = string.Empty;
        [Column(TypeName = "decimal(18,2)")]
        public decimal SBPrice { get; set; }
        [Column(TypeName = "decimal(18,2)")]
        public decimal DBPrice { get; set; }
        [Column(TypeName = "decimal(18,2)")]
        public decimal FamPrice { get; set; }
    }
}
```

```
        public string? Description { get; set; }  
    }  
}
```

### BlazorHotelBooking/Shared/HotelBooking.cs

```
using System.ComponentModel.DataAnnotations;  
using System.ComponentModel.DataAnnotations.Schema;  
  
namespace BlazorHotelBooking.Shared  
{  
    public class HotelBooking  
    {  
        [Key]  
        public string Id { get; set; } = Guid.NewGuid().ToString();  
        public int HotelId { get; set; }  
        [Required(ErrorMessage = "Room Type required.")]  
        public string RoomType { get; set; }  
        [Required(ErrorMessage = "Check-In Date required.")]  
        public DateTime CheckIn { get; set; } =  
DateTime.Now.Date.AddMonths(2);  
        public DateTime CheckOut { get; set; }  
        [Required(ErrorMessage = "Number of Nigts required.")]  
        public int NumberOfNights { get; set; }  
        [Column(TypeName = "decimal(18,2)")]  
        public decimal TotalPrice { get; set; }  
        [Column(TypeName = "decimal(18,2)")]  
        public decimal DepositAmountPaid { get; set; }  
        public string UserId { get; set; }  
        public DateTime BookingDate { get; set; } = DateTime.Now.Date;  
        public bool PaidInfull { get; set; } = false;  
        public bool IsCancelled { get; set; } = false;  
        public DateTime PaymentDueDate { get; set; }  
    }  
}
```

### BlazorHotelBooking/Shared/PackageBooking.cs

```
using System.ComponentModel.DataAnnotations;  
using System.ComponentModel.DataAnnotations.Schema;  
  
namespace BlazorHotelBooking.Shared  
{  
    public class PackageBooking  
    {  
        public string Id { get; set; } = Guid.NewGuid().ToString();  
        public int HotelId { get; set; }  
        public string RoomType { get; set; }  
    }  
}
```

```

        public DateTime HotelCheckIn { get; set; } =
DateTime.Now.Date.AddMonths(2);
        public DateTime HotelCheckOut { get; set; }
        public int NumberOfNights { get; set; }
        public int TourId { get; set; }
        [Required(ErrorMessage = "Commencement Date required.")]
        public DateTime TourStartDate { get; set; } =
DateTime.Now.Date.AddMonths(2);
        public DateTime TourEndDate { get; set; }
        [Column(TypeName = "decimal(18,2)")]
        public decimal TotalPrice { get; set; }
        public int NumberOfPeopleOnTour { get; set; }
        [Column(TypeName = "decimal(18,2)")]
        public decimal DepositAmountPaid { get; set; }
        public string UserId { get; set; }
        public DateTime BookingDate { get; set; } = DateTime.Now.Date;
        public bool PaidInfull { get; set; } = false;
        public bool IsCancelled { get; set; } = false;
        public DateTime PaymentDueDate { get; set; }
    }
}

```

### BlazorHotelBooking/Shared/Payments.cs

```

namespace BlazorHotelBooking.Shared
{
    public class Payments
    {
        public string Id { get; set; } = Guid.NewGuid().ToString();
        public string UserId { get; set; }
        public string bookingId { get; set; }
        public string bookingType { get; set; }
        public string paymentType { get; set; }
        public DateTime PaymentDate { get; set; } = DateTime.Now;
        public decimal AmountPaid { get; set; }
    }
}

```

### BlazorHotelBooking/Shared/Tour.cs

```

using System.ComponentModel.DataAnnotations.Schema;

namespace BlazorHotelBooking.Shared
{
    public class Tour
    {
        public int Id { get; set; }
        public string Name { get; set; } = string.Empty;
    }
}

```

```
        [Column(TypeName = "decimal(18,2)"]
        public decimal Cost { get; set; }
        public int DurationInDays { get; set; }
        public int MaxNumberOfGuests { get; set; }
        public string? Description { get; set; }
    }
}
```

### BlazorHotelBooking/Shared/TourBooking.cs

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace BlazorHotelBooking.Shared
{
    public class TourBooking
    {
        [Key]
        public string Id { get; set; } = Guid.NewGuid().ToString();
        public int TourId { get; set; }
        [Required(ErrorMessage = "Commencement Date required.")]
        public DateTime CommencementDate { get; set; } =
            DateTime.Now.Date.AddMonths(2);
        public DateTime EndDate { get; set; }
        [Column(TypeName = "decimal(18,2)"]
        public decimal TotalPrice { get; set; }
        [Column(TypeName = "decimal(18,2)"]
        public decimal DepositAmountPaid { get; set; }
        public int NumberOfPeople { get; set; }
        public string UserId { get; set; }
        public DateTime BookingDate { get; set; } = DateTime.Now.Date;
        public bool PaidInfull { get; set; } = false;
        public bool IsCancelled { get; set; } = false;
        public DateTime PaymentDueDate { get; set; }
    }
}
```