Flight Manager

Цел на проекта

- Целта на проекта е да се създаде програма за управление на полети. Чрез програмата се добавят потребители от администраторски профил, създават се полети и могат да се създават резервации без потребилтелски вход.

Линк за проекта в Github:
https://github.com/AdamS839/FlightManager

Етапи в разработката

1. Първи етап – създаване на repository в github и създаване на моделите за базата данни и създаване на миграции.
2. Втори етап – създаване на контролери и развиване на функционалностите по потребителски вход и добавяне на роли.
3. Трети етап – създаване на забрани по роли и оправяне на Create, Edit, Details и Delete за страниците.
4. Четвърти етап – разработка на други функционалности за успешна работа с данните

Използвани технологии:

- Microsoft.EntityFrameworkCore
- Microsoft.AspNetCore.Identity.UI
- Microsoft.VisualStudio.Web.CodeGenerating

По-важни методи

Данни в модела dbUser

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Configuration;
using System.Diagnostics.CodeAnalysis;
using System.Linq;
using System.Reflection.Metadata.Ecma335;
using System.Text;
using System.Threading.Tasks;

namespace Data.Models
{
    [Index(nameof(dbUser.EGN), IsUnique = true)]
    36 references
    public class dbUser : IdentityUser
    {
        [Required]
        [StringLength(30, ErrorMessage = "First name must be longer than 2 letters.", MinimumLength = 3)]
        [RegularExpression(@"^[a-zA-Z]+$", ErrorMessage = "First name must contain only letters.")]
        [Display(Name = "First Name")]
        17 references
        public string FirstName { get; set; }
        [Required]
        [StringLength(30, ErrorMessage = "Last name must be longer than 2 letters.", MinimumLength = 3)]
        [RegularExpression(@"^[a-zA-Z]+$", ErrorMessage = "Last name must contain only letters.")]
        [Display(Name = "Last Name")]
        17 references
        public string LastName { get; set; }
        [Required]
        [StringLength(10, ErrorMessage = "EGN must be exactly 10 digits long.", MinimumLength = 10)]
        [RegularExpression("^[0-9]*$", ErrorMessage = "EGN must contain only digits.")]
        [Display(Name = "EGN")]
        18 references
        public string EGN { get; set; }
        [Required]
        [Display(Name = "Address")]
        [StringLength(60, ErrorMessage = "Address must contain more than 10 characters.", MinimumLength = 11)]
        17 references
        public string Address { get; set; }
    }
}
```

В модела на dbUser се добавят само полетата за имена, ЕГН и адрес, защото dbUser наследява IndentityUser и там вече има другите полета за Id, Username, Email, Phone number и се получава грешка, защото имаме 2 еднакви полета.

Променяне на данните в dbUsersController.cs

Добавя се нов потребител в dbUser, за да може директно в него да се променят данните, които добавяме или искаме да променим.

За създаване:

```
public async Task<IActionResult> Create([Bind("UserName,FirstName,LastName,EGN,Address,Id,Email,PasswordHash,PhoneNumber")] dbUser User)
{
    string password = User.PasswordHash;
    dbUser user = new dbUser
    {
        UserName = User.Email,
        Email = User.Email,
        Address = User.Address,
        EGN = User.EGN,
        FirstName = User.FirstName,
        LastName = User.LastName,
        PhoneNumber = User.PhoneNumber,
    };
    if (ModelState.IsValid)
    {
        //Creating CreateAsync for the new user with the hashed password
        var _user = await _userManager.FindByNameAsync(user.UserName);
        if (_user == null)
        {
            IdentityResult checkUser = await _userManager.CreateAsync(user, password);
            if (checkUser.Succeeded)
            {
                await _userManager.AddToRoleAsync(user, "Employee");
            }
        }

        /*
        _context.Add(User);
        await _context.SaveChangesAsync();*/

        return RedirectToAction(nameof(Index));

    }
    return View(User);
}
```

Намираме вече създаден потребител и променяме неговите данни.

За редактиране:

```
0 references
public async Task<IActionResult> Edit(string id, [Bind("UserName,FirstName,LastName,EGN,Address,Id,Email,PasswordHash,PhoneNumber")] dbUser dbUser)
{
    if (id != dbUser.Id)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            var editted = await _context.Users.FindAsync(id);
            editted.FirstName = dbUser.FirstName;
            editted.LastName = dbUser.LastName;
            editted.EGN = dbUser.EGN;
            editted.Address = dbUser.Address;
            editted.PhoneNumber = dbUser.PhoneNumber;
            editted.Email = dbUser.Email;
            editted.UserName = dbUser.Email;
            _context.Update(editted);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!dbUserExists(dbUser.Id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(dbUser);
}
```

Грешка при създаване на администраторски акаунт

Метода проверява дали ролите Admin и Employee съществуват и ако не съществуват ги създава. Щом не съществуват роли, то администраторски акаунт също не съществува. След това създава нов акаунт с роля администратор.

Тук имаше грешка при влизането в админ акаунта (log in) и решението на проблема е, че трябва потребителското име (username) и имейла (email) трябва да бъдат еднакви като username-а е равен на емейла.

```
1 reference
private async Task CreateRole(IServiceProvider serviceProvider)
{
    var RoleManager = serviceProvider.GetRequiredService<RoleManager<IdentityRole>>();
    var UserManager = serviceProvider.GetRequiredService<UserManager<dbUser>>();
    string[] roles = { "Admin", "Employee" };
    IdentityResult result;

    //Checks if roles Admin and Employee exist. If not - they get created
    foreach (var role in roles)
    {
        var check = await RoleManager.RoleExistsAsync(role);
        if (!check)
        {
            result = await RoleManager.CreateAsync(new IdentityRole(role));
        }
    }
    // Add admin user
    var admin = new dbUser
    {
        UserName = "Admin@admin.bg",
        FirstName = "Admin",
        LastName = "Admin",
        EGN = "0000000000",
        Address = "AdminNoAddress123",
        Email = "Admin@admin.bg",
        PhoneNumber = "1234567890",
        Id = Guid.NewGuid().ToString()
    };

    string passwordUser = "_Password123";

    //Checks if admin user exists. If it doesn't, it creates a admin user with role ,,Admin''
    var _user = await UserManager.FindByNameAsync(admin.UserName);
    if (_user == null)
    {
        IdentityResult checkUser = await UserManager.CreateAsync(admin, passwordUser);
        if (checkUser.Succeeded)
        {
            await UserManager.AddToRoleAsync(admin, "Admin");
        }
    }

}
```

Проверка за налични полети

```
0 references
public async Task<IActionResult> Create([Bind("Id,FirstName,MiddleName,LastName,EGN,PhoneNumber,Nationality,TicketType,FlightId")] Reservation reservation)
{
    if (ModelState.IsValid)
    {
        var flight = await _context.Flights
            .Include(f => f.Reservations)
            .FirstOrDefaultAsync(f => f.Id == reservation.FlightId);

        if (flight == null)
        {
            // Flight not found
            ModelState.AddModelError(string.Empty, "Invalid flight selection.");
            ViewData["FlightId"] = new SelectList(_context.Flights, "Id", "LocationFrom", reservation.FlightId);
            return View(reservation);
        }

        // Check flight capacity
        int bookedBusinessSeats = flight.Reservations.Where(r => r.TicketType == "Business").Sum(r => r.Id);
        int bookedOrdinarySeats = flight.Reservations.Where(r => r.TicketType == "Ordinary").Sum(r => r.Id);

        int availableBusinessSeats = flight.BusinessPassengerCapacity - bookedBusinessSeats;
        int availableOrdinarySeats = flight.PassangerCapacity - bookedOrdinarySeats;

        int requestedSeats = reservation.Id;
        int availableBSeats = reservation.TicketType == "Business" ? availableBusinessSeats : availableOrdinarySeats;
        int availableOSeats = reservation.TicketType == "Ordinary" ? availableBusinessSeats : availableOrdinarySeats;

        if (availableBSeats < requestedSeats || availableOSeats < requestedSeats)
        {
            // Not enough available seats
            ModelState.AddModelError(string.Empty, $"Not enough available seats for {reservation.TicketType} ticket.");
            ViewData["FlightId"] = new SelectList(_context.Flights, "Id", "LocationFrom", reservation.FlightId);
            return View(reservation);
        }

        // Create the reservation
        _context.Add(reservation);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }

    // Model state is not valid, return the view with validation errors
    ViewData["FlightId"] = new SelectList(_context.Flights, "Id", "LocationFrom", reservation.FlightId);
    return View(reservation);
}
```

Проверява се дали има свободни места за двата вида билета – бизнес и обикновен и спрямо това кой тип потребителят е въвел в полето за билет, се връща съобщение, че няма свободно място или се запазва резервацията.

Благодаря за вниманието!