

НП “Обучение за ИТ Умения и кариера“

Име на проект: Flight Manager

Екип: Александър Девинизов, Никола Димов

Хасково 2024г.

Линк за проекта в github:

<https://github.com/AdamS839/FlightManager>

1.Цел на проекта

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

2.Разпределение на ролите

Работата по проекта е разпределена 50% на 50%.

3.Етапи в реализиране на проекта

Първи етап – създаване на repository в github и създаване на моделите за базата данни и създаване на миграции.

Втори етап – създаване на контролери и развиване на функционалностите по потребителски вход и добавяне на роли.

Трети етап – създаване на забрани по роли и оправяне на Create, Edit, Details и Delete за страниците.

Четвърти етап – разработка на други функционалности за успешна работа с данните

4.Реализация:

Използвани технологии:

Microsoft.EntityFrameworkCore

Microsoft.AspNetCore.Identity.UI

Microsoft.VisualStudio.Web.CodeGenerating

Описание на по-важните методи и функционалности в кода:

- Модела на 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 IdentityUser и там има другите полета за Id, Username, Email, Phone number и се получава грешка, защото имаме 2 еднакви полета.

- Променяне на данните в dbUsersController.cs

```
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);
}
```

Трябва да се добави нов потребител в dbUser, защото да може директно в него да се променят данните, които вкарваме.

- Функционалността на проверка за налични полети за резервация

```
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);
}
```

Тук се проверява дали има свободни места за двата вида билета – бизнес и обикновен и спрямо това връща съобщение, че няма свободно място или се запазва резервацията

- Метод за задаване на админ акаунт и задаване на роли

```
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");
        }
    }
}
```

Проверява се дали има роли Admin и Employee в базата данни и ако няма биват създадени и след това се създава администраторски акаунт, на който бива дадена ролята Admin. Ако вече съществува администраторски акаунт, кода не създава втори такъв.

- Проверка за налична Admin роля

```

<td>
    @if (this.User.IsInRole("Admin"))
    {
        <a asp-action="Edit" asp-route-id="@item.Id">Edit</a>
        <a asp-action="Details" asp-route-id="@item.Id">Details</a>
        <a asp-action="Delete" asp-route-id="@item.Id">Delete</a>
    }
</td>

```

В cshtml файловете добавяме проверка дали потребителят е във роля Admin и ако е ще види допълнителните бутони за редактиране, детайли, изтриване и създаване.

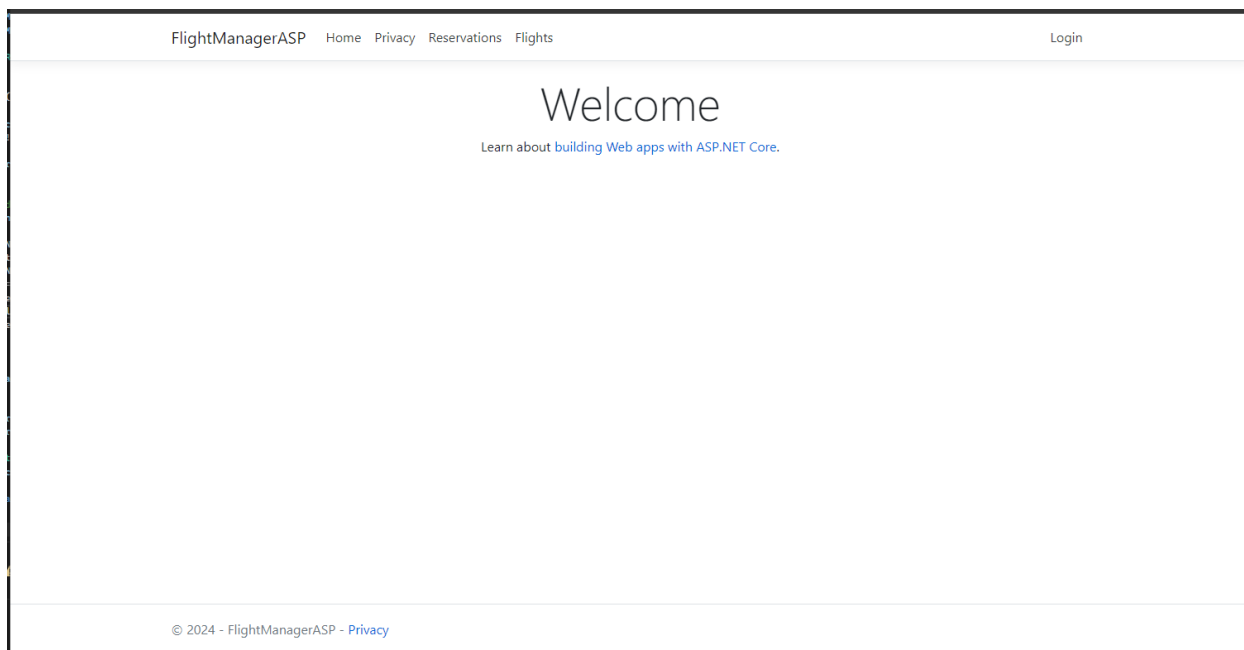
```

</li>
@if (this.User.IsInRole("Admin"))
{
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="dbUsers" asp-action="Index">Users</a>
    </li>
}
</li>

```

Тук ако потребителят е администратор, единствено той вижда полето за Users и само той може да влиза там и да прави промени.

Страницата на проекта без потребителски акаунт:



Страницата на проекта в администраторски акаунт:

