LABORATORY ASSIGNMENT №4

CRUD operations using EF Core. Data validation.

Goal: acquire skills of implementing operations of creating, reading, updating and deleting data using ORM.

Workflow:

Task 1. Using the Code First approach, create a model that, together with the existing model, will form a one-to-many relationship.

I’ve created Booking model and added some navigation properties to Booking and Event models:

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

namespace TicketBookingWebsite.Models
{
    public class Event
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public DateTime Date { get; set; }
        public string Location { get; set; }

        [Column(TypeName = "decimal(8, 2)")]
        public decimal Price { get; set; }

        public int AvailableTickets { get; set; }

        public ICollection<Booking>? Bookings { get; set; }
    }
      }


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

namespace TicketBookingWebsite.Models
{
    public class Booking
    {
        public int Id { get; set; }

        [Required]
        public string CustomerName { get; set; }
        [Required]
        public string CustomerEmail { get; set; }

        public int Quantity { get; set; }

        public DateTime BookingDate { get; set; } = DateTime.Now;

        public int EventId { get; set; }

        public Event? Event { get; set; }
    }
}
```

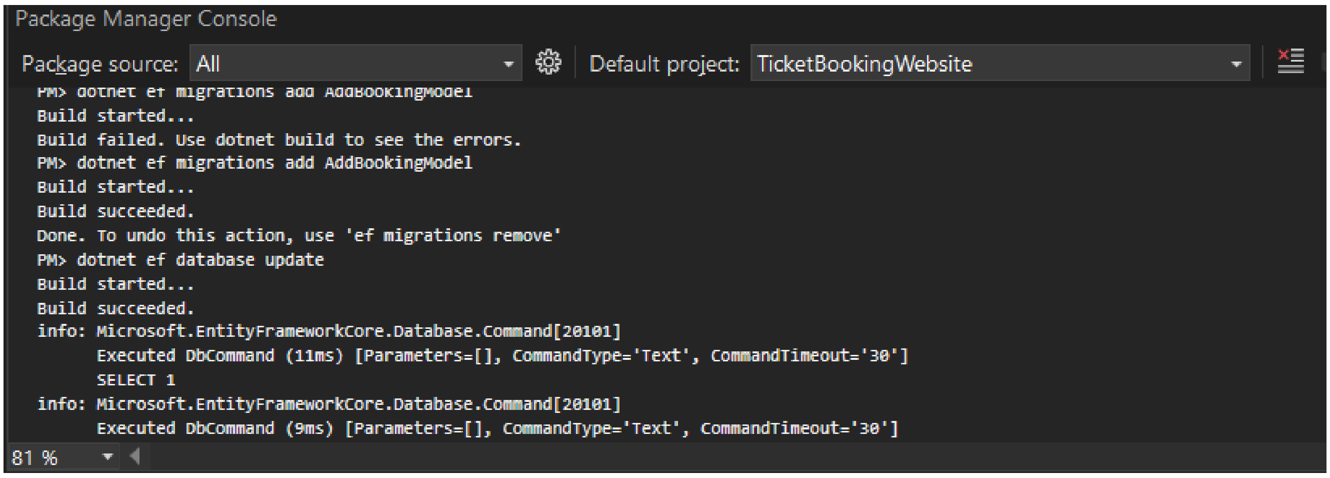
Then i`ve modified DbContext:

```
using Microsoft.EntityFrameworkCore;
using TicketBookingWebsite.Models;

namespace TicketBookingWebsite.Data
{
    public class WebsiteDbContext : DbContext
    {
        public WebsiteDbContext(DbContextOptions<WebsiteDbContext> options) :
base(options) { }

        public DbSet<Event> Events => Set<Event>();
        public DbSet<Booking> Bookings => Set<Booking>();
    }
}
```

Then I've created migration, where tables with relationship will be created in the database:



```
PM> dotnet ef migrations add AddBookingModel
Build started...
Build failed. Use dotnet build to see the errors.
PM> dotnet ef migrations add AddBookingModel
Build started...
Build succeeded.
Done. To undo this action, use 'ef migrations remove'
PM> dotnet ef database update
Build started...
Build succeeded.
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (11ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT 1
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (9ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
```

Task 2. Implement CRUD (Create, Read, Update, Delete) operations for both models.

I’ve created Interface and Repository for each model, here is code for Event model, the other repository is almost same:

```
using TicketBookingWebsite.Models;

namespace TicketBookingWebsite.Repositories.Interfaces
{
    public interface IEventRepository
    {
        IQueryable<Event> Events { get; }

        void CreateEvent(Event e);
        void UpdateEvent(Event e);
        void DeleteEvent(Event e);
        Event? GetEventById(int id);
    }
}
```

```
using TicketBookingWebsite.Data;
using TicketBookingWebsite.Models;
using TicketBookingWebsite.Repositories.Interfaces;

namespace TicketBookingWebsite.Repositories
{
    public class EventRepository : IEventRepository
    {
        private WebsiteDbContext context;

        public EventRepository(WebsiteDbContext ctx)
        {
            context = ctx;
        }

        public IQueryable<Event> Events => context.Events;

        public void CreateEvent(Event e)
        {
            context.Events.Add(e);
            context.SaveChanges();
        }

        public void UpdateEvent(Event e)
        {
            context.Events.Update(e);
            context.SaveChanges();
        }

        public void DeleteEvent(Event e)
        {
            context.Events.Remove(e);
            context.SaveChanges();
        }

        public Event? GetEventById(int id)
        {
            return context.Events.FirstOrDefault(e => e.Id == id);
        }
    }
}
```

The next step is to create Controller and Views for it, here is example for Event model:

```
using Microsoft.AspNetCore.Mvc;
using TicketBookingWebsite.Models;
using TicketBookingWebsite.Repositories.Interfaces;

namespace TicketBookingWebsite.Controllers
{
    public class EventController : Controller
    {
        private readonly IEventRepository _repository;

        public EventController(IEventRepository repository)
        {
            _repository = repository;
        }

        public IActionResult Index()
        {
            return View(_repository.Events.ToList());
        }

        public IActionResult Details(int id)
        {
            var ev = _repository.GetEventById(id);
            if (ev == null) return NotFound();
            return View(ev);
        }

        public IActionResult Create()
        {
            return View();
        }

        [HttpPost]
        public IActionResult Create(Event ev)
        {
            if (ModelState.IsValid)
            {
                _repository.CreateEvent(ev);
                return RedirectToAction("Index");
            }
            return View(ev);
        }

        public IActionResult Edit(int id)
        {
            var ev = _repository.GetEventById(id);
            if (ev == null) return NotFound();
            return View(ev);
        }

        [HttpPost]
        public IActionResult Edit(Event ev)
        {
            if (ModelState.IsValid)
            {
                _repository.UpdateEvent(ev);
                return RedirectToAction("Index");
            }
            return View(ev);
        }

        public IActionResult Delete(int id)
        {
            var ev = _repository.GetEventById(id);
            if (ev == null) return NotFound();
            return View(ev); // confirmation page
        }

        [HttpPost, ActionName("Delete")]
        public IActionResult DeleteConfirmed(int id)
        {
            var ev = _repository.GetEventById(id);
            if (ev == null) return NotFound();

            _repository.DeleteEvent(ev);
            return RedirectToAction("Index");
        }
    }
}
```

I don’t want to provide Views code, cause there is to many of it and it is already in GitHub repository.

Task 3. Add server-side validation to validate user input before performing CRUD operations.

Updated models:

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

namespace TicketBookingWebsite.Models
{
    public class Booking
    {
        public int Id { get; set; }

        [Required(ErrorMessage = "Customer name is required.")]
        [StringLength(100, ErrorMessage = "Name cannot be longer than 100 characters.")]
        public string CustomerName { get; set; }
        [Required(ErrorMessage = "Email is required.")]
        [EmailAddress(ErrorMessage = "Invalid email address.")]
        public string CustomerEmail { get; set; }

        [Required(ErrorMessage = "Please specify ticket quantity.")]
        [Range(1, 100, ErrorMessage = "You must book at least 1 ticket.")]
        public int Quantity { get; set; }

        public DateTime BookingDate { get; set; } = DateTime.Now;

        [Required(ErrorMessage = "Event selection is required.")]
        public int EventId { get; set; }

        public Event? Event { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace TicketBookingWebsite.Models
{
    public class Event
    {
        public int Id { get; set; }

        [Required(ErrorMessage = "Event name is required.")]
        [StringLength(100, ErrorMessage = "Name cannot exceed 100 characters.")]
        public string Name { get; set; }

        [Required(ErrorMessage = "Description is required.")]
        [StringLength(1000, ErrorMessage = "Description cannot exceed 1000 characters.")]
        public string Description { get; set; }

        [Required(ErrorMessage = "Event date is required.")]
        [DataType(DataType.Date)]
        public DateTime Date { get; set; }

        [Required(ErrorMessage = "Location is required.")]
        [StringLength(200, ErrorMessage = "Location cannot exceed 200 characters.")]
        public string Location { get; set; }

        [Required(ErrorMessage = "Price is required.")]
        [Range(0.01, 10000, ErrorMessage = "Please enter a valid price.")]
        [Column(TypeName = "decimal(8, 2)")]
        public decimal Price { get; set; }

        [Required(ErrorMessage = "Available ticket number is required.")]
        [Range(0, 10000, ErrorMessage = "Tickets must be a non-negative number.")]
        public int AvailableTickets { get; set; }

        public ICollection<Booking>? Bookings { get; set; }
    }
}
```

Conclusion: I acquired skills of implementing operations of creating, reading, updating and deleting data using ORM.