ЛАБОРАТОРНА РОБОТА № 4

CRUD-операції в EF Core. Валідація даних.

Мета: набути навичок роботи з ORM, навчитися реалізовувати операції створення, читання, оновлення та видалення даних з використанням EF Core, набути навичок роботи з механізмами валідації даних.

Хід роботи

Завдання 1. Використовуючи підхід Code First, створити модель, яка разом з існуючою моделлю, утворюватиме зв’язок один-до-багатьох.

У межах завдання було розширено модель даних проєкту таким чином, щоб разом з уже наявною сутністю Event вона формувала зв’язок один-до-багатьох. Для предметної області TicketBooking логічною моделлю для зв’язку є сутність Booking (бронювання/квиток), де одна подія (Event) може мати багато бронювань (Booking) та кожне бронювання (Booking) належить одній події (Event).

Для реалізації зв’язку створено клас Booking з зовнішнім ключем EventId та навігаційною властивістю Event. У клас Event додано колекцію Bookings, яка представляє всі бронювання для конкретної події.

Лістинг моделі Booking:

```
using System.ComponentModel.DataAnnotations;

namespace TicketBooking.Models {
	public class Booking {
		public long BookingId { get; set; }
```

```
        [Required]
        [StringLength(100)]
        public string CustomerName { get; set; } = string.Empty;

        [Required]
        [EmailAddress]
        [StringLength(120)]
        public string CustomerEmail { get; set; } = string.Empty;

        [Range(1, 10)]
        public int Quantity { get; set; } = 1;

        // FK -> Event
        public long EventId { get; set; }

        // Navigation property
        public Event? Event { get; set; }

        public DateTime CreatedAtUtc { get; set; } = DateTime.UtcNow;
    }
}
```

Лістинг оновленої моделі Event:

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

namespace TicketBooking.Models {
    public class Event {
        public long? EventId { get; set; }

        public string Title { get; set; } = string.Empty;
        public string Description { get; set; } = string.Empty;
        public string Location { get; set; } = string.Empty;

        public DateTime StartDate { get; set; }
        [Range(0, double.MaxValue, ErrorMessage = "Ticket's price can't be negative")]
        [Column(TypeName = "decimal(10, 2)")]   // 99999999.99 max value
        public decimal BaseTicketPrice { get; set; }

        public ICollection<Booking> Bookings { get; set; } = new List<Booking>();
    }
}
```

Далі оновлено TicketBookingDbContext: додано DbSet<Booking> і зафіксовано зв’язок Fluent API у OnModelCreating (HasOne/WithMany/HasForeignKey). Це дозволяє явно задокументувати тип зв’язку та поведінку при видаленні (cascade).

Лістинг оновленого класу TicketBookingDbContext:

```
using Microsoft.EntityFrameworkCore;

namespace TicketBooking.Models {
    public class TicketBookingDbContext : DbContext {
        public TicketBookingDbContext(DbContextOptions<TicketBookingDbContext> options)
            : base(options) { }

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

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

        protected override void OnModelCreating(ModelBuilder modelBuilder) {
            modelBuilder.Entity<Booking>()
                .HasOne(b => b.Event)
                .WithMany(e => e.Bookings)
                .HasForeignKey(b => b.EventId)
                .OnDelete(DeleteBehavior.Cascade);
        }
    }
}
```

Після внесення змін було створено міграцію та оновлено базу даних командою dotnet ef database update. У результаті створено таблицю Bookings з зовнішнім ключем на Events.

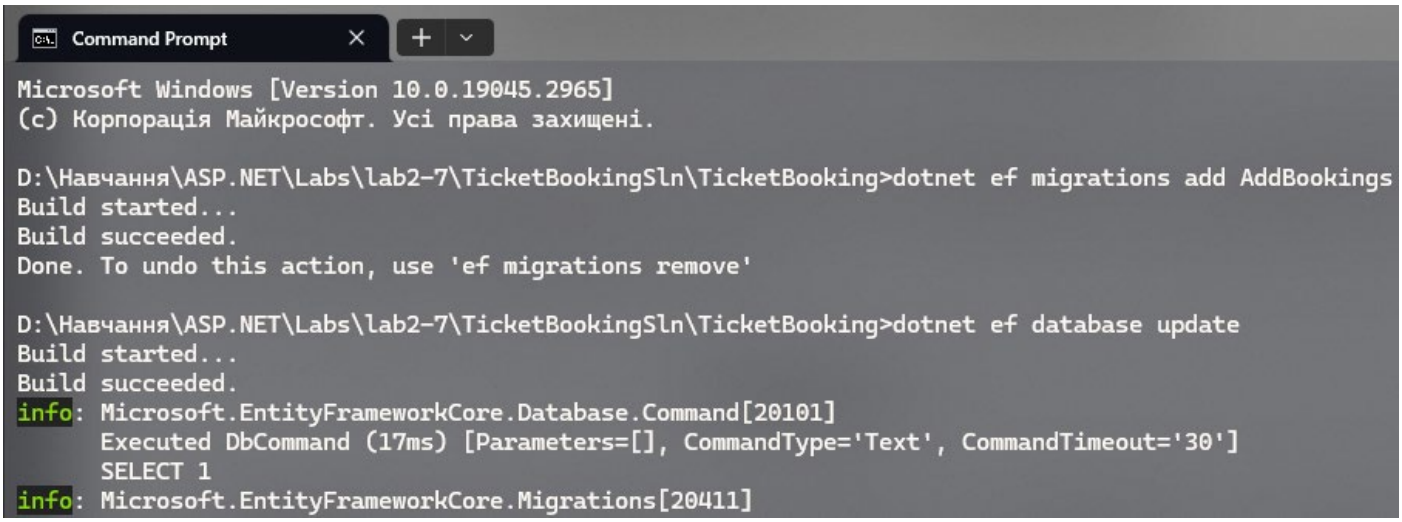

Рис. 1. Створення та застосування міграції для зв’язку один-до-багатьох.

Для підготовки даних для тестування (і подальшої перевірки CRUD) у SeedData додано наповнення таблиці бронювань тестовими записами. Бронювання прив’язуються до вже створених подій через їх EventId.

Фрагмент SeedData, де додаються тестові Booking:

```
if (!context.Bookings.Any()) {
    var firstEvent = context.Events.OrderBy(e => e.EventId).FirstOrDefault();
    var secondEvent = context.Events.OrderBy(e => e.EventId).Skip(1).FirstOrDefault();
    var thirdEvent = context.Events.OrderBy(e => e.EventId).Skip(2).FirstOrDefault();

    var bookings = new List<Booking>();

    if (firstEvent != null) {
        bookings.AddRange(new[] {
            new Booking
```

```
                {
                        CustomerName = "Denys Test",
                        CustomerEmail = "denys.test@example.com",
                        Quantity = 2,
                        EventId = firstEvent.EventId ?? 0
                },
                new Booking
                {
                        CustomerName = "Anna Test",
                        CustomerEmail = "anna.test@example.com",
                        Quantity = 1,
                        EventId = firstEvent.EventId ?? 0
                }
        });
    }

    if (secondEvent != null) {
        bookings.Add(new Booking {
                CustomerName = "Ivan Test",
                CustomerEmail = "ivan.test@example.com",
                Quantity = 3,
                EventId = secondEvent.EventId ?? 0
        });
    }

    if (thirdEvent != null) {
        bookings.Add(new Booking {
                CustomerName = "Olha Test",
                CustomerEmail = "olha.test@example.com",
                Quantity = 1,
                EventId = thirdEvent.EventId ?? 0
        });
    }

    if (bookings.Count > 0) {
        context.Bookings.AddRange(bookings);
        context.SaveChanges();
    }
}
```

Завдання 2. Навігація по категоріям.

Реалізувати CRUD (Create, Read, Update, Delete) операції для обох моделей, враховуючи наступні вимоги:

- Реалізувати представлення з формами для операцій Create та Update
- Для виконання операції Read реалізувати окрему Details сторінку, на якій будуть виводитися всі дані про сутність.
- Перед виконанням операції Delete запитати користувача, підтвердження видалення запису.

У цьому завданні було реалізовано повний набір CRUD-операцій для сутностей Event та Booking. Реалізація виконана з використанням патерну «Repository», який

інкапсулює доступ до EF Core контексту та надає методи створення, оновлення й видалення записів.

Для сутності Event розширено інтерфейс репозиторію методами CreateEvent, UpdateEvent, DeleteEvent, а в реалізації EFEventRepository додано відповідні виклики EF Core (Add/Update/Remove + SaveChanges).

Лістинг інтерфейсу IEventRepository:

```
namespace TicketBooking.Models {
	public interface IEventRepository {
		IQueryable<Event> Events { get; }

		void CreateEvent(Event e);
		void UpdateEvent(Event e);
		void DeleteEvent(Event e);
	}
}
```

Лістинг класу EFEventRepository:

```
namespace TicketBooking.Models {
	public class EFEventRepository : IEventRepository {
		private readonly TicketBookingDbContext _context;

		public EFEventRepository(TicketBookingDbContext context) {
			_context = context;
		}

		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();
		}
	}
}
```

Для доступу до CRUD-сторінок створено EventController, який містить:

- Index — перелік подій (адмін-список);
- Details — окрема сторінка читання повних даних (Read);

- Create (GET/POST) — форма створення;
- Edit (GET/POST) — форма редагування;
- Delete (GET підтвердження / POST підтверджене видалення).

Лістинг контроллера EventController:

```
using Microsoft.AspNetCore.Mvc;
using TicketBooking.Models;

namespace TicketBooking.Controllers {
	public class EventController : Controller {
		private readonly IEventRepository _repo;

		public EventController(IEventRepository repo) {
			_repo = repo;
		}

		public IActionResult Index() {
			var events = _repo.Events.OrderBy(e => e.EventId).ToList();
			return View(events);
		}

		public IActionResult Details(long id) {
			var ev = _repo.Events.FirstOrDefault(e => e.EventId == id);
			if (ev == null)
				return NotFound();
			return View(ev);
		}

		[HttpGet]
		public IActionResult Create() {
			return View(new Event());
		}

		[HttpPost]
		[ValidateAntiForgeryToken]
		public IActionResult Create(Event model) {
			if (!ModelState.IsValid)
				return View(model);

			_repo.CreateEvent(model);
			return RedirectToAction(nameof(Index));
		}

		[HttpGet]
		public IActionResult Edit(long id) {
			var ev = _repo.Events.FirstOrDefault(e => e.EventId == id);
			if (ev == null)
				return NotFound();
			return View(ev);
		}

		[HttpPost]
		[ValidateAntiForgeryToken]
		public IActionResult Edit(Event model) {
			if (!ModelState.IsValid)
				return View(model);

			_repo.UpdateEvent(model);
			return RedirectToAction(nameof(Index));
```

```
        }

        [HttpGet]
        public IActionResult Delete(long id) {
            var ev = _repo.Events.FirstOrDefault(e => e.EventId == id);
            if (ev == null)
                return NotFound();
            return View(ev);
        }

        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public IActionResult DeleteConfirmed(long id) {
            var ev = _repo.Events.FirstOrDefault(e => e.EventId == id);
            if (ev == null)
                return NotFound();

            _repo.DeleteEvent(ev);
            return RedirectToAction(nameof(Index));
        }
    }
}
```

Далі у папці Views/Event створено представлення для кожної операції. Для Create та Edit використано форму (можливе винесення в часткове представлення, щоб уникнути дублювання). Для Delete реалізовано сторінку підтвердження видалення з кнопкою «Yes, delete».

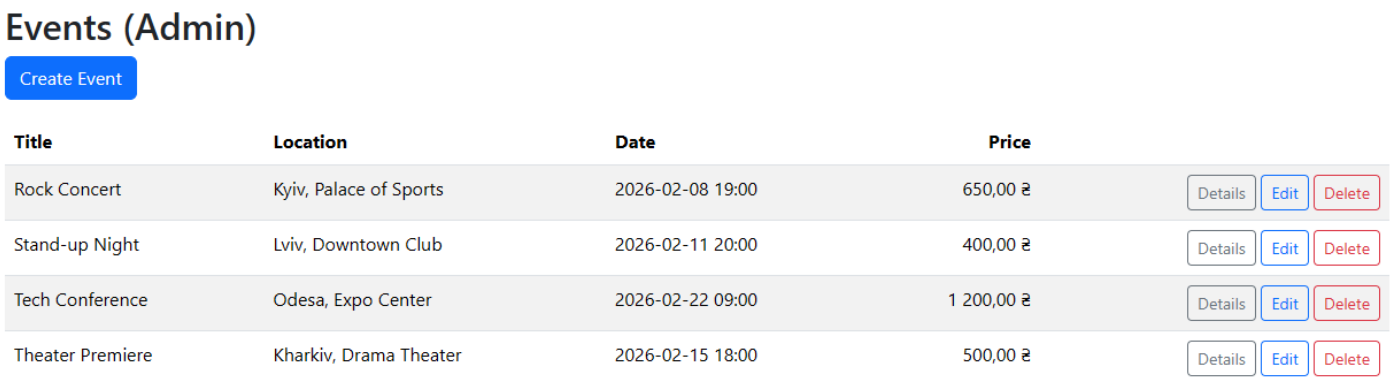

Рис. 2.1. Адміністрування подій (Event): список записів та дії CRUD.

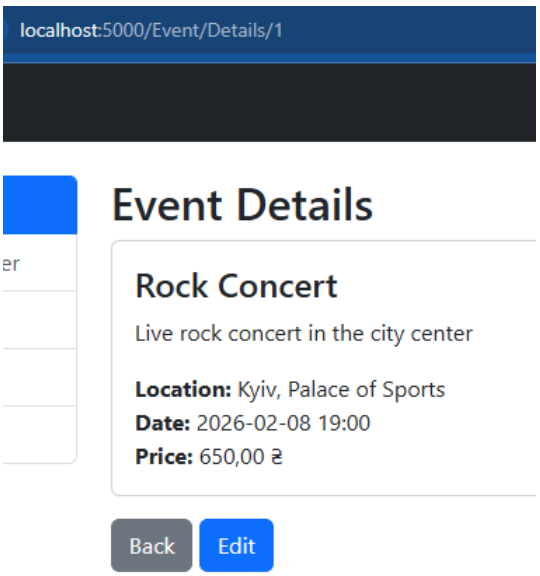

Рис. 2.2. Details-сторінка для сутності Event.

Рис. 2.3. Форма створення/редагування Event.

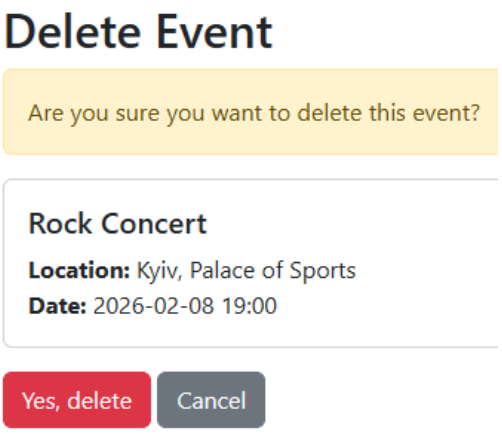

Рис. 2.4. Підтвердження видалення запису Event.

Для сутності Booking реалізовано окремий репозиторій (IBookingRepository та EFBookingRepository). У запитах читання застосовано Include(b => b.Event) для відображення пов’язаної події у списках та Details.

Лістинг інтерфейсу IBookingRepository:

```
namespace TicketBooking.Models {
	public interface IBookingRepository {
		IQueryable<Booking> Bookings { get; }

		void CreateBooking(Booking b);
		void UpdateBooking(Booking b);
		void DeleteBooking(Booking b);
	}
}
```

Лістинг класу EFBookingRepository:

```
using Microsoft.EntityFrameworkCore;

namespace TicketBooking.Models {
	public class EFBookingRepository : IBookingRepository {
		private readonly TicketBookingDbContext _context;

		public EFBookingRepository(TicketBookingDbContext context) {
			_context = context;
		}

		public IQueryable<Booking> Bookings =>
			_context.Bookings.Include(b => b.Event);

		public void CreateBooking(Booking b) {
			_context.Bookings.Add(b);
			_context.SaveChanges();
		}

		public void UpdateBooking(Booking b) {
			_context.Bookings.Update(b);
			_context.SaveChanges();
		}

		public void DeleteBooking(Booking b) {
			_context.Bookings.Remove(b);
			_context.SaveChanges();
		}
	}
}
```

Також створено BookingController з повним набором CRUD-дій. Для створення/редагування бронювання додано вибір події зі списку (SelectList), що дозволяє встановити зовнішній ключ EventId з UI.

Лістинг контроллеру BookingController:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using TicketBooking.Models;

namespace TicketBooking.Controllers {
	public class BookingController : Controller {
		private readonly IBookingRepository _bookingRepo;
		private readonly IEventRepository _eventRepo;

		public BookingController(IBookingRepository bookingRepo, IEventRepository
eventRepo) {
			_bookingRepo = bookingRepo;
			_eventRepo = eventRepo;
		}

		public IActionResult Index() {
			var data = _bookingRepo.Bookings.OrderByDescending(b =>
b.BookingId).ToList();
			return View(data);
		}

		public IActionResult Details(long id) {
			var booking = _bookingRepo.Bookings.FirstOrDefault(b => b.BookingId ==
id);
			if (booking == null)
				return NotFound();
			return View(booking);
		}

		[HttpGet]
		public IActionResult Create() {
			ViewBag.Events = new SelectList(_eventRepo.Events.ToList(), "EventId",
"Title");
			return View(new Booking());
		}

		[HttpPost]
		[ValidateAntiForgeryToken]
		public IActionResult Create(Booking model) {
			if (!ModelState.IsValid) {
				ViewBag.Events = new SelectList(_eventRepo.Events.ToList(),
"EventId", "Title", model.EventId);
				return View(model);
			}

			_bookingRepo.CreateBooking(model);
			return RedirectToAction(nameof(Index));
		}

		[HttpGet]
		public IActionResult Edit(long id) {
			var booking = _bookingRepo.Bookings.FirstOrDefault(b => b.BookingId ==
id);
			if (booking == null)
				return NotFound();

			ViewBag.Events = new SelectList(_eventRepo.Events.ToList(), "EventId",
"Title", booking.EventId);
			return View(booking);
		}

		[HttpPost]
		[ValidateAntiForgeryToken]
```

```
        public IActionResult Edit(Booking model) {
            if (!ModelState.IsValid) {
                ViewBag.Events = new SelectList(_eventRepo.Events.ToList(),
"EventId", "Title", model.EventId);
                return View(model);
            }

            _bookingRepo.UpdateBooking(model);
            return RedirectToAction(nameof(Index));
        }

        [HttpGet]
        public IActionResult Delete(long id) {
            var booking = _bookingRepo.Bookings.FirstOrDefault(b => b.BookingId ==
id);
            if (booking == null)
                return NotFound();
            return View(booking);
        }

        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public IActionResult DeleteConfirmed(long id) {
            var booking = _bookingRepo.Bookings.FirstOrDefault(b => b.BookingId ==
id);
            if (booking == null)
                return NotFound();

            _bookingRepo.DeleteBooking(booking);
            return RedirectToAction(nameof(Index));
        }
    }
}
```

У Views/Booking створено представлення Index/Details/Create/Edit/Delete. Сторінка Delete містить підтвердження перед видаленням.

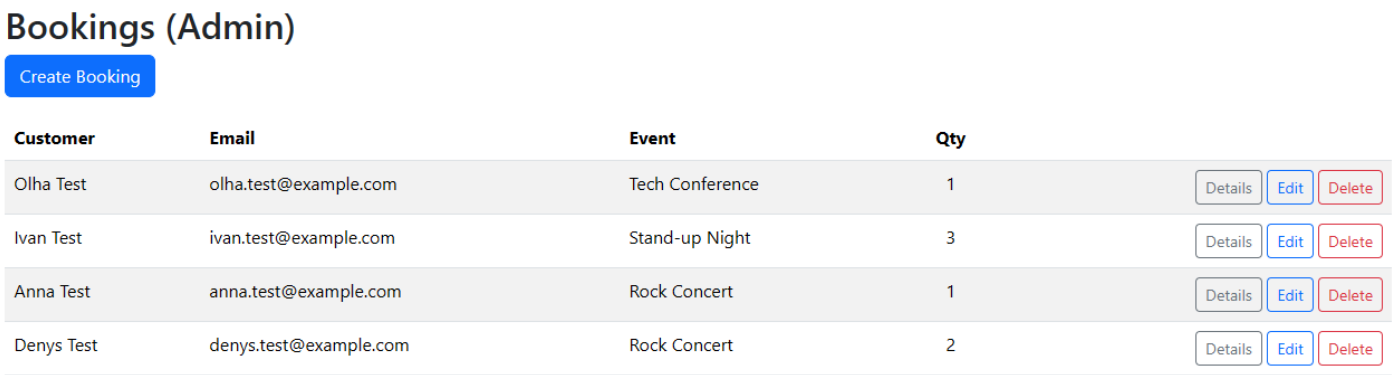

Рис. 2.5. Адміністрування бронювань (Booking): список записів та дії CRUD.

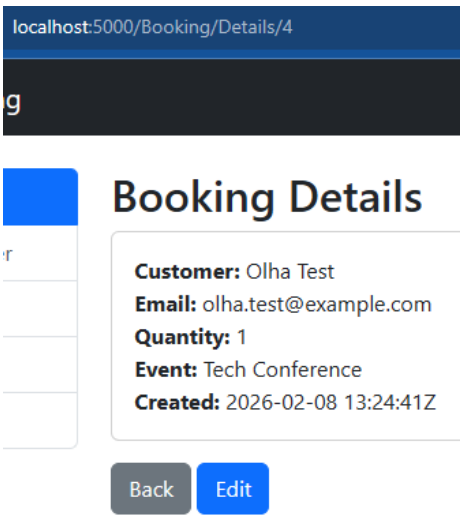

Рис. 2.6. Details-сторінка для сутності Booking.

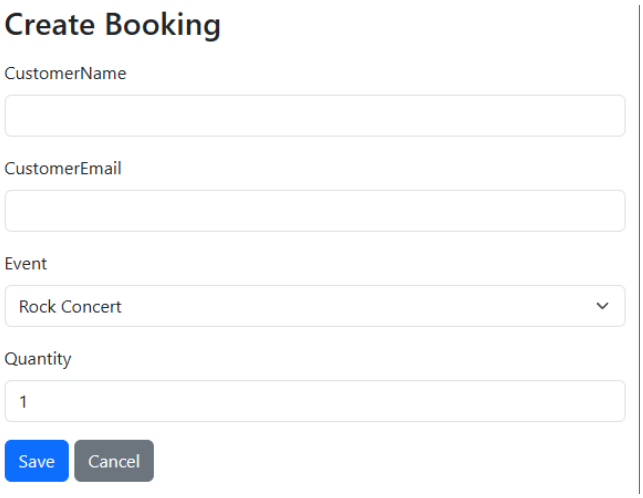

Рис. 2.7.1. Форма створення Booking.

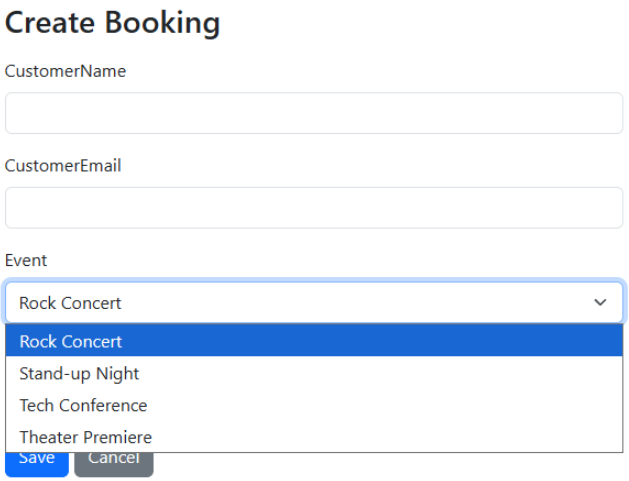

Рис. 2.7.2. Випадаючий список вибору пов’язаної події у формі створення Booking.

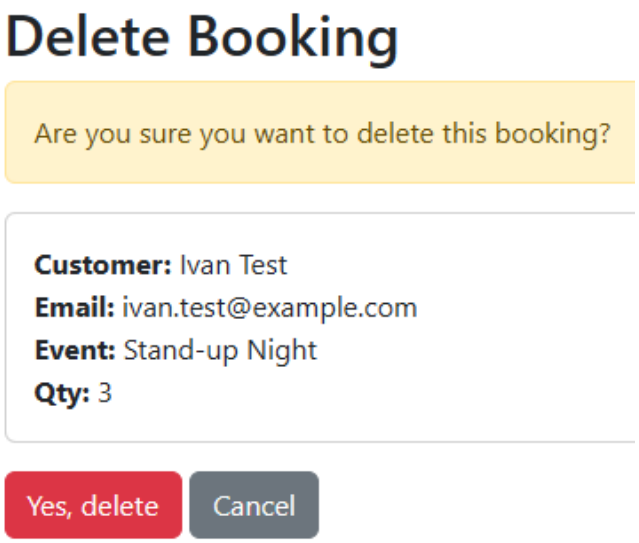

Рис. 2.8. Підтвердження видалення запису Booking.

Для зручної перевірки роботи CRUD у навігаційну панель додано стилізовані кнопки переходу до адміністративних сторінок подій та бронювань (не у вигляді звичайних текстових посилань). Це спрощує демонстрацію функціоналу викладачу.

Лістинг фрагменту коду _Layout.cshtml з кнопками адміністрування Events та Bookings:

```
<header>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <div class="container">
            <a class="navbar-brand" asp-controller="Home" asp-action="Index">
                TicketBooking
            </a>
            <a class="btn btn-outline-light" asp-controller="Cart" asp-action="Index">
                Booking
            </a>
        </div>
        <div class="dropdown">
            <button class="btn btn-warning btn-sm dropdown-toggle" type="button" data-bs-
toggle="dropdown" aria-expanded="false">
                Admin
            </button>

            <ul class="dropdown-menu dropdown-menu-end">
                <li>
                    <a class="dropdown-item" asp-controller="Event" asp-action="Index">
                        Events
                    </a>
                </li>
                <li>
                    <a class="dropdown-item" asp-controller="Booking" asp-action="Index">
                        Bookings
                    </a>
                </li>
            </ul>
```

```
        </div>
    </nav>
</header>
```

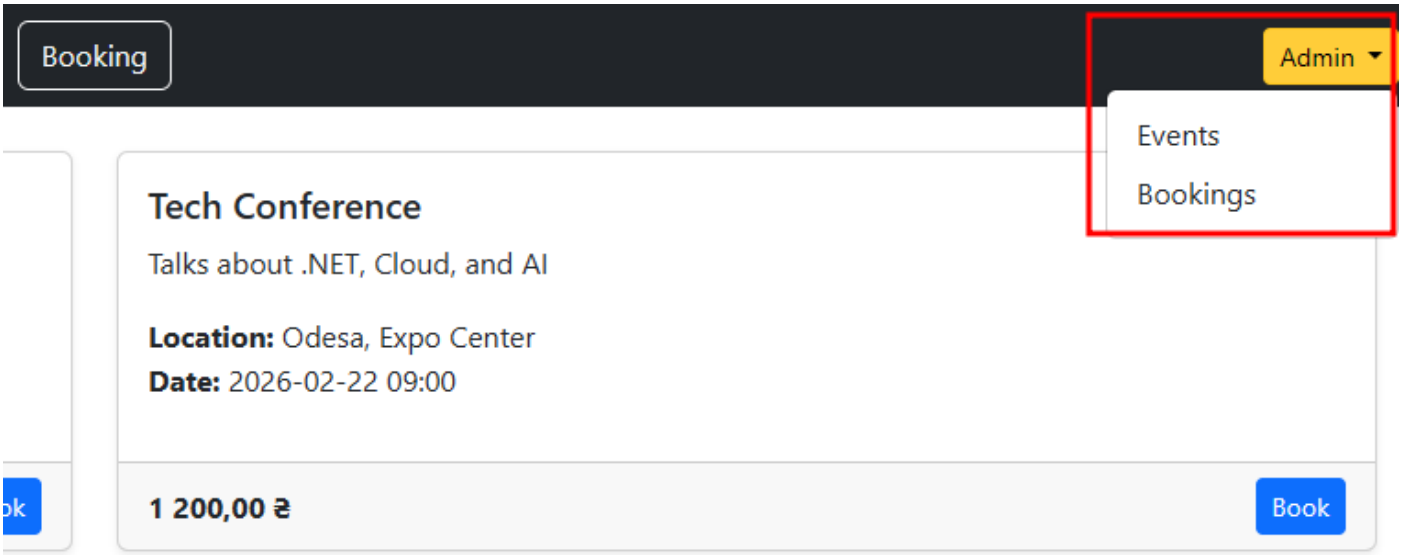

Рис. 2.9. Стилізовані кнопки навігації до сторінок адміністрування.

Завдання 3. Додати серверну валідацію для перевірки введених користувачем даних перед виконанням CRUD-операцій. Використати DataAnnotations для реалізації перевірки введених даних.

Перед виконанням CRUD-операцій додано серверну валідацію для перевірки введених користувачем даних. Для цього у моделі Event та Booking застосовано атрибути DataAnnotations, зокрема Required, StringLength, Range, EmailAddress. Такий підхід дозволяє: відхиляти некоректні дані ще до звернення до бази даних, формувати повідомлення про помилки у ModelState, відображати помилки безпосередньо у формах Create/Edit.

Для Event налаштовано перевірки обов'язковості полів, довжини тексту та коректності ціни (значення має бути додатним). Для Booking додано перевірки імені, email, допустимого діапазону кількості квитків та вибору події.

Лістинг класу Event:

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

namespace TicketBooking.Models {
	public class Event {
		public long? EventId { get; set; }

	  [Required(ErrorMessage = "Please enter a title")]
	  [StringLength(120, ErrorMessage = "Title must be up to 120 characters")]
		public string Title { get; set; } = string.Empty;

		[Required(ErrorMessage = "Please enter a description")]
		[StringLength(1000, ErrorMessage = "Description must be up to 1000 characters")]
		public string Description { get; set; } = string.Empty;

		[Required(ErrorMessage = "Please enter a location")]
		[StringLength(80, ErrorMessage = "Location must be up to 80 characters")]
		public string Location { get; set; } = string.Empty;

		[Required(ErrorMessage = "Please select a date and time")]
		[FutureDate(ErrorMessage = "Start date must be in the future")]
		public DateTime StartDate { get; set; }

		[Required]
		[Range(0.01, 10000000, ErrorMessage = "Please enter a positive ticket price")]
		[Column(TypeName = "decimal(10, 2)")]
		public decimal BaseTicketPrice { get; set; }

		public ICollection<Booking> Bookings { get; set; } = new List<Booking>();
	}
}
```

Лістинг класу Booking:

```
using System.ComponentModel.DataAnnotations;

namespace TicketBooking.Models {
	public class Booking {
		public long BookingId { get; set; }

		[Required(ErrorMessage = "Please enter customer name")]
		[StringLength(100, ErrorMessage = "Name must be up to 100 characters")]
		public string CustomerName { get; set; } = string.Empty;

		[Required(ErrorMessage = "Please enter an email")]
		[EmailAddress(ErrorMessage = "Please enter a valid email address")]
		[StringLength(120)]
		public string CustomerEmail { get; set; } = string.Empty;

		[Required]
		[Range(1, 10, ErrorMessage = "Quantity must be between 1 and 10")]
		public int Quantity { get; set; } = 1;

		// FK -> Event
		[Range(1, long.MaxValue, ErrorMessage = "Please select an event")]  // This
ensures that the EventId must be a positive number, which is important for a valid foreign key
reference.
		public long EventId { get; set; }

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

```
        public DateTime CreatedAtUtc { get; set; } = DateTime.UtcNow;
    }
}
```

У POST-методах Create/Edit контролерів використано перевірку ModelState.IsValid. Якщо дані не проходять валідацію, сторінка повертається з повідомленнями про помилки. Для Booking-форм важливо повторно заповнювати SelectList подій при помилці, щоб користувач міг виправити дані без втрати вибору.

Лістинг фрагментів коду POST-методів Create/Edit у EventController:

```
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Create(Event model) {
    if (!ModelState.IsValid)
        return View(model);

    _repo.CreateEvent(model);
    return RedirectToAction(nameof(Index));
}
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Edit(Event model) {
    if (!ModelState.IsValid)
        return View(model);

    _repo.UpdateEvent(model);
    return RedirectToAction(nameof(Index));
}
```

Лістинг фрагментів коду POST-методів Create/Edit у BookingController:

```
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Create(Booking model) {
    if (!ModelState.IsValid) {
        ViewBag.Events = new SelectList(_eventRepo.Events.ToList(), "EventId", "Title",
model.EventId);
        return View(model);
    }

    _bookingRepo.CreateBooking(model);
    return RedirectToAction(nameof(Index));
}
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Edit(Booking model) {
    if (!ModelState.IsValid) {
        ViewBag.Events = new SelectList(_eventRepo.Events.ToList(), "EventId", "Title",
model.EventId);
        return View(model);
    }

    _bookingRepo.UpdateBooking(model);
    return RedirectToAction(nameof(Index));
}
```

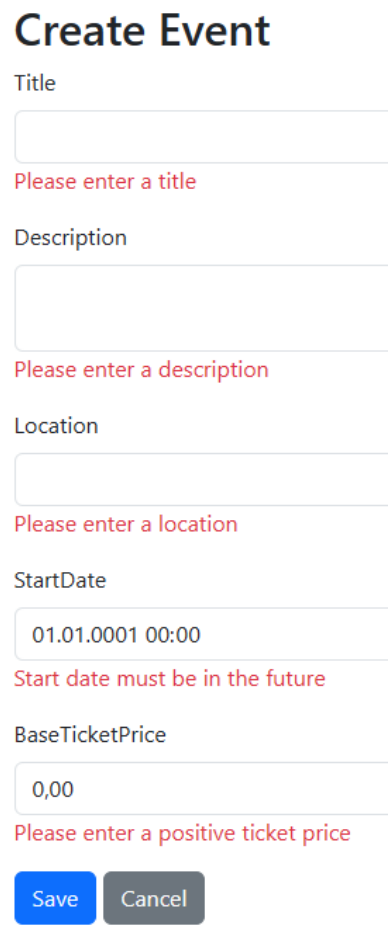

Рис. 3.1. Приклад серверної валідації даних для Event.

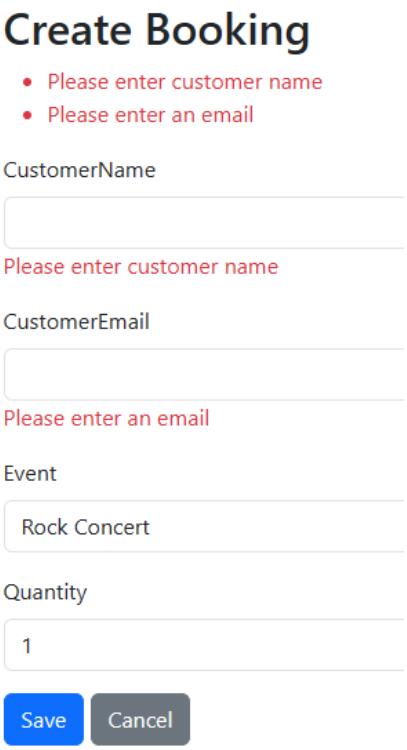

Рис. 3.2.Приклад серверної валідації даних для Booking.

Висновок: У ході виконання лабораторної роботи №4 проєкт TicketBooking було розширено новою сутністю Booking, яка разом із Event утворює зв’язок один-до-багатьох у підході Code First. На основі EF Core та MVC реалізовано повний набір CRUD-операцій для обох моделей із формами Create/Edit, сторінками Details та підтвердженням видалення. Додатково впроваджено серверну валідацію даних через DataAnnotations, що забезпечило перевірку коректності введення перед збереженням у БД та відображення помилок у формах користувача.