Booking controller

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using EventEase3.Data;
using EventEase3.Models;

namespace EventEase3.Controllers
{
    public class BookingController : Controller
    {
        private readonly EventEaseDbContext _context;

        public BookingController(EventEaseDbContext context)
        {
            _context = context;
        }

        // GET: Booking
        public async Task<IActionResult> Index()
        {
            var eventEaseDbContext = _context.Bookings.Include(b => b.Event).Include(b =>
b.Venue);
            return View(await eventEaseDbContext.ToListAsync());
        }

        // GET: Booking/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var booking = await _context.Bookings
                .Include(b => b.Event)
                .Include(b => b.Venue)
                .FirstOrDefaultAsync(m => m.BookingId == id);
            if (booking == null)
            {
                return NotFound();
            }

            return View(booking);
        }

        // GET: Booking/Create
        public IActionResult Create()
        {
            ViewData["EventId"] = new SelectList(_context.Events, "EventId", "EventId");
            ViewData["VenueId"] = new SelectList(_context.Venues, "VenueId", "VenueId");
            return View();
        }

        // POST: Booking/Create
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("BookingId,EventId,VenueId,BookingDate")] Booking booking)
        {
            if (ModelState.IsValid)
            {
                _context.Add(booking);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            ViewData["EventId"] = new SelectList(_context.Events, "EventId", "EventId",
booking.EventId);
            ViewData["VenueId"] = new SelectList(_context.Venues, "VenueId", "VenueId",
booking.VenueId);
            return View(booking);
        }

        // GET: Booking/Edit/5
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var booking = await _context.Bookings.FindAsync(id);
            if (booking == null)
            {
                return NotFound();
            }
            ViewData["EventId"] = new SelectList(_context.Events, "EventId", "EventId",
booking.EventId);
            ViewData["VenueId"] = new SelectList(_context.Venues, "VenueId", "VenueId",
booking.VenueId);
            return View(booking);
        }

        // POST: Booking/Edit/5
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
[Bind("BookingId,EventId,VenueId,BookingDate")] Booking booking)
        {
            if (id != booking.BookingId)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(booking);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!BookingExists(booking.BookingId))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            ViewData["EventId"] = new SelectList(_context.Events, "EventId", "EventId",
booking.EventId);
            ViewData["VenueId"] = new SelectList(_context.Venues, "VenueId", "VenueId",
booking.VenueId);
            return View(booking);
        }

        // GET: Booking/Delete/5
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var booking = await _context.Bookings
                .Include(b => b.Event)
                .Include(b => b.Venue)
                .FirstOrDefaultAsync(m => m.BookingId == id);
            if (booking == null)
            {
                return NotFound();
            }

            return View(booking);
        }

        // POST: Booking/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var booking = await _context.Bookings.FindAsync(id);
            if (booking != null)
            {
                _context.Bookings.Remove(booking);
            }

            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

        private bool BookingExists(int id)
        {
            return _context.Bookings.Any(e => e.BookingId == id);
        }
    }
}
```

"The `BookingController` in this ASP.NET Core MVC application follows the conventional Model-View-Controller pattern for handling web requests (Microsoft, n.d.a). Data interaction within the controller is facilitated by Entity Framework Core, which provides a way to interact with the underlying database using C# objects (Microsoft, n.d.b)."

**Reference List:**

Microsoft. (n.d.a). *ASP.NET Core Documentation*. [online]. Available from: https://learn.microsoft.com/en-us/aspnet/core/ [Accessed 6  April 2025].

Microsoft. (n.d.b). *Entity Framework Core Documentation*. [online]. Available from: https://learn.microsoft.com/en-us/ef/core/ [Accessed 6 April 2025].

Events Controller

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using EventEase3.Data;

using EventEase3.Models;
```

```
namespace EventEase3.Controllers
{
    public class EventsController : Controller
    {
        private readonly EventEaseDbContext _context;

        public EventsController(EventEaseDbContext context)
        {
            _context = context;
        }

        // GET: Events
        public async Task<IActionResult> Index()
        {
            var eventEaseDbContext = _context.Events.Include(e => e.Venue);
            return View(await eventEaseDbContext.ToListAsync());
        }

        // GET: Events/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var @event = await _context.Events
                .Include(e => e.Venue)
                .FirstOrDefaultAsync(m => m.EventId == id);

            if (@event == null)
            {
                return NotFound();
            }

            return View(@event);
        }

        // GET: Events/Create
        public IActionResult Create()
        {
            ViewData["VenueId"] = new SelectList(_context.Venues, "VenueId", "VenueName");
            return View();
        }

        // POST: Events/Create
        [HttpPost]
        [ValidateAntiForgeryToken]
```

```
        public async Task<IActionResult>
Create([Bind("EventId,EventName,EventDate,Description,VenueId")] Event @event)
        {
            if (ModelState.IsValid)
            {
                _context.Add(@event);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            ViewData["VenueId"] = new SelectList(_context.Venues, "VenueId", "VenueName",
@event.VenueId);
            return View(@event);
        }

        // GET: Events/Edit/5
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var @event = await _context.Events.FindAsync(id);
            if (@event == null)
            {
                return NotFound();
            }
            ViewData["VenueId"] = new SelectList(_context.Venues, "VenueId", "VenueName",
@event.VenueId);
            return View(@event);
        }

        // POST: Events/Edit/5
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
[Bind("EventId,EventName,EventDate,Description,VenueId")] Event @event)
        {
            if (id != @event.EventId)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(@event);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!EventExists(@event.EventId))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            ViewData["VenueId"] = new SelectList(_context.Venues, "VenueId", "VenueName",
@event.VenueId);
            return View(@event);
        }


        // GET: Events/Delete/5
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
```

```
    var @event = await _context.Events
        .Include(e => e.Venue)
        .FirstOrDefaultAsync(m => m.EventId == id);

    if (@event == null)
    {
        return NotFound();
    }

    return View(@event);
}

// POST: Events/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var @event = await _context.Events.FindAsync(id);
    if (@event == null)
    {
        return NotFound();
    }
```

```
            _context.Events.Remove(@event);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

        private bool EventExists(int id)
        {
            return _context.Events.Any(e => e.EventId == id);
        }
    }
}
```

"The `EventsController` in this ASP.NET Core MVC application follows the conventional Model-View-Controller pattern for handling web requests (Microsoft, n.d.a). Data interaction within the controller is facilitated by Entity Framework Core, which provides a way to interact with the underlying database using C# objects (Microsoft, n.d.b)."

**Reference List**

Microsoft. (n.d.a). *ASP.NET Core Documentation*. [online]. Available from: https://learn.microsoft.com/en-us/aspnet/core/ [Accessed 6 April 2025].

Microsoft. (n.d.b). *Entity Framework Core Documentation*. [online]. Available from: https://learn.microsoft.com/en-us/ef/core/ [Accessed 6 April 2025].

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using EventEase3.Data;
using EventEase3.Models;

namespace EventEase3.Controllers
{
    public class VenueController : Controller
    {
        private readonly EventEaseDbContext _context;

        public VenueController(EventEaseDbContext context)
        {
            _context = context;
        }

        // GET: Venue
        public async Task<IActionResult> Index()
        {
            return View(await _context.Venues.ToListAsync());
        }
```

```
// GET: Venue/Details/5
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var venue = await _context.Venues
        .FirstOrDefaultAsync(m => m.VenueId == id);
    if (venue == null)
    {
        return NotFound();
    }

    return View(venue);
}

// GET: Venue/Create
public IActionResult Create()
{
    return View();
}
```

```
        // POST: Venue/Create
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("VenueId,VenueName,Location,ImageUrl")]
Venue venue)
        {
            if (ModelState.IsValid)
            {
                _context.Add(venue);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            return View(venue);
        }

        // GET: Venue/Edit/5
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var venue = await _context.Venues.FindAsync(id);
            if (venue == null)
            {
                return NotFound();
            }
            return View(venue);
        }

        // POST: Venue/Edit/5
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
    [Bind("VenueId,VenueName,Location,ImageUrl")] Venue venue)
        {
            if (id != venue.VenueId)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
        {
            _context.Update(venue);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!VenueExists(venue.VenueId))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(venue);
}

// GET: Venue/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var venue = await _context.Venues
        .FirstOrDefaultAsync(m => m.VenueId == id);
    if (venue == null)
    {
        return NotFound();
    }

    return View(venue);
}

// POST: Venue/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var venue = await _context.Venues.FindAsync(id);
    if (venue != null)
    {
        _context.Venues.Remove(venue);
    }
```

```
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

        private bool VenueExists(int id)
        {
            return _context.Venues.Any(e => e.VenueId == id);
        }
    }
}
```

"The `VenueController` in this ASP.NET Core MVC application manages operations related to event venues, adhering to the framework's request handling conventions (Microsoft, n.d.a). Data persistence and retrieval for venue information are handled using Entity Framework Core, interacting with the `EventEaseDbContext` (Microsoft, n.d.b)."

**Reference List**

Microsoft. (n.d.a). *ASP.NET Core Documentation*. [online]. Available from: https://learn.microsoft.com/en-us/aspnet/core/ [Accessed 6 April 2025].

Microsoft. (n.d.b). *Entity Framework Core Documentation*. [online]. Available from: https://learn.microsoft.com/en-us/ef/core/ [Accessed 6 April 2025].

```
using System;
using System.Collections.Generic;
using EventEase3.Models;
using Microsoft.EntityFrameworkCore;

namespace EventEase3.Data;

public partial class EventEaseDbContext : DbContext
{
    public EventEaseDbContext()
    {
    }

    public EventEaseDbContext(DbContextOptions<EventEaseDbContext> options)
        : base(options)
    {
    }

    public virtual DbSet<Booking> Bookings { get; set; }

    public virtual DbSet<Event> Events { get; set; }

    public virtual DbSet<Venue> Venues { get; set; }
    protected override void OnModelCreating(ModelBuilder modelBuilder)

    {

        modelBuilder.Entity<Booking>(entity =>

        {

            entity.HasKey(e => e.BookingId).HasName("PK__Booking__73951ACD5BBF644A");



            entity.ToTable("Booking");



            entity.HasIndex(e => new { e.VenueId, e.BookingDate },
"CK_UniqueVenueBookingTime").IsUnique();



            entity.Property(e => e.BookingId).HasColumnName("BookingID");

            entity.Property(e => e.BookingDate).HasColumnType("datetime");

            entity.Property(e => e.EventId).HasColumnName("EventID");

            entity.Property(e => e.VenueId).HasColumnName("VenueID");



            entity.HasOne(d => d.Event).WithMany(p => p.Bookings)

                .HasForeignKey(d => d.EventId)

                .HasConstraintName("FK__Booking__EventID__3D5E1FD2");



            entity.HasOne(d => d.Venue).WithMany(p => p.Bookings)

                .HasForeignKey(d => d.VenueId)

                .HasConstraintName("FK__Booking__VenueID__3E52440B");

        });
```

```
modelBuilder.Entity<Event>(entity =>
{
    entity.HasKey(e => e.EventId).HasName("PK__Event__7944C8708E6D48FB");

    entity.ToTable("Event");

    entity.Property(e => e.EventId).HasColumnName("EventID");
    entity.Property(e => e.Description).IsUnicode(false);
    entity.Property(e => e.EventDate).HasColumnType("datetime");
    entity.Property(e => e.EventName)
        .HasMaxLength(255)
        .IsUnicode(false);
    entity.Property(e => e.VenueId).HasColumnName("VenueID");

    entity.HasOne(d => d.Venue).WithMany(p => p.Events)
        .HasForeignKey(d => d.VenueId)
        .HasConstraintName("FK__Event__VenueID__398D8EEE");
});

modelBuilder.Entity<Venue>(entity =>
{
    entity.HasKey(e => e.VenueId).HasName("PK__Venues__3C57E5D275A210CD");
```

```
                entity.Property(e => e.VenueId).HasColumnName("VenueID");

                entity.Property(e => e.ImageUrl)

                    .IsUnicode(false)

                    .HasColumnName("ImageURL");

                entity.Property(e => e.Location)

                    .HasMaxLength(255)

                    .IsUnicode(false);

                entity.Property(e => e.VenueName)

                    .HasMaxLength(255)

                    .IsUnicode(false);

            });


            OnModelCreatingPartial(modelBuilder);

        }


        partial void OnModelCreatingPartial(ModelBuilder modelBuilder);

    }
```

- "The `EventEaseDbContext` class inherits from `DbContext`, providing a connection to the database using Entity Framework Core (Microsoft, n.d.b)."
- "The `DbSet<Booking>`, `DbSet<Event>`, and `DbSet<Venue>` properties represent the tables in the database and allow for querying and saving of `Booking`, `Event`, and `Venue` entities respectively (Microsoft, n.d.b)."
- "The `OnModelCreating` method is used to configure the database schema, including defining primary keys and foreign key relationships, as seen in the configuration for the `Booking` entity (Microsoft, n.d.b)."

- "The unique index `CK_UniqueVenueBookingTime` on the `Booking` table, enforced through EF Core's configuration, ensures that a venue cannot be booked for two different events at the exact same time (Microsoft, n.d.b)."
- "The foreign key relationships, such as `HasOne(d => d.Event).WithMany(p => p.Bookings)`, define how the `Booking` entity relates to the `Event` entity in the database (Microsoft, n.d.b)."
- "The use of `IsUnicode(false)` and `HasMaxLength()` on string properties like `EventName` and `Location` defines the data types and constraints in the database schema (Microsoft, n.d.b)."
- "The specific database provider being used is SQL Server, as indicated by the scaffolding command using `Microsoft.EntityFrameworkCore.SqlServer` (Microsoft, n.d.c)." (Only include this if you discuss SQL Server-specific aspects).

**Reference List:**

Microsoft. (n.d.b). *Entity Framework Core Documentation*. [online]. Available from: https://learn.microsoft.com/en-us/ef/core/ [Accessed 6 April 2025].

Microsoft. (n.d.c). *Microsoft.EntityFrameworkCore.SqlServer*. [online]. Available from:

https://learn.microsoft.com/en-us/ef/core/providers/sql-server?tabs=dotnet-core-cli [Accessed 6 April 2025]. (Only include if you discuss SQL Server-specific aspects).

Venue index

```
@model IEnumerable<EventEase3.Models.Venue>


@{

    ViewData["Title"] = "Index";

}


<h1>Index</h1>


<p>

    <a asp-action="Create">Create New</a>
```

```
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.VenueName)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Location)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.ImageUrl)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.VenueName)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Location)
            </td>
            <td>
                    @* @Html.DisplayFor(modelItem => item.ImageUrl)*@
                <img src="@item.ImageUrl" alt="" style="width: 150px"/>
            </td>
            <td>
                <a asp-action="Edit" asp-route-id="@item.VenueId">Edit</a> |
                <a asp-action="Details" asp-route-id="@item.VenueId">Details</a> |
                <a asp-action="Delete" asp-route-id="@item.VenueId">Delete</a>
            </td>
        </tr>
}
    </tbody>
</table>
```

Focusing specifically on the line `<img src="@item.ImageUrl" alt="" style="width: 150px"/>`

- "The `<img>` HTML tag is used to embed images within a web page, with the `src` attribute specifying the image source (World Wide Web Consortium, n.d.a)."
- "In this ASP.NET Core MVC View, the `@item.ImageUrl` Razor syntax is used to dynamically inject the image URL from the `ImageUrl` property of the `Venue` model into the `src` attribute of the `<img>` tag (Microsoft, n.d.a)."
- "The `alt` attribute provides alternative text for the image if it cannot be displayed, improving accessibility (World Wide Web Consortium, n.d.b)."
- "Inline CSS styles, such as `style="width: 150px"`, can be used to control the visual presentation of the image directly within the HTML tag (World Wide Web Consortium, n.d.c)."

**Reference List:**

Microsoft. (n.d.a). *ASP.NET Core Documentation*. [online]. Available from: https://learn.microsoft.com/en-us/aspnet/core/ [Accessed 7 April 2025].

World Wide Web Consortium. (n.d.c). *Cascading Style Sheets (CSS) - Inline Styles*. [online]. Available from: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/style [Accessed 7 April 2025].

Appsetting.json

"ASP.NET Core utilizes the `appsettings.json` file for configuration settings, including database connection strings which are typically defined under the `ConnectionStrings` section (Microsoft, n.d.e)."

**Reference List:**

Microsoft. (n.d.e).

*Configuration in ASP.NET Core*. [online]. Available from: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-8.0 [Accessed 7 April 2025].