

```
// Seed bookings
public static async Task SeedBookingsAsync(ApplicationDbContext context)
{
    if (!await context.Booking.AnyAsync())
    {
        var Room1 = await context.Rooms.FirstOrDefaultAsync(r => r.RoomName == "Deluxe Suite");
        if (Room1 == null)
            return;

        var bookings = new Booking
        {
            RoomId = Room1.RoomId,
            StaffId = 1,
            GuestName = "Michael Jackson",
            NumberOfGuests = 2,
            BookingDate = DateOnly.FromDateTime(DateTime.Now),
            CheckInDate = DateTime.Now,
            CheckOutDate = DateTime.Now,
            CreatedAt = DateTime.Now,
            UpdatedAt = DateTime.Now
        };

        await context.Booking.AddAsync(bookings);

        var staff = await context.Staff.FindAsync(s => s.Id == 1);
        if (staff == null)
        {
            staff = new Staff { StaffId = 1 };
            await context.Staff.AddAsync(staff);
            await context.SaveChangesAsync();
        }
        await context.SaveChangesAsync();
    }
}
```

Looks like you're seeding a booking, but there are a few gotchas in this code. I'll walk through the issues and then show a cleaned-up, safer version.

Things to watch out for:

1. Foreign key order (big one)

You're creating a `Booking` that references `StaffId = 1` before you ever create the `Staff` entity. If your DB enforces FK constraints → **boom, exception.**

Rule of thumb:

Seed parent entities (Staff, Room) before child entities (Booking)

2. Hard-coding IDs

Manually setting `StaffId = 1` is risky if:

- Identity columns are enabled
- The DB already has data
- Migrations reseed differently across environments

Better: fetch the staff first and use its ID.

3. DateOnly + DateTime mismatch

You're mixing:

csharp

```
Copy code = DateOnly.FromDateTime(DateTime.Now), CheckInDate = DateTime.Now
```

That's fine if your model is intentional—but it often causes confusion between different date types.

4. Multiple SaveChangesAsync calls

You're calling `SaveChangesAsync()` twice. It works, but it's unnecessary.

A safer, cleaner version ✓

csharp

```
Copy codeic async Task SeedBookingsAsync(ApplicationDbContext context)
{
    if (!await context.Booking.AnyAsync()) return;
    var room = await context.Rooms.FirstOrDefaultAsync(r => r.RoomName == "Deluxe Suite");
    if (room == null) return;
    var staff = await context.Staff.FindAsync(s => s.Id == 1);
    if (staff == null) return;
    var bookings = new Booking
    {
        RoomId = room.Id,
        StaffId = staff.Id,
        GuestName = "Michael Jackson",
        NumberOfGuests = 2,
        BookingDate = DateOnly.FromDateTime(DateTime.Now),
        CheckInDate = DateTime.Now,
        CheckOutDate = DateTime.Now,
        CreatedAt = DateTime.Now,
        UpdatedAt = DateTime.Now
    };
    await context.Booking.AddAsync(bookings);
    await context.SaveChangesAsync();
}
```

ChatGPT

```
"Johnson" }; await context.Staff.AddAsync(staff); await context.
StaffId is generated } var booking = new Booking { RoomId = room
GuestName = "Michael Jackson", NumberOfGuests = 2, BookingDate =
DateOnly.FromDateTime(DateTime.Now), CheckInDate = DateTime.Now..
DateTime.Now.AddDays(10), CreatedAt = DateTime.Now, UpdatedAt =
context.Booking.AddAsync(booking); await context.SaveChangesAsync
```

Extra polish ideas ✨

- Use `DateTime.UtcNow` instead of `Now`
- Avoid magic values ("Deluxe Suite", 1) by constants
- Seed Rooms → Staff → Bookings in that order
- Consider `EnsureCreated()` vs migrations for dev seeding

If you want, tell me:

- what your entity models look like, or
- whether this is EF Core + SQL Server / SQLite,

and I'll tailor it perfectly to your setup.

```
using CityPoint.Data;
using Microsoft.EntityFrameworkCore;
using Microsoft.Identity.Client;
using Task2.Models;
```

```
namespace Task2.Data
{
    public class SeedData
    {
        public static async Task SeedRoom(
            context)
        {
            // Seed Rooms
            if (!await context.Room.AnyAs(
            {
                var rooms = new List<Room>
                {
                    new Room
                    {
                        RoomName = "Deluxe",
                        Roomnumber = 101,
                        Capacity = 2,
                        Description = "A luxuri",
                        PricePerNight = 250,
                        RoomSize = 45.0m,
                        IsAvailable = true
                    },
                    new Room
                    {
                        RoomName = "Standar",
                        Roomnumber = 102,
                        Capacity = 2,
                        Description = "A comf",
                        PricePerNight = 100,
                        RoomSize = 25.0m,
                        IsAvailable = true
                    },
                    new Room
                    {
                        RoomName = "Deluxe",
                        Roomnumber = 103,
                        Capacity = 2,
                        Description = "A spaci",
                        and city views."
                    }
                };
            
```

```
            foreach (var room in rooms)
            {
                context.Room.Add(room);
            }
            await context.SaveChangesAsync();
        }
    }
}
```

```
    public static async Task SeedStaff(
        context)
    {
        if (!await context.Staff.AnyAs(
        {
            var staff = new Staff
            {
                StaffName = "John Doe",
                Email = "john.doe@example.com",
                PasswordHash = "password123",
                Role = "Manager"
            };
            context.Staff.Add(staff);
        }
    );
}
```

```
    public static async Task SeedBookings(
        context)
    {
        if (!await context.Booking.AnyAs(
        {
            var booking = new Booking
            {
                RoomId = 101,
                GuestName = "Michael Jackson",
                NumberOfGuests = 2,
                BookingDate = DateOnly.FromDateTime(DateTime.Now),
                CheckInDate = DateTime.Now.AddDays(10),
                CreatedAt = DateTime.Now,
                UpdatedAt = DateTime.Now
            };
            context.Booking.Add(booking);
        }
    );
}
```

```
    public static async Task SeedRooms(
        context)
    {
        if (!await context.Room.AnyAs(
        {
            var rooms = new List<Room>
            {
                new Room
                {
                    RoomName = "Deluxe",
                    Roomnumber = 101,
                    Capacity = 2,
                    Description = "A luxuri",
                    PricePerNight = 250,
                    RoomSize = 45.0m,
                    IsAvailable = true
                },
                new Room
                {
                    RoomName = "Standar",
                    Roomnumber = 102,
                    Capacity = 2,
                    Description = "A comf",
                    PricePerNight = 100,
                    RoomSize = 25.0m,
                    IsAvailable = true
                },
                new Room
                {
                    RoomName = "Deluxe",
                    Roomnumber = 103,
                    Capacity = 2,
                    Description = "A spaci",
                    and city views."
                }
            };
            foreach (var room in rooms)
            {
                context.Room.Add(room);
            }
            await context.SaveChangesAsync();
        }
    );
}
```

```
    public static async Task SeedBookings(
        context)
    {
        if (!await context.Booking.AnyAs(
        {
            var booking = new Booking
            {
                RoomId = 101,
                GuestName = "Michael Jackson",
                NumberOfGuests = 2,
                BookingDate = DateOnly.FromDateTime(DateTime.Now),
                CheckInDate = DateTime.Now.AddDays(10),
                CreatedAt = DateTime.Now,
                UpdatedAt = DateTime.Now
            };
            context.Booking.Add(booking);
        }
    );
}
```

```
    public static async Task SeedRooms(
        context)
    {
        if (!await context.Room.AnyAs(
        {
            var rooms = new List<Room>
            {
                new Room
                {
                    RoomName = "Deluxe",
                    Roomnumber = 101,
                    Capacity = 2,
                    Description = "A luxuri",
                    PricePerNight = 250,
                    RoomSize = 45.0m,
                    IsAvailable = true
                },
                new Room
                {
                    RoomName = "Standar",
                    Roomnumber = 102,
                    Capacity = 2,
                    Description = "A comf",
                    PricePerNight = 100,
                    RoomSize = 25.0m,
                    IsAvailable = true
                },
                new Room
                {
                    RoomName = "Deluxe",
                    Roomnumber = 103,
                    Capacity = 2,
                    Description = "A spaci",
                    and city views."
                }
            };
            foreach (var room in rooms)
            {
                context.Room.Add(room);
            }
            await context.SaveChangesAsync();
        }
    );
}
```

```
    public static async Task SeedRooms(
        context)
    {
        if (!await context.Room.AnyAs(
        {
            var rooms = new List<Room>
            {
                new Room
                {
                    RoomName = "Deluxe",
                    Roomnumber = 101,
                    Capacity = 2,
                    Description = "A luxuri",
                    PricePerNight = 250,
                    RoomSize = 45.0m,
                    IsAvailable = true
                },
                new Room
                {
                    RoomName = "Standar",
                    Roomnumber = 102,
                    Capacity = 2,
                    Description = "A comf",
                    PricePerNight = 100,
                    RoomSize = 25.0m,
                    IsAvailable = true
                },
                new Room
                {
                    RoomName = "Deluxe",
                    Roomnumber = 103,
                    Capacity = 2,
                    Description = "A spaci",
                    and city views."
                }
            };
            foreach (var room in rooms)
            {
                context.Room.Add(room);
            }
            await context.SaveChangesAsync();
        }
    );
}
```

```

        Description = "Ideal for
larger living area.",
        PricePerNight = 180,
        RoomSize = 40.0m,
        IsAvailable = true
    },

    new Room
    {
        RoomName = "Executive",
        Roomnumber = 201,
        Capacity = 2,
        Description = "A premium
area and luxury amenities.",
        PricePerNight = 250,
        RoomSize = 55.0m,
        IsAvailable = true
    },

    new Room
    {
        RoomName = "Single",
        Roomnumber = 105,
        Capacity = 1,
        Description = "A compact
room for one person.",
        PricePerNight = 80,
        RoomSize = 18.0m,
        IsAvailable = false
    },

    new Room
    {
        RoomName = "Twin Room",
        Roomnumber = 106,
        Capacity = 2,
        Description = "A practical
room for colleagues or friends.",
        PricePerNight = 120,
        RoomSize = 28.0m,
        IsAvailable = true
    },

    new Room
    {
        RoomName = "President's Suite",
        Roomnumber = 301,
        Capacity = 4,
        Description = "A luxury
suite featuring premium furnishings, and exclusive services.
        PricePerNight = 400,
        RoomSize = 85.0m,
        IsAvailable = true
    }
};

await context.Room.AddRange(Rooms);
await context.SaveChangesAsync();
}

// Seed bookings
public static async Task SeedBookings(DbContext context)
{
    if (!await context.Booking.Any())
        return;

    var Room = await context.Rooms
        .Where(r => r.RoomName == "Deluxe Suite")
        .FirstOrDefaultAsync();
    if (Room == null)
        return;

    var staff = await context.Staff
        .Where(s => s.StaffID == 1)
        .FirstOrDefaultAsync();

```

```

if (staff == null)
{
    staff = new Staff { StaffId =
        await context.Staff.AddAsyn
        await context.SaveChanges,
    }
}

var bookings = new Booking
{
    RoomId = Room.RoomId,
    StaffId = 1,
    GuestName = "Michael Jack",
    NumberOfGuests = 2,
    BookingDate = DateOnly.Fr
    CheckInDate = DateTime.N
    CheckOutDate = DateTime.
    CreatedAt = DateTime.Now
    UpdatedAt = DateTime.Now
};

await context.Booking.AddAsy
await context.SaveChangesAsy
}
}

```

Nice, this helps a lot

You're *very* close — there's just one logic bug in your booking seeding tweaks.

💡 Main bug (why bookings never seed)

This line:

```
csharp  
    Copy codecontext.Booking.AnyAsync()) return;
```

s as:

- If there are no bookings, exit.

That's the opposite of what you want 😊

so your booking

 Correct logic

You s

csharp

[Copy code](#)

⚠ Secondary issue: hard-coded StaffID

100

- Let EF generate the ID

Then use `start!Start!Id`

5

```
Here's a corrected and safer version of your method:  
csharp  
    Copy codeic async Task SeedBookingsAsync(ApplicationDbContext exist, do nothing if (await context.Booking.AnyAsync()) return;  
    .SingleOrDefaultAsync(r => r.RoomName == "Deluxe Suite"); if (ro  
    await context.Staff .SingleOrDefaultAsync(s => s.LastName == "Jo  
    = new Staff { LastName = "Johnson" }; await context.Staff.AddAsy  
    context.SaveChangesAsync(); // ensures StaffId is generated } va  
    room.RoomId, StaffId = staff.StaffId, GuestName = "Michael Jacks  
    BookingDate = DateOnly.FromDateTime(DateTime.Now), CheckInDate =  
    CheckOutDate = DateTime.Now.AddDays(10), CreatedAt = DateTime.No  
    await context.Booking.AddAsync(bookings); await context.SaveChangesAsync();
```

► Optional improvements (not required, but nice)

- Use `DateTime.UtcNow` instead of `Now`

ChatGPT

- Seed order should be:
 1. Rooms
 2. Staff
 3. Bookings
- Avoid `SingleOrDefault` if duplicates are possible → use `FirstOrDefault`

If you want, next we can:

- Refactor this into one unified `SeedAsync()`
- Convert this to EF Core `HasData()`
- Or debug why staff/rooms aren't showing up in the DB

Just tell me 🌟

```
public static async Task SeedBooking(
    context)
{
    if (await context.Booking.AnyAsync())
        return;

    var Room = await context.Room.Singl
    if (Room == null)
        return;

    var staff = await context.Staff.Singl
    if (staff == null)
    {
        staff = new Staff
        {
            FirstName = "Fred",
            LastName = "Johnson",
            Department = "Default Depa
            Email = "staff123@gmail.com"
            PhoneNumber = "123-456-7
            IsActive = true
        };

        await context.Staff.AddAsync(sta
        await context.SaveChangesAsync()
    }

    var bookings = new Booking
    {
        RoomId = Room.RoomId,
        StaffId = staff.StaffId,
        GuestName = "Michael Jackson",
        NumberOfGuests = 2,
        BookingDate = DateOnly.FromI
        CheckInDate = DateTime.Now.,
        CheckOutDate = DateTime.Nov
        CreatedAt = DateTime.Now,
        UpdatedAt = DateTime.Now
    };
    await context.Booking.AddAsync(b
    await context.SaveChangesAsync()
} check for any errors and add 2 extr
```

You're in good shape 🌟

No compile-time errors here, and the logic is mostly solid. I'll first point out some pitfalls, then show you a version with 2 extra bookings added cleanly.

Code check (what's good & what to tweak)

What's correct

- `AnyAsync()` guard is now correct
- Room lookup is safe
- Staff is created before booking
- `StaffId = staff.StaffId` (no hard-coding)

ChatGPT

- SaveChanges placement is correct

⚠ Minor improvements (recommended)

1. SingleOrDefaultAsync can throw

If someone accidentally inserts another "Deluxe Suite", this will

👉 Safer:

csharp

Copy code

- defaultAsync(...)

2. Use UtcNow for timestamps

Not required, but best practice for real apps.

3. Check date logic

Your dates are valid (check-in < check-out 👍).

Version with 3 bookings total (original + 2 extra)

This keeps your structure but adds two more bookings with different c
csharp

```
Copy codeic async Task SeedBookingsAsync(ApplicationDbContext context)
{
    Booking? room = await context.Rooms
        .FirstOrDefaultAsync(r => r.RoomName == "Deluxe Suite");
    if (room == null) return;
    var staff = await context.Staff
        .FirstOrDefaultAsync(s => s.StaffId == 1);
    if (staff == null) return;
    var fred = new Staff
    {
        FirstName = "Fred",
        LastName = "Johnson",
        Department = "Default",
        Email = "staff123@gmail.com",
        PhoneNumber = "123-456-7890",
        IsActive = true
    };
    context.Staff.AddAsync(staff);
    await context.SaveChangesAsync();
    var booking1 = new Booking
    {
        RoomId = room.RoomId,
        StaffId = staff.StaffId,
        GuestName = "Sarah Connor",
        NumberOfGuests = 1,
        BookingDate = DateTime.UtcNow.AddDays(7),
        CheckOutDate = DateTime.UtcNow.AddDays(14),
        UpdatedAt = DateTime.UtcNow,
        CreatedAt = DateTime.UtcNow,
        UpdateBooking = false
    };
    var booking2 = new Booking
    {
        RoomId = room.RoomId,
        StaffId = staff.StaffId,
        GuestName = "Sarah Connor",
        NumberOfGuests = 1,
        BookingDate = DateTime.UtcNow.AddDays(14),
        CheckInDate = DateTime.UtcNow.AddDays(2),
        CheckOutDate = DateTime.UtcNow.AddDays(22),
        CreatedAt = DateTime.UtcNow,
        UpdatedAt = DateTime.UtcNow
    };
    context.Bookings.AddRange(booking1, booking2);
    await context.SaveChangesAsync();
}
```

🔥 Optional next upgrades

If you want to level this up:

- Seed different rooms per booking
- Prevent overlapping bookings
- Seed bookings only if none exist for that room
- Extract dates into helper methods for readability

Tell me what you want next and we'll polish it further 🤝

```
using CityPoint.Data;
using Microsoft.EntityFrameworkCore;
using Microsoft.Identity.Client;
using Task2.Models;

namespace Task2.Data
{
    public class SeedData
    {
        public static async Task SeedRoomsAsync(IApplicationBuilder app)
        {
            // Seed Rooms
            if (!await context.Rooms.AnyAsync())
            {
                var rooms = new List<Room>
                {
                    new Room
                    {
                        RoomName = "Deluxe Suite",
                        RoomNumber = 101,
                        Capacity = 2,
                        Description = "A luxury room with a balcony",
                        PricePerNight = 250,
                        IsActive = true
                    }
                };
                context.Rooms.AddRange(rooms);
                await context.SaveChangesAsync();
            }
        }
    }
}
```

```
RoomSize = 45.0m,  
IsAvailable = true  
},  
new Room  
{  
    RoomName = "Standar  
    Roomnumber = 102,  
    Capacity = 2,  
    Description = "A comf  
    PricePerNight = 100,  
    RoomSize = 25.0m,  
    IsAvailable = true  
},  
new Room  
{  
    RoomName = "Deluxe  
    Roomnumber = 103,  
    Capacity = 2,  
    Description = "A spaci  
and city views.",  
    PricePerNight = 150,  
    RoomSize = 30.0m,  
    IsAvailable = false  
},  
new Room  
{  
    RoomName = "Family  
    Roomnumber = 104,  
    Capacity = 4,  
    Description = "Ideal fo  
larger living area.",  
    PricePerNight = 180,  
    RoomSize = 40.0m,  
    IsAvailable = true  
},  
new Room  
{  
    RoomName = "Executive  
    Roomnumber = 201,  
    Capacity = 2,  
    Description = "A prem  
area and luxury amenities.",  
    PricePerNight = 250,  
    RoomSize = 55.0m,  
    IsAvailable = true  
},  
new Room  
{  
    RoomName = "Single  
    Roomnumber = 105,  
    Capacity = 1,  
    Description = "A comp  
    PricePerNight = 80,  
    RoomSize = 18.0m,  
    IsAvailable = false  
},  
new Room  
{  
    RoomName = "Twin R  
    Roomnumber = 106,  
    Capacity = 2,  
    Description = "A pract  
for colleagues or friends.",  
    PricePerNight = 120,  
    RoomSize = 28.0m,  
    IsAvailable = true  
},  
new Room  
{  
    RoomName = "Preside
```

```

        Roomumber = 301,
        Capacity = 4,
        Description = "A luxury room featuring a king-size bed, a large en-suite bathroom with a walk-in shower, a spacious living area with a sofa and armchairs, a dining table for four, and a fully equipped kitchenette. The room is decorated with high-end furniture and lighting, and offers panoramic views of the surrounding landscape through floor-to-ceiling windows. It includes a private balcony with outdoor seating and a small barbecue area. The room is perfect for couples or small groups looking for a comfortable and indulgent stay."}),
        PricePerNight = 400,
        RoomSize = 85.0m,
        IsAvailable = true
    }
}
await context.Room.AddRoom(r)
await context.SaveChangesAsync()
}
}

```

```

// Seed bookings
public static async Task SeedBookings(DbContext context)
{
    if (await context.Booking.AnyAsync())
        return;

    var room = await context.Rooms
        .FirstOrDefaultAsync(r => r.Id == 1);
    if (room == null)
        return;

    var staff = await context.Staff
        .FirstOrDefaultAsync(s => s.Id == 1);
    if (staff == null)
    {
        staff = new Staff
        {
            FirstName = "Fred",
            LastName = "Johnson",
            Department = "Default Department",
            Email = "staff123@gmail.com",
            PhoneNumber = "123-4567-8901",
            IsActive = true
        };
        await context.Staff.AddAsync(staff);
        await context.SaveChangesAsync();
    }

    var bookings = new List<Booking>
    {
        new Booking
        {
            RoomId = room.RoomId,
            StaffId = staff.StaffId,
            GuestName = "Michael Jackson",
            NumberOfGuests = 2,
            BookingDate = DateOnly.FromDateTime(DateTime.Now),
            CheckInDate = DateTime.Now,
            CheckOutDate = DateTime.Now.AddDays(1),
            CreatedAt = DateTime.UtcNow,
            UpdatedAt = DateTime.UtcNow
        },
        new Booking
        {
            RoomId = room.RoomId,
            StaffId = staff.StaffId,
            GuestName = "Sarah Connor",
            NumberOfGuests = 3,
            BookingDate = DateOnly.FromDateTime(DateTime.Now),
            CheckInDate = DateTime.Now,
            CheckOutDate = DateTime.Now.AddDays(2),
            CreatedAt = DateTime.UtcNow,
            UpdatedAt = DateTime.UtcNow
        },
        new Booking
        {
            RoomId = room.RoomId,
            StaffId = staff.StaffId,
            GuestName = "John Connor",
            NumberOfGuests = 1,
            BookingDate = DateOnly.FromDateTime(DateTime.Now),
            CheckInDate = DateTime.Now,
            CheckOutDate = DateTime.Now.AddDays(1),
            CreatedAt = DateTime.UtcNow,
            UpdatedAt = DateTime.UtcNow
        }
    };
}
```

```

        RoomId = room.RoomId,
        StaffId = staff.StaffId,
        GuestName = "John Wick",
        NumberOfGuests = 2,
        BookingDate = DateOnly.F
        CheckInDate = DateTime.U
        CheckOutDate = DateTime
        CreatedAt = DateTime.Utc
        UpdatedAt = DateTime.Utc
    },
};

};

await context.Booking.AddRar
await context.SaveChangesAsy
}
}

```

the other bookings aren't seeding

Yep — this is a classic seeding trap, and you're actually hitting two separate seeding problems. The first run correctly inserts all three bookings. However, the second run (or after a migration restart) finds the first booking already exists, so it only inserts the second and third bookings. This is because the check is based on the primary key (RoomId), which is unique but not guaranteed to be the same across runs.

Let's break it down cleanly 🤝

⚠️ Main reason the other bookings aren't seeding

This line is the culprit:

csharp

```
Copy codeoncontext.Booking.AnyAsync() return;
```

What's happening

- First run → no bookings → all 3 get inserted ✅
- Second run (or after a migration restart) → 1 booking already exists → only the first booking ever appears
- Result: only the first booking ever appears

This is expected behavior with your current logic.

✓ Fix option 1 (most common): seed only once per room

Instead of checking *any* booking, check bookings for that room:

csharp

```
Copy codeoncontext.Booking.AnyAsync(b => b.RoomId == room.RoomId)
```

⚠️ But this requires `room` to be loaded first.

Correct order:

csharp

```
Copy codeawait context.Room .FirstOrDefaultAsync(r => r.RoomNa
null) return; if (await context.Booking.AnyAsync(b => b.RoomId =
```

⚠️ Second problem: capacity mismatch (this can silently fail)

Your Deluxe Suite has:

csharp

```
Copy code2
```

But this booking has:

csharp

```
Copy code "Sarah Connor", NumberOfGuests = 3, // ❌ exceeds capacity
```

If you have:

- validation logic
- FluentValidation
- DB constraint
- or business rules in `SaveChanges`

👉 This booking will not be saved, and EF may stop processing the batch.

✓ Fix it

Change to:

csharp

```
Copy codests = 2
```

or use a larger room.

✓ Fully fixed `SeedBookingsAsync` (safe & predictable)

csharp

```
Copy code
```

ChatGPT

```
public static async Task SeedBookingsAsync(ApplicationDbContext context)
{
    Room room = await context.Room.FirstOrDefaultAsync(r => r.RoomName == "Deluxe Suite");
    if (room == null) return;
    var staff = await context.Staff.FirstOrDefaultAsync(s => s.FirstName == "Fred" & s.LastName == "Default Department");
    if (staff == null) staff = new Staff { FirstName = "Fred", LastName = "Default Department", Email = "staff123@gmail.com", PhoneNumber = "(555) 123-4567" };
    await context.Staff.AddAsync(staff);
    await context.SaveChangesAsync();
    var bookings = new List<Booking> {
        new Booking { RoomId = room.Id, StaffId = staff.Id, GuestName = "Michael Jackson", NumberOfGuests = 2, CheckInDate = DateTime.UtcNow.AddDays(7), CheckOutDate = DateTime.UtcNow.AddDays(14), CreatedAt = DateTime.UtcNow, UpdatedAt = DateTime.UtcNow },
        new Booking { RoomId = room.Id, StaffId = staff.Id, GuestName = "Sarah Connor", NumberOfGuests = 2, CheckInDate = DateTime.UtcNow.AddDays(12), CheckOutDate = DateTime.UtcNow.AddDays(14), CreatedAt = DateTime.UtcNow, UpdatedAt = DateTime.UtcNow }
    };
    context.Bookings.AddRange(bookings);
    await context.SaveChangesAsync();
}
```

Pro tip (for debugging)

Wrap `SaveChangesAsync()` in a `try/catch` while developing:

csharp

```
Copy code context.SaveChanges(); } catch (Exception ex) { throw; }
```

This instantly reveals why EF skips records.

If you want next:

- Seed bookings across multiple rooms
- Prevent overlapping dates
- Auto-generate realistic bookings
- Or convert this to EF `HasData()`

Just say the word

Quick check first — *where* are you trying to highlight columns?

Your `j=` looks like a typo, so I'll cover the most common cases, and you can apply them to your specific needs.

Excel / Google Sheets

Highlight an entire column

- Click the column letter (A, B, C...)
- Or use Ctrl + Space

Conditional highlight (e.g. based on value)

1. Select the column
2. Home → Conditional Formatting
3. Choose a rule (Greater Than, Text Contains, etc.)

HTML table (CSS)

Highlight a whole column

html

```
Copy code nth-child(2), th:nth-child(2) { background-color: #ffccbc; }
```

This highlights column 2

JavaScript (highlight column on click)

javascript

```
Copy code querySelectorAll("td").forEach(td => { td.addEventListener("click", () => { td.cellIndex + 1; document.querySelector(`td:nth-child(${td.cellIndex + 1})`).style.backgroundColor = "yellow"; }); });
});
```

Click any cell → its column highlights

C# WinForms DataGridView

csharp

```
Copy code w1.Columns[2].DefaultCellStyle.BackColor = Color.LightGreen;
```

ASP.NET / Razor table

html

```
Copy code <td highlight>...</td>
```

css

```
Copy code { background-color: #d1ecf1; }
```

 If you tell

There are Indentation err

Ahhh got it — you're fighting indentation, not "columns" 😅

Good news: you do NOT have to fix it line by line.

Here's how to fix everything at once, depending on your editor 

Visual Studio (most likely, since EF Core)

- ◆ Auto-format the entire file

Shortcut:

mathematica

Copy code 

 This instantly fixes all indentation in the current file.

- ◆ Auto-format a selected block

1. Select the messy code

2. Press:

r

Copy code 

VS Code

- ◆ Format entire file

r

Copy code + 

- ◆ Format selection only

1. Highlight the code

2. Right-click → Format Selection

Just fixing indentation manually (fast way)

Indent everything

1. Select all ()

2. Press:

mathematica

Copy code

Un-indent everything

mathematica

Copy code

 If formatting STILL doesn't work

Make sure C# formatting is enabled:

Visual Studio

Tools → Options → Text Editor → C# → Formatting → General

Check Automatically format on paste

VS Code

Install:

- C# Dev Kit
- C# (by Microsoft)

 Pro tip (EF Core lifesaver)

Turn on format on save so this never happens again:

VS Code

json

Copy code  : true

If you tell me which editor you're using (Visual Studio / VS Code / Rider) you never see indentation errors again 

```
using CityPoint.Data;
using Microsoft.EntityFrameworkCore;
using Microsoft.Identity.Client;
using Task2.Models;
```

```
namespace Task2.Data
{
    public class SeedData
    {
        public static async Task SeedRoomContext()
        {
            // Seed Rooms
            if (!await context.Room.AnyAs<...>
            {
                var rooms = new List<Room>
                {
                    new Room
                    {
                        RoomName = "Deluxe",
                        Roomnumber = 101,
                        Capacity = 2,
                        Description = "A luxuri...
                        PricePerNight = 250,
                        RoomSize = 45.0m,
                        IsAvailable = true
                    },
                    new Room
                    {
                        RoomName = "Standar...
                        Roomnumber = 102,
                        Capacity = 2,
                        Description = "A comf...
                        PricePerNight = 100,
                        RoomSize = 25.0m,
                        IsAvailable = true
                    },
                    new Room
                    {
                        RoomName = "Deluxe",
                        Roomnumber = 103,
                        Capacity = 2,
                        Description = "A spaci...
                        and city views.",
                        PricePerNight = 150,
                        RoomSize = 30.0m,
                        IsAvailable = false
                    },
                    new Room
                    {
                        RoomName = "Family",
                        Roomnumber = 104,
                        Capacity = 4,
                        Description = "Ideal fo...
                        larger living area."
                        PricePerNight = 180,
                        RoomSize = 40.0m,
                        IsAvailable = true
                    },
                    new Room
                    {
                        RoomName = "Executive",
                        Roomnumber = 201,
                        Capacity = 2,
                        Description = "A prem...
                        area and luxury amenities."
                        PricePerNight = 250,
                        RoomSize = 55.0m,
                        IsAvailable = true
                    },
                    new Room
                    {
                        RoomName = "Single",
                        Roomnumber = 105,
                        Capacity = 1,
                        Description = "A comp...
                        PricePerNight = 80,
                    }
                };
            }
        }
    }
}
```

```

        RoomSize = 18.0m,
        IsAvailable = false
    },

    new Room
    {
        RoomName = "Twin R
        Roomnumber = 106,
        Capacity = 2,
        Description = "A pract
for colleagues or friends.",
        PricePerNight = 120,
        RoomSize = 28.0m,
        IsAvailable = true
    },

    new Room
    {
        RoomName = "Preside
        Roomnumber = 301,
        Capacity = 4,
        Description = "A luxur
premium furnishings, and exclusive se
        PricePerNight = 400,
        RoomSize = 85.0m,
        IsAvailable = true
    }
};

await context.Room.AddRar
await context.SaveChanges(
)
}

// Seed bookings
public static async Task SeedBoo
context)
{
    var room = await context.Roo
    r.RoomName == "Deluxe Suite");
    if (room == null)
        return;

    if (await context.Booking.AnyA
    room.RoomId))
        return;

    var staff = await context.Staff.I
1);
    if (staff == null)
    {
        staff = new Staff
        {
            FirstName = "Fred",
            LastName = "Johnson",
            Department = "Default D
            Email = "staff123@gmail
            PhoneNumber = "123-45
            IsActive = true
        };
        await context.Staff.AddAsyn
        await context.SaveChanges(
    }

    var now = DateTime.UtcNow;
    var bookings = new List<Book
    {
        new Booking
        {
            RoomId = room.RoomId,
            StaffId = staff.StaffId,
            GuestName = "Michael J.
    }
}

```

```

        NumberOfGuests = 2,
        BookingDate = DateOnly
        CheckInDate = now.Add[1]
        CheckOutDate = now.Ad
        CreatedAt = now,
        UpdatedAt = now
    },
    new Booking
    {
        RoomId = room.RoomId,
        StaffId = staff.StaffId,
        GuestName = "Sarah Cor
        NumberOfGuests = 2, //
        BookingDate = DateOnly
        CheckInDate = now.Add[1]
        CheckOutDate = now.Ad
        CreatedAt = now,
        UpdatedAt = now
    },
    new Booking
    {
        RoomId = room.RoomId,
        StaffId = staff.StaffId,
        GuestName = "John Wicl
        NumberOfGuests = 2,
        BookingDate = DateOnly
        CheckInDate = now.Add[1]
        CheckOutDate = now.Ad
        CreatedAt = now,
        UpdatedAt = now
    }
};

await context.Booking.AddRar
await context.SaveChangesAsy
}

```

}

} only one booking still posts

You're not crazy — your code is doing *exactly* what you told it to do 🎉

This is a seeding logic issue, not an EF bug.

Let's walk through it slowly and clearly.

⚠️ Why only ONE booking appears

This line is the reason 👇

csharp

Copy code`context.Booking.AnyAsync(b => b.RoomId == room.RoomId)`
What actually happens in real life

1. You previously ran the app
2. ONE booking for Deluxe Suite already exists in the database
3. You run the app again
4. This condition becomes true
5. Method exits immediately
6. ✗ The other two bookings are never inserted

So EF is not “posting one booking” —

👉 it's refusing to seed again because the room already has a booking

✅ Important concept (this trips everyone up)

Seed methods only run once unless you explicitly allow more than one

Your database keeps old data between runs.

💡 How to PROVE this right now

Run this in SQL Server / SQLite / whatever DB you're using:

sql

Copy code`OM Booking;

You'll see only one existing row, from an earlier run.

- Option 1: Reset the database (FASTEST fix)

If this is a dev project (which it clearly is):

Visual Studio

- Open Package Manager Console

powershell

Copy code `Update-Database`

Then run the app again → all 3 bookings will appear

- Option 2: Allow multiple bookings per room (recommended for least privilege)

Change your guard to something more specific, like this:

csharp

Copy code `context.Booking.AnyAsync(b => b.GuestName == "Michael")`

That way:

- Seed runs once
- All 3 bookings get inserted
- Future runs don't duplicate them

- Option 3: Remove the guard entirely (dev-only)

csharp

Copy code `HIS // if (await context.Booking.AnyAsync(b => b.Room`

⚠ This will insert duplicates every time you run the app

Good for testing, bad for production.

💡 Why your logic is still GOOD

Your current logic is correct for real apps:

- Prevents duplicate bookings
- Protects data integrity
- Mimics real-world booking rules

The confusion comes from persistent databases during development.

- TL;DR

- ✗ Nothing is wrong with EF
- ✗ Nothing is wrong with your code
- Your DB already has a booking
- Seeding is intentionally stopping
- 🔥 Drop the DB or loosen the guard

If you want, next I can:

- Make a proper SeedAsync pipeline
- Add room availability logic
- Prevent overlapping bookings
- Show how to see exactly what EF is inserting

Just say the word 💡

```
@model IEnumerable<Task2.Models.I
```

```
@{
    ViewData["Title"] = "Index";
}
```

```
<h1>Index</h1>
```

```
<p>
    <a asp-action="Create">Create New Room</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.RoomName)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.RoomType)
            </th>
        </tr>
    <thead>
    <tbody>
        <tr>
            <td>Room A</td>
            <td>Single</td>
        </tr>
        <tr>
            <td>Room B</td>
            <td>Double</td>
        </tr>
        <tr>
            <td>Room C</td>
            <td>Double</td>
        </tr>
    <tbody>
</table>
```

```

<th>
    @Html.DisplayNameFor(mc
</th>
<th></th>
</tr>
</thead>
<tbody>
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem =>
                item.RoomId
        </td>
        <td>
            @Html.DisplayFor(modelItem =>
                item.RoomName
        </td>
        <td>
            @Html.DisplayFor(modelItem =>
                item.RoomType
        </td>
        <td>
            @Html.DisplayFor(modelItem =>
                item.RoomCapacity
        </td>
        <td>
            @Html.DisplayFor(modelItem =>
                item.RoomStatus
        </td>
        <td>
            @Html.DisplayFor(modelItem =>
                item.RoomDescription
        </td>
        <td>
            @Html.DisplayFor(modelItem =>
                item.RoomAddress
        </td>
        <td>
            @Html.DisplayFor(modelItem =>
                item.RoomPhone
        </td>
        <td>
            @Html.DisplayFor(modelItem =>
                item.RoomEmail
        </td>
        <td>
            <a href="#" asp-action="Edit" asp-route-id="@item.RoomId">Edit</a> |
            <a href="#" asp-action="Details" asp-route-id="@item.RoomId">Details</a> |
            <a href="#" asp-action="Delete" asp-route-id="@item.RoomId">Delete</a>
        </td>
    </tr>
}
</tbody>
</table> Add Filters for available roo

```

Nice, this is a classic MVC Index page upgrade 🌟

We'll do this cleanly and *properly*, not with hacks.

You want filters for:

1.  Available rooms
 2.  Best prices
 3.  Capacity

That requires two parts:

- Controller logic
 - Small UI changes in the view

1 Update the Controller (RoomsController)

Add filter parameters to `Index`

csharp

```
        Copy codec Task<IActionResult> Index( bool? available, int? mi  
rooms = _Context.Room.AsQueryable(); if (available == true) { ro  
r.IsAvailable); } if (minCapacity.HasValue) { rooms = rooms.Whe
```

ChatGPT

```
minCapacity.Value); } if (maxPrice.HasValue) { rooms = rooms.Where(maxPrice.Value); } return View(await rooms.ToListAsync()); }
```

💡 This keeps it:

- Fast
- SQL-translated
- Clean

② Add Filter UI to the View (Index.cshtml)

- ◆ Put this above the table

html

```
Copy code<get class="mb-3"> <div class="row g-2"> <!-- Available Only --> <div class="form-check mt-2"> <input class="form-check-input" type="checkbox" value="true" id="available" @(_Context.Request.Query["available"])> <label class="form-check-label" for="available"> Available Only Capacity --> <div class="col-md-3"> <input type="number" name="minCapacity" placeholder="Min Capacity" value="@_Context.Request.Query["minCapacity"]"> <div class="col-md-3"> <input type="number" step="0.01" name="maxPrice" placeholder="Max Price" value="@_Context.Request.Query["maxPrice"]"> <button type="submit" class="btn btn-primary w-100"> Search </button> </div> </div> </form>
```

③ What each filter does (plain English)

Filter	Meaning
Available Only	Shows rooms where IsAvailable == true
Min Capacity	Rooms that fit X or more guests
Max Price	"Best price" under a chosen limit

✓ Result

You now have:

- ✓ Dynamic filtering
- ✓ Query-string based URLs (bookmarkable)
- ✓ No JS required
- ✓ Works with EF Core efficiently

💡 Optional upgrades (tell me if you want them)

- Sort by cheapest first
- Dropdown for capacity instead of input
- Show number of matching rooms
- Combine filters with pagination
- Highlight unavailable rooms visually

You're building this the right way — say what you want next and we'll

```
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 CityPoint.Data;
using Task2.Models;

namespace CityPoint.Controllers
{
    public class RoomsController : Controller
    {
        private readonly ApplicationDbContext _context;

        public RoomsController(ApplicationDbContext context)
        {
            _context = context;
        }

        // GET: Rooms
        public async Task<IActionResult>
```

```

    {
        return View(await _context.Ro
    }

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

        var room = await _context.Ro
            .FirstOrDefaultAsync(m =>
        if (room == null)
        {
            return NotFound();
        }

        return View(room);
    }

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

    // POST: Rooms/Create
    // To protect from overposting a
    you want to bind to.
    // For more details, see http://gc
    LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
    Create([Bind("RoomId,RoomName,Ro
    nNight,RoomSize,IsAvailable")] Room
    {
        if (ModelState.IsValid)
        {
            _context.Add(room);
            await _context.SaveChangesAsync();
            return RedirectToAction(nar
        }
        return View(room);
    }

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

        var room = await _context.Ro
        if (room == null)
        {
            return NotFound();
        }
        return View(room);
    }

    // POST: Rooms/Edit/5
    // To protect from overposting a
    you want to bind to.
    // For more details, see http://gc
    LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
    [Bind("RoomId,RoomName,Roomumk
    RoomSize,IsAvailable")] Room room)

```

```

    {
        if (id != room.RoomId)
        {
            return NotFound();
        }

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

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

        var room = await _context.Rooms
            .FirstOrDefaultAsync(m =>
        if (room == null)
        {
            return NotFound();
        }

        return View(room);
    }

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

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

    private bool RoomExists(int id)
    {
        return _context.Rooms.Any(e =>
    }
} This is the room controller
}

```

Perfect — thanks for pasting the controller 👍

Right now your RoomsController is still using the basic Index(), so the · work yet.

We just need to upgrade the Index action. Everything else in your cont

● What's wrong right now

Your current Index action:

csharp

Copy codec Task<IActionResult> Index() { return View(await _co

This:

- Ignores query parameters
- Always returns all rooms
- Makes filters in the view do nothing

Fix: Replace Index with a filter-aware version

Replace your Index() method with this:

csharp

Copy codec Task<IActionResult> Index(bool? available, int? mi
rooms = _context.Room.AsQueryable(); // Available rooms filter i
rooms.Where(r => r.IsAvailable); } // Capacity filter if (minCap
rooms.Where(r => r.Capacity >= minCapacity.Value); } // Best pri
rooms = rooms.Where(r => r.PricePerNight <= maxPrice.Value); } r
rooms.ToListAsync()); }

This works automatically with query strings

No routing changes

No JS needed

💡 Why this works

When you submit the form in your view:

html

Copy codec `d="get">`

The browser sends a URL like:

arduino

Copy codec `able=true&minCapacity=2&maxPrice=200`

ASP.NET MVC:

- Matches those values to the action parameters
- Applies filters
- Returns only matching rooms

You do NOT need to change:

- Create
- Edit
- Delete
- Details
- Views (besides the filter UI you already added)

💡 Quick test checklist

Try these URLs manually in your browser:

arduino

Copy codec `able=true`
`/Rooms?minCapacity=4`
`/Rooms?maxPrice=150`
`/Rooms?available=true&minCapacity=2&maxPrice=200`

If those work → your UI will work too

🔥 Optional next upgrades (easy wins)

- Sort by price (low → high)
- Add “Clear filters” button
- Show count of results
- Highlight unavailable rooms
- Add paging (VERY common interview topic)

Say which one you want next and we'll bolt it on cleanly 🌟

```
<form method="get" class="mb-3">
  <div class="row g-2">
```

```
    <!-- Available -->
```

```

<div class="col-md-3">
  <div class="form-check mt-2">
    <input class="form-check-input" type="checkbox" name="available" value="true" id="available" @(Context.Request.Query["Available Only"])
  />
  <label class="form-check-label"> Available Only </label>
</div>
</div>

<!-- Minimum Capacity -->
<div class="col-md-3">
  <input type="number" name="minCapacity" class="form-control" placeholder="Min Capacity" value="@Context.Reques
</div>

<!-- Maximum Capacity -->
<div class="col-md-3">
  <input type="number" name="maxCapacity" class="form-control" placeholder="Max Capacity" value="@Context.Reques
</div>

<!-- Best Price -->
<div class="col-md-3">
  <input type="number" step="0.01" name="maxPrice" class="form-control" placeholder="Max Price" value="@Context.Reques
</div>

<!-- Submit -->
<div class="col-md-3">
  <button type="submit" class="btn-primary" value="Apply Filters"> Apply Filters </button>
</div>

</div>
</form> ADDED A Form for max price

```

Nice 🌟 the form itself is mostly correct, but right now one of your filters is up in the controller.

You added Max Capacity in the view — but the controller doesn't know about it. Let's fix this cleanly.

❗ What's missing right now

In your form you have:

html

Copy code`capacity"`
...but your controller does NOT accept `maxCapacity` yet.

So MVC just ignores it.

✓ Fix 1: Update the Index action (REQUIRED)

Replace your Index method with this version:

csharp

```
Copy codeC Task<IActionResult> Index( bool? available, int? minCapacity? maxPrice) { var rooms = _context.Room.AsQueryable(); if (rooms.Where(r => r.IsAvailable); } if (minCapacity.HasValue) { if (r >= minCapacity.Value); } if (maxCapacity.HasValue) { rooms = rooms.Where(r => r <= maxCapacity.Value); }
}
```

ChatGPT

```
maxCapacity.Value); } if (maxPrice.HasValue) { rooms = rooms.Where(r => r.Price == maxPrice.Value); } return View(await rooms.ToListAsync()); }
```

- ✓ Now all four filters work together

⚠ Small UI issue (layout)

You currently have 5 `.col-md-3` columns in a single row.

Bootstrap rows are 12 columns wide → this will wrap awkwardly.

🔧 Quick fix

Change them to `col-md-2` or split into two rows.

Example:

html

Copy code `"col-md-2">...</div>`

💡 How to test quickly

Paste these into your browser:

arduino

Copy code `label=true`

`/Rooms?minCapacity=2&maxCapacity=4`

`/Rooms?maxPrice=150`

`/Rooms?available=true&minCapacity=2&maxCapacity=4&maxPrice=200`

If these work → the form will too ✓

🔥 Optional next upgrades

- Add sorting (price ↑↓)
- Add Clear Filters button
- Disable impossible inputs (min > max)
- Show active filters as badges
- Add pagination

Say the word and we'll bolt it on properly 👉

Any time I tap on the is available filter
show

Ah! I see exactly what's happening 😬 — this is a common MVC mistake.
Let's break it down carefully.

1 Why it's not working

Your checkbox is rendered like this:

html

```
Copy code<input type="checkbox" name="available" @(Context.Request.Query["available"] == "true" ? "checked" : "")
```

Problem:

- When a checkbox is unchecked, the browser does NOT send any value.
- So `available` in the controller is null, not false.
- Your controller logic:

csharp

```
Copy codeif (available == true) { rooms = rooms.Where(r => r.IsAvailable) }
```

- This only runs if `available == true`.

- But if you check and uncheck, your page reload may still keep old values clearing correctly.

- Also, your controller never handles `available == false`.

💡 In other words: the filter only ever works if the checkbox is checked.

2 Correct way to handle a checkbox filter in MVC

Option A: Keep "Available Only" (simplest)

Controller logic:

csharp

```
Copy codeif (available == true) // only filter when checked { rooms = rooms.Where(r => r.IsAvailable) }
```

- ✓ This is correct.

- ✗ Make sure your form doesn't leave old query parameters.

- When you reload the page, unchecking the box should remove a parameter. Otherwise, the filter still applies.

Option B: Two-state "Available" filter (true/false)

If you want a checkbox that also hides unavailable rooms when unchecked

```
Copy code<available class="form-select"> <option value="">A
value="true" @(Context.Request.Query["available"]=="true"?"selec
</select>
```

- Controller stays the same.
- User can explicitly select "All" or "Available Only".
- Works reliably with MVC query binding.

③ Make sure your Index action is updated

csharp

```
Copy codeTask<IActionResult> Index( bool? available, int? mi
decimal? maxPrice) { var rooms = _context.Room.AsQueryable(); if
rooms.Where(r => r.IsAvailable); if (minCapacity.HasValue) rooms =
minCapacity.Value); if (maxCapacity.HasValue) rooms = rooms.Where(r
maxCapacity.Value); if (maxPrice.HasValue) rooms = rooms.Where(r
maxPrice.Value); return View(await rooms.ToListAsync()); }
```

- Only filters when checked.
- If unchecked or removed from URL, shows all rooms.

④ TL;DR Fix

1. Keep the checkbox but remember it only works when checked.
2. Or switch to a dropdown with "All / Available Only" — more predictable.
3. Ensure the form clears query params when box is unchecked.
4. Your controller logic is actually correct for a "Available Only" filter.

If you want, I can rewrite your Index form and controller together so that it properly every time, without weird edge cases.

Do you want me to do that?

```
<form method="get" class="mb-3">
<div class="row g-2">

    <!-- Available -->
    <div class="col-md-2">
        <div class="form-check mt-2">
            <input class="form-check-input"
                type="checkbox"
                name="available"
                value="true"
                id="available"
                @(Context.Request.Query["
            />
            <label class="form-check-label"
                Available Only
            </label>
        </div>
    </div>

    <!-- Minimum Capacity -->
    <div class="col-md-2">
        <input type="number"
            name="minCapacity"
            class="form-control"
            placeholder="Min Capacity"
            value="@Context.Reques
        </div>

    <!-- Maximum Capacity -->
    <div class="col-md-2">
        <input type="number"
            name="maxCapacity"
            class="form-control"
            placeholder="Max Capacity"
            value="@Context.Reques
        </div>

    <!-- Best Price -->
```

```

<div class="col-md-2">
  <input type="number"
    step="0.01"
    name="maxPrice"
    class="form-control"
    placeholder="Max Price"
    value="@Context.Request.Query["maxPrice"]" />
</div>

<!-- Submit -->
<div class="col-md-3">
  <button type="submit" class="btn btn-primary w-100" value="Apply Filters">Apply Filters</button>
</div>

</div>
</form> Update The file

```

Got it! Let's update your Index view form properly so the "Available Or capacity and price filters. I'll clean it up and make it robust.

Updated Index.cshtml Filter Form

razor

```

Copy coded="get" class="mb-3">
  <div class="row g-2">
    <!-- Available Only Capacity -->
    <div class="form-check mt-2">
      <input class="form-check-input" type="checkbox" checked="" id="available" name="available" value="true" />
      <label class="form-check-label" for="available"> Available Only Capacity -->
    </div>
    <div class="col-md-2">
      <input type="number" name="minCapacity" placeholder="Min Capacity" value="@Context.Request.Query["minCapacity"]" />
    </div>
    <div class="col-md-2">
      <input type="number" name="maxCapacity" placeholder="Max Capacity" value="@Context.Request.Query["maxCapacity"]" />
    </div>
    <div class="col-md-2">
      <input type="number" step="0.01" name="maxPrice" placeholder="Max Price" value="@Context.Request.Query["maxPrice"]" />
    </div>
    <div class="col-md-2">
      <button type="submit" class="btn btn-primary w-100" value="Apply Filters">Apply Filters</button>
    </div>
    <!-- Clear Filters Button -->
    <div class="col-md-2">
      <button type="button" class="btn btn-secondary w-100" value="Clear Filters">Clear Filters</button>
    </div>
  </div>

```

✓ What's changed

1. Added a Clear Filters button:

- Resets all filters.
- Prevents the checkbox from "sticking" when unchecked.

2. Adjusted Bootstrap columns to fit in one row (col-md-2 × 6 = 12)

3. Checkbox logic remains:

razor

```

Copy code
query["available"] == "true" ? "checked" : ""

```

- This ensures it's only checked if the query string has available set.
- If unchecked, the parameter is not sent, and the controller handles it.

4. Added proper placeholders and kept the values persistent.

⚡ Controller Reminder

Make sure your `Index` action is like this:

csharp

```

Copy code
Task Index(bool? available, int? minCapacity, decimal? maxPrice) {
  var rooms = _context.Room.AsQueryable();
  if (available != null) rooms = rooms.Where(r => r.IsAvailable);
  if (minCapacity.HasValue) rooms = rooms.Where(r => r.Capacity >= minCapacity.Value);
  if (maxCapacity.HasValue) rooms = rooms.Where(r => r.Capacity <= maxCapacity.Value);
  if (maxPrice.HasValue) rooms = rooms.Where(r => r.Price <= maxPrice.Value);
  return View(await rooms.ToListAsync());
}

```

This combination ensures:

- Available Only works correctly.
- Capacity and price filters also work.
- Users can clear all filters in one click.

If you want, I can also highlight unavailable rooms in red and highlight obvious which rooms are available — that's a nice UI touch.

Do you want me to do that?

2/6/26, 11:35 AM

ChatGPT

Is there

You've reached our limit of messages per hour. Please try again later.