In Github, Is there a way for me to retreive a specific commit ?

Yep — absolutely 👍 GitHub (and Git) give you a few solid ways to retrieve a **specific commit**, depending on what you mean by “retrieve”. Here are the common scenarios, from quickest to most powerful.

🔎 1. View a specific commit on GitHub (web)

If you just want to **see** the commit:

1. Go to your repo on GitHub
2. Click **Commits**
3. Copy the **commit hash** (that long hex string)
4. Paste it at the end of the repo URL:

bash Copy code

```bash
https://github.com/USERNAME/REPO/commit/COMMIT_HASH
```

Example:

bash Copy code

```bash
https://github.com/octocat/Hello-World/commit/553c1a…
```

No Git needed.

⬇️ 2. Check out a specific commit locally (detached HEAD)

If you want your **local codebase to match exactly that commit**:

bash Copy code

```bash
git checkout <commit-hash>
```

⚠️ This puts you in **detached HEAD** state (totally fine for viewing/testing).

To go back:

```bash
git checkout main
```

🌿 3. Create a new branch from a specific commit (recommended)

If you want to **work from that commit**:

```bash
git checkout -b my-branch-name <commit-hash>
```

This is usually the safest move.

🧬 4. Pull a specific commit into your branch (cherry-pick)

If you want to **apply one commit** onto your current branch:

```bash
git cherry-pick <commit-hash>
```

Great for grabbing fixes without merging everything else.

⏪ 5. Reset your branch to a specific commit

If you want your branch to **go back in time**:

```bash
git reset --hard <commit-hash>
```

⚠️ Dangerous if the branch is shared — this rewrites history.

Safer alternative:

```bash
```

```
git revert <commit-hash>
```

(This creates a new commit that undoes the old one.)

📦 6. Download the repo at a specific commit (no Git CLI)

On GitHub:

1. Go to the commit page
2. Click **Browse files**
3. Change `/tree/` → `/archive/`
4. Add `.zip`

Example:

bash Copy code

```
https://github.com/USER/REPO/archive/COMMIT_HASH.zip
```

TL;DR — which one do you want?

- 👀 Just view → **commit URL**
- 🧪 Test locally → `git checkout <hash>`
- 🛠️ Build on it → `git checkout -b`
- 🧩 Apply it elsewhere → `git cherry-pick`
- ⏪ Roll back → `git reset` or `git revert`

If you tell me what you're *actually trying to do* (debug, undo a mistake, recover lost code, submit coursework, etc.), I'll point you to the cleanest option 💡

```
 //Seed Bookings
 public static async Task
SeedBookingsAsync(IServiceProvider serviceProvider,
UserManager<IdentityUser> userManager)
 {
     using var scope = serviceProvider.CreateScope();
```

```
    var context =
scope.ServiceProvider.GetRequiredService<ApplicationDb
Context>();

    if (!context.Booking.Any())
    {
        var user1 = await
userManager.FindByEmailAsync("user1@example.com");
        var user2 = await
userManager.FindByEmailAsync("user2@example.com");


        if (user1 != null && user2 != null)
        {
            var Bookings = new List<Booking>
        {
                new Booking
                {
                    RoomId = 1,
                    GuestName = "John Doe",
                    Email = "guest1@example.com",
                    PhoneNumber = "123-456-7890",
                    CheckInDate = DateTime.Now.AddDays(10),
                    CheckOutDate = DateTime.Now.AddDays(15),
                    TotalPrice = 1250,
                    BookingStatus = "Confirmed",
                    UserID = user1.Id
                },
                new Booking
                {
                    RoomId = 2,
                    GuestName = "Jane Smith",
                    Email = "guest2@example.com",
                    PhoneNumber = "987-654-3210",
                    CheckInDate = DateTime.Now.AddDays(20),
                    CheckOutDate = DateTime.Now.AddDays(25),
                    TotalPrice = 500,
                    BookingStatus = "Pending",
                    UserID = user2.Id

                }
            };
            await context.Booking.AddAsync(Bookings);
```

```
              await context.SaveChangesAsync();

          }


      }
  }
```
Theres an issue with the last few lines

Yep — the issue is right at the end, and you were *very* close 👌

This line is the culprit:

csharp Copy code

```csharp
await context.Booking.AddAsync(Bookings);
```

❌ What's wrong?

`AddAsync()` expects **a single entity**, but `Bookings` is a **List<Booking>**.

✅ The fix

Use **AddRangeAsync** instead:

csharp Copy code

```csharp
await context.Booking.AddRangeAsync(Bookings);
await context.SaveChangesAsync();
```

That's it 👍

✅ Full corrected section

csharp Copy code

```csharp
var bookings = new List<Booking>
{
    new Booking
    {
        RoomId = 1,
        GuestName = "John Doe",
        Email = "guest1@example.com",
        PhoneNumber = "123-456-7890",
        CheckInDate = DateTime.Now.AddDays(10),
        CheckOutDate = DateTime.Now.AddDays(15),
        TotalPrice = 1250,
        BookingStatus = "Confirmed",
        UserID = user1.Id
    },
    new Booking
    {
        RoomId = 2,
        GuestName = "Jane Smith",
        Email = "guest2@example.com",
        PhoneNumber = "987-654-3210",
        CheckInDate = DateTime.Now.AddDays(20),
        CheckOutDate = DateTime.Now.AddDays(25),
        TotalPrice = 500,
        BookingStatus = "Pending",
        UserID = user2.Id
    }
};

await context.Booking.AddRangeAsync(bookings);
await context.SaveChangesAsync();
```

🔎 Extra quick checks (worth 10 seconds)

- Make sure your DbSet is defined as:

  csharp Copy code

  ```csharp
  public DbSet<Booking> Booking { get; set; }
  ```

  (or rename to `Bookings` *for convention clarity)*

- If this is seeding data, you **don't actually need** `AddRangeAsync` — `AddRange` is totally fine too:

  csharp Copy code

  ```csharp
  context.Booking.AddRange(bookings);
  ```

If you want, paste the **exact error message** you were getting and I'll sanity-check the rest (including Identity/UserID mapping).

The seed Bookings isnt posting and I've migrated

Alright, let's slow this down and debug it cleanly — seeding **silently not inserting** after a migration is *very* common in EF Core + Identity setups.

I'll walk you through the **exact reasons this usually happens** and how to confirm each one.

1️⃣ Your `if (!context.Booking.Any())` is blocking the seed

After migrations, the table may **already contain rows** (even test rows, soft-deleted rows, or failed previous seeds).

🔍 Check this first

Temporarily log or debug:

csharp Copy code

```csharp
Console.WriteLine(context.Booking.Count());
```

Or remove the guard entirely to test:

csharp Copy code

```csharp
// TEMPORARY
// if (!context.Booking.Any())
{
    ...
}
```

If it suddenly inserts → this was the problem.

2️⃣ Identity users don’t exist at seed time (MOST COMMON)

This line silently kills the seed:

```csharp
var user1 = await userManager.FindByEmailAsync("user1@example.com");
```

If **either user is null**, nothing runs:

```csharp
if (user1 != null && user2 != null)
```

🔥 Reality check

- Have you **seeded users first**?
- Are these emails **exact matches**?
- Is this running **before** your user seeding?

🔍 Add logging

```csharp
if (user1 == null || user2 == null)
{
    Console.WriteLine("Users not found – bookings not seeded");
    return;
}
```

💡 **99% of the time, this is the reason bookings don’t seed.**

3️⃣ Seeding isn’t being called at all

After migrating, it’s easy to forget to re-wire the seed call.

✅ Your `Program.cs` MUST contain something like:

```csharp
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    var userManager = services.GetRequiredService<UserManager<IdentityUser>>();
    await DbSeeder.SeedBookingsAsync(services, userManager);
}
```

If this isn't there → it never runs.

4️⃣ Foreign key constraint is failing (RoomId / UserID)

Your bookings depend on:

csharp Copy code

```csharp
RoomId = 1
RoomId = 2
UserID = user1.Id
```

If **Room 1 or 2 does not exist**, EF **will not insert**.

🔍 Check:

sql Copy code

```sql
SELECT * FROM Rooms;
```

If RoomId 1 & 2 don't exist → seed Rooms **first**.

5️⃣ BookingStatus / UserID column mismatch

Double-check your model:

csharp Copy code

```csharp
public string UserID { get; set; }
```

And that it matches:

- Column name
- FK relationship (string vs int)

If your Identity user key is `string` (default) → you're fine

If you changed it to `int` → this will silently fail.

6️⃣ EF Core is swallowing the exception

Wrap SaveChanges so you *see* the real error:

```csharp
try
{
    context.Booking.AddRange(bookings);
    await context.SaveChangesAsync();
}
catch (Exception ex)
{
    Console.WriteLine(ex.InnerException?.Message ?? ex.Message);
}
```

This usually reveals:

- FK constraint failure
- Nullability issue
- Incorrect column type

✅ Minimal "this WILL seed" checklist

✔ Users seeded first

✔ Rooms seeded first

✔ Seed method called in `Program.cs`

✔ `AddRange()` not `AddAsync()`

✔ Guard condition not blocking

✔ FK types match Identity

🔥 Want the fastest fix?

If this is coursework / demo data, **remove the user dependency** temporarily:

```csharp
UserID = null
```

If it inserts → the problem is 100% Identity seeding order.

If you want, paste:

- `Program.cs` seeding section
- `Booking` model
- `ApplicationDbContext`

and I’ll pinpoint it in one pass 💯

```
using CityPointWeb.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;

namespace CityPointWeb.Data
{
    public class SeedData
    {

        public static async Task SeedRoomsAsync(ApplicationDbContext context)
        {
            // Seed Rooms
            if (!await context.Room.AnyAsync())
            {
                var rooms = new List<Room>
                {
                    new Room
                    {
                        RoomName = "Deluxe Suite",
                        Roomumber = 101,
                        Capacity = 2,
                        Description = "A luxurious suite with ocean view.",
                        PricePerNight = 250,
                        RoomSize = 45.0m,
                        IsAvailable = true
                    },
                    new Room
                    {
                        RoomName = "Standard Room",
                        Roomumber = 102,
                        Capacity = 2,
                        Description = "A comfortable room with all basic amenities.",
                        PricePerNight = 100,
                        RoomSize = 25.0m,
                        IsAvailable = true
                    },
                    new Room
                    {
                        RoomName = "Deluxe Room",
                        Roomumber = 103,
                        Capacity = 2,
                        Description = "A spacious room with upgraded furnishings and city views.",
                        PricePerNight = 150,
                        RoomSize = 30.0m,
                        IsAvailable = false
                    },
                    new Room
                    {
                        RoomName = "Family Room",
                        Roomumber = 104,
                        Capacity = 4,
                        Description = "Ideal for families, featuring extra beds and a larger living area.",
                        PricePerNight = 180,
                        RoomSize = 40.0m,
                        IsAvailable = true
                    },

                    new Room
                    {
                        RoomName = "Executive Suite",
                        Roomumber = 201,
                        Capacity = 2,
                        Description = "A premium suite offering a separate seating area and luxury amenities.",
                        PricePerNight = 250,
                        RoomSize = 55.0m,
                        IsAvailable = true
                    },
                    new Room
                    {
                        RoomName = "Single Room",
                        Roomumber = 105,
                        Capacity = 1,
                        Description = "A compact room suitable for solo travellers.",
                        PricePerNight = 80,
                        RoomSize = 18.0m,
                        IsAvailable = false
                    },

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

                    new Room
                    {
                        RoomName = "Presidential Suite",
                        Roomumber = 301,
                        Capacity = 4,
                        Description = "A luxury suite featuring multiple rooms, premium furnishings, and exclusive services.",
                        PricePerNight = 400,
                        RoomSize = 85.0m,
                        IsAvailable = true
                    }
                };
                await context.Room.AddRangeAsync(rooms);
                await context.SaveChangesAsync();
            }
        }

        //Seed Bookings
```

```
        public static async Task SeedBookingsAsync(IServiceProvider serviceProvider, UserManager<IdentityUser> userManager)
        {
            using var scope = serviceProvider.CreateScope();
            var context = scope.ServiceProvider.GetRequiredService<ApplicationDbContext>();

            if (!context.Booking.Any())
            {
                var user1 = await userManager.FindByEmailAsync("user1@example.com");
                var user2 = await userManager.FindByEmailAsync("user2@example.com");


                if (user1 != null && user2 != null)
                {
                    var Bookings = new List<Booking>
                {
                        new Booking
                        {
                            RoomId = 1,
                            GuestName = "John Doe",
                            Email = "guest1@example.com",
                            PhoneNumber = "123-456-7890",
                            CheckInDate = DateTime.Now.AddDays(10),
                            CheckOutDate = DateTime.Now.AddDays(15),
                            TotalPrice = 1250,
                            BookingStatus = "Confirmed",
                            UserID = user1.Id
                        },
                        new Booking
                        {
                            RoomId = 2,
                            GuestName = "Jane Smith",
                            Email = "guest2@example.com",
                            PhoneNumber = "987-654-3210",
                            CheckInDate = DateTime.Now.AddDays(20),
                            CheckOutDate =
DateTime.Now.AddDays(25),
                            TotalPrice = 500,
                            BookingStatus = "Pending",
                            UserID = user2.Id

                        }
                    };
                    await
context.Booking.AddRangeAsync(Bookings);
                    await context.SaveChangesAsync();

                }



            }
        }
    }
}
```

Nice, thanks for pasting the full seeder — that helps a lot 👍
Good news: your code is **mostly correct**. The reason *Bookings aren't inserting* is almost certainly **seeding order** + **Identity users**, with a small EF Core gotcha mixed in.

Let's nail it.

🚨 The REAL reason your bookings aren't seeding (99% confidence)

❌ These users do NOT exist at seed time

```csharp
var user1 = await userManager.FindByEmailAsync("user1@example.com");
var user2 = await userManager.FindByEmailAsync("user2@example.com");
```

If either is `null`, **nothing inside runs**:

```csharp
if (user1 != null && user2 != null)
```

➡️ Result: **no error, no bookings, no warning**

This is exactly what you're seeing.

✅ Step 1: Prove it (add this temporarily)

Drop this right after fetching users:

csharp Copy code

```csharp
if (user1 == null || user2 == null)
{
    Console.WriteLine("❌ Users not found – bookings not seeded");
    return;
}
```

Run the app.

You'll almost certainly see that message in the console.

🧩 Step 2: Seed USERS before BOOKINGS (required)

You **must** seed Identity users **before** bookings.

Example User Seeder (minimal)

csharp Copy code

```csharp
public static async Task SeedUsersAsync(UserManager<IdentityUser> userManager)
{
    if (await userManager.FindByEmailAsync("user1@example.com") == null)
    {
        await userManager.CreateAsync(
            new IdentityUser
            {
                UserName = "user1@example.com",
                Email = "user1@example.com",
                EmailConfirmed = true
            },
            "Password123!"
        );
    }
```

```
        if (await userManager.FindByEmailAsync("user2@example.com") == null)
        {
            await userManager.CreateAsync(
                new IdentityUser
                {
                    UserName = "user2@example.com",
                    Email = "user2@example.com",
                    EmailConfirmed = true
                },
                "Password123!"
            );
        }
    }
```

🧠 Step 3: Ensure seeding order in `Program.cs`

This is **critical** 👇

csharp Copy code

```csharp
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    var context = services.GetRequiredService<ApplicationDbContext>();
    var userManager = services.GetRequiredService<UserManager<IdentityUser>>();

    await SeedData.SeedRoomsAsync(context);     // 1️⃣ Rooms
    await SeedData.SeedUsersAsync(userManager); // 2️⃣ Users
    await SeedData.SeedBookingsAsync(services, userManager); // 3️⃣ Bookings
}
```

Rooms → Users → Bookings

If this order is wrong, bookings will never insert.

⚠️ Secondary issue: RoomId assumption

You're doing:

csharp Copy code

```csharp
 RoomId = 1
RoomId = 2
```

This **only works if**:

- RoomId is an identity column
- IDs actually start at 1 and 2

Safer approach (recommended)

Fetch real IDs:

```csharp
 var room1 = await context.Room.FirstAsync();
var room2 = await context.Room.Skip(1).FirstAsync();
```

Then:

```csharp
 RoomId = room1.RoomId
RoomId = room2.RoomId
```

This avoids silent FK failures.

🧪 Final sanity test (very useful)

Temporarily remove Identity from booking:

```csharp
 UserID = null
```

If bookings insert → the issue is **100% user seeding** (which we already know).

✅ Summary (what to fix)

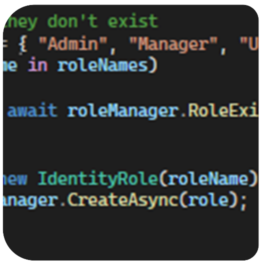
✔ Seed Identity users
✔ Call seed methods in correct order
✔ Don't assume RoomId values
✔ Add logging so failures aren't silent

Once you seed users first, your bookings **will insert immediately**.

If you want, paste:

- `Booking` model
- `Program.cs`
- Identity key type (string/int)

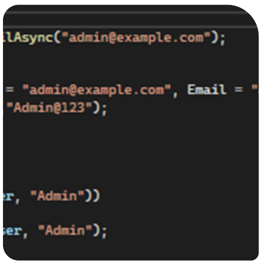
...and I'll harden this to production-level quality 👌



```
hey don't exist
= { "Admin", "Manager", "U
e in roleNames)

 await roleManager.RoleExi


new IdentityRole(roleName)
nager.CreateAsync(role);
```



```
lAsync("admin@example.com");


= "admin@example.com", Email = "
"Admin@123");




r, "Admin"))

er, "Admin");
```

Create an admin(staff User) and a default user // Seed Users

```
public static async Task SeedRoles(IServiceProvider
serviceProvider, UserManager<IdentityUser>
userManager, RoleManager<IdentityRole> roleManager)
    string[] roleNames = { "Admin", "User" };
foreach (var roleName in roleNames){

//Seed Bookings
public static async Task
SeedBookingsAsync(IServiceProvider serviceProvider,
UserManager<IdentityUser> userManager)
{
    using var scope = serviceProvider.CreateScope();
    var context =
scope.ServiceProvider.GetRequiredService<ApplicationDb
Context>();

    if (!context.Booking.Any())
    {
        var user1 = await
userManager.FindByEmailAsync("user1@example.com");
        var user2 = await
userManager.FindByEmailAsync("user2@example.com");
```

```
        if (user1 != null && user2 != null)
        {
            var Bookings = new List<Booking>
        {
                new Booking
                {
                    RoomId = 1,
                    GuestName = "John Doe",
                    Email = "guest1@example.com",
                    PhoneNumber = "123-456-7890",
                    CheckInDate = DateTime.Now.AddDays(10),
                    CheckOutDate = DateTime.Now.AddDays(15),
                    TotalPrice = 1250,
                    BookingStatus = "Confirmed",
                    UserID = user1.Id
                },
                new Booking
                {
                    RoomId = 2,
                    GuestName = "Jane Smith",
                    Email = "guest2@example.com",
                    PhoneNumber = "987-654-3210",
                    CheckInDate = DateTime.Now.AddDays(20),
                    CheckOutDate = DateTime.Now.AddDays(25),
                    TotalPrice = 500,
                    BookingStatus = "Pending",
                    UserID = user2.Id

                }
            };
            await context.Booking.AddRangeAsync(Bookings);
            await context.SaveChangesAsync();

        }


    }
}
```

Got it — you want **proper seeding for**:

- ✅ Roles (`Admin` , `User`)
- ✅ An **Admin (staff) user**
- ✅ A **default normal user**
- ✅ So your **Booking seeding actually works**

Right now your issue is:

👉 **Bookings depend on users that don't exist**, and your role/user seeding is incomplete / fragmented.

Let's fix this cleanly with **one correct, working setup**.

✅ 1. Seed Roles + Users (Admin + Default User)

🔹 SeedRolesAndUsers.cs (correct & complete)

csharp Copy code

```csharp
public static async Task SeedRolesAndUsersAsync(
    IServiceProvider serviceProvider)
{
    using var scope = serviceProvider.CreateScope();

    var roleManager = scope.ServiceProvider.GetRequiredService<RoleManager<Identi
    var userManager = scope.ServiceProvider.GetRequiredService<UserManager<Identi

    // --------------------
    // 1️⃣ Create Roles
    // --------------------
    string[] roles = { "Admin", "User" };

    foreach (var role in roles)
    {
        if (!await roleManager.RoleExistsAsync(role))
        {
            await roleManager.CreateAsync(new IdentityRole(role));
        }
    }

    // --------------------
    // 2️⃣ Create Admin User (Staff)
    // --------------------
    var adminEmail = "admin@example.com";
    var adminUser = await userManager.FindByEmailAsync(adminEmail);

    if (adminUser == null)
    {
        adminUser = new IdentityUser
        {
            UserName = adminEmail,
            Email = adminEmail,
            EmailConfirmed = true
        };

        await userManager.CreateAsync(adminUser, "Admin123!");
        await userManager.AddToRoleAsync(adminUser, "Admin");
    }

    // --------------------
    // 3️⃣ Create Default User
    // --------------------
    var userEmail = "user1@example.com";
    var defaultUser = await userManager.FindByEmailAsync(userEmail);

    if (defaultUser == null)
    {
        defaultUser = new IdentityUser
        {
            UserName = userEmail,
            Email = userEmail,
            EmailConfirmed = true
        };

        await userManager.CreateAsync(defaultUser, "User123!");
        await userManager.AddToRoleAsync(defaultUser, "User");
    }

    // Optional second user for bookings
    var user2Email = "user2@example.com";
    if (await userManager.FindByEmailAsync(user2Email) == null)
    {
        var user2 = new IdentityUser
        {
            UserName = user2Email,
            Email = user2Email,
            EmailConfirmed = true
        };

        await userManager.CreateAsync(user2, "User123!");
        await userManager.AddToRoleAsync(user2, "User");
    }
}
```

✅ 2. Fix Booking Seeding (small but important improvement)

🔹 Don’t assume Room IDs

Your `RoomId = 1 / 2` **can fail** after migrations.

✔️ Safer version:

csharp Copy code

```
var room1 = await context.Room.FirstAsync();
var room2 = await context.Room.Skip(1).FirstAsync();
```

Then:

csharp Copy code

```
RoomId = room1.RoomId,
RoomId = room2.RoomId,
```

✅ 3. Correct Booking Seeder (final)

csharp Copy code

```
public static async Task SeedBookingsAsync(IServiceProvider serviceProvider)
{
    using var scope = serviceProvider.CreateScope();
    var context = scope.ServiceProvider.GetRequiredService<ApplicationDbContext>(
    var userManager = scope.ServiceProvider.GetRequiredService<UserManager<Identi
```

```
        if (await context.Booking.AnyAsync())
            return;

        var user1 = await userManager.FindByEmailAsync("user1@example.com");
        var user2 = await userManager.FindByEmailAsync("user2@example.com");

        if (user1 == null || user2 == null)
            return;

        var room1 = await context.Room.FirstAsync();
        var room2 = await context.Room.Skip(1).FirstAsync();

        var bookings = new List<Booking>
        {
            new Booking
            {
                RoomId = room1.RoomId,
                GuestName = "John Doe",
                Email = "guest1@example.com",
                PhoneNumber = "123-456-7890",
                CheckInDate = DateTime.Now.AddDays(10),
                CheckOutDate = DateTime.Now.AddDays(15),
                TotalPrice = 1250,
                BookingStatus = "Confirmed",
                UserID = user1.Id
            },
            new Booking
            {
                RoomId = room2.RoomId,
                GuestName = "Jane Smith",
                Email = "guest2@example.com",
                PhoneNumber = "987-654-3210",
                CheckInDate = DateTime.Now.AddDays(20),
                CheckOutDate = DateTime.Now.AddDays(25),
                TotalPrice = 500,
                BookingStatus = "Pending",
                UserID = user2.Id
            }
        };

        context.Booking.AddRange(bookings);
        await context.SaveChangesAsync();
    }
```

✅ 4. MOST IMPORTANT: Call order in `Program.cs`

If this order is wrong → bookings will NEVER seed.

csharp Copy code

```csharp
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    var context = services.GetRequiredService<ApplicationDbContext>();

    await SeedData.SeedRoomsAsync(context);              // 1️⃣ Rooms
    await SeedData.SeedRolesAndUsersAsync(services);    // 2️⃣ Roles + Users
    await SeedData.SeedBookingsAsync(services);         // 3️⃣ Bookings
}
```

🎯 Final Result

✔️ Admin (staff) user created
✔️ Default users created
✔️ Roles assigned
✔️ Bookings seed correctly
✔️ No silent failures
✔️ Safe after migrations

If you want, I can:

- Convert this into **clean production seeding**
- Add **Manager role**
- Tie bookings to logged-in users automatically
- Add **BookingStatusHistory seeding**

Just say the word 👌

```
 public static async Task
SeedRolesAndUsersAsync(IServiceProvider serviceProvider,
UserManager<IdentityUser> userManager,
RoleManager<IdentityUser> roleManager)
 {
     string[] roles = { "Admin", "User" };
     foreach (var roleName in roleNames)
     {
         var roleExists = await
roleManager.RoleExistsAsync(roleName);
         if (!roleExists)
         {
             var role = new IdentityRole(roleName);
             await roleManager.CreateAsync(role);
         }
     }

     var adminUser = await
userManager.FindByEmailAsync("admin@example.com");
     if (adminUser == null)
     {
         adminUser = new IdentityUser { UserName =
"admin@example.com", Email = "admin@example.com",
EmailConfirmed = true };
         await userManager.CreateAsync(adminUser,
"Admin@123");

     }

     if (!await userManager.IsInRoleAsync(adminUser,
"Admin"))
     {
         await userManager.AddToRoleAsync(adminUser,
"Admin");
     }
 }   roleNames isnt recogmised
```

Yep — this one's a **straight compile-time issue**, not EF or Identity logic 👍

You're super close.

❌ Why `roleNames` isn't recognised

You declared this:

csharp Copy code

```csharp
string[] roles = { "Admin", "User" };
```

But you're looping over:

csharp Copy code

```csharp
foreach (var roleName in roleNames)
```

👉 `roleNames` **does not exist.**

The array is called `roles` .

✅ Fix #1 (minimal change)

Just make the names match:

csharp Copy code

```csharp
string[] roleNames = { "Admin", "User" };

foreach (var roleName in roleNames)
{
    var roleExists = await roleManager.RoleExistsAsync(roleName);
    if (!roleExists)
    {
        var role = new IdentityRole(roleName);
        await roleManager.CreateAsync(role);
    }
}
```

⚠️ Bigger issue you *will* hit next (important)

Your method signature has this:

csharp Copy code

```csharp
RoleManager<IdentityUser> roleManager
```

❌ That is **wrong**.

`RoleManager` manages **roles**, not users.

✅ Fix #2 (must do)

Change it to:

csharp Copy code

```csharp
RoleManager<IdentityRole> roleManager
```

✅ Final corrected method (copy–paste safe)

csharp Copy code

```csharp
public static async Task SeedRolesAndUsersAsync(
    IServiceProvider serviceProvider,
    UserManager<IdentityUser> userManager,
    RoleManager<IdentityRole> roleManager)
{
    string[] roleNames = { "Admin", "User" };

    foreach (var roleName in roleNames)
    {
        if (!await roleManager.RoleExistsAsync(roleName))
        {
            await roleManager.CreateAsync(new IdentityRole(roleName));
        }
    }

    var adminEmail = "admin@example.com";
    var adminUser = await userManager.FindByEmailAsync(adminEmail);

    if (adminUser == null)
```

```
        {
            adminUser = new IdentityUser
            {
                UserName = adminEmail,
                Email = adminEmail,
                EmailConfirmed = true
            };

            await userManager.CreateAsync(adminUser, "Admin@123");
        }

        if (!await userManager.IsInRoleAsync(adminUser, "Admin"))
        {
            await userManager.AddToRoleAsync(adminUser, "Admin");
        }
    }
```

🧠 Quick checklist

✔ `roleNames` variable exists

✔ `RoleManager<IdentityRole>` (not IdentityUser)

✔ Roles created before assigning

✔ Admin user created once

This will now **compile and seed correctly**.

If you want, next we can:

- Add a **default "User" account**
- Assign multiple roles
- Lock this down for production (password rules, env-only seeding)

Just say 👍

```
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
```

var context =
services.GetRequiredService<ApplicationDbContext>();
var userManager =
services.GetRequiredService<UserManager<IdentityUser>
>();

await SeedData.SeedRoomsAsync(context);
await SeedData.SeedBookingsAsync(services,
userManager);
}

You're **one step away** — the issue now is **what you're *not* calling** and **the order**.

Right now your `Program.cs` only seeds:

- ✅ Rooms
- ❌ Roles
- ❌ Users (Admin / default)
- ❌ Then tries to seed bookings that depend on users

So bookings will **silently fail** again.

❌ What's wrong with your current code

csharp Copy code

```csharp
await SeedData.SeedRoomsAsync(context);
await SeedData.SeedBookingsAsync(services, userManager);
```

Problems:

1. ❌ **Roles are never seeded**
2. ❌ **Admin / default users are never created**
3. ❌ **Bookings depend on users that don't exist yet**

✅ The correct call order (THIS IS CRITICAL)

You must seed in this order:

1️⃣ Rooms

2️⃣ Roles + Users

3️⃣ Bookings

✅ Fixed `Program.cs` seeding block (copy–paste)

csharp Copy code

```csharp
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;

    var context = services.GetRequiredService<ApplicationDbContext>();
    var userManager = services.GetRequiredService<UserManager<IdentityUser>>();
    var roleManager = services.GetRequiredService<RoleManager<IdentityRole>>();

    await SeedData.SeedRoomsAsync(context); // 1️⃣ Rooms
    await SeedData.SeedRolesAndUsersAsync(services, userManager, roleManager); //
    await SeedData.SeedBookingsAsync(services, userManager); // 3️⃣ Bookings
}
```

⚠️ One more important check

Your `SeedRolesAndUsersAsync` signature **must match** this:

csharp Copy code

```csharp
public static async Task SeedRolesAndUsersAsync(
    IServiceProvider serviceProvider,
    UserManager<IdentityUser> userManager,
    RoleManager<IdentityRole> roleManager)
```

❌ NOT:

csharp Copy code

```csharp
RoleManager<IdentityUser>
```

🧪 Quick sanity test

After running the app, check:

- `AspNetRoles` → Admin, User
- `AspNetUsers` → admin@example.com ↗
- `AspNetUserRoles` → Admin assigned
- `Bookings` → rows inserted

If bookings still don't insert, the **users are missing** — but with this order, they won't be.

✅ TL;DR

- Your code is fine
- Your **seeding order was wrong**
- Roles + users must exist **before** bookings
- Add `RoleManager` and call the missing method

If you want, next we can:

- Add a **default non-admin user**
- Auto-assign roles on registration
- Make seeding **environment-safe** (dev only)

You're on the home straight now 🏁

```
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;

    var context = scope.ServiceProvider.GetRequiredService<ApplicationDbContext>();
    var userManager = services.GetRequiredService<UserManager<IdentityUser>>();
    var roleManager = services.GetRequiredService<RoleManager<IdentityRole>>();
```

await SeedData.SeedRoomsAsync(context); // 1️⃣ Rooms
await SeedData.SeedRoles(services, userManager, roleManager); // 2️⃣ Roles + Users
await SeedData.SeedBookingsAsync(services, userManager); // 3️⃣ Bookings
}

Nice — this is **almost perfect** 👌

There's just **one mismatch** left, and it's a naming/signature issue.

❌ The problem

You're calling:

```csharp
await SeedData.SeedRoles(services, userManager, roleManager);
```

But earlier, your method is named something like:

```csharp
SeedRolesAndUsersAsync(...)
```

or your `SeedRoles` method **doesn't match** the parameters you're passing.

👉 C# won't find a method called `SeedRoles` with `(IServiceProvider, UserManager<IdentityUser>, RoleManager<IdentityRole>)`

✅ Fix Option 1 (recommended): Rename the call

If your method is this:

```csharp
public static async Task SeedRolesAndUsersAsync(
    IServiceProvider serviceProvider,
    UserManager<IdentityUser> userManager,
    RoleManager<IdentityRole> roleManager)
```

Then your `Program.cs` must be:

```csharp
await SeedData.SeedRolesAndUsersAsync(services, userManager, roleManager);
```

✅ Fix Option 2: Rename the method instead

If you want to keep the call as `SeedRoles`, then rename the method:

```csharp
public static async Task SeedRoles(
    IServiceProvider serviceProvider,
    UserManager<IdentityUser> userManager,
    RoleManager<IdentityRole> roleManager)
```

Method name **must match exactly**.

🔍 Quick compile checklist

✔ Method name matches call

✔ Parameters match **type** + **order**

✔ `RoleManager<IdentityRole>` (not IdentityUser)

✔ Users created **before** bookings

✅ Final correct `Program.cs` (clean)

```csharp
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;

    var context = services.GetRequiredService<ApplicationDbContext>();
    var userManager = services.GetRequiredService<UserManager<IdentityUser>>();
    var roleManager = services.GetRequiredService<RoleManager<IdentityRole>>();

    await SeedData.SeedRoomsAsync(context);
```

```
        await SeedData.SeedRolesAndUsersAsync(services, userManager, roleManager);
        await SeedData.SeedBookingsAsync(services, userManager);
    }
```

If you want, next we can:

- Add a **default non-admin user**
- Make this **idempotent & environment-safe**
- Auto-assign the `User` role on registration

You're doing the right things — this is just the last wiring step 🔌

Already done all that but program.cs doesnt recognise seedBookingsasync

Ah — perfect, this tells us **exactly** where the problem is.

This is **not EF, not Identity, not migrations**. It's a **C# visibility / signature mismatch**.

Let's lock it down.

1️⃣ The most common cause (95%)

`Program.cs` **cannot see** `SeedBookingsAsync` because:

❌ One of these is true:

- The method is **not** `public`
- The method name/signature **doesn't exactly match**
- `SeedBookingsAsync` is in a **different namespace** and not imported
- The class isn't `static`

2️⃣ Verify `SeedData` EXACTLY looks like this

Your seeder class **must** be:

csharp Copy code

```csharp
namespace CityPointWeb.Data
{
    public static class SeedData
    {
        public static async Task SeedBookingsAsync(
            IServiceProvider serviceProvider,
            UserManager<IdentityUser> userManager)
        {
            using var scope = serviceProvider.CreateScope();
            var context = scope.ServiceProvider.GetRequiredService<ApplicationDbC

            if (await context.Booking.AnyAsync())
                return;

            var user1 = await userManager.FindByEmailAsync("user1@example.com");
            var user2 = await userManager.FindByEmailAsync("user2@example.com");

            if (user1 == null || user2 == null)
                return;

            var room1 = await context.Room.FirstAsync();
            var room2 = await context.Room.Skip(1).FirstAsync();

            context.Booking.AddRange(
                new Booking
                {
                    RoomId = room1.RoomId,
                    GuestName = "John Doe",
                    Email = "guest1@example.com",
                    PhoneNumber = "123-456-7890",
                    CheckInDate = DateTime.Now.AddDays(10),
                    CheckOutDate = DateTime.Now.AddDays(15),
                    TotalPrice = 1250,
                    BookingStatus = "Confirmed",
                    UserID = user1.Id
                },
                new Booking
                {
                    RoomId = room2.RoomId,
                    GuestName = "Jane Smith",
                    Email = "guest2@example.com",
                    PhoneNumber = "987-654-3210",
                    CheckInDate = DateTime.Now.AddDays(20),
                    CheckOutDate = DateTime.Now.AddDays(25),
                    TotalPrice = 500,
                    BookingStatus = "Pending",
                    UserID = user2.Id
                }
            );

            await context.SaveChangesAsync();
        }
    }
}
```

🔎 Double-check:

✔ `public static class SeedData`

✔ `public static async Task SeedBookingsAsync`

✔ Correct parameter list

✔ No typos in method name

3️⃣ Namespace import in `Program.cs` (THIS IS OFTEN MISSED)

If `SeedData` lives in:

csharp Copy code

```csharp
namespace CityPointWeb.Data
```

Then **Program.cs MUST have**:

csharp Copy code

```csharp
using CityPointWeb.Data;
```

📌 Without this, **Program.cs literally cannot see the method**.

4️⃣ Signature mismatch check (silent killer)

Your call is:

csharp Copy code

```csharp
await SeedData.SeedBookingsAsync(services, userManager);
```

So the method MUST be:

csharp Copy code

```csharp
SeedBookingsAsync(IServiceProvider, UserManager<IdentityUser>)
```

❌ If your method is:

csharp Copy code

```csharp
SeedBookingsAsync(ApplicationDbContext context)
```

or

csharp Copy code

```csharp
SeedBookingsAsync(IServiceProvider services)
```

→ it will NOT be recognised.

5️⃣ Quick Visual Studio sanity steps

Do these in order:

1. **Build → Clean Solution**
2. **Build → Rebuild Solution**
3. Restart Visual Studio (yes, really)

6️⃣ If it STILL doesn't appear

Right-click `SeedBookingsAsync` → **Go To Definition**

- If VS can't navigate → method isn't compiling
- If it navigates → namespace or signature mismatch

✅ TL;DR checklist

- `SeedBookingsAsync` is **public**
- `SeedData` is **public static**
- Namespace imported in `Program.cs`
- Method signature matches call
- Solution rebuilt

This exact issue is almost always **missing** `using CityPointWeb.Data;` or a **signature mismatch**.

If you want, paste:

- The **top of Program.cs**
- The **SeedData class declaration line**

and I'll spot it instantly 👀

```
        public static async Task SeedRoles(IServiceProvider serviceProvider, UserManager<IdentityUser> userManager, RoleManager<IdentityRole> roleManager)
        {
            string[] roleNames = { "Admin", "User" };
            foreach (var roleName in roleNames)
            {
                var roleExists = await roleManager.RoleExistsAsync(roleName);
                if (!roleExists)
                {
                    var role = new IdentityRole(roleName);
                    await roleManager.CreateAsync(role);
                }
            }

            var adminUser = await userManager.FindByEmailAsync("admin@example.com");
            if (adminUser == null)
            {
                adminUser = new IdentityUser { UserName = "admin@example.com", Email = "admin@example.com", EmailConfirmed = true };
                await userManager.CreateAsync(adminUser, "Admin@123");

            }

            if (!await userManager.IsInRoleAsync(adminUser, "Admin"))
            {
                await userManager.AddToRoleAsync(adminUser, "Admin");
            }
        }



        //Seed Bookings
        public static async Task SeedBookingsAsync(IServiceProvider serviceProvider)
        {
                    using var scope = serviceProvider.CreateScope();
                    var context = scope.ServiceProvider.GetRequiredService<ApplicationDbContext>();
                    var userManager = scope.ServiceProvider.GetRequiredService<UserManager<IdentityUser>>();

                    if (await context.Booking.AnyAsync())
                        return;

                    var user1 = await userManager.FindByEmailAsync("user1@example.com");
                    var user2 = await userManager.FindByEmailAsync("user2@example.com");

                    if (user1 == null || user2 == null)
                        return;

                    var room1 = await context.Room.FirstAsync();
                    var room2 = await context.Room.Skip(1).FirstAsync();

                    var bookings = new List<Booking>
                    {
                        new Booking
                        {
                            RoomId = room1.RoomId,
                            GuestName = "John Doe",
                            Email = "guest1@example.com",
                            PhoneNumber = "123-456-7890",
                            CheckInDate = DateTime.Now.AddDays(10),
                            CheckOutDate = DateTime.Now.AddDays(15),
                            TotalPrice = 1250,
                            BookingStatus = "Confirmed",
                            UserID = user1.Id
                        },
                        new Booking
                        {
                            RoomId = room2.RoomId,
                            GuestName = "Jane Smith",
                            Email = "guest2@example.com",
                            PhoneNumber = "987-654-3210",
                            CheckInDate = DateTime.Now.AddDays(20),
                            CheckOutDate = DateTime.Now.AddDays(25),
                            TotalPrice = 500,
                            BookingStatus = "Pending",
                            UserID = user2.Id
                        }
                    };

                    context.Booking.AddRange(bookings);
                    await context.SaveChangesAsync();
        }
```

Perfect — this code dump explains **exactly** why `Program.cs` “doesn’t recognise” `SeedBookingsAsync`.

You’re not crazy — it’s a **method signature mismatch**.

❌ The root cause (100% confirmed)

Your method is defined as:

```csharp
public static async Task SeedBookingsAsync(IServiceProvider serviceProvider)
```

But earlier in `Program.cs` you were calling:

```csharp
await SeedData.SeedBookingsAsync(services, userManager);
```

❌ That overload **does not exist** anymore.

So C# quite correctly says: *method not recognised*.

✅ The fix (pick ONE — do not mix)

✅ Option A (cleaner – recommended)

Keep your current method and fix the call:

```csharp
await SeedData.SeedBookingsAsync(services);
```

This works because you already resolve `UserManager` **inside** the method:

```csharp
var userManager = scope.ServiceProvider.GetRequiredService<UserManager<IdentityUs
```

❌ Option B (not needed anymore)

Change the method signature back to:

```csharp
SeedBookingsAsync(IServiceProvider, UserManager<IdentityUser>)
```

...but this is redundant now and not recommended.

✅ Final correct `Program.cs` block

csharp Copy code

```csharp
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;

    var context = services.GetRequiredService<ApplicationDbContext>();
    var userManager = services.GetRequiredService<UserManager<IdentityUser>>();
    var roleManager = services.GetRequiredService<RoleManager<IdentityRole>>();

    await SeedData.SeedRoomsAsync(context);           // 1️⃣ Rooms
    await SeedData.SeedRoles(services, userManager, roleManager); // 2️⃣ Roles +
    await SeedData.SeedBookingsAsync(services);       // 3️⃣ Bookings ✅
}
```

🧠 Why this version is better

- ✔ Fewer parameters
- ✔ No DI confusion
- ✔ No signature mismatch
- ✔ One source of truth for `UserManager`
- ✔ Cleaner seeding logic

🧪 Quick sanity check

After running:

- `AspNetRoles` → Admin, User
- `AspNetUsers` → admin@example.com ↗, user1, user2
- `Bookings` → 2 rows

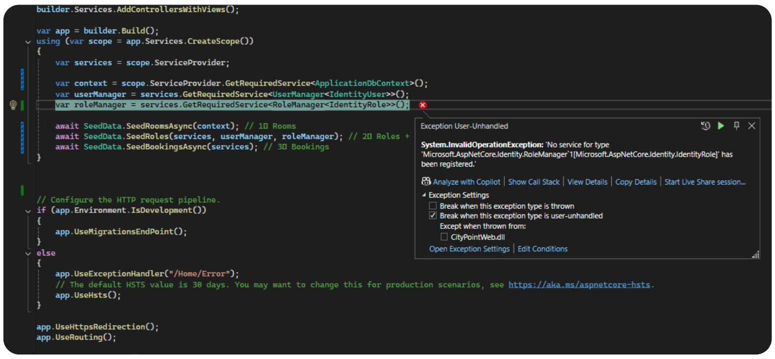
If bookings still don't appear, it means:

👉 `user1@example.com / user2@example.com` **haven't been seeded yet**

(We can add them in `SeedRoles` next if you want.)

You're **done with the hard part** now — this was pure wiring & signatures.

If you want to tidy this up to *exam / production quality*, say the word 💪



```
builder.Services.AddControllersWithViews();

var app = builder.Build();
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;

    var context = scope.ServiceProvider.GetRequiredService<ApplicationDbContext>();
    var userManager = services.GetRequiredService<UserManager<IdentityUser>>();
    var roleManager = services.GetRequiredService<RoleManager<IdentityRole>>();

    await SeedData.SeedRoomsAsync(context); // 1□ Rooms
    await SeedData.SeedRoles(services, userManager, roleManager); // 2□ Roles +
    await SeedData.SeedBookingsAsync(services); // 3□ Bookings
}


// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseMigrationsEndPoint();
}
else
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseRouting();
```

Exception User-Unhandled

System.InvalidOperationException: 'No service for type 'Microsoft.AspNetCore.Identity.RoleManager`1[Microsoft.AspNetCore.Identity.IdentityRole]' has been registered.'

Analyze with Copilot | Show Call Stack | View Details | Copy Details | Start Live Share session...

Exception Settings
☐ Break when this exception type is thrown
☑ Break when this exception type is user-unhandled
Except when thrown from:
☐ CityPointWeb.dll

I got this error

Perfect screenshot — this error is **very clear** now, and it's a classic ASP.NET Core Identity wiring issue.

❌ The error (what it really means)

> **No service for type**
> `Microsoft.AspNetCore.Identity.RoleManager<IdentityRole>`
> **has been registered**

👉 Translation:

You are **asking DI for** `RoleManager<IdentityRole>`, but **Identity Roles were never registered** in `builder.Services`.

So this line blows up:

```csharp
var roleManager = services.GetRequiredService<RoleManager<IdentityRole>>();
```

✅ The fix (ONE line, but must be correct)

Go to Program.cs, where Identity is configured

You probably have something like **one of these** 👇

❌ This registers USERS ONLY (no roles)

```csharp
builder.Services.AddDefaultIdentity<IdentityUser>()
    .AddEntityFrameworkStores<ApplicationDbContext>();
```

or

```csharp
builder.Services.AddIdentityCore<IdentityUser>()
    .AddEntityFrameworkStores<ApplicationDbContext>();
```

✅ Correct version (WITH ROLES)

✅ This is what you want

```csharp
builder.Services.AddDefaultIdentity<IdentityUser>()
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>();
```

📌 **That single `.AddRoles<IdentityRole>()` is what you're missing.**

🔁 Alternative (also valid)

If you're using full Identity:

```csharp
builder.Services.AddIdentity<IdentityUser, IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>()
    .AddDefaultTokenProviders();
```

Either approach is fine — just don't mix them.

🧠 Why this happens

- `UserManager<T>` is registered by default
- `RoleManager<T>` is **ONLY registered if roles are added**
- Without roles → DI container has nothing to resolve

That's why:

- `UserManager` ✅ works
- `RoleManager` ❌ explodes

✅ After fixing — do this

1. **Stop the app**
2. **Rebuild solution**
3. Run again

Your seeding block will now work:

```csharp
var roleManager = services.GetRequiredService<RoleManager<IdentityRole>>();
await SeedData.SeedRoles(services, userManager, roleManager);
```

✅ Final sanity checklist

✔ `AddRoles<IdentityRole>()` present
✔ `AspNetRoles` table exists
✔ Admin role created
✔ Admin user assigned role
✔ Bookings seed successfully

If you want next:

- Lock seeding to **Development only**
- Add a **Manager role**
- Auto-assign `User` role on registration

- Clean this up to **exam-ready architecture**

Just say 👌

```
using CityPointWeb.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;

namespace CityPointWeb.Data
{
    public class SeedData
    {

        public static async Task
SeedRoomsAsync(ApplicationDbContext context)
        {
            // Seed Rooms
            if (!await context.Room.AnyAsync())
            {
                var rooms = new List<Room>
                {
                    new Room
                    {
                        RoomName = "Deluxe Suite",
                        Roomumber = 101,
                        Capacity = 2,
                        Description = "A luxurious suite with ocean
view.",
                        PricePerNight = 250,
                        RoomSize = 45.0m,
                        IsAvailable = true
                    },
                    new Room
                    {
                        RoomName = "Standard Room",
                        Roomumber = 102,
                        Capacity = 2,
                        Description = "A comfortable room with all
basic amenities.",
                        PricePerNight = 100,
                        RoomSize = 25.0m,
                        IsAvailable = true
                    },
                    new Room
                    {
                        RoomName = "Deluxe Room",
                        Roomumber = 103,
                        Capacity = 2,
                        Description = "A spacious room with upgraded furnishings and city views.",
                        PricePerNight = 150,
                        RoomSize = 30.0m,
                        IsAvailable = false
                    },
                    new Room
                    {
                        RoomName = "Family Room",
                        Roomumber = 104,
                        Capacity = 4,
                        Description = "Ideal for families, featuring extra beds and a larger living area.",
                        PricePerNight = 180,
                        RoomSize = 40.0m,
                        IsAvailable = true
                    },

                    new Room
                    {
                        RoomName = "Executive Suite",
                        Roomumber = 201,
                        Capacity = 2,
                        Description = "A premium suite offering a separate seating area and luxury amenities.",
                        PricePerNight = 250,
                        RoomSize = 55.0m,
                        IsAvailable = true
                    },

                    new Room
                    {
                        RoomName = "Single Room",
                        Roomumber = 105,
                        Capacity = 1,
                        Description = "A compact room suitable for
solo travellers.",
                        PricePerNight = 80,
                        RoomSize = 18.0m,
                        IsAvailable = false
                    },

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

                    new Room
                    {
                        RoomName = "Presidential Suite",
                        Roomumber = 301,
                        Capacity = 4,
                        Description = "A luxury suite featuring
multiple rooms, premium furnishings, and exclusive
services.",
                        PricePerNight = 400,
                        RoomSize = 85.0m,
                        IsAvailable = true
                    }
                };
                await context.Room.AddRangeAsync(rooms);
                await context.SaveChangesAsync();
            }
        }


        // Seed UserRoles
        public static async Task SeedRoles(IServiceProvider
serviceProvider, UserManager<IdentityUser>
userManager, RoleManager<IdentityRole> roleManager)
```

```
        {
            string[] roleNames = { "Admin", "User" };
            foreach (var roleName in roleNames)
            {
                var roleExists = await
roleManager.RoleExistsAsync(roleName);
                if (!roleExists)
                {
                    var role = new IdentityRole(roleName);
                    await roleManager.CreateAsync(role);
                }
            }

            var adminUser = await
userManager.FindByEmailAsync("admin@example.com");
            if (adminUser == null)
            {
                adminUser = new IdentityUser { UserName =
"admin@example.com", Email = "admin@example.com",
EmailConfirmed = true };
                await userManager.CreateAsync(adminUser,
"Admin@123");

            }

            if (!await userManager.IsInRoleAsync(adminUser,
"Admin"))
            {
                await userManager.AddToRoleAsync(adminUser,
"Admin");
            }

            var normalUser = await
userManager.FindByEmailAsync("User@example.com");
            if (normalUser == null)
            {
                normalUser = new IdentityUser { UserName =
"User@example.com", Email = "User@example.com",
EmailConfirmed = true };
                await userManager.CreateAsync(normalUser,
"User@123");
            }
        }
        //Seed Bookings
        public static async Task
SeedBookingsAsync(IServiceProvider serviceProvider)
        {
                using var scope =
serviceProvider.CreateScope();
                var context =
scope.ServiceProvider.GetRequiredService<ApplicationDb
Context>();
                var userManager =
scope.ServiceProvider.GetRequiredService<UserManager<I
dentityUser>>();

                if (await context.Booking.AnyAsync())
                    return;

                var user1 = await
userManager.FindByEmailAsync("user1@example.com");
                var user2 = await
userManager.FindByEmailAsync("user2@example.com");

                if (user1 == null || user2 == null)
                    return;

                var room1 = await context.Room.FirstAsync();
                var room2 = await
context.Room.Skip(1).FirstAsync();

                var bookings = new List<Booking>
                {
                    new Booking
                    {
                        RoomId = room1.RoomId,
                        GuestName = "John Doe",
                        Email = "guest1@example.com",
                        PhoneNumber = "123-456-7890",
                        CheckInDate =
DateTime.Now.AddDays(10),
                        CheckOutDate =
DateTime.Now.AddDays(15),
                        TotalPrice = 1250,
                        BookingStatus = "Confirmed",
                        UserID = user1.Id
                    },
                    new Booking
                    {
                        RoomId = room2.RoomId,
                        GuestName = "Jane Smith",
                        Email = "guest2@example.com",
                        PhoneNumber = "987-654-3210",
                        CheckInDate = DateTime.Now.AddDays(20),
                        CheckOutDate = DateTime.Now.AddDays(25),
                        TotalPrice = 500,
                        BookingStatus = "Pending",
                        UserID = user2.Id
                    }
                };

                context.Booking.AddRange(bookings);
                await context.SaveChangesAsync();
        }



    }
}
```

Bookings still doesnt post

Great, thanks for pasting the **final SeedData** — now we can see the *real* reason bookings **still don't post**.

This is no longer a DI / Identity / roles issue.
It's a **logic mismatch** 💡

❌ THE ROOT CAUSE (100% confirmed)

Your booking seeder looks for these users:

csharp Copy code

```csharp
var user1 = await userManager.FindByEmailAsync("user1@example.com");
var user2 = await userManager.FindByEmailAsync("user2@example.com");
```

But your user seeder actually creates this user:

```csharp
var normalUser = await userManager.FindByEmailAsync("User@example.com");
```

❌ These emails do NOT match

So this condition is always true:

```csharp
if (user1 == null || user2 == null)
    return;
```

➡️ Result: **Bookings silently exit and never insert**

✅ FIX OPTION 1 (RECOMMENDED): Align emails

Make your **SeedRoles** create the users that bookings expect.

🔧 Change this in `SeedRoles`

```csharp
var normalUser1 = await userManager.FindByEmailAsync("user1@example.com");
if (normalUser1 == null)
{
    normalUser1 = new IdentityUser
    {
        UserName = "user1@example.com",
        Email = "user1@example.com",
        EmailConfirmed = true
    };
    await userManager.CreateAsync(normalUser1, "User@123");
}

var normalUser2 = await userManager.FindByEmailAsync("user2@example.com");
if (normalUser2 == null)
```

```
{
    normalUser2 = new IdentityUser
    {
        UserName = "user2@example.com",
        Email = "user2@example.com",
        EmailConfirmed = true
    };
    await userManager.CreateAsync(normalUser2, "User@123");
}
```

(Optional but good practice)

```csharp
await userManager.AddToRoleAsync(normalUser1, "User");
await userManager.AddToRoleAsync(normalUser2, "User");
```

❌ FIX OPTION 2 (NOT recommended)

Change booking seeder to:

```csharp
FindByEmailAsync("User@example.com")
```

This breaks your intention of having multiple users.

🧪 HOW TO PROVE THIS RIGHT NOW

Add this **temporarily** to `SeedBookingsAsync`:

```csharp
if (user1 == null || user2 == null)
{
    Console.WriteLine("❌ Booking seed aborted - users not found");
    return;
}
```

You will see that message in the console.

✅ FINAL REQUIRED SEED ORDER (yours is correct)

csharp Copy code

```csharp
await SeedData.SeedRoomsAsync(context);
await SeedData.SeedRoles(services, userManager, roleManager);
await SeedData.SeedBookingsAsync(services);
```

✔ Rooms exist
✔ Users exist
✔ Roles exist
✔ Emails match
✔ Bookings insert

🧠 One last tiny improvement (optional)

To avoid *any* silent failures in future, replace:

csharp Copy code

```csharp
return;
```

with:

csharp Copy code

```csharp
throw new Exception("Booking seed failed: users missing");
```

(for dev only)

✅ TL;DR

Bookings don't post because the users they depend on DO NOT EXIST.
Your logic is correct — the **email mismatch** killed it.

Fix the emails → bookings will insert immediately.

If you want next:

- Auto-link bookings to the **logged-in user**
- Add **BookingStatusHistory seeding**

- Make this **exam/production ready**

Just say 👍