Day 2

Here's your full 📅 **Day 2 – Backend with ASP.NET Core** schedule, in your preferred format.

Your focus today is on building and exposing the API from your .NET Core backend to power your React frontend (which you set up on Day 1).

📅 Day 2 – Build ASP.NET Core API (Backend for Task Manager)

🎯 Goal: Create backend logic to store, fetch, and manage tasks using .NET Core Web API + SQL Server

🕒 Time: ~5 hours

✅ Task 1: Setup Models and DbContext (~1 hr)

🧠 Why it matters:

This defines the **data structure** and lets Entity Framework Core generate SQL tables automatically.

📘 Concepts explained:

- `Task.cs` model class
- `ApplicationDbContext.cs` : where your DB tables are mapped
- EF Core: ORM (Object Relational Mapper)
- `DbSet<T>` = table
- `Data` folder holds DB context
- `Models` folder holds your C# entities

🔨 Example Code:

```
// Models/Task.cs
public class Task {
  public int Id { get; set; }
  public string Title { get; set; }
  public string Description { get; set; }
}
```

```
// Data/ApplicationDbContext.cs
public class ApplicationDbContext : DbContext {
  public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options) { }
  public DbSet<Task> Tasks { get; set; }
}
```

✅ Task 2: Add DB Connection String + EF Core Setup (~30 mins)

🧠 Why it matters:

Links your app to **SQL Server** and prepares it for migrations (automatic DB table creation).

📘 Concepts explained:

- `appsettings.json` : contains connection string
- `AddDbContext` : registers EF in program
- `dotnet ef migrations add` : generates DB table scripts
- `dotnet ef database update` : creates tables

🔨 Example:

```
// appsettings.json
"ConnectionStrings": {
  "DefaultConnection": "Server=.;Database=TaskManagerDb;Trusted_Connection=True;"
}
```

```
// Program.cs or Startup.cs
builder.Services.AddDbContext<ApplicationDbContext>(options =>
  options.UseSqlServer(builder.Configuration.GetConnectionString("Default
Connection")));
```

Then run:

```
dotnet ef migrations add InitialCreate
dotnet ef database update
```

✅ Task 3: Create TaskController (API Routes) (~1 hr)

🧠 Why it matters:

This is the **core of your backend** — it exposes endpoints like `GET /task`, `POST /task`, etc.

📘 Concepts explained:

- API Controller: uses `[ApiController]` and `[Route("api/[controller]")]`
- REST methods: `GET`, `POST`, `PUT`, `DELETE`
- `FromBody`: binds JSON to C# object

🔨 Example:

```
[ApiController]
[Route("api/[controller]")]
public class TaskController : ControllerBase {
  private readonly ApplicationDbContext _context;

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

  [HttpGet]
  public async Task<IActionResult> GetTasks() {
    var tasks = await _context.Tasks.ToListAsync();
    return Ok(tasks);
  }
```

```
    [HttpPost]
    public async Task<IActionResult> AddTask([FromBody] Task task) {
      _context.Tasks.Add(task);
      await _context.SaveChangesAsync();
      return Ok(task);
    }
  }
```

✅ Task 4: Test API in Swagger/Postman (~30 mins)

🧠 Why it matters:

You need to confirm the API **works correctly** before you call it from the frontend.

📘 Concepts explained:

- Swagger UI for testing endpoints
- Postman: powerful REST client

🔨 Steps:

1. Run app → Swagger opens → test `GET /task`, `POST /task`
2. Use Postman to send JSON:

```
{
  "title": "Learn EF Core",
  "description": "Understand DbContext & Migration"
}
```

✅ Task 5: Push to GitHub (~30 mins)

🧠 Why it matters:

Keeps your code versioned and ready for collaboration or showcasing in interviews.

📘 Concepts explained:

- Commit history shows professionalism

- Good commit messages = better resume appeal

🔨 Example:

```
git add .
git commit -m "feat: backend API for tasks using EF Core"
git push
```

✅ By End of Day 2, You Will Have:

Backend Feature	Status
`Task` Model	✅
`DbContext` setup with SQL Server	✅
Connection string + EF migration setup	✅
`GET` , `POST` API in `TaskController`	✅
API tested in Swagger/Postman	✅
Code pushed to GitHub	✅

Let me know when you're ready for **Day 3**, where we start connecting this backend to the **React frontend using Axios**, render task lists, and add/delete tasks 💪