Contents

[How The Code All Works Together (Step-by-Step) - Scenario: User sends POST /api/books 2](#_Toc203346824)

[OVERVIEW 2](#_Toc203346825)

[1a. of 4. Web API Controller: 3](#_Toc203346826)

[1b. or 4. BookDto (Application Layer) 3](#_Toc203346827)

[2. of 4. MediatR Pipeline: Finds CreateBookCommandHandler 4](#_Toc203346828)

[\* MediatR is a library in .NET used to implement the **Mediator pattern** 4](#_Toc203346829)

[3. of 4. Command Handler: 5](#_Toc203346830)

[4. of 4.Repository: 6](#_Toc203346831)

# How The Code All Works Together (Step-by-Step) - Scenario: User sends POST /api/books

### OVERVIEW

|  |
| --- |
| **1. Web API Controller:**   1. Receives CreateBookCommand via [HttpPost] 2. Passes command to MediatR.Send()   **2. MediatR Pipeline:**   1. Finds CreateBookCommandHandler 2. Executes Handle()   **3. Command Handler:**   1. Creates a Book entity 2. Calls \_repo.AddAsync(book)   **4. Repository:**   1. Uses AppDbContext (EF Core) to add the entity 2. Saves to SQL Server database via SaveChangesAsync |

## 1a. of 4. Web API Controller via [HttpPost]:

1. Receives CreateBookCommand object via [HttpPost]
2. Passes command to MediatR.Send()

[Route("api/[controller]")]

[ApiController]

public class BooksController : ControllerBase

{

private readonly IMediator \_mediator;

public BooksController(IMediator mediator) => \_mediator = mediator;

[HttpGet]

public async Task<ActionResult<IEnumerable<BookDto>>> Get()

=> Ok(await \_mediator.Send(new GetAllBooksQuery()));

[HttpPost]

public async Task<ActionResult<Guid>> Post(CreateBookCommand cmd)

{

var id = await \_mediator.Send(cmd);

return CreatedAtAction(nameof(Get), new { id }, id);

}

}

## 1b. or 4. BookDto (Application Layer)

public record BookDto(Guid Id, Guid ResourceId, string Title, string Author, int TotalPages, int PagesRead);

Used for **output** to the client, not for internal domain logic.

## 2. of 4. MediatR Pipeline: Finds CreateBookCommandHandler

### \* MediatR is a library in .NET used to implement the **Mediator pattern**

1. Finds CreateBookCommandHandler
2. Executes Handle()

**CQRS: Command & Handler**

public record CreateBookCommand(Guid ResourceId, string Title, string Author, int TotalPages) : IRequest<Guid>;

public class CreateBookCommandHandler : IRequestHandler<CreateBookCommand, Guid>

{

private readonly IBookRepository \_repo;

public CreateBookCommandHandler(IBookRepository repo) => \_repo = repo;

public async Task<Guid> Handle(CreateBookCommand request, CancellationToken ct)

{

var book = new Book

{

Id = Guid.NewGuid(),

ResourceId = request.ResourceId,

Title = request.Title,

Author = request.Author,

TotalPages = request.TotalPages,

PagesRead = 0

};

await \_repo.AddAsync(book);

return book.Id;

}

## 3. of 4. Command Handler:

1. Creates a Book entity
2. Calls \_repo.AddAsync(book)

**Book Entity (Domain Layer)**

public class Book

{

public Guid Id { get; set; }

public Guid ResourceId { get; set; } // FK

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

public string Author { get; set; } = string.Empty;

public int TotalPages { get; set; }

public int PagesRead { get; set; }

// Domain logic

public int PagesLeftToRead() => Math.Max(0, TotalPages - PagesRead);

public double PercentageRead() => TotalPages == 0 ? 0 : Math.Round((double)PagesRead / TotalPages \* 100, 2);

}

## 4. of 4.Repository:

1. Uses AppDbContext (EF Core) to add the entity
2. Saves to SQL Server database via SaveChangesAsync

**BookRepository Implementation (Infrastructure Layer)**

public class BookRepository : IBookRepository

{

private readonly AppDbContext \_context;

public BookRepository(AppDbContext context) => \_context = context;

//GET ALL

public async Task<IEnumerable<Book>> GetAllAsync() => await \_context.Books.ToListAsync();

//GET BY ID

public async Task<Book?> GetByIdAsync(Guid id) => await \_context.Books.FindAsync(id);

//INSERT

public async Task AddAsync(Book book)

{

await \_context.Books.AddAsync(book);

await \_context.SaveChangesAsync();

}

//UPDATE

public async Task UpdateAsync(Book book)

{

\_context.Books.Update(book);

await \_context.SaveChangesAsync();

}

//DELETE

public async Task DeleteAsync(Guid id)

{

var book = await GetByIdAsync(id);

if (book != null)

{

\_context.Books.Remove(book);

await \_context.SaveChangesAsync();

}

}

}