Using EF Core in ASP.NET Core Apps

Julie Lerman

EF Core Expert and Software Coach

@julielerman | thedatafarm.com

Module Overview

Review lifecycle of DbContext in web apps

Create an ASP.NET Core API project

Use a template to create an endpoints class to perform read & write tasks for Authors

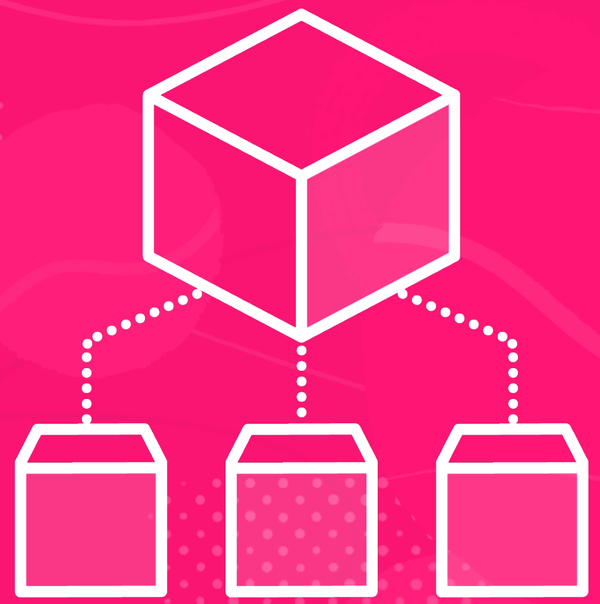
Wire up the API to the PubContext

Learn how to combine entities and DTOs

Learn tips for debugging and logging

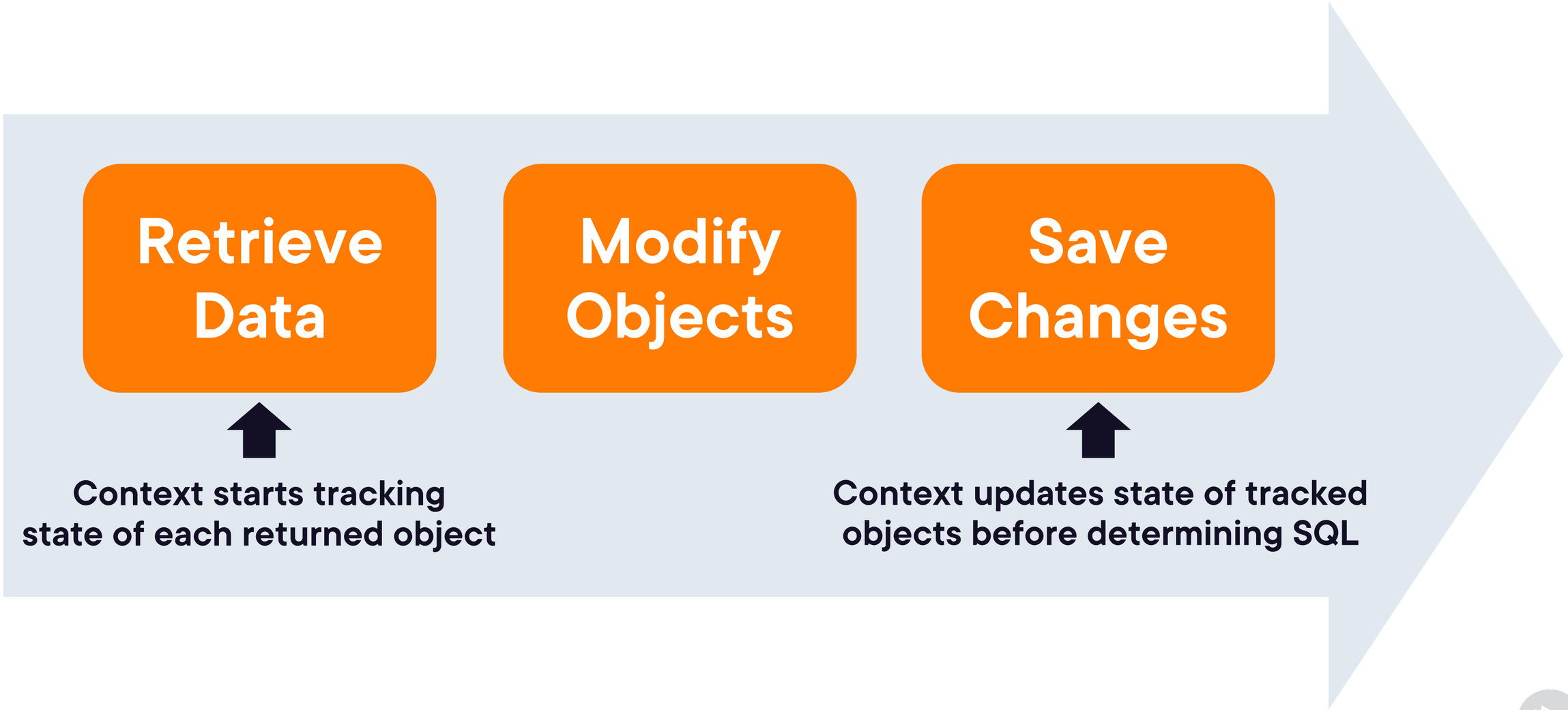
Interact with API to read, write and delete data

We won't do a lot of work with *related* data in this demo.



Reviewing EF Core's Lifecycle in Disconnected Apps

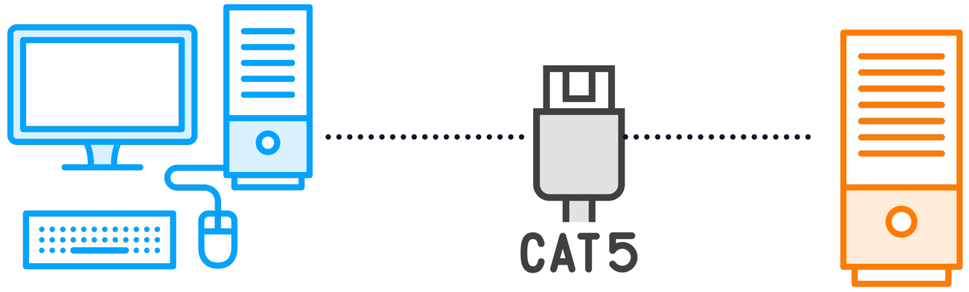
Working in a Single DbContext Instance



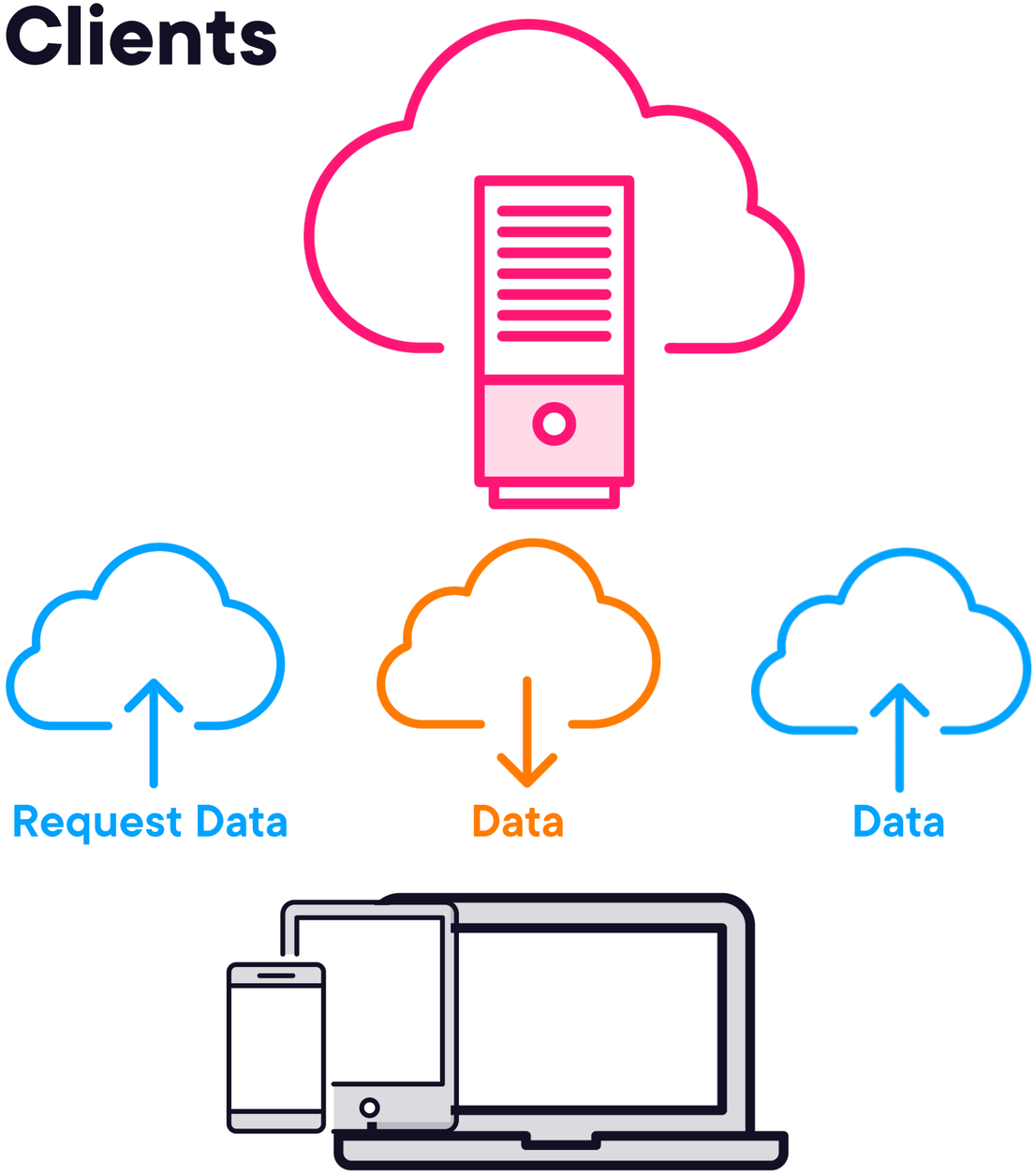
Connected Data Access

Client Storing Data Locally

Network Connected Clients

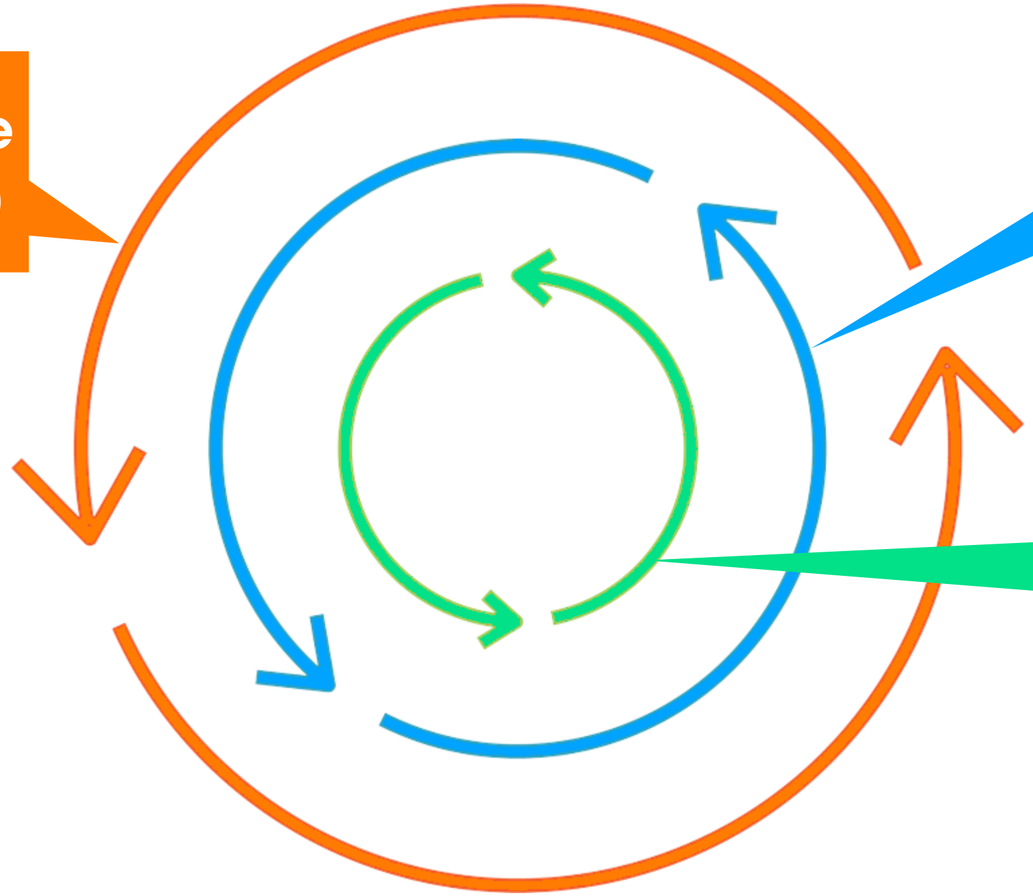


Disconnected Clients



Short-Lived DbContexts

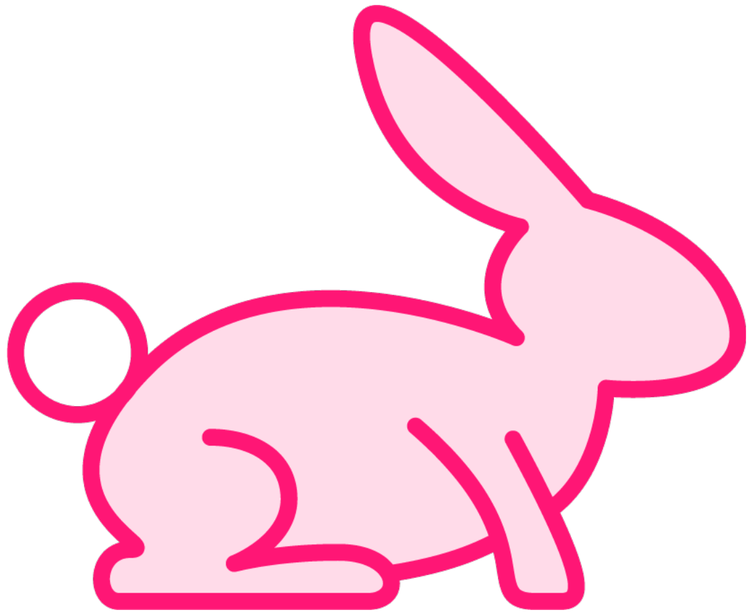
Web application lifetime
(app restart to recycle)



Controller instance
per Request

DbContext per
controller instance

Query Performance Bonus in Web Apps



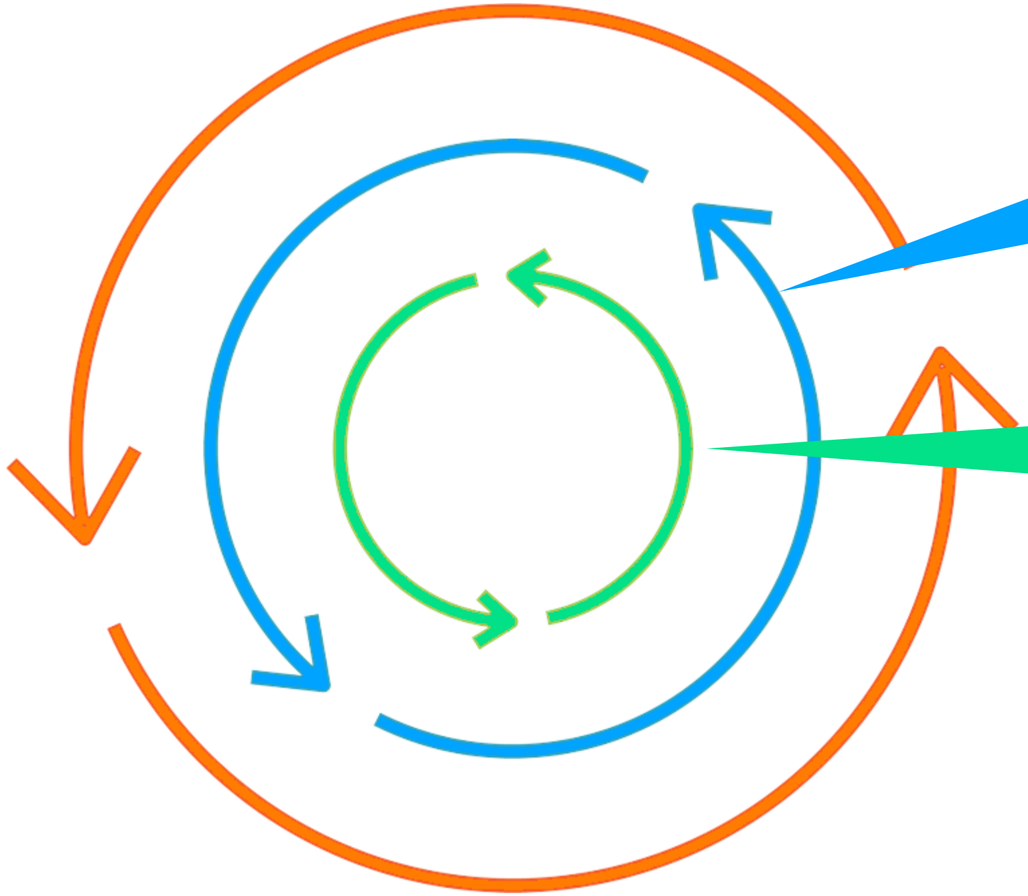
The context can’t track data on the client device

Don’t waste time and resources tracking

Use no-tracking queries

In fact … just make the entire DbContext a no-tracking context

Short-Lived DbContexts

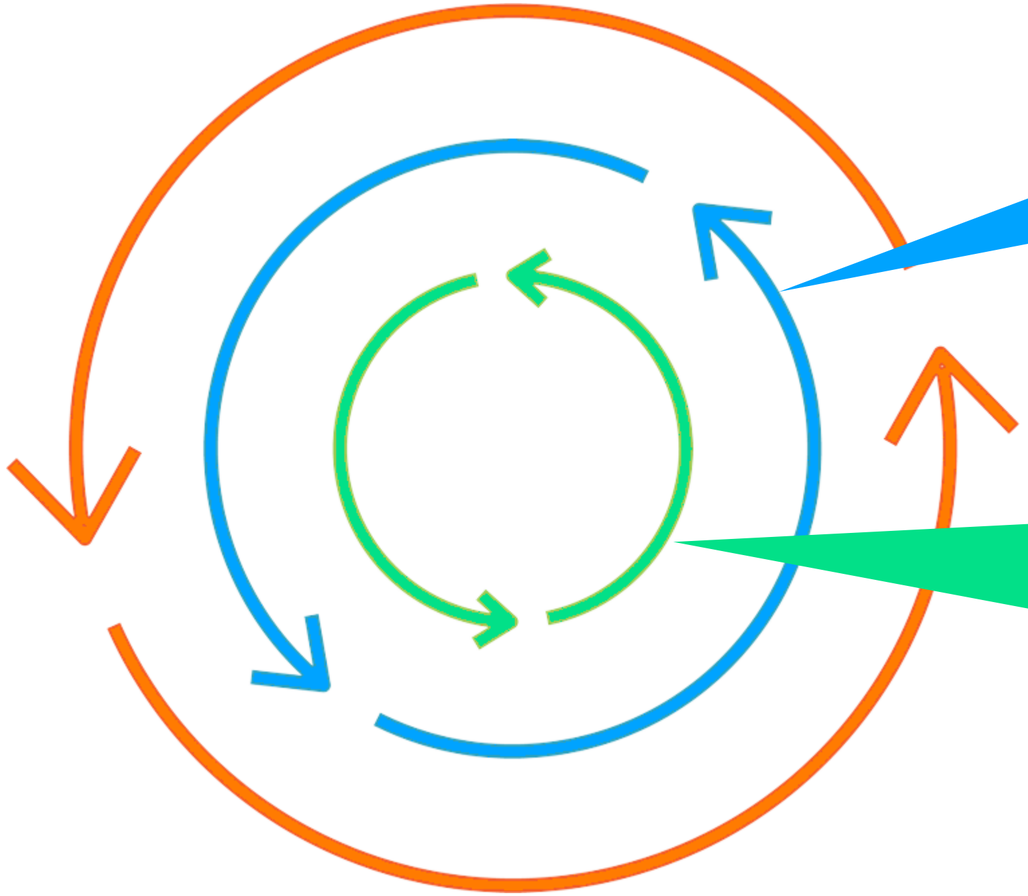


App: Request some data

DbContext: Query and return results

When the DbContext is disposed, any change tracking objects disappear. Creating them would be a wasted effort

Short-Lived DbContexts



App: Save changes to that data

DbContext knows nothing about this data! Needs to know if it should add, update or delete!

Various Ways to Inform Context of State

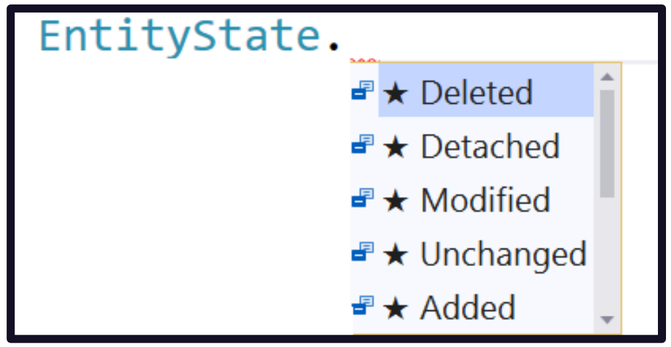
DbSet Methods

```
Authors.Add(newAuth);

Authors.Update(existingAuth);

Authors.Remove(existingAuth);
```

Set DbEntry.State



Retrieve and modify from database

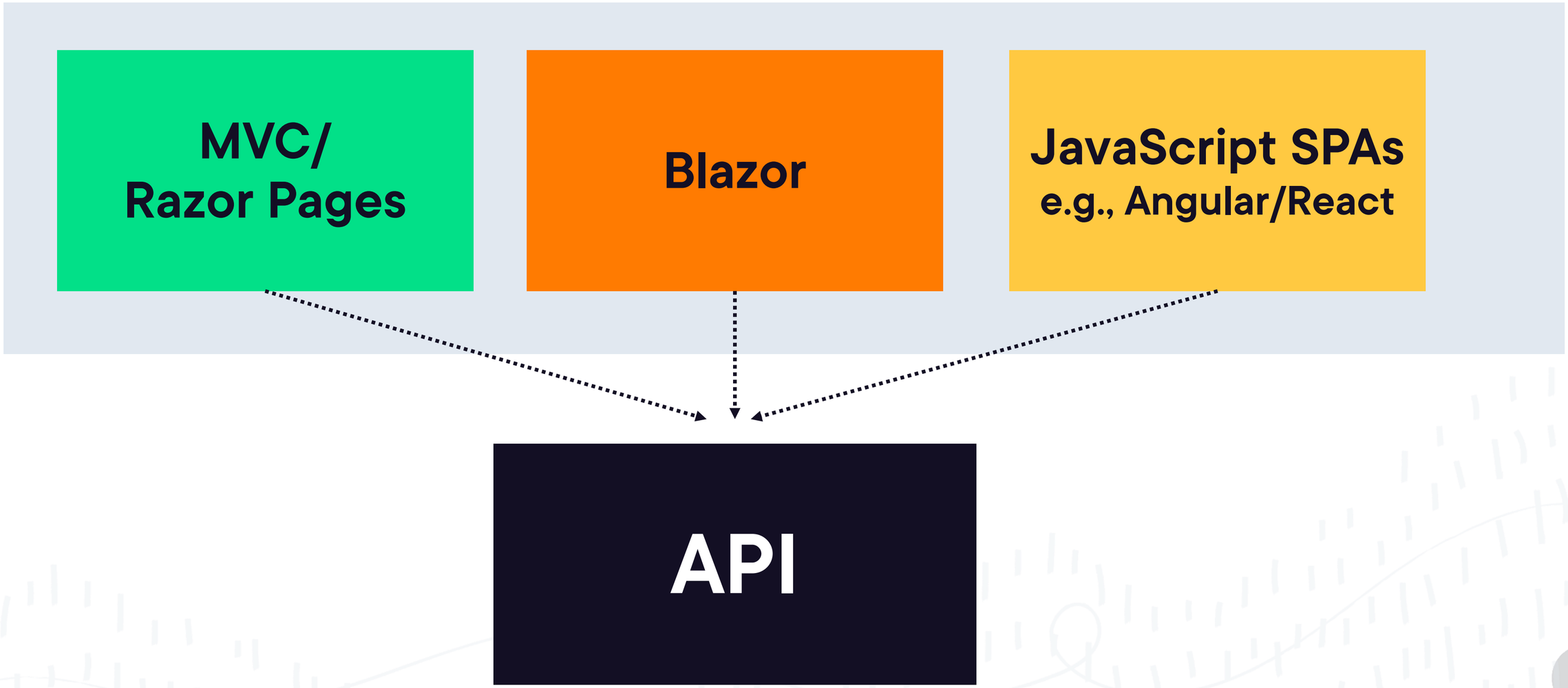
```
void UpdateDBAuthorValues(Author aFromRequest)
{
    var a = _context.Authors
        .Find(aFromRequest.AuthorId);
    //set values with aFromRequestValues
}
```

In disconnected scenarios, it’s up to you to inform the context about object state.

Not an EF Core problem, but a standard conundrum across all tech stacks

Adding the ASP.NET Core Project

Front Ends Talk to Server-Side APIs



Creating Author Endpoints for the API

Wiring up ASP.NET Core with EF Core

Add an Authors Endpoints

1. Add references to projects with entities and DbContext
2. Add endpoint class
   (“API with read/write endpoints using EF Core”)

Generated API Endpoints with EF Core

Stake in the ground starting point

Returning domain objects to [or expecting them from the caller] is an anti-pattern

Further on, we'll refactor to align with preferred practices

Wiring up the ASP.NET Core App with the DbContext

Wiring up ASP.NET Core with EF Core

Add an Authors Endpoints

1. Add references to projects with entities and DbContext
2. Add endpoint class
   (“API with read/write endpoints using EF Core”)

Program.cs

3. Add services for DbContext with UseSqlServer to program.cs

appsettings.json

4. Add connection string config
5. Add EF Core logging config

PubContext.cs

6. Add constructor that takes in DbContextOptions
7. Remove optionsBuilder from OnConfiguring
8. Clean up using statements

Wiring up ASP.NET Core with EF Core

Add an Authors Endpoints

1. Add references to projects with entities and DbContext
2. Add endpoint class
   (“API with read/write endpoints using EF Core”)

Program.cs

3. Add services for DbContext with UseSqlServer to program.cs

appsettings.json

4. Add connection string config
5. Add EF Core logging config

We’re almost there!

Dependency Injection

C# 10 Dependency Injection

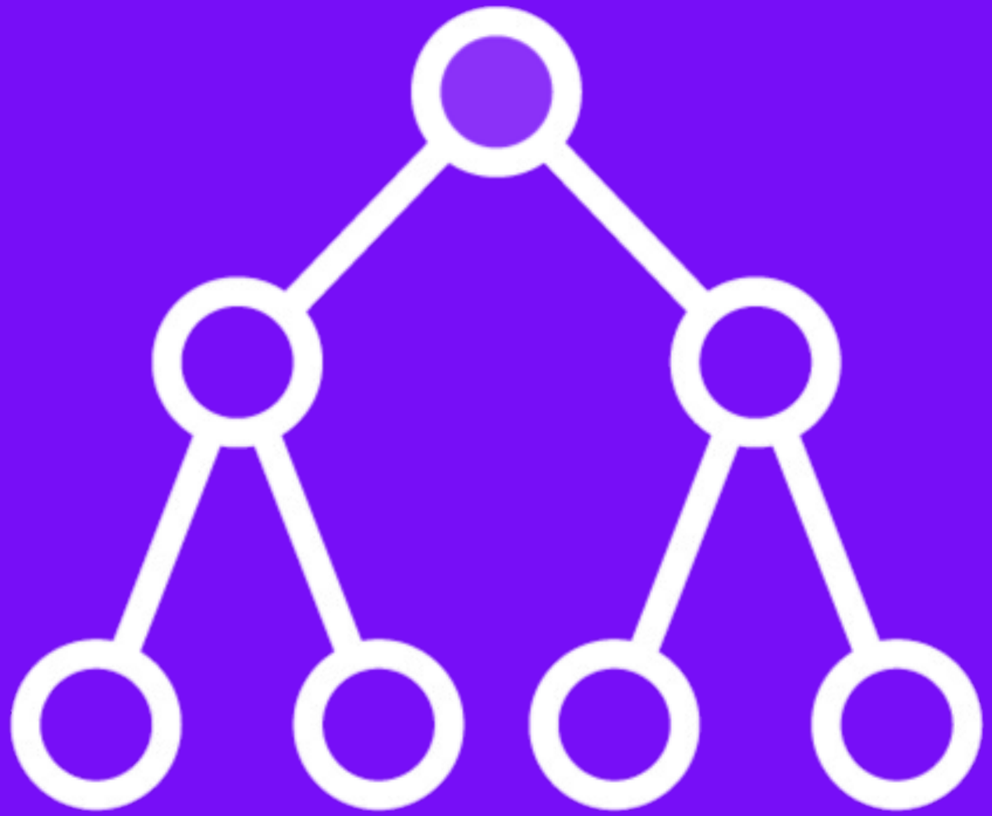
Henry Been

Loose coupling

SOLID Principals for C# Developers

Steve Smith

Running the API to See the Output and Logs



Getting Related Data

An important lesson about recursive data in our web application!

Refactoring the API to Align with Common Practices Using DTOs

Generated API Endpoints with EF Core

Stake in the ground starting point

Returning domain objects to [or expecting them from the caller] is an anti-pattern

NOW

Further on, we'll refactor to align with preferred practices

Data Transfer Object (DTO)

Simple class to transfer data between processes

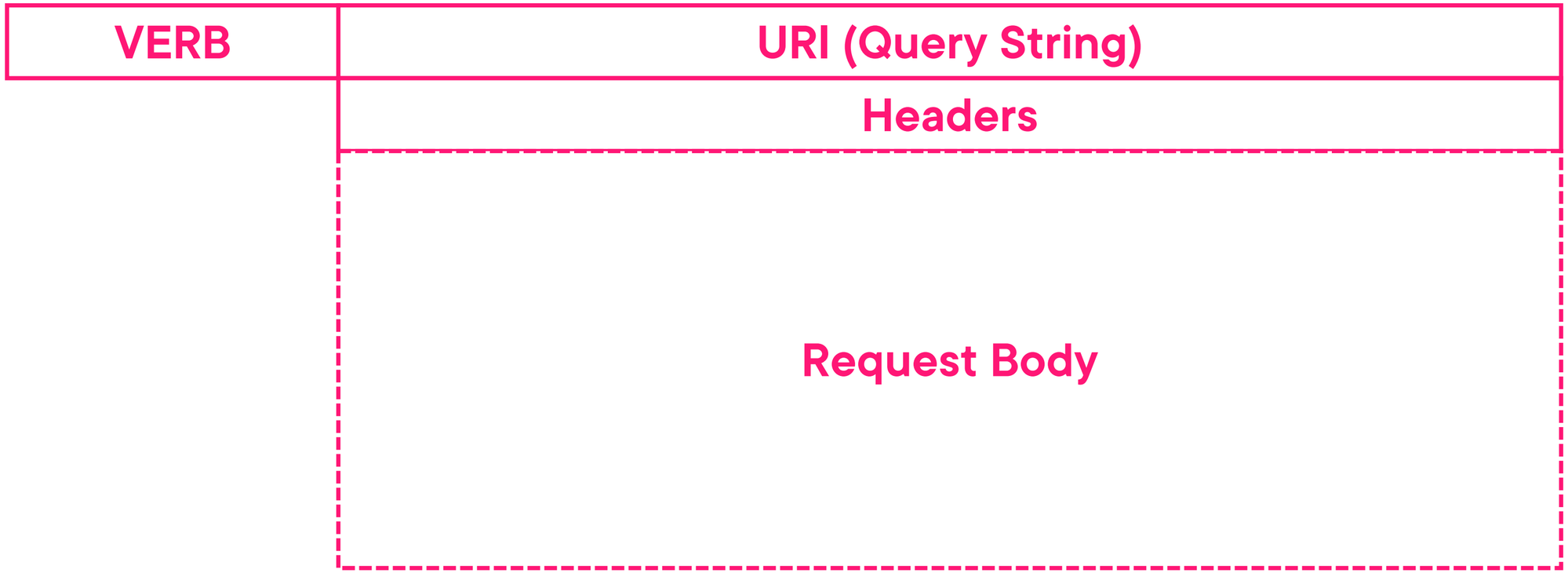
API Endpoints with DTOs

Convert DTOs to entities
Convert entities to DTOs

Exploring and Debugging Insert, Update & Delete API Methods

For these persistence methods, we will only work with author objects, not relationships.

REST APIs Have Several Parts



From Shawn Wildermuth's Designing RESTful Web APIs course

The template- generated API with EF Core code is a pretty good stake in the ground.

Review

For disconnected apps:

- Short-lived DbContexts
- Asynchronous methods
- Non-tracking queries/DbContext

ASP.NET Core can do the hard work for D.I. and logging

Template controller is a stake in the ground

Use DTOs to communicate with calling client and use entities with EF Core

Up Next:

Testing with EF Core

Resources

EF Core Documentation: learn.microsoft.com/ef

Swagger documentation: swagger.io

JSONVue extension for Google Chrome:
github.com/gildas-lormeau/JSONVue

C# 10 Dependency Injection, Henry Been
app.pluralsight.com/profile/author/henry-been

Resources Cont.

Designing RESTful Web APIs, Shawn Wildermuth
app.pluralsight.com/library/courses/designing-restful-web-apis

SOLID Principles for C# Developers
app.pluralsight.com/library/courses/csharp-solid-principles/table-of-contents

Logging in .NET Core and ASP.NET Core
learn.microsoft.com/aspnet/core/fundamentals/logging

Blazor Sample App with EF Core
learn.microsoft.com/aspnet/core/blazor/blazor-server-ef-core