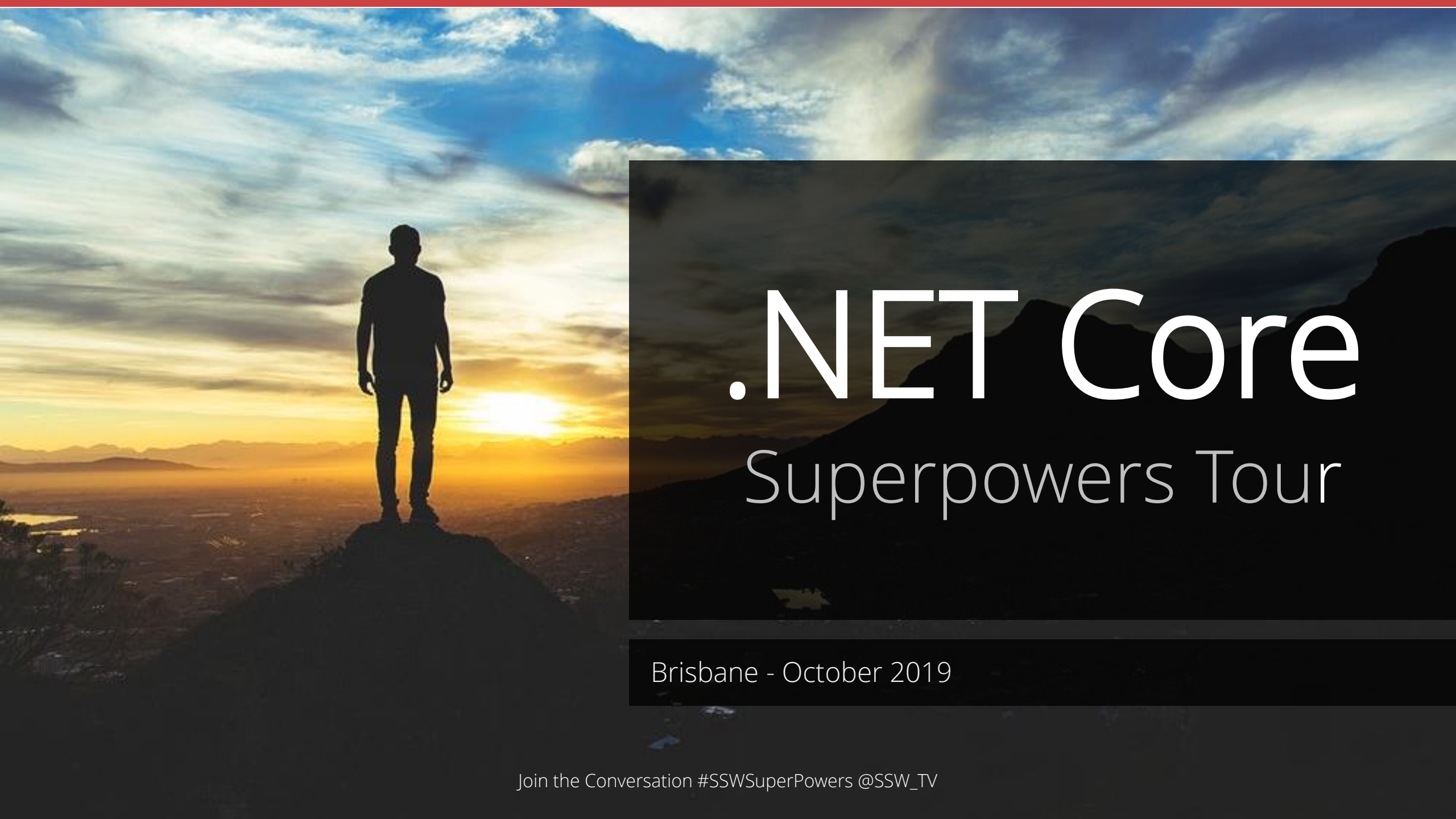


.NET Core

Superpowers Tour

Brisbane - October 2019

Join the Conversation #SSWSuperPowers @SSW_TV

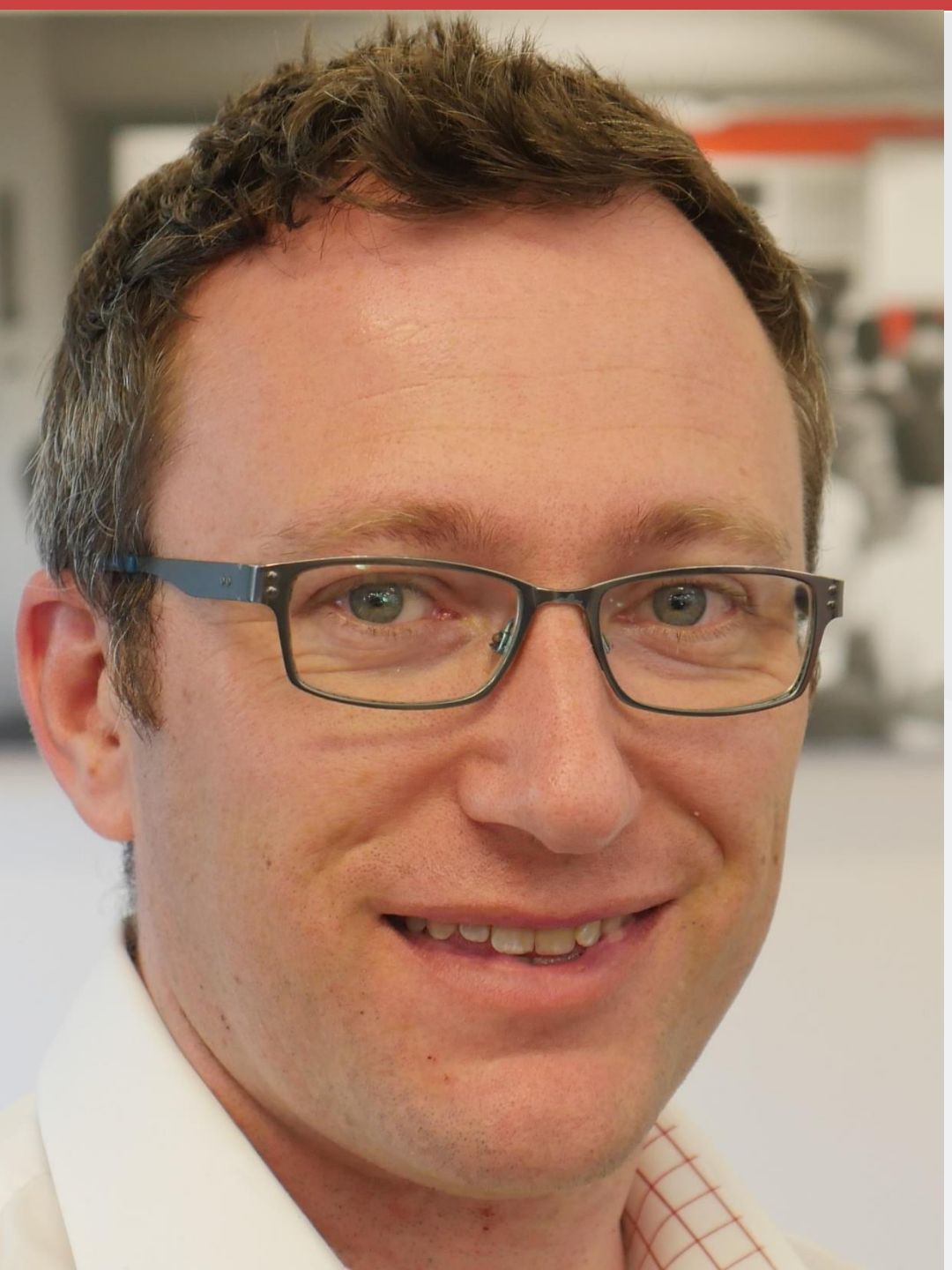
House Keeping

Facilities

WiFi

Timetable

09:00	Registration
10:30 – 10:50	Morning Tea
13:00 – 14:00	Lunch
15:30 – 15:50	Afternoon Tea
17:00	Finish



Brendan Richards

SSW Solution Architect

Linux user since 1995

.NET Developer since 2004

.NET Core Since Beta 2

brendanssw

github.com/brendanrichards

Join the Conversation #SSWSuperPowers @SSW_TV

Jason Taylor

SSW Solution Architect

Linux user in 1997, 2005, 2013, 2019

.NET Developer since 2002

.NET Core Since 1.0

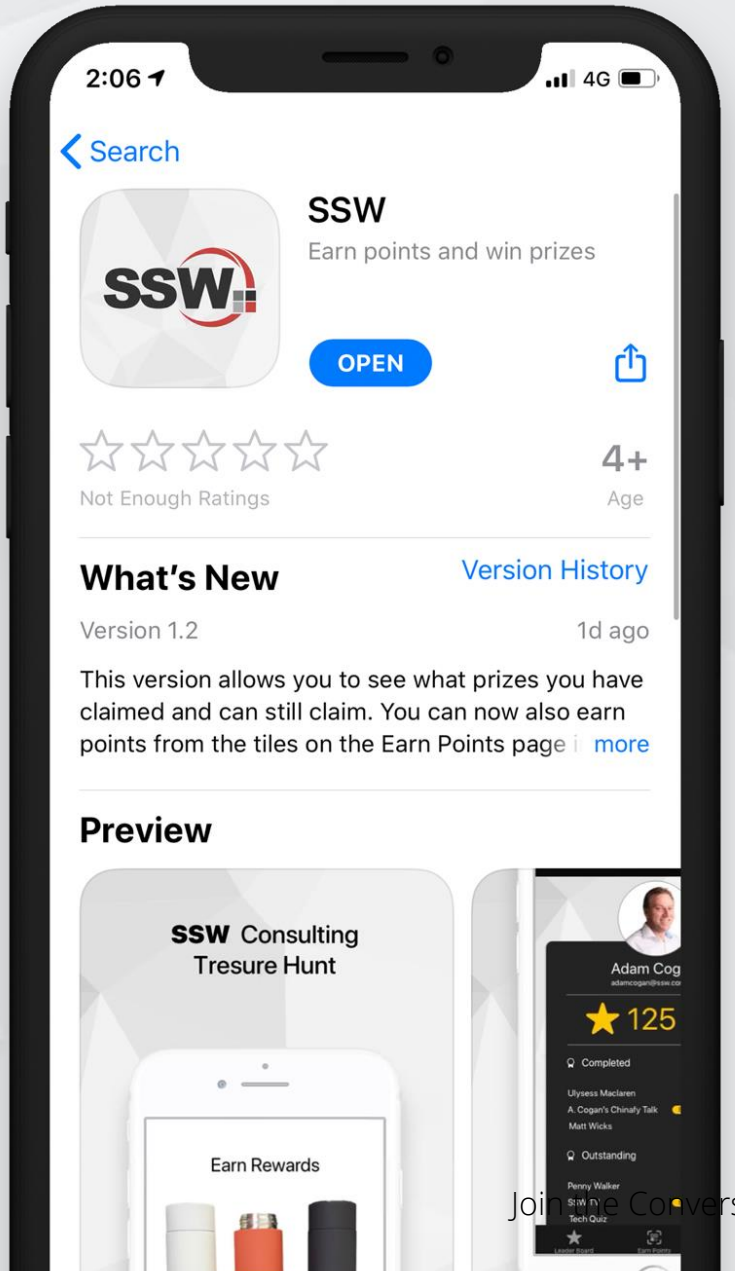
jasongtau

github.com/jasongt

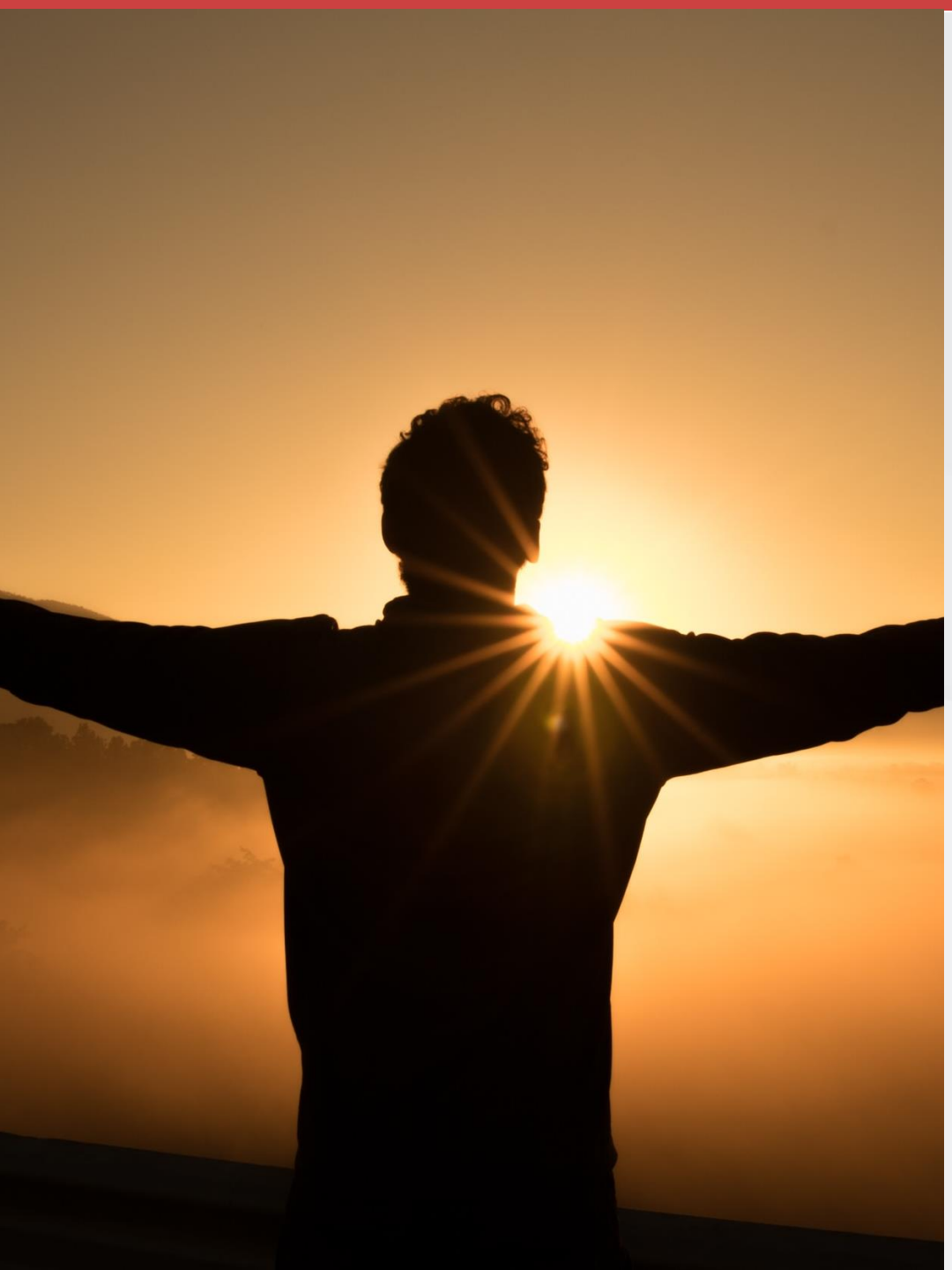
codingflow.net

youtube.com/jasongt

Join the Conversation #SSWSuperPowers @SSW_TV



1. Download SSW App
2. Scan QR codes to win prizes



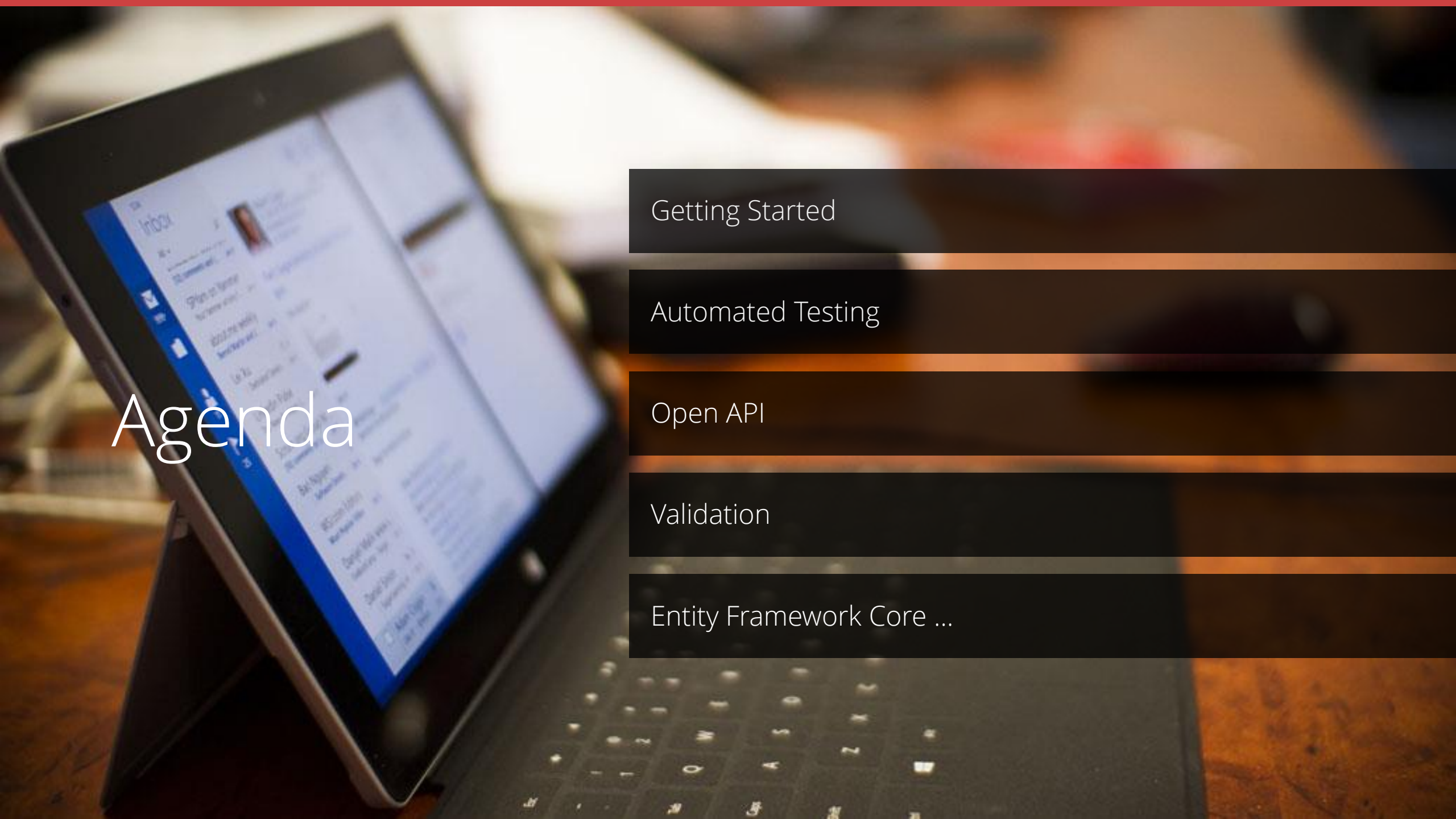
About You

What's your name?

What's your goal for the day?

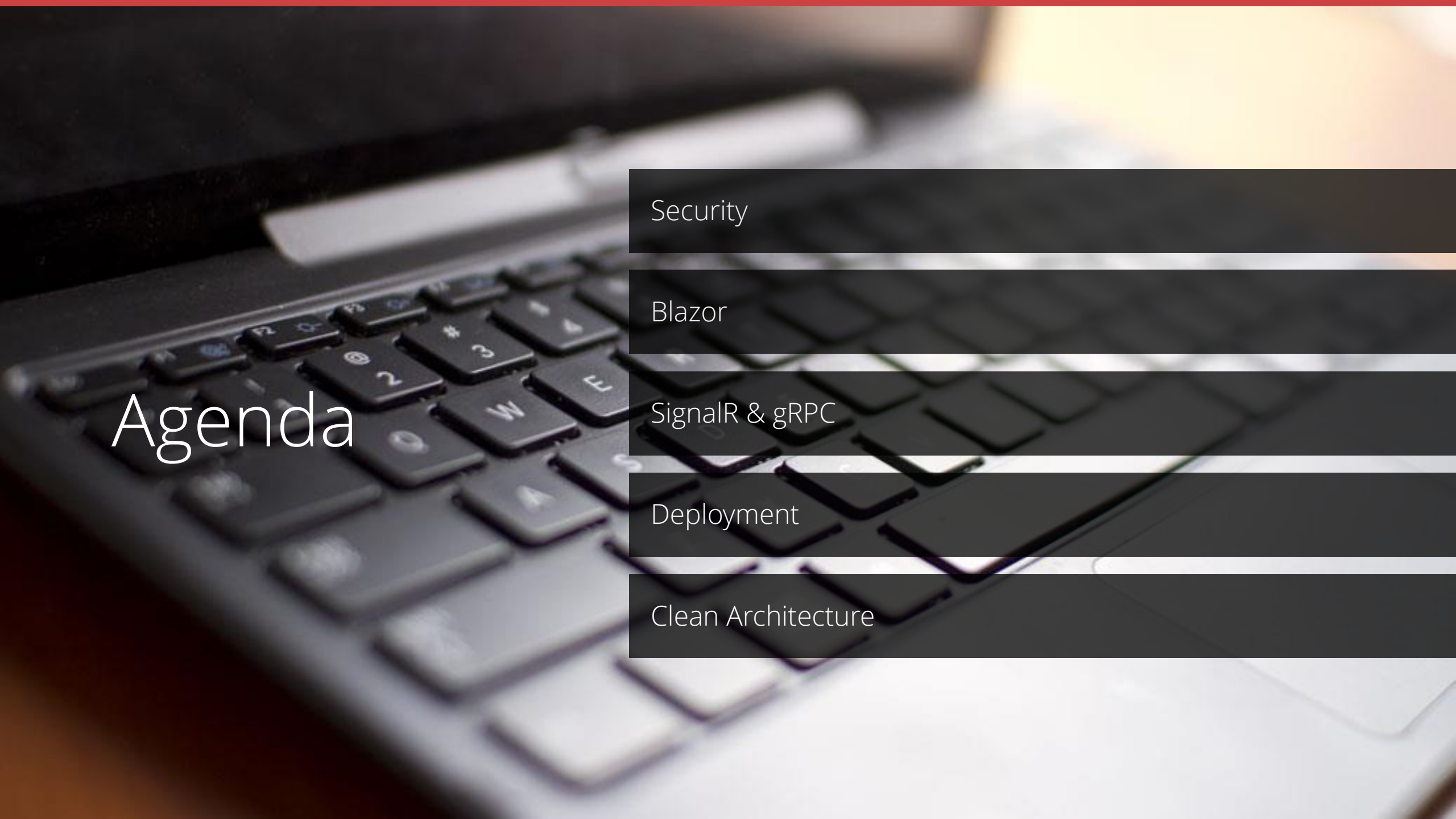
How much experience do you have with .NET?

Coffee? ☕



Agenda

- Getting Started
- Automated Testing
- Open API
- Validation
- Entity Framework Core ...



Agenda

- Security
- Blazor
- SignalR & gRPC
- Deployment
- Clean Architecture

Agenda

- Getting Started
- Automated Testing
- Open API
- Entity Framework Core
- Validation ...

Why .NET Core?

Simple answer: Azure

Cross platform

Open source (MIT License)

Optimised for server / cloud / containers

Faster

Version	Release Date	Released with
.NET Core 1.0[17]	2016-06-27	Visual Studio 2015 Update 3
.NET Core 1.1[18]	2016-11-16	Visual Studio 2017 Version 15.0
.NET Core 2.0[19]	2017-08-14	Visual Studio 2017 Version 15.3
.NET Core 2.1[20]	2018-05-30	Visual Studio 2017 Version 15.7
.NET Core 2.2[21]	2018-12-04	Visual Studio 2017 Version 15.9
.NET Core 3.0[22]	2019-09-23[13]	Visual Studio 2019 Version 16.3
.NET 5[23]	2020-11 (projected)	

Upcoming Ship Dates

Milestone	Release Date
.NET Core 2.1.x, 2.2.x, 3.0.x (servicing)	Approximately every 1-2 months or as needed (see also releases)
.NET Core 3.1	LTS (Long Term Support) release, scheduled for December 2019
.NET 5.0	Release scheduled for November 2020
.NET 6.0	LTS (Long Term Support) release, scheduled for November 2021
.NET 7.0	Release scheduled for November 2022
.NET 8.0	LTS (Long Term Support) release, scheduled for November 2023

Introducing .NET Standard

Formal specification of .NET APIs

Establishes greater uniformity in the .NET ecosystem

Implemented by .NET Core, .NET Framework, Mono, ...

bit.ly/2ys2Jvx

.NET Standard	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0	2.1
.NET Core	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0	3.0
.NET Framework [1]	4.5	4.5	4.5.1	4.6	4.6.1	4.6.1 [2]	4.6.1 [2]	4.6.1 [2]	N/A[3]
Mono	4.6	4.6	4.6	4.6	4.6	4.6	4.6	5.4	6.4
Xamarin.iOS	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.14	12.16
Xamarin.Mac	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.8	5.16
Xamarin.Android	7.0	7.0	7.0	7.0	7.0	7.0	7.0	8.0	10.0
Universal Windows Platform	10.0	10.0	10.0	10.0	10.0	10.0.16299	10.0.16299	10.0.16299	TBD
Unity	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	TBD

.NET Standard	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0	2.1
.NET Core	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0	3.0
.NET Framework [1]	4.5	4.5	4.5.1	4.6	4.6.1	4.6.1 [2]	4.6.1 [2]	4.6.1 [2]	N/A[3]
Mono	4.6	4.6	4.6	4.6	4.6	4.6	4.6	5.4	6.4
Xamarin.iOS	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.14	12.16
Xamarin.Mac	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.8	5.16
Xamarin.Android	7.0	7.0	7.0	7.0	7.0	7.0	7.0	8.0	10.0
Universal Windows Platform	10.0	10.0	10.0	10.0	10.0	10.0.16299	10.0.16299	10.0.16299	TBD
Unity	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	TBD

nuget Packages Upload Statistics Documentation Downloads Blog Sign

Search for packages...

Newtonsoft.Json 12.0.1-beta1

Json.NET

Json.NET is a popular high-performance JSON framework for .NET

ⓘ This is a prerelease version of Newtonsoft.Json.

Requires NuGet 2.12 or higher.

Package Manager | .NET CLI | Paket CLI

```
PM> Install-Package Newtonsoft.Json -Version 12.0.1-beta1
```

Info

last updated 22 days ago

Project Site

Source Code

License Info

Contact owners

Report

Download Package (3.24 MB)

Statistics

163,264,397 total downloads

20,988 downloads of latest

> Dependencies

Dependencies

.NETFramework 2.0

No dependencies.

.NETFramework 3.5

No dependencies.

.NETFramework 4.0

No dependencies.

.NETFramework 4.5

No dependencies.

.NETStandard 1.0

Microsoft.CSharp (>= 4.3.0)
NETStandard.Library (>= 1.6.1)
System.ComponentModel.TypeConverter (>= 4.3.0)
System.Runtime.Serialization.Primitives (>= 4.3.0)

.NETStandard 1.3

Microsoft.CSharp (>= 4.3.0)
NETStandard.Library (>= 1.6.1)
System.ComponentModel.TypeConverter (>= 4.3.0)
System.Runtime.Serialization.Formatters (>= 4.3.0)
System.Runtime.Serialization.Primitives (>= 4.3.0)
System.Xml.XmlDocument (>= 4.3.0)

.NETStandard 2.0

No dependencies.

163,264,397 total downloads

20,988 downloads of latest version

56,787 downloads per day (avg)

View full stats

Owners


jamesnk

newtonsoft

Authors

James Newton-King

Copyright

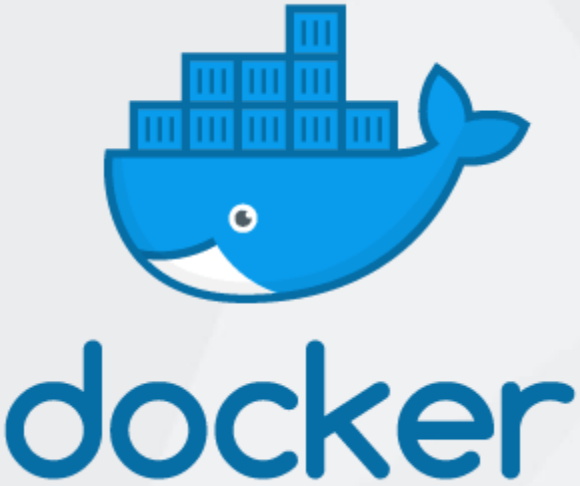
Copyright © James Newton-King 2008

Tags

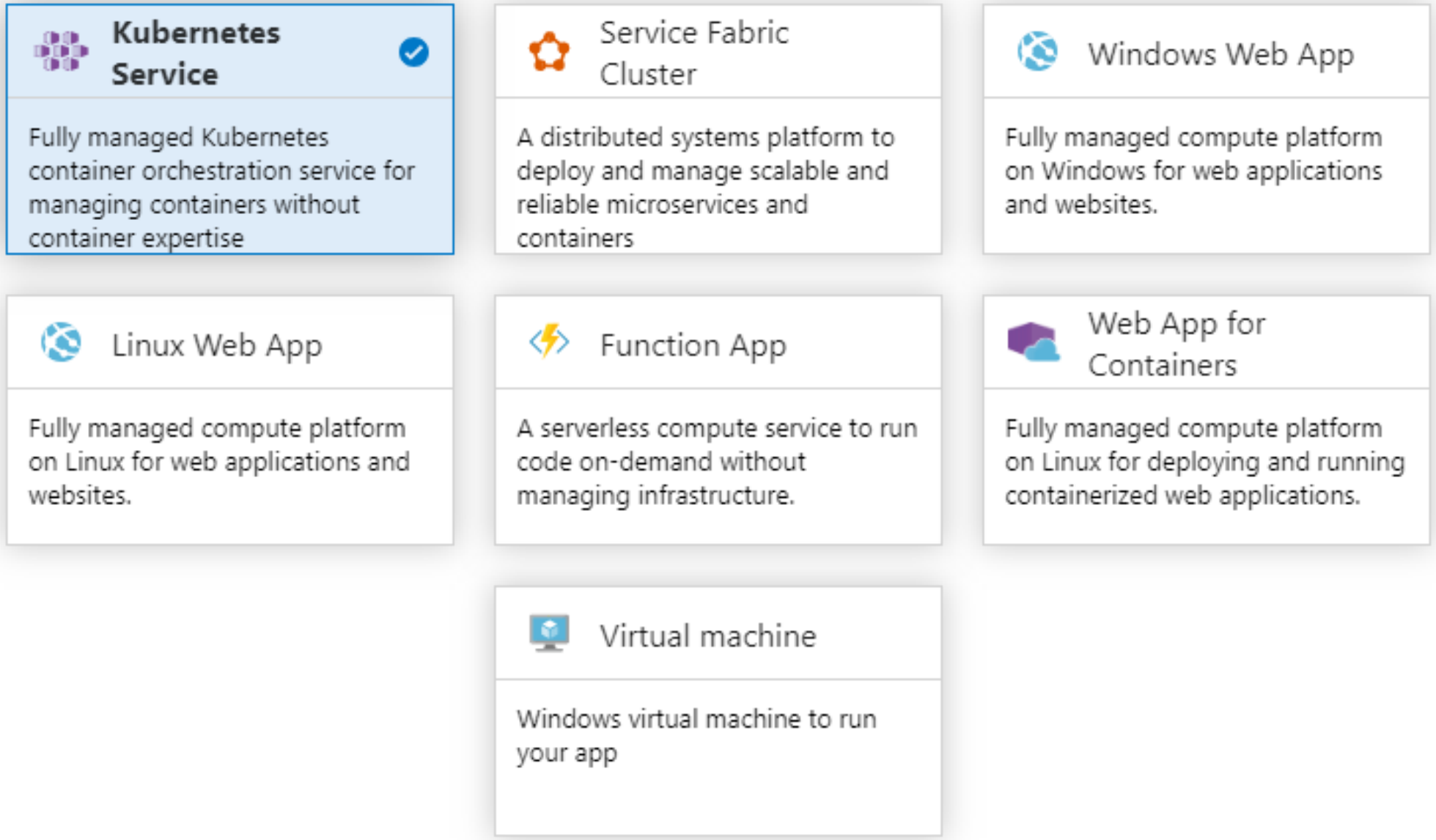
json

Run Anywhere!

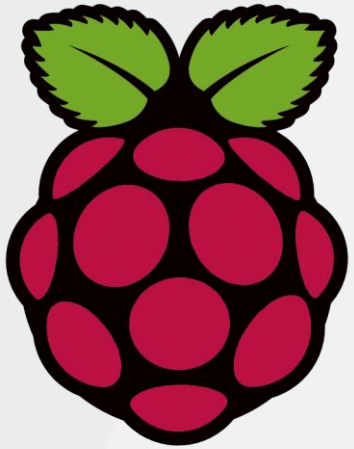
Run Anywhere!

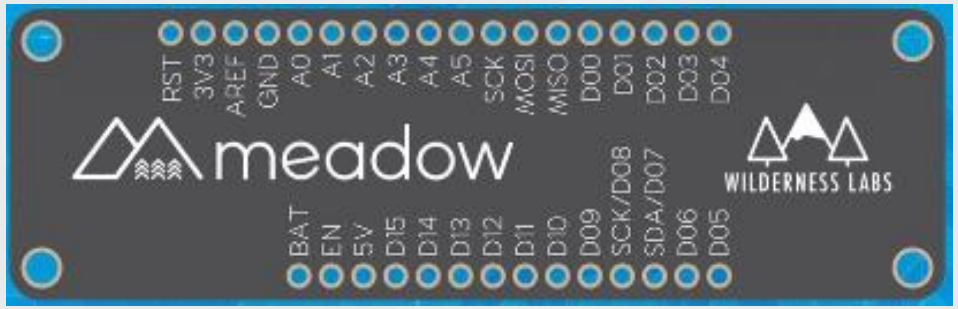


Select an Azure service to deploy the application



Run Anywhere!





Run Anywhere!

WEBASSEMBLY

Key Points

- .NET Core is the future of .Net
- Cross-platform & open source
- Runs in many more places
- .NET Standard specifies common APIs across runtimes

Demo: The Obligatory Hello World!

Create a console app using the .NET Core CLI

dotnet new

dotnet restore / build / run

dotnet publish

Demo: ASP .NET Core

.csproj

Program.cs

Configuration

Startup.cs

Problem:

There is always one team member who has their local database configured differently.

Solution:

Add an optional appsettings.local.json

```
1 reference | Jason Taylor, 166 days ago | 2 authors, 4 changes | 1 work item | 0 exceptions
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    new WebHostBuilder()
        .UseKestrel()
        .UseContentRoot(Directory.GetCurrentDirectory())
        .ConfigureAppConfiguration((hostingContext, config) =>
        {
            var env = hostingContext.HostingEnvironment;
            config.AddJsonFile(path: "appsettings.json", optional: true, reloadOnChange: true)
                  .AddJsonFile(path: $"appsettings.Local.json", optional: true, reloadOnChange: true)
                  .AddJsonFile(path: $"appsettings.{env.EnvironmentName}.json", optional: true, reloadOnChange: true);
            config.AddEnvironmentVariables();
        })
```

Demo: Middleware and Routing

Creating Middleware

Endpoint Routing

Built-in Dependency Injection

ASP.NET Core supports and leverages dependency injection

The default implementation provides a minimal feature set and services basic needs

Developers can integrate the built-in container with their preferred container

Libraries will often register themselves

Service Lifetimes

Transient – Created each time they're requested

Scoped – Created once per request (Http)

Singleton – Created once when they're requested, subsequent requests use the same instance

Demo: Dependency Injection

Logging

Built-in support for structured logging

Supports numerous providers

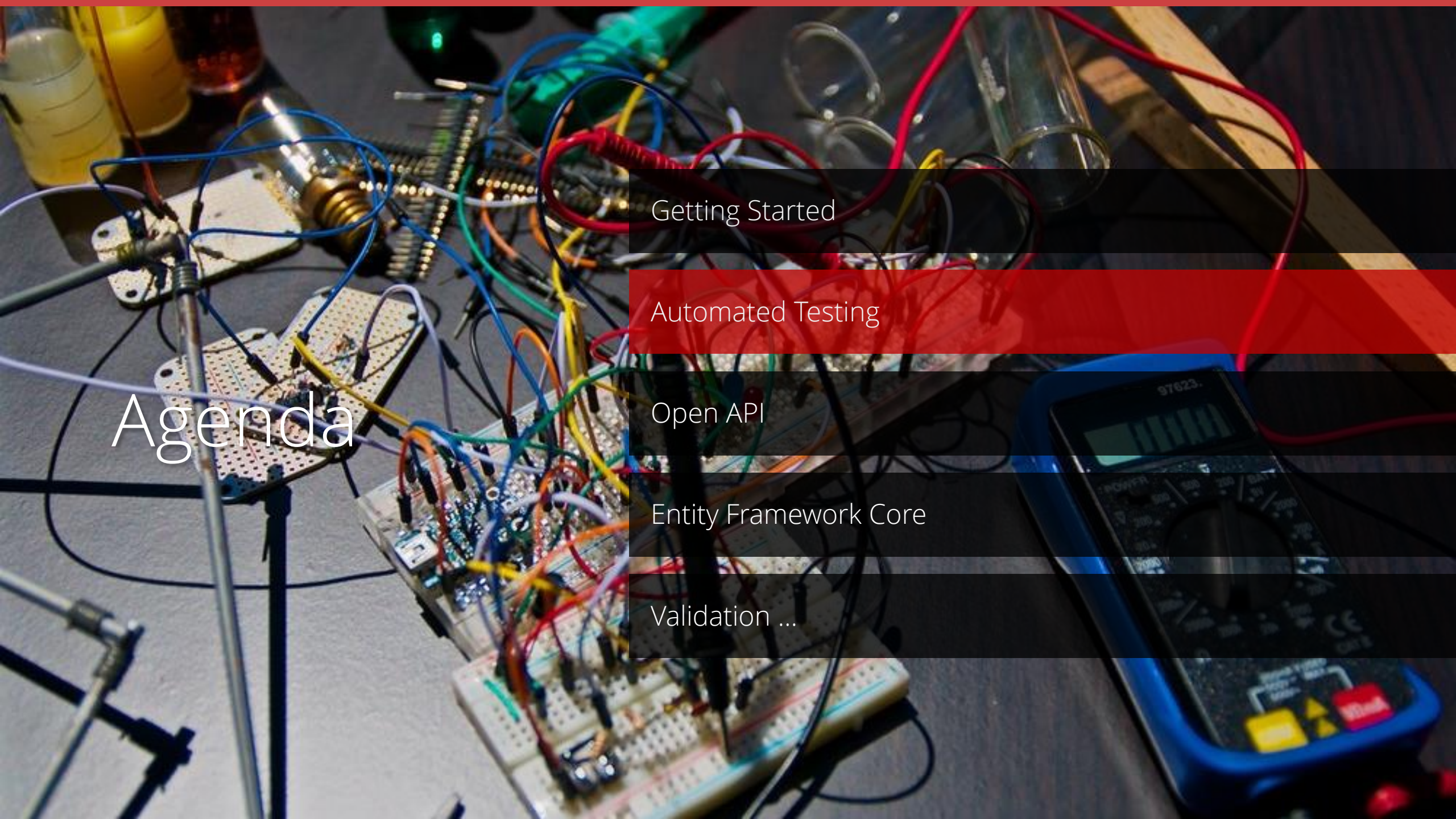
Logging providers take some action on logged data, e.g. display on console, store in event log

Supports third-party providers such as NLog or Seq

Demo: Logging with SEQ

Key Points

- Web apps are just console apps with more libraries
- Break free from IIS
- Dependency Injection everywhere
- Flexible configuration
- Request handling pipeline
- Structured Logging



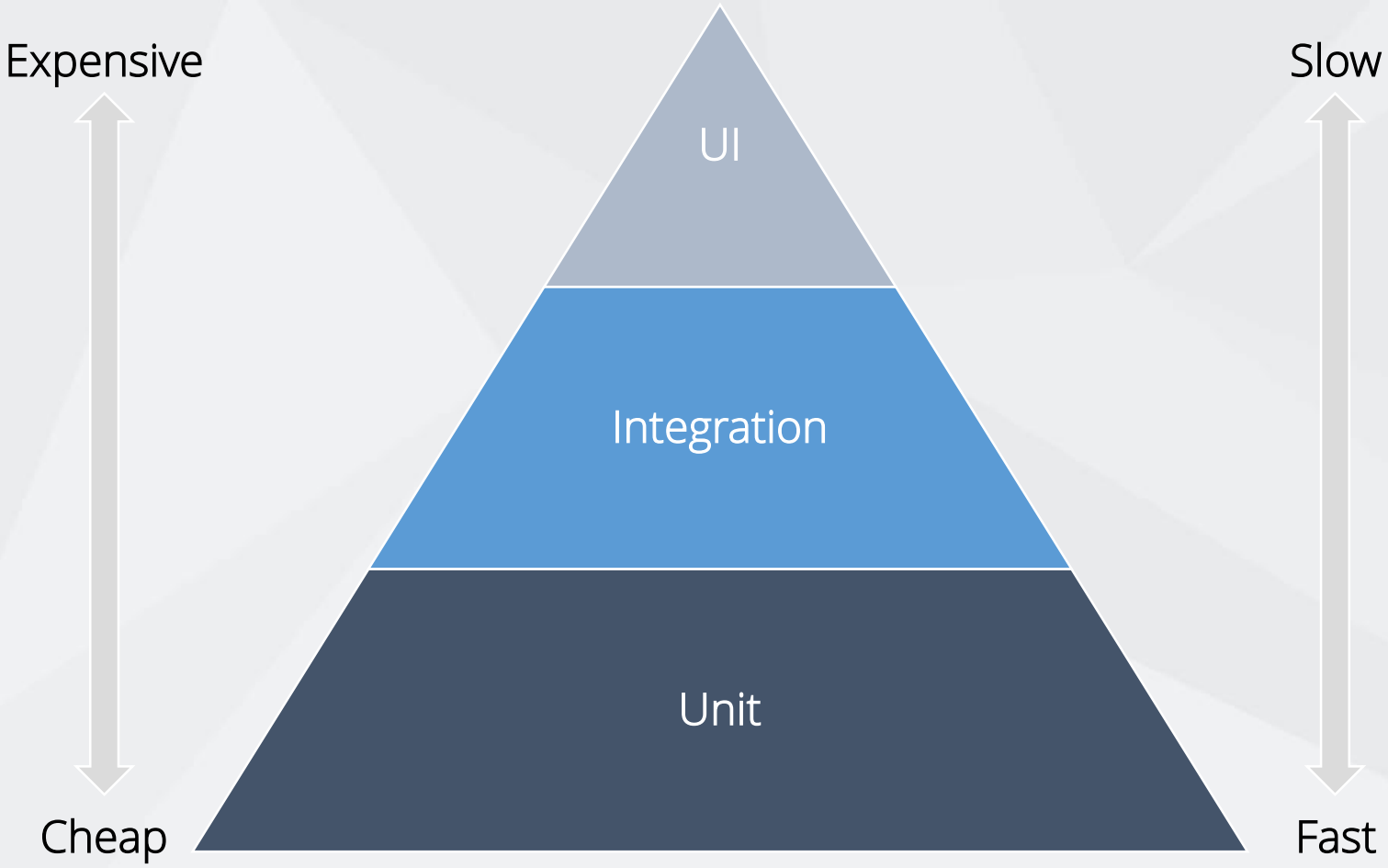
Agenda

- Getting Started
- Automated Testing
- Open API
- Entity Framework Core
- Validation ...

Guess Who?

Testing Frameworks

Testing Pyramid



Testing Tools

xUnit + Shoudly

EF Core InMemory

Moq

Live Unit Testing, .NET Watch

Selenium or ~~Coded UI~~

Subcutaneous Tests

A test that operates just below the UI, e.g. API layer

Tests the basic inputs and outputs of the system

Ideal if you keep logic out of the UI (you should)

Reduces the need for UI tests

Easy to write, easy to maintain

Test-Driven Development (TDD)

A workflow for writing test first unit tests

The cycle is:

1. Add a failing test
2. Run all tests and see the new one fail (Red)
3. Write some code
4. Run all tests again and see the new one pass (Green)
5. Refactor changed code (Refactor)

Why is it important to write a failing test first?

Demo: .NET Core CLI

Creating and running tests using the .NET Core CLI

Demo: Unit Testing Logic

Getting started with xUnit, Shouldly, EF Core InMemory

Recommended Resources

Unit Testing in .NET Core (bit.ly/testing-dotnet-core)

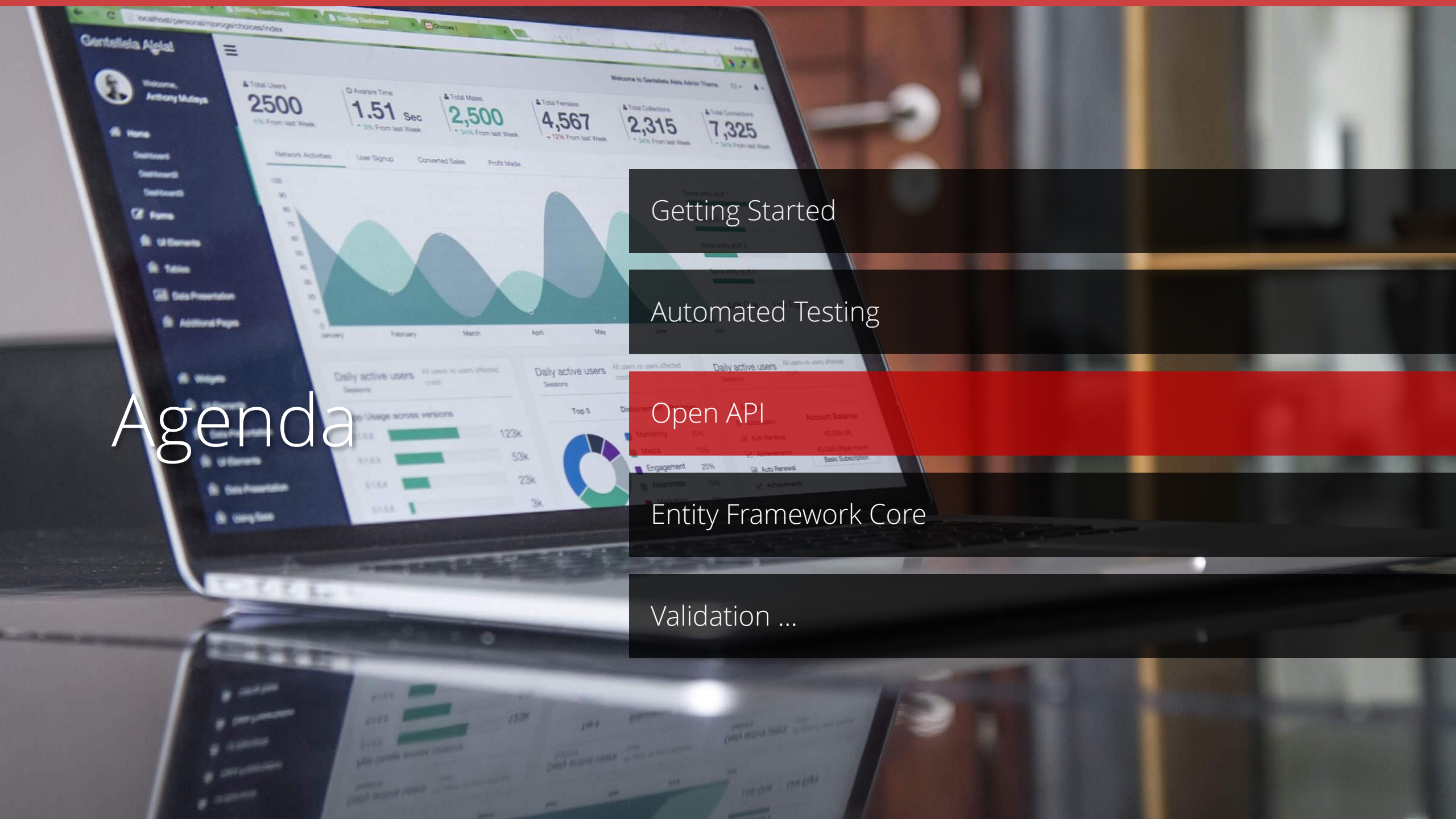
The Bowling Game Kata (bit.ly/bowling-game-kata)

Test-Driven Development (bit.ly/tdd-beck)

The Art of Unit Testing (bit.ly/aut-osherove)

Key Points

- Write your tests first
- Focus on writing unit tests
- Integration tests, if cannot be unit tested
- UI tests, if cannot be integration tested
- Use Shouldly for readable assertions
- Use EF Core InMemory Provider for easier testing



Agenda

- Getting Started
- Automated Testing
- Open API
- Entity Framework Core
- Validation ...

Open API

Formally known as Swagger

Used to describe the capabilities of Web APIs

Allows both humans and computers to understand a service

Useful for generating documentation and client code

Work with tools such as Swagger, NSwag & Swashbuckle

Single Page Applications

ASP.NET Core has built-in support for SPA

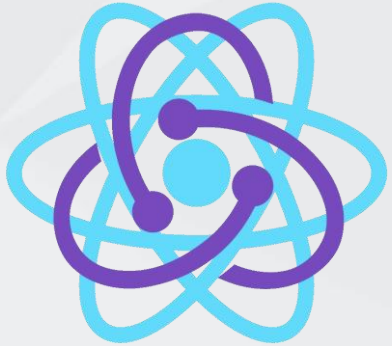
Supports ~~all~~ major client-side frameworks

Cross-platform Windows, Mac, or Linux

SPA with .NET Core 3.0

Angular

React

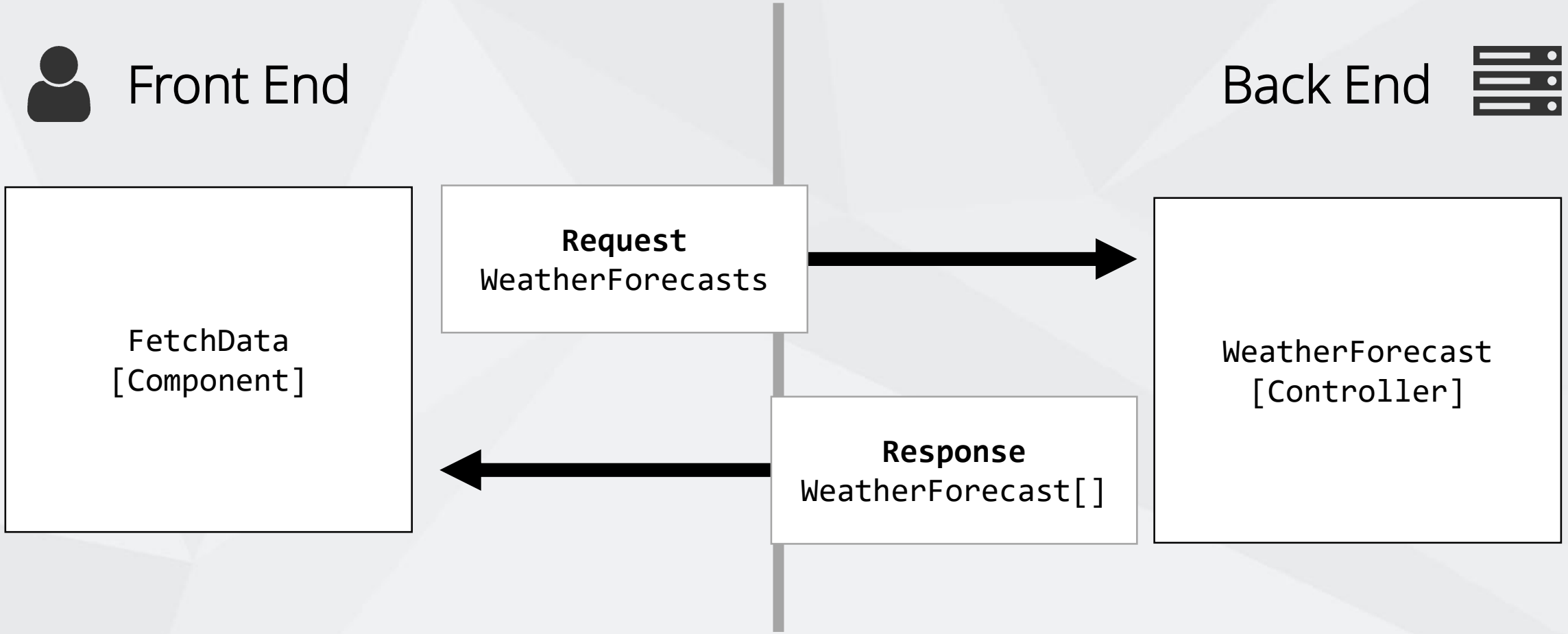


React + Redux

Demo: Create a SPA

Create from .NET Core CLI

Fetch Data Sample



```
@Component({
  selector: 'app-fetch-data',
  templateUrl: './fetch-data.component.html'
})
export class FetchDataComponent {
  public forecasts: WeatherForecast[];

  constructor(http: HttpClient, @Inject('BASE_URL') baseUrl: string) {
    http.get<WeatherForecast[]>(baseUrl + 'api/SampleData/WeatherForecasts')
      .subscribe(result => {
        this.forecasts = result;
      }, error => console.error(error));
  }
}
```

```
interface WeatherForecast {
  dateFormatted: string;
  temperatureC: number;
  temperatureF: number;
  summary: string;
}
```

- ✖ Requires knowledge of API
- ✖ Hard-coded URLs
- ✖ Duplication of back end types
- ✖ Breaking changes are hard to detect

Demo

Implementing Open API

```
import { Component } from '@angular/core';
import { WeatherForecast, SampleDataClient } from '../awesome-spa-api.generated';

@Component({
  selector: 'app-fetch-data',
  templateUrl: './fetch-data.component.html'
})
export class FetchDataComponent {
  public forecasts: WeatherForecast[];

  constructor(client: SampleDataClient) {
    client.weatherForecasts().subscribe(result =>
      this.forecasts = result,
      error => console.error(error));
  }
}
```

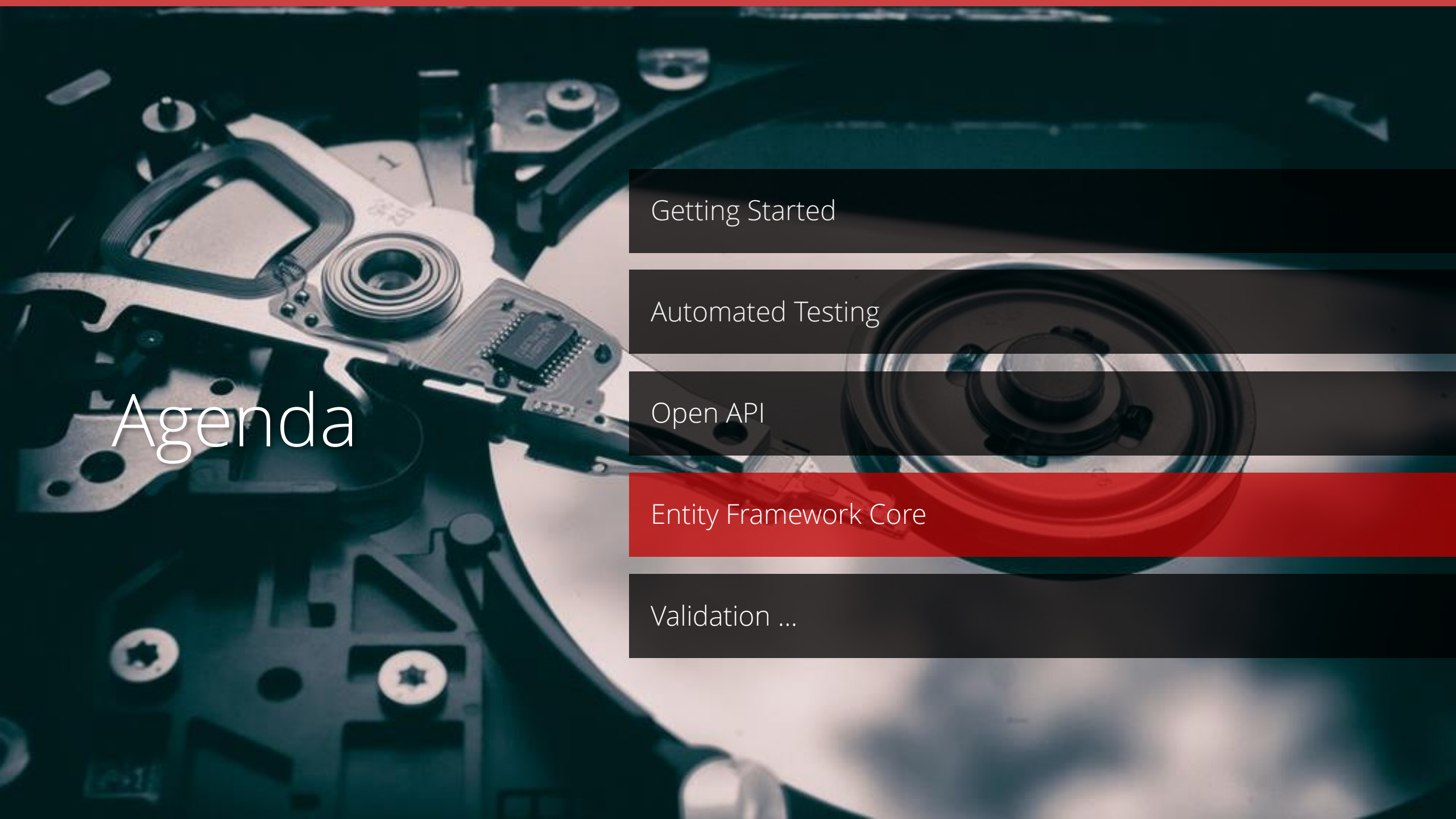
- ✔ Can explore API endpoints
- ✔ No hard-coded URLs
- ✔ No duplication of back end types
- ✔ Breaking changes are easy to detect

Recommend Resources

Open API initiative	openapis.org
NSwag + ASP.NET Core	bit.ly/aspnetcore-nswag
NSwag docs	github.com/RSuter/NSwag

Key Points

- Generate Open API specifications
- Documentation for developers
- UI to interact with API endpoints
- Automatic generation of API clients
- Use NSwag (best end-to-end experience)



Agenda

- Getting Started
- Automated Testing
- Open API
- Entity Framework Core
- Validation ...

Entity Framework Core

10/27/2016 • 2 minutes to read •  +2

Entity Framework (EF) Core is a lightweight, extensible, open source and cross-platform version of the popular Entity Framework data access technology.

EF Core can serve as an object-relational mapper (O/RM), enabling .NET developers to work with a database using .NET objects, and eliminating the need for most of the data-access code they usually need to write.

EF Core supports many database engines, see Database Providers for details.

https://docs.microsoft.com/en-us/ef/core/

ORM – Impedance Mismatch

Id: int	Name: varchar(50)

ORM – Impedance Mismatch

Id: int	Name: varchar(50)
1	Brendan
2	Jason

ORM – Impedance Mismatch

Id: int	Name: varchar(50)
1	Brendan
2	Jason

Id: int	FK: Person(Id)	Skills
1	1	EF Core
2	2	Clean Code

ORM – Impedance Mismatch

Id: int	Name: varchar(50)
1	Brendan
2	Jason

Id: int	FK: Person(Id)	Skills
1	1	EF Core
2	2	Clean Code

ORM – Impedance Mismatch

Id: int	Name: varchar(50)
1	Brendan
2	Jason

One-To-One Relationship

Id: int	FK: Person(Id)	Skills
1	1	EF Core
2	2	Clean Code

ORM – Impedance Mismatch

Id: int	Name: varchar(50)
1	Brendan
2	Jason

One-To-Many Relationship

Id: int	FK: Person(Id)	Skills
1	1	EF Core
2	2	Clean Code
3	2	VueJs

ORM – Impedance Mismatch

Id: int	Name: varchar(50)
1	Brendan
2	Jason

Many-To-Many Relationship

FK: Person(Id)	FK: Skill(Id)
1	1
2	1

Id: int	Skills
1	EF Core
2	Clean Code
3	VueJs

ORM – Impedance Mismatch

```
public class Person
{
    public int id { get; set; }
    public string Name { get; set; }
}
```

```
public class Skill
{
    public int id { get; set; }
    public string Name { get; set; }
}
```

ORM – Impedance Mismatch

One-To-One Relationship

```
public class Person
{
    public int id { get; set; }
    public string Name { get; set; }
    public Skill Skill { get; set; }
}
```

```
public class Skill
{
    public int id { get; set; }
    public string Name { get; set; }
}
```

ORM – Impedance Mismatch

One-To-One Relationship

```
public class Person
{
    public int id { get; set; }
    public string Name { get; set; }
}
```

```
public class Skill
{
    public int id { get; set; }
    public string Name { get; set; }
    public Person Person { get; set; }
}
```

ORM – Impedance Mismatch

Bi-Directional One-To-One Relationship

```
public class Person
{
    public int id { get; set; }
    public string Name { get; set; }
    public Skill Skill { get; set; }
}
```

```
public class Skill
{
    public int id { get; set; }
    public string Name { get; set; }
    public Person Person { get; set; }
}
```

ORM – Impedance Mismatch

One-To-Many Relationship

```
public class Person
{
    public int id { get; set; }
    public string Name { get; set; }
    public ICollection<Skill> Skills { get; set; }
        = new List<Skill>();
}
```

```
public class Skill
{
    public int id { get; set; }
    public string Name { get; set; }
    public Person Person { get; set; }
}
```

ORM – Impedance Mismatch

One-To-Many Relationship

```
public class Person
{
    public int id { get; set; }
    public string Name { get; set; }
    public ICollection<Skill> Skills { get; set; }
        = new List<Skill>();
}
```

```
public class Skill
{
    public int id { get; set; }
    public string Name { get; set; }
    public Person Person { get; set; }
}
```

ORM – Impedance Mismatch

Many-To-Many Relationship

```
public class Person
{
    public int id { get; set; }
    public string Name { get; set; }
    public ICollection<Skill> Skills { get; set; }
        = new List<Skill>();
}
```

```
public class Skill
{
    public int id { get; set; }
    public string Name { get; set; }
    public ICollection<Person> Persons { get; set; }
        = new List<Person>();
}
```

Object Oriented Concepts

Data plus Behaviour

Classes, Inheritance, Polymorphism, Encapsulation

Pointers and Collections to model relationships

OO provides a richer modelling toolkit

The DBContext

Most important class in Entity Framework

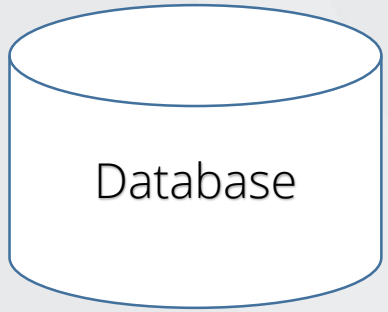
Interface to data via DBSet

Provides a Unit Of Work, Tracks Changes

Sorts out relationships between Added, Modified and Unchanged Entities when you save

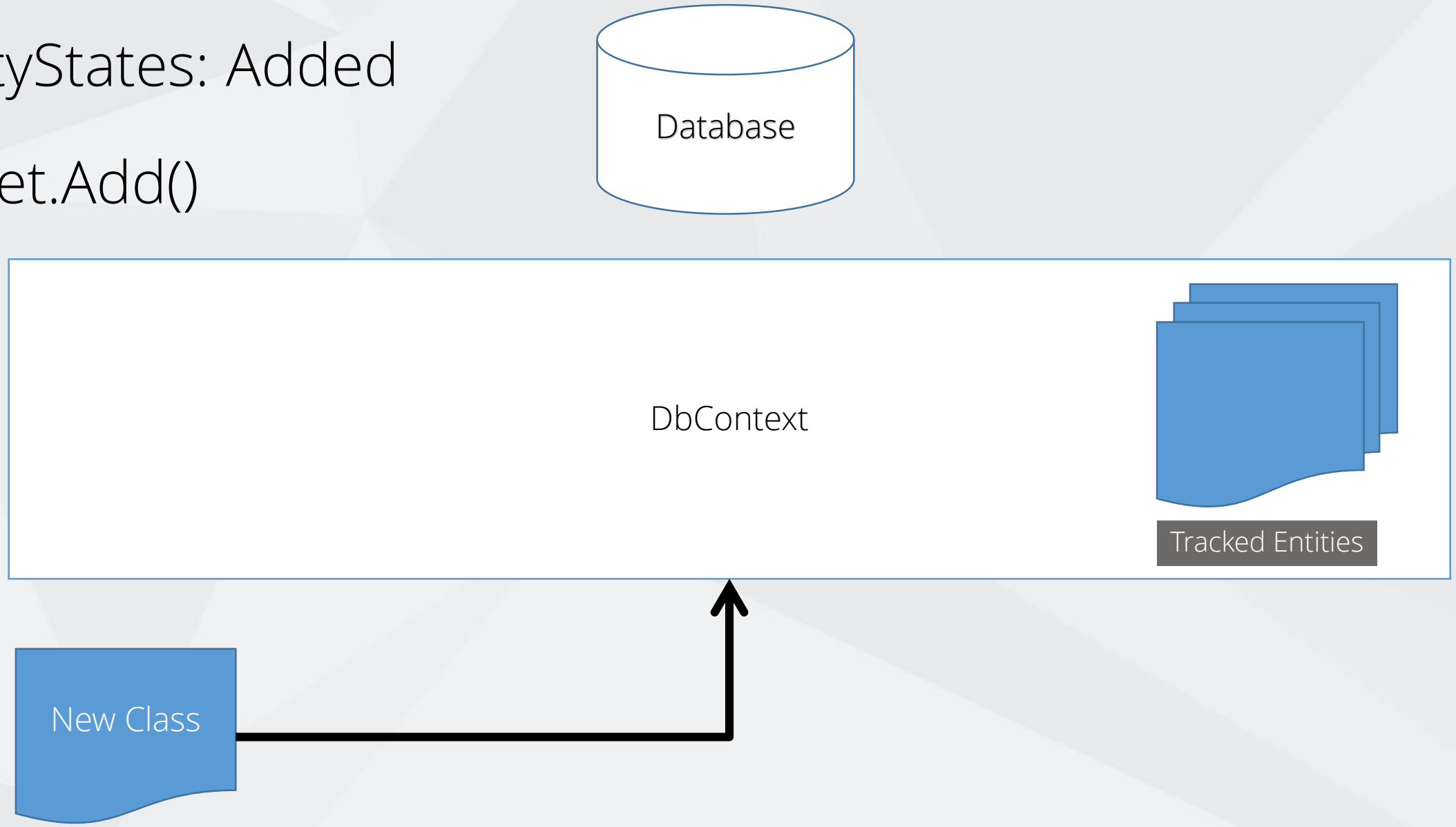
Important to manage lifecycle of DBContext

EntityStates



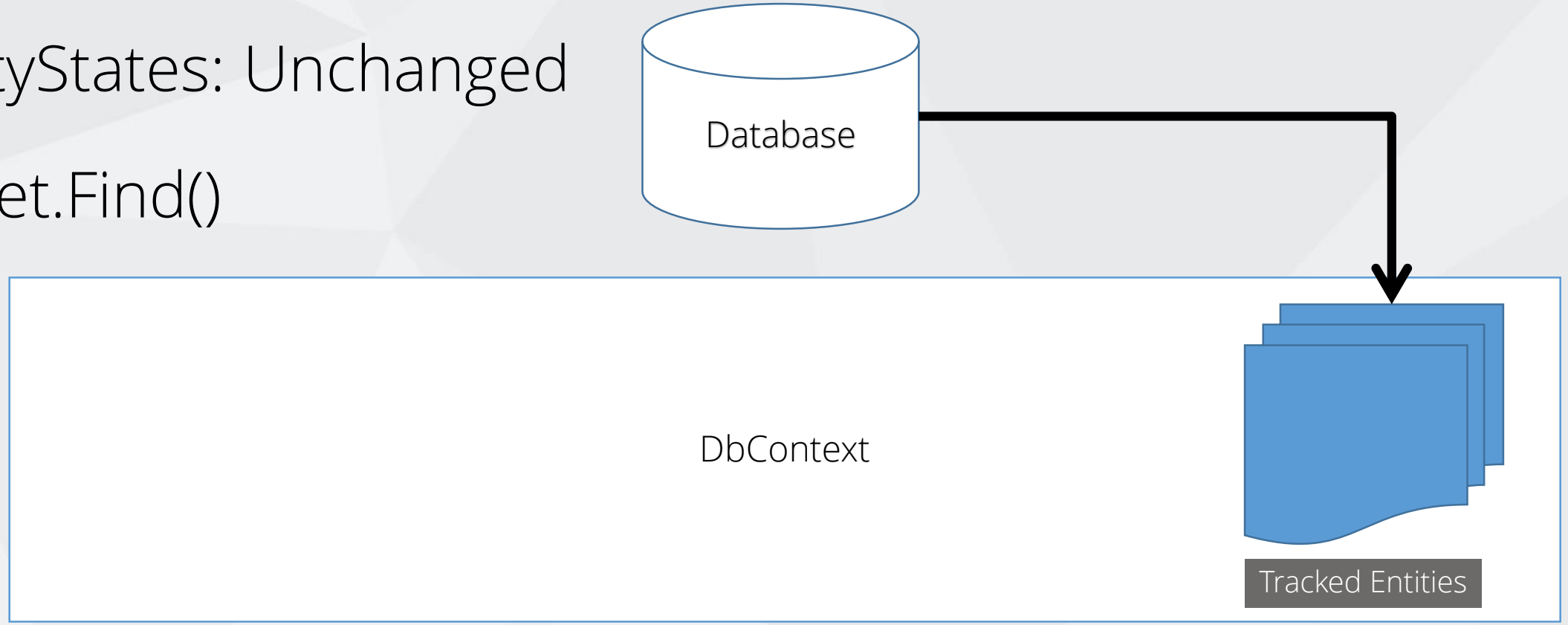
EntityStates: Added

DbSet.Add()

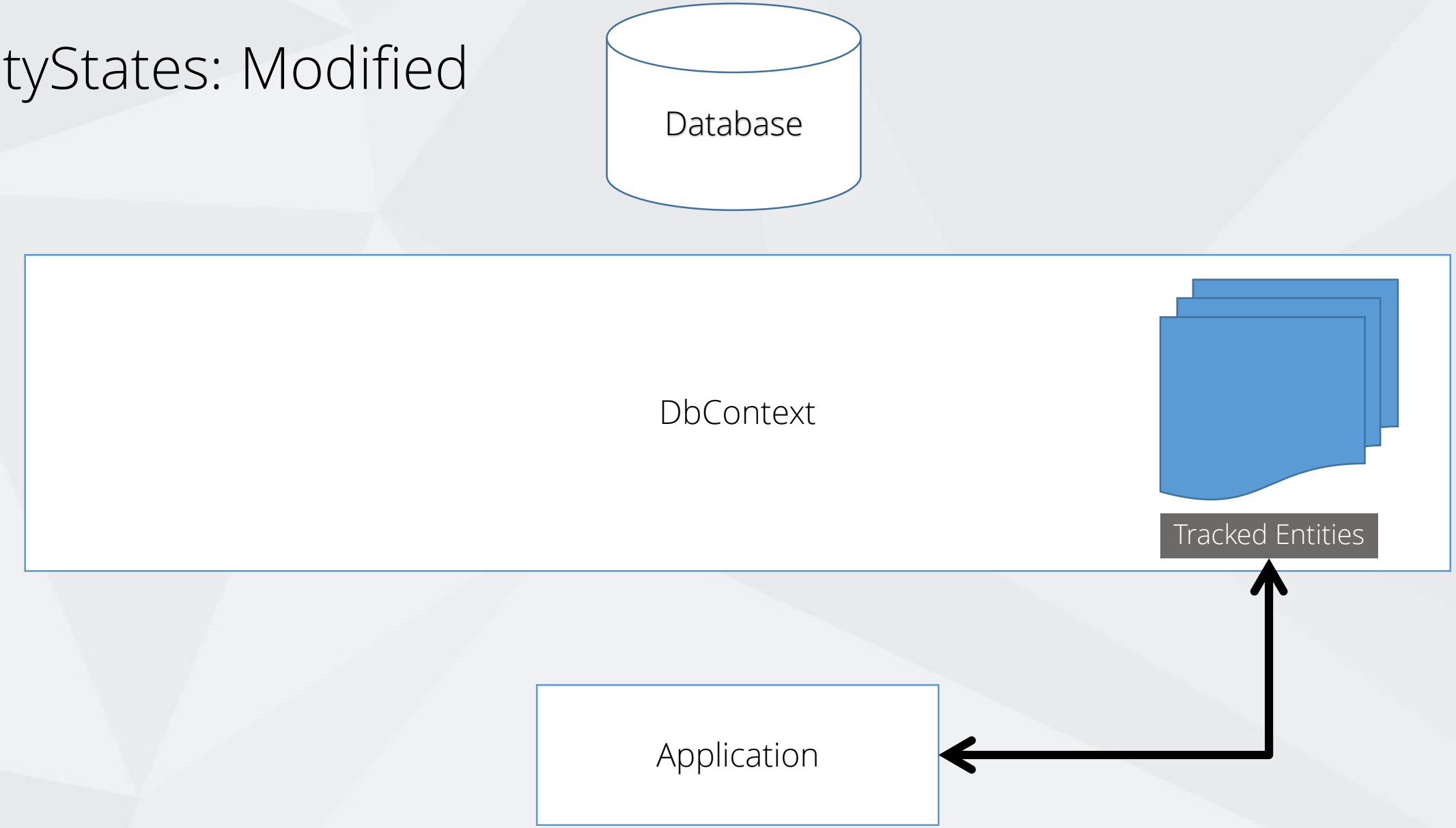


EntityStates: Unchanged

DbSet.Find()

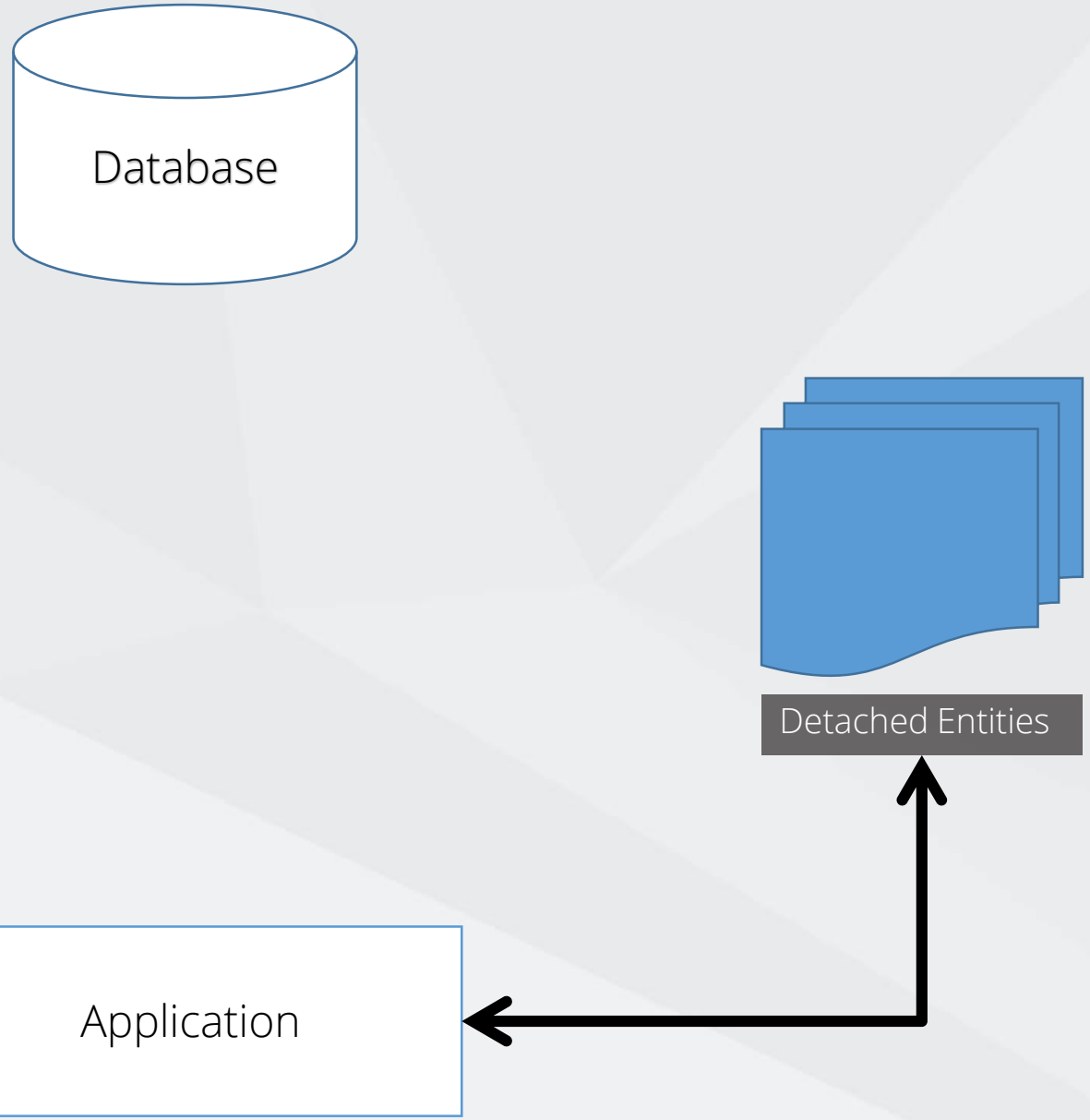


EntityStates: Modified

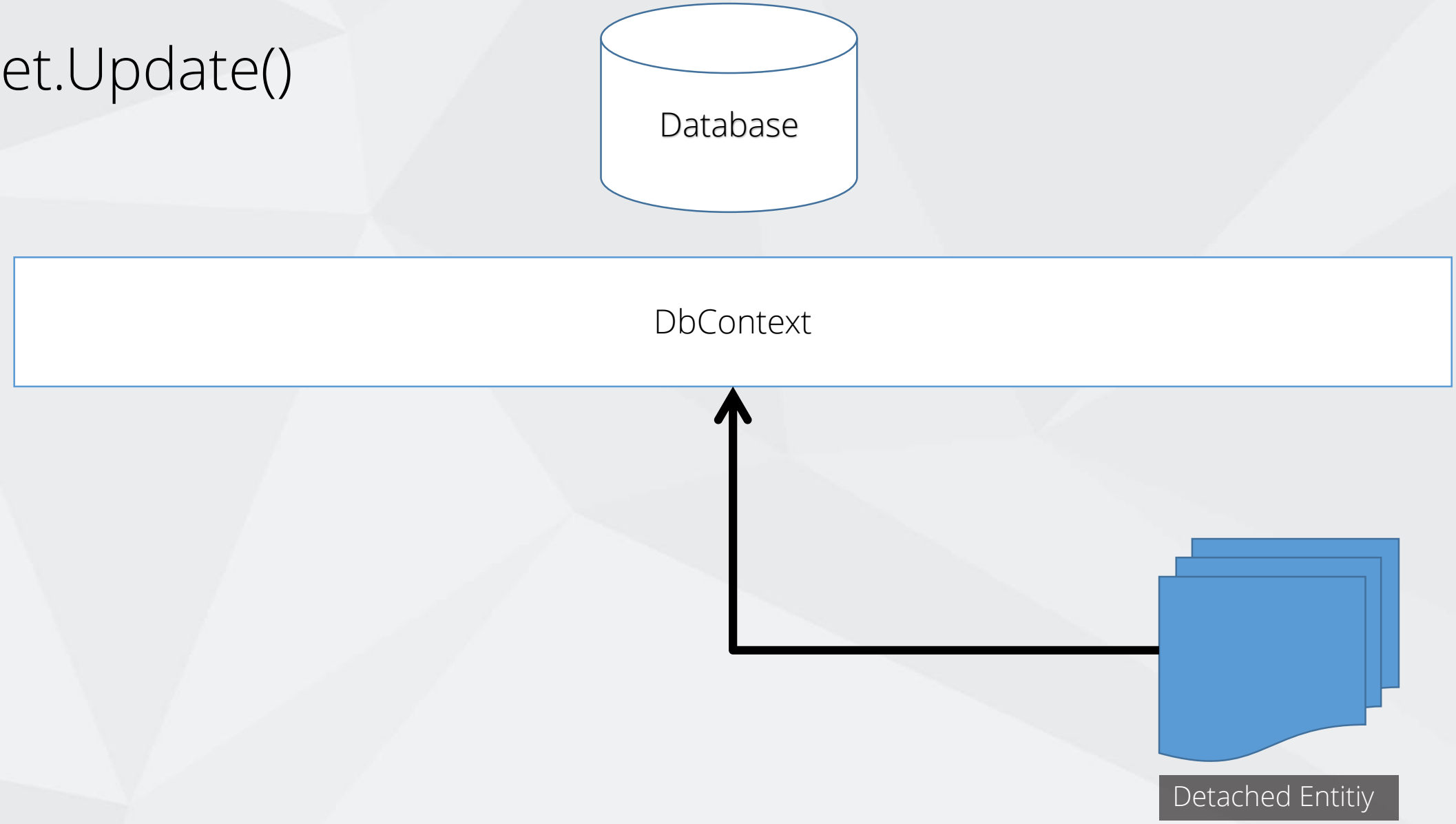


EntityStates: Detached

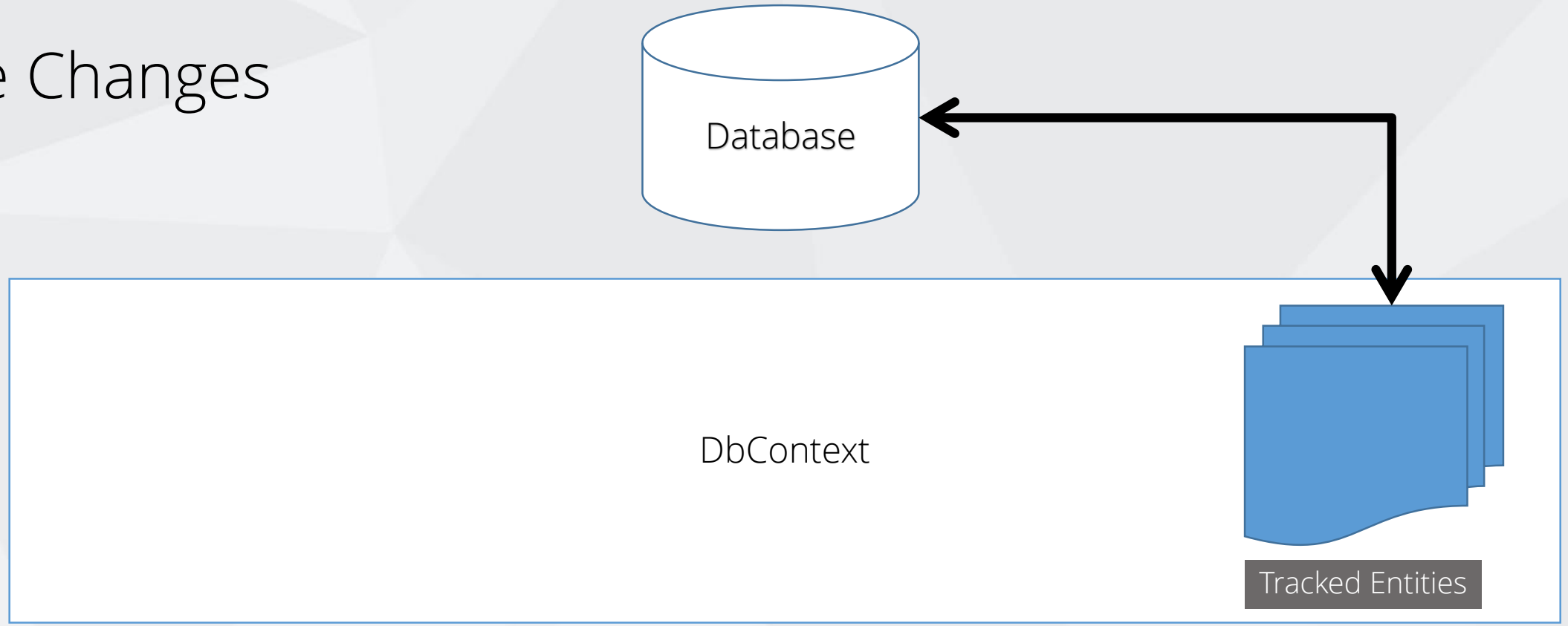
Lazy Loading breaks here!



DbSet.Update()



Save Changes



Define Some Entities

```
public class Customer
{
    public Customer()
    {
        Orders = new HashSet<Order>();
    }

    public string CustomerId { get; set; }
    public string CompanyName { get; set; }
    public string ContactName { get; set; }
    public string ContactTitle { get; set; }
    public string Address { get; set; }
    public string City { get; set; }
    public string Region { get; set; }
    public string PostalCode { get; set; }
    public string Country { get; set; }
    public string Phone { get; set; }
    public string Fax { get; set; }
```

Create a DBContext

```
public class NorthwindDbContext: DbContext, INorthwindDbContext
{

    public NorthwindDbContext(DbContextOptions options)
        : base(options)
    {
    }

    public DbSet<Customer> Customers { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.ApplyConfigurationsFromAssembly(typeof(NorthwindDbContext).Assembly);
```

Add Configuration In Startup

```
// This method gets called by the runtime. Use this method to add services to the container.
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers();

    services.AddDbContext<NorthwindDbContext>(opts =>
        opts.UseSqlServer(Configuration.GetConnectionString("NorthwindDbContext"),
            b => b.MigrationsAssembly("Northwind.EF.Persistence.MSSQL")));
```

Demo: Migrations

Publishing schema changes.

Dependency Injection

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

    public ScaffoldedCustomersController(NorthwindDbContext context)
    {
        _context = context;
    }

    // GET: api/ScaffoldedCustomers
    [HttpGet]
    public async Task<ActionResult<IEnumerable<Customer>>> GetCustomers()
    {
        return await _context.Customers.ToListAsync();
    }
```

But....

Incoming request

```
// GET: api/ScaffoldedCustomers/5
[HttpGet("{id}")]
public async Task<ActionResult<Customer>> GetCustomer(string id)
{
    var customer = await _context.Customers.FindAsync(id);
    if (customer == null)
    {
        return NotFound();
    }
    return customer;
}
```

Database layer

But....

```
// GET: api/ScaffoldedCustomers/5
[HttpGet("{id}")]
public async Task<ActionResult<Customer>> GetCustomer(string id)
{
    var customer = await _context.Customers.FindAsync(id);
    if (customer == null)
    {
        return NotFound();
    }
    return customer;
}
```

Should we send everything in the DB to the client?

But....

```
// GET: api/ScaffoldedCustomers/5
[HttpGet("{id}")]
public async Task<ActionResult<Customer>> GetCustomer(string id)
{
    var customer = await _context.Customers.FindAsync(id);
    if (customer == null)
    {
        return NotFound();
    }
    return customer;
}
```

JSON Seriallization does not like bidirectional relationships!

But....

```
// PUT: api/ScaffoldedCustomers/5
[HttpPut("{id}")]
public async Task<IActionResult> PutCustomer(string id, Customer customer)
{
    if (id != customer.CustomerId)
    {
        return BadRequest();
    }

    _context.Entry(customer).State = EntityState.Modified;

    try
    {
        await _context.SaveChangesAsync();
    }
```

CQRS

Command Query Responsibility Segregation

Separate reads (queries) from writes (commands)

Single Responsibility Principle

Easy to add new features, just add a new query or command

Easy to maintain, changes only affect one command or query

Can be used with repositories

Revisiting CQRS

~~Event Sourcing~~

~~Separate data stores for read and writes~~

Clearly separated command and query operations

Separate process for reads and writes

CQRS and EF Core

Read Operations

- Select / Project into View Models
- No Change Tracking

Query Object

```
public class GetCustomerOrders : IRequest<IEnumerable<CustomerOrdersModel>>
{
    public int? PageIndex { get; set; } = 0;

    public int? PageSize { get; set; } = 5;

    public string NameSearch { get; set; }

}
```

Query Handler

```
public async Task<IEnumerable<CustomerOrdersModel>> Handle(GetCustomerOrders re
{
    var efQuery = _context.Set<Customer>().Where(c => true);

    efQuery =  nlyFilters(efQuery, request);
    efQuery = Ap aging(efQuery, request);

    return av ry);
```

IQueryable<Customer>

Query Handler

```
public async Task<IEnumerable<CustomerOrdersModel>> Handle(GetCustomerOrders requ
{
    var efQuery = _context.Set<Customer>().Where(c => true);

    efQuery = ApplyFilters(efQuery, request);
    efQuery = ApplyPaging(efQuery, request);

    return await SimpleSelect(efQuery);
```

```
private IQueryable<Customer> ApplyPaging(IQueryable<Customer> efQuery, GetCustomerOrders request)
{
    if (request.PageIndex.HasValue && request.PageSize.HasValue)
    {
        return efQuery
            .Skip(request.PageIndex.Value * request.PageSize.Value)
            .Take(request.PageSize.Value)
            .OrderBy(c => c.CompanyName);
    }
    return efQuery;
}

private IQueryable<Customer> ApplyFilters(IQueryable<Customer> efQuery, GetCustomerOrders request)
{
    if (!string.IsNullOrWhiteSpace(request.NameSearch))
    {
        efQuery = efQuery.Where(c => c.CompanyName.Contains(request.NameSearch));
    }
    return efQuery;
}
```

```
private async Task<IEnumerable<CustomerOrdersModel>> SimpleSelect(IQueryable<Customer> efQuery)
{
    return await efQuery
        .Select(c => new CustomerOrdersModel()
        {
            Name = c.CompanyName,
            CustomerId = c.CustomerId,
            CountOrders = c.Orders.Count(),
            Orders = c.Orders.Select(o => new CustomerOrderModel()
            {
                OrderId = o.OrderId,
                EmployeeId = o.EmployeeId,
                EmployeeName = o.Employee != null
                    ? o.Employee.FirstName + " " + o.Employee.LastName
                    : string.Empty,
                OrderDate = o.OrderDate
            })
        }).ToListAsync();
}
```

```
private async Task<IEnumerable<CustomerOrdersModel>> SimpleSelect(IQueryable<Customer> efQuery)
{
    return await efQuery
        .Select(c => new CustomerOrdersModel()
        {
            Name = c.CompanyName,
            CustomerId = c.CustomerId,
            CountOrders = c.Orders.Count(),
            Orders = c.Orders.Select(o => new CustomerOrderModel()
            {
                OrderId = o.OrderId,
                EmployeeId = o.EmployeeId,
                EmployeeName = o.Employee != null
                    ? o.Employee.FirstName + " " + o.Employee.LastName
                    : string.Empty,
                OrderDate = o.OrderDate
            })
        }).ToListAsync();
}
```

```
private async Task<IEnumerable<CustomerOrdersModel>> ToListSelect(IQueryable<Customer> efQuery)
{
    var list = await efQuery
        .Include(c => c.Orders).ThenInclude(o => o.Employee)
        .ToListAsync();
    return list.Select(c => new CustomerOrdersModel()
    {
        Name = c.CompanyName,
        CustomerId = c.CustomerId,
        CountOrders = c.Orders.Count(),
        Orders = c.Orders.Select(o => new CustomerOrderModel()
        {
            OrderId = o.OrderId,
```

```
private async Task<IEnumerable<CustomerOrdersModel>> SelectAnon(IQueryable<Customer> efQuery)
{
    var list = await efQuery.Select(c => new
    {
        Name = c.CompanyName,
        CustomerId = c.CustomerId,
        CountOrders = c.Orders.Count(),
        Orders = c.Orders.Select(o => new
        {
            OrderId = o.OrderId,
            EmployeeId = o.EmployeeId,
            EmployeeName = o.Employee != null
                ? o.Employee.FirstName + " " + o.Employee.LastName
                : string.Empty,
            OrderDate = o.OrderDate
        })
    }).ToListAsync();
    return list.Select(c => new CustomerOrdersModel()
    {
        Name = c.Name,
```

```
private async Task<IEnumerable<CustomerOrdersModel>> AutoMapper(IQueryable<Customer> efQuery)
{
    return await efQuery
        .ProjectTo<CustomerOrdersModel>(MapperConfig)
        .ToListAsync();
}
```

CQRS and EF Core

- Read Operations
  - Select / Project into View Models
  - No Change Tracking
- Write Operations
  - Work with entities
  - DbContext / Unit of work

```
3 references
public async Task<Unit> Handle(UpdateCustomerCommand request, CancellationToken cancellationToken)
{
    var entity = await _context.Customers
        .SingleOrDefaultAsync(c => c.CustomerId == request.Id, cancellationToken);

    if (entity == null)
    {
        throw new NotFoundException(nameof(Customer), request.Id);
    }

    entity.Address = request.Address;
    entity.City = request.City;
    entity.CompanyName = request.CompanyName;
    entity.ContactName = request.ContactName;
    entity.ContactTitle = request.ContactTitle;
    entity.Country = request.Country;
    entity.Fax = request.Fax;
    entity.Phone = request.Phone;
    entity.PostalCode = request.PostalCode;

    await _context.SaveChangesAsync(cancellationToken);

    return Unit.Value;
}
```

~~Query Types~~ – HasNoKey()

```
public class CustomersMostPurchasedProducts
{
    public string CustomerID { get; set; }

    public string CompanyName { get; set; }

    public int ProductID { get; set; }

    public string ProductName { get; set; }

    public int QuantityPurchased { get; set; }
}
```

```
modelBuilder.Entity<CustomersMostPurchasedProducts>()
    .HasNoKey()
    .ToQuery(() => Set<CustomersMostPurchasedProducts>().FromSqlRaw(@"
select
    c.CustomerID
    , c.CompanyName
    , p.ProductID
    , p.ProductName
    , qtyCounts.QuantityPurchased
    from Customers c
    inner join
        (select
            o.CustomerID
            , od.ProductID
            ,sum(od.Quantity) as QuantityPurchased
        from [Order Details] od
        inner join [Orders] o on od.OrderID = o.OrderID
        group by o.CustomerID, od.ProductID)  qtyCounts on c.CustomerID = qtyCounts.CustomerID
    inner join Products p on p.ProductID = qtyCounts.ProductID
"));
```

```
public async Task<IEnumerable<CustomersMostPurchasedViewModel>> Handle(Get
{
    return await _context.Set<CustomersMostPurchasedProducts>()
        .OrderByDescending(c => c.QuantityPurchased)
        .Skip(request.PageIndex * request.PageSize)
        .Take(request.PageSize)
        .Select(r => new CustomersMostPurchasedViewModel
        {
            CompanyName = r.CompanyName,
            CustomerId = r.CustomerID,
            ProductId = r.ProductID,
```

```
public async Task<IEnumerable<CustomersMostPurchasedViewModel>> Handle(Get
{
    return await _context.Set<CustomersMostPurchasedProducts>()
        .OrderByDescending(c => c.QuantityPurchased)
        .Skip(request.PageIndex * request.PageSize)
        .Take(request.PageSize)
        .Select(r => new CustomersMostPurchasedViewModel
        {
            CompanyName = r.CompanyName,
            CustomerId = r.CustomerID,
            ProductId = r.ProductID,
```

Key Points

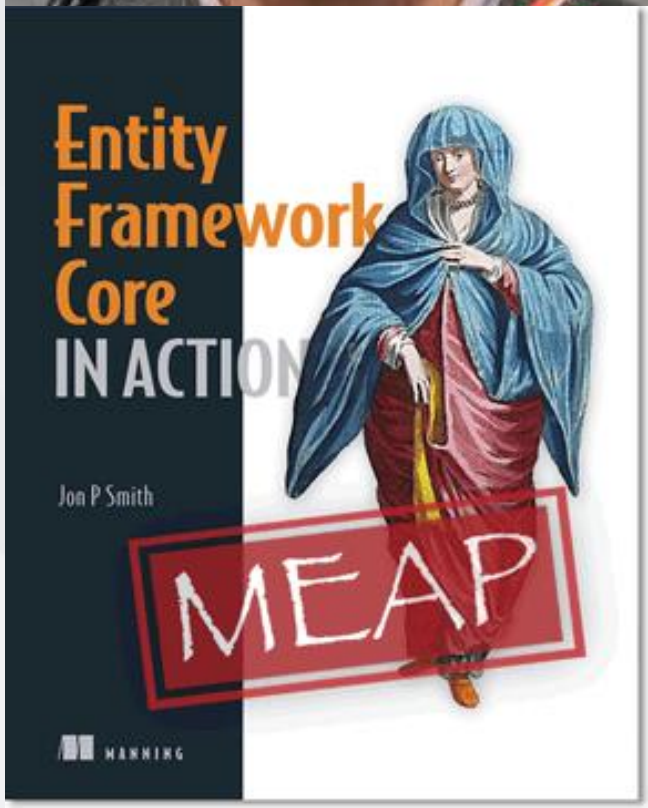
- EF Core is a 'heavyweight' ORM that tracks changes
- Understand DBContext and Entity States
- CQRS separates reads from writes
- EF can be used differently for reads and writes

John P Smith

TheReformedProgrammer.net

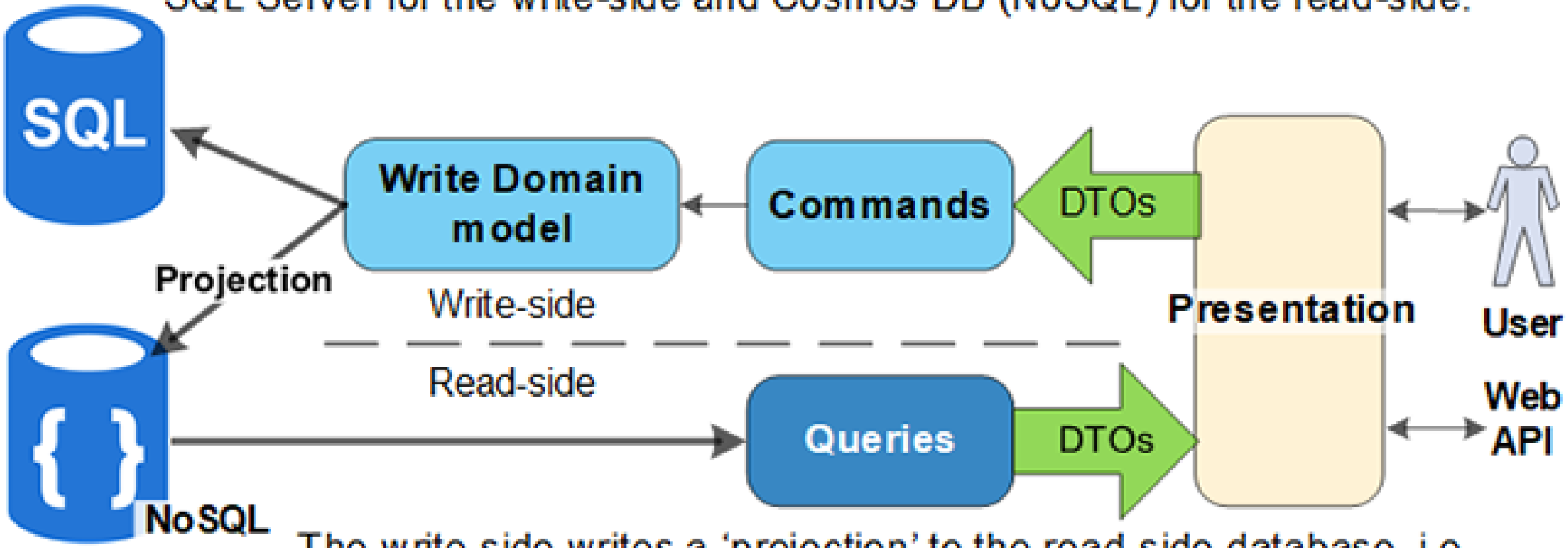
Building a robust CQRS database with EF Core and Cosmos DB

https://www.thereformedprogrammer.net/building-a-robust-cqrs-database-with-ef-core-and-cosmos-db/



CQRS database pattern, with separate write and read databases.

SQL Server for the write-side and Cosmos DB (NoSQL) for the read-side.



The write-side writes a 'projection' to the read-side database, i.e. the data is written in a form that is ready to display to the user.

EF Core Performance

Generated code almost never faster than writing SQL.

Use EF Core to make CRUD quicker to write.

Measure Performance.

Focus optimisation work for best business value.

Integration tests to reflect actual and predicted load.

Be ready to bypass EF Core when you need to.

EF Core and Ilogger<>

Easy to get logs out of Entity Framework

Integrates well with Serilog, SEQ, Application Insights

Measure the number of SQL statements executed per command or per request.

Use Query Tags to identify the code that invoked a query

The Performance Timebomb

Sub-optimal SQL can appear fine when developing against an empty DB

These problems can surface in Production – when you have lots of upset users!

You can't red-green refactor for performance without a failing test!

Bogus

Generate large amounts of human-friendly test data

Procedurally generated

Use a seed value to make data set repeatable

```
public IEnumerable<Customer> Generate()
{
    int id = 0;
    Randomizer.Seed = new Random(1234);

    var customerFaker = new Faker<Customer>()
        .RuleFor(t => t.CustomerId, f => "T" + (++id))
        .RuleFor(t => t.CompanyName, f => f.Company.CompanyName())
        .RuleFor(t => t.ContactName, f => f.Name.FullName())
        .RuleFor(t => t.Address, f => f.Address.StreetAddress());

    return customerFaker.Generate(9999);
}
```

```
public IEnumerable<Customer> Generate()
{
    int id = 0;
    Randomizer.Seed = new Random(1234);

    var customerFaker = new Faker<Customer>()
        .RuleFor(t => t.CustomerId, f => "T" + (++id))
        .RuleFor(t => t.CompanyName, f => f.Company.CompanyName())
        .RuleFor(t => t.ContactName, f => f.Name.FullName())
        .RuleFor(t => t.Address, f => f.Address.StreetAddress());

    return customerFaker.Generate(9999);
}
```

```
public IEnumerable<Customer> Generate()
{
    int id = 0;
    Randomizer.Seed = new Random(1234);

    var customerFaker = new Faker<Customer>()
        .RuleFor(t => t.CustomerId, f => "T" + (++id))
        .RuleFor(t => t.CompanyName, f => f.Company.CompanyName())
        .RuleFor(t => t.ContactName, f => f.Name.FullName())
        .RuleFor(t => t.Address, f => f.Address.StreetAddress());

    return customerFaker.Generate(9999);
}
```

```
public IEnumerable<Customer> Generate()
{
    int id = 0;
    Randomizer.Seed = new Random(1234);

    var customerFaker = new Faker<Customer>()
        .RuleFor(t => t.CustomerId, f => "T" + (++id))
        .RuleFor(t => t.CompanyName, f => f.Company.CompanyName())
        .RuleFor(t => t.ContactName, f => f.Name.FullName())
        .RuleFor(t => t.Address, f => f.Address.StreetAddress());

    return customerFaker.Generate(9999);
}
```

76	SPLIR	Split Rail Beer & Ale	Art Braunschwei...	Sales Manager	P.O. Box 555
77	SUPRD	Suprêmes délices	Pascale Cartrain	Accounting Manager	Boulevard Tirou, 255
78	T1	Tremblay - Hamill	Ubaldo Hayes	NULL	489 Nolan Harbor
79	T10	Trantow, Wintheiser and Schneider	Alycia Glover	NULL	253 Flatley Junctions
80	T100	O'Reilly, Renner and Welch	Abraham Kertzm...	NULL	725 Raynor Hollow
81	T1000	Marks, Kuhic and Hilll	Clemmie Blick	NULL	3342 Bayer Avenue
82	T1001	Bins, Ondricka and Mayer	Einar Bogisich	NULL	29745 Abshire Branch
83	T1002	Reinger Inc	Lillian Sanford	NULL	904 Tamara Parkway
84	T1003	Ullrich - Robel	Boyd Harber	NULL	204 Grover Center
85	T1004	Hettinger, Deckow and Schaden	Sarah Prosacco	NULL	7452 Nolan Cove

```
private TestContext()
{
    var builder = new DbContextOptionsBuilder<NorthwindDbContext>();
    builder.UseSqlServer(
        "Data Source=(LocalDb)\\mssqllocaldb;Initial Catalog=Northwind_loadtest;
        b => b.MigrationsAssembly("Northwind.EF.Persistence.MSSQL"));

    // use a  dbcontext to perform setup
    using (var initContext = new MsSqlNorthwindDbContext(builder.Options))
    {
        initContext.Database.EnsureDeleted();
        initContext.Database.Migrate();
        NorthwindInitializer.Initialize(initContext);

        var fakeCustomers = new TestCustomerGenerator().Generate();

        // todo: try bulkinsert again after release of ef core 3
        //NorthwindDbContext.BulkInsert<Customer>(fakeCustomers.ToList());
        initContext.Customers.AddRange(fakeCustomers);
        initContext.SaveChanges();
    }
    NorthwindDbContext = new MsSqlNorthwindDbContext(builder.Options);
}
```

```
private TestContext()
{
    var builder = new DbContextOptionsBuilder<NorthwindDbContext>();
    builder.UseSqlServer(
        "Data Source=(LocalDb)\\mssqllocaldb;Initial Catalog=Northwind_loadtest;
        b => b.MigrationsAssembly("Northwind.EF.Persistence.MSSQL"));

    // use a  dbcontext to perform setup
    using (var initContext = new MsSqlNorthwindDbContext(builder.Options))
    {
        initContext.Database.EnsureDeleted();
        initContext.Database.Migrate();
        NorthwindInitializer.Initialize(initContext);

        var fakeCustomers = new TestCustomerGenerator().Generate();

        // todo: try bulkinsert again after release of ef core 3
        //NorthwindDbContext.BulkInsert<Customer>(fakeCustomers.ToList());
        initContext.Customers.AddRange(fakeCustomers);
        initContext.SaveChanges();
    }
    NorthwindDbContext = new MsSqlNorthwindDbContext(builder.Options);
}
```

```
private TestContext()
{
    var builder = new DbContextOptionsBuilder<NorthwindDbContext>();
    builder.UseSqlServer(
        "Data Source=(LocalDb)\\mssqllocaldb;Initial Catalog=Northwind_loadtest;
        b => b.MigrationsAssembly("Northwind.EF.Persistence.MSSQL"));

    // use a  dbcontext to perform setup
    using (var initContext = new MsSqlNorthwindDbContext(builder.Options))
    {
        initContext.Database.EnsureDeleted();
        initContext.Database.Migrate();
        NorthwindInitializer.Initialize(initContext);

        var fakeCustomers = new TestCustomerGenerator().Generate();

        // todo: try bulkinsert again after release of ef core 3
        //NorthwindDbContext.BulkInsert<Customer>(fakeCustomers.ToList());
        initContext.Customers.AddRange(fakeCustomers);
        initContext.SaveChanges();
    }
    NorthwindDbContext = new MsSqlNorthwindDbContext(builder.Options);
}
```

Performance Summary

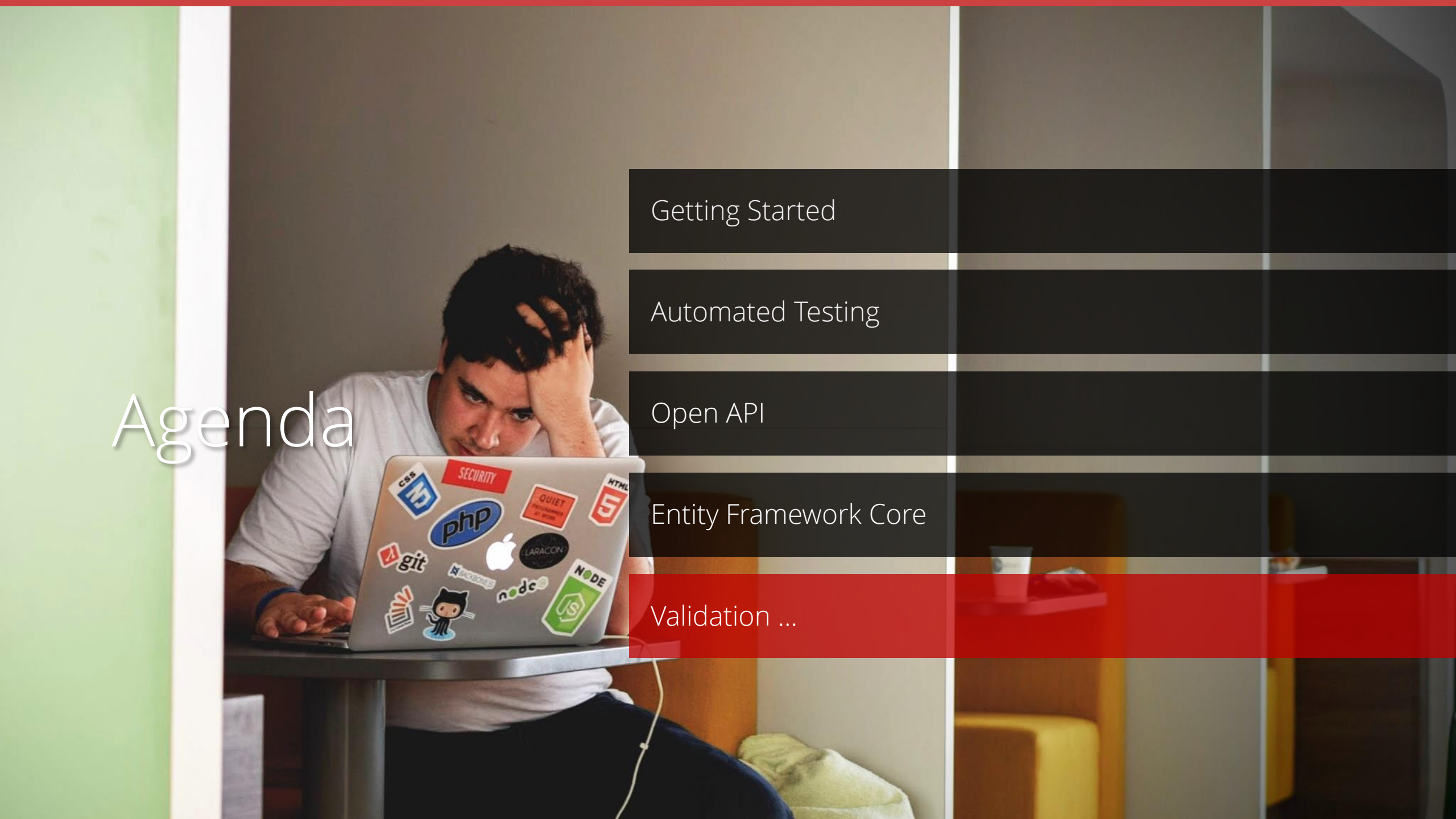
Evaluate the DB work performed for an operation – look for logical errors

Measure and prioritise optimization efforts

Optimisation Options: rewrite linq, compiled queries, map to hand- written SQL

Caching is the most effective way to speed up most applications

Agenda



- Getting Started
- Automated Testing
- Open API
- Entity Framework Core
- Validation ...

Demo: Data Annotations

Validating with data annotations

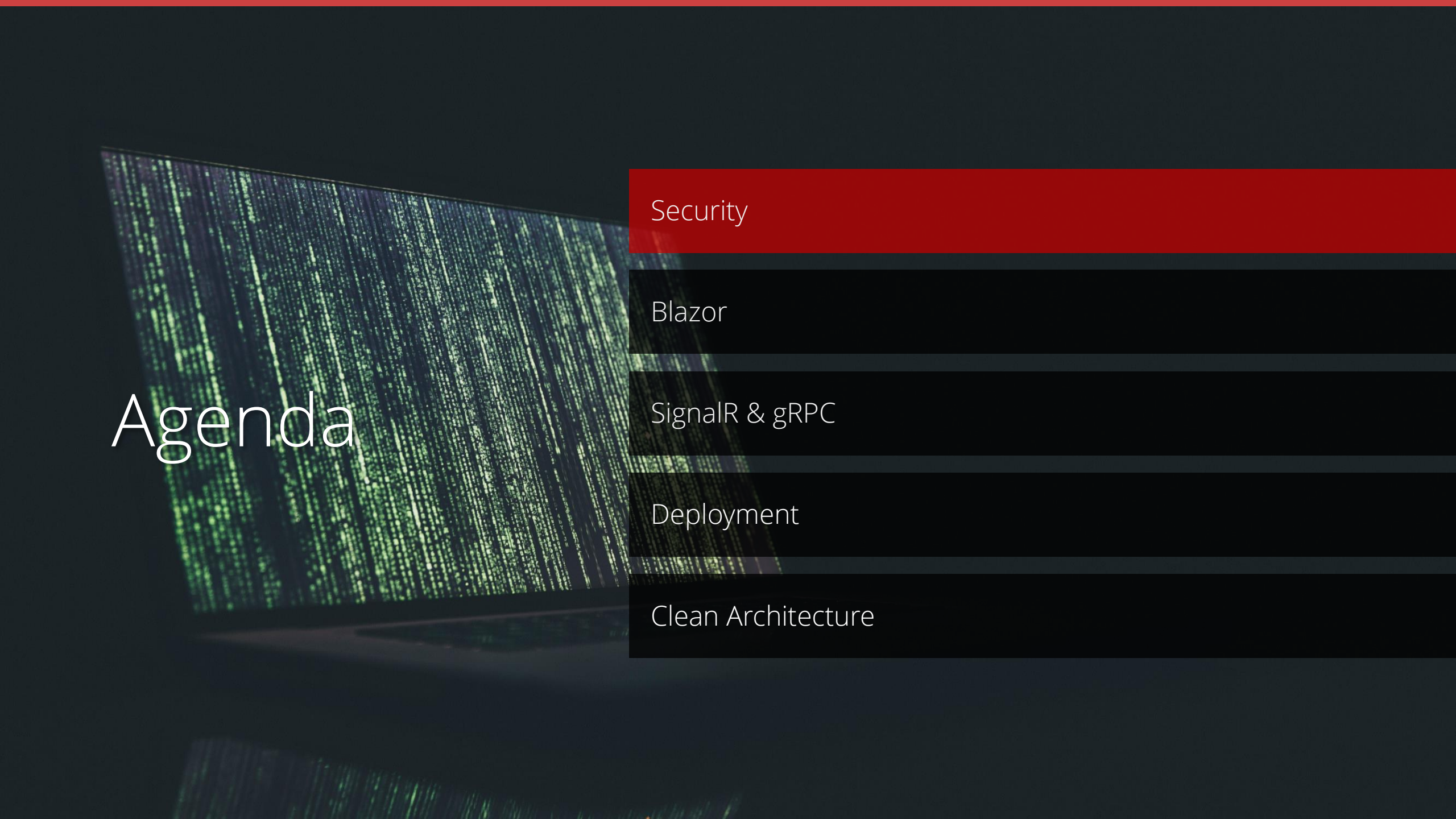
Demo: Fluent Validation

Validating with Fluent Validation

Client-Side Validation

Key Points

- Avoid using data annotations for validation
- Data annotations are only useful for simple scenarios
- Fluent Validation is useful for all validation scenarios
- Server-side validation is always required
- Client-side validation is optional



Agenda

- Security
- Blazor
- SignalR & gRPC
- Deployment
- Clean Architecture

Security Choices

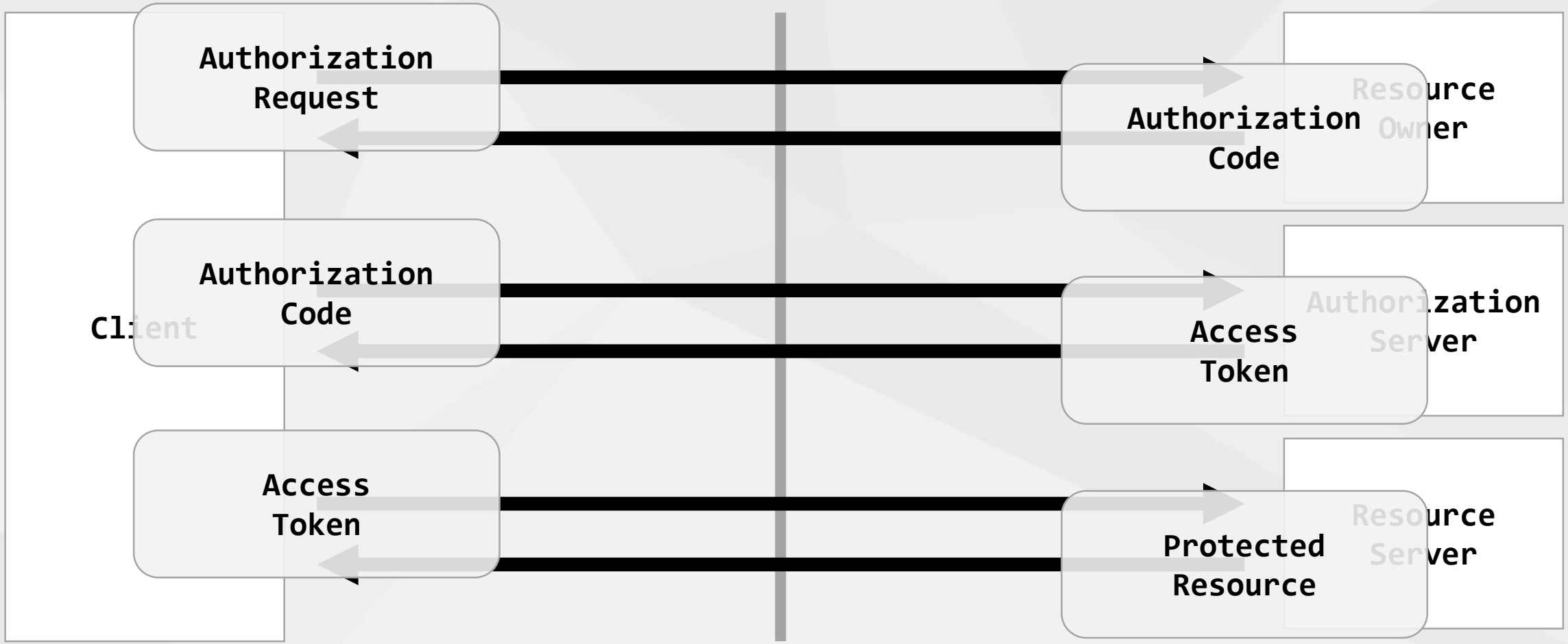
No Security

Windows Authentication

ASP.NET Core Identity

IdentityServer4

OAuth 2.0 (Abstract Flow)



Windows Authentication

Easy to implement (hosted on windows)

Good option for Intranet

Can use reverse proxy such as F5 to publish to internet

System.DirectoryServices – windows only via compatibility pack

ADFS can support OAuth 2.0 and OpenID Connect (OIDC)

Windows Authentication: Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<IISOptions>(options => options.ForwardWindowsAuthentication = true);
    // Add framework services.
    services.AddMvc();
}
```

Windows Authentication: web.config

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <system.webServer>
        <aspNetCore forwardWindowsAuthToken="true" processPath="%LAUNCHER_PATH%" arguments="%LAUNCHER_ARGS%" /
        <handlers>
            <add name="aspNetCore" path="*" verb="*" modules="AspNetCoreModule" resourceType="Unspecified" />
        </handlers>
    </system.webServer>
</configuration>
```

ASP.NET Core Identity

Membership provider: Registration, Reset Password, Attempt Limits

Support for 2FA – interfaces for Email & SMS Senders

Persistence to SQL Server (via EF Core) or your own persistence store

Support for external providers via OAuth 2.0 & OIDC

ASP.NET Core Identity - Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();
```

Startup.cs – Configure(IApplicationServices app)

```
app.UseStaticFiles();

app.UseIdentity();

app.UseMvc(routes =>
```

```
// Configure Identity
services.Configure<IdentityOptions>(options =>
{
    // Password settings
    options.Password.RequireDigit = true;
    options.Password.RequiredLength = 8;
    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequireUppercase = true;
    options.Password.RequireLowercase = false;

    // Lockout settings
    options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(30);
    options.Lockout.MaxFailedAccessAttempts = 10;

    // Cookie settings
    options.Cookies.ApplicationCookie.ExpireTimeSpan = TimeSpan.FromDays(150);
    options.Cookies.ApplicationCookie.LoginPath = "/Account/LogIn";
    options.Cookies.ApplicationCookie.LogoutPath = "/Account/LogOut";

    // User settings
```

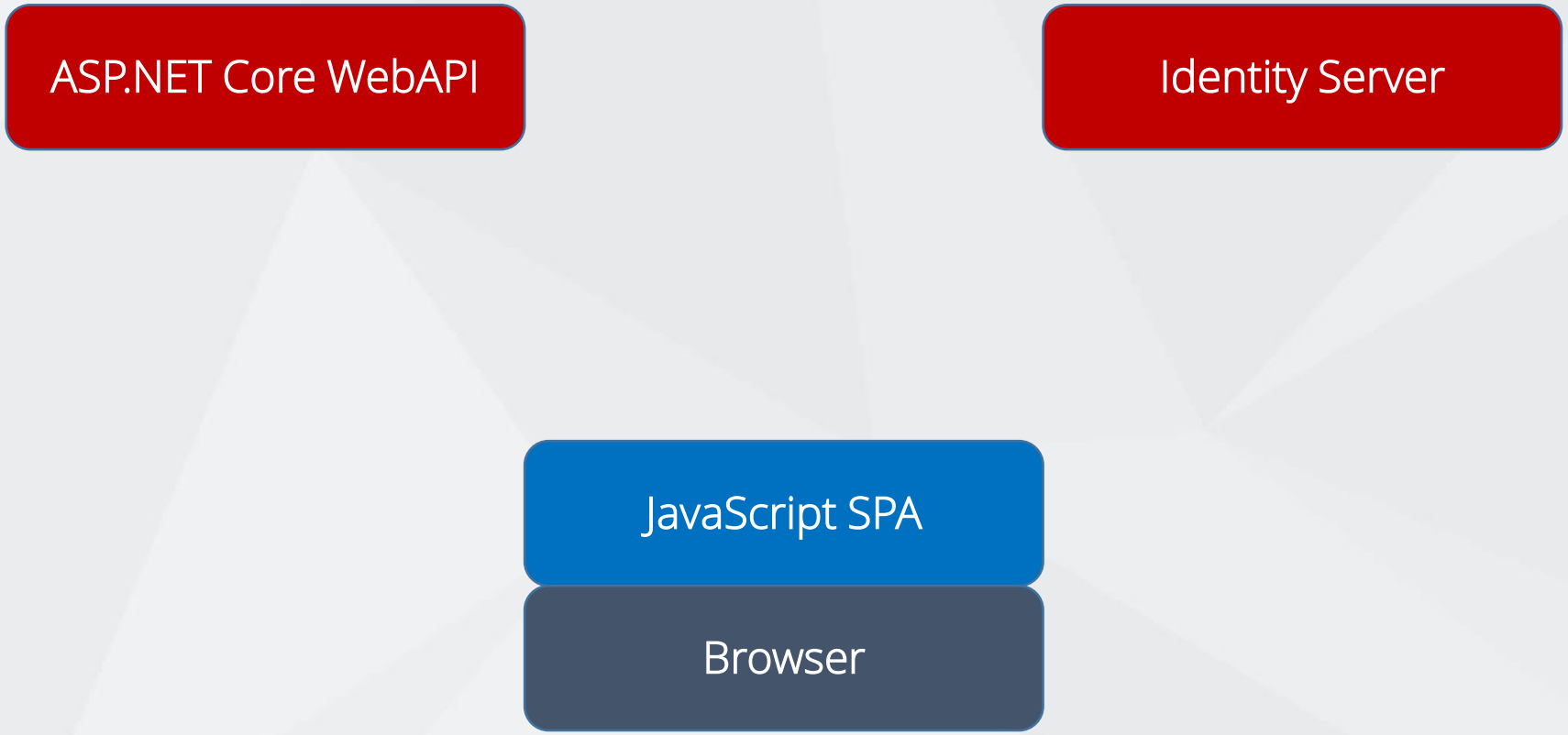
IdentityServer4

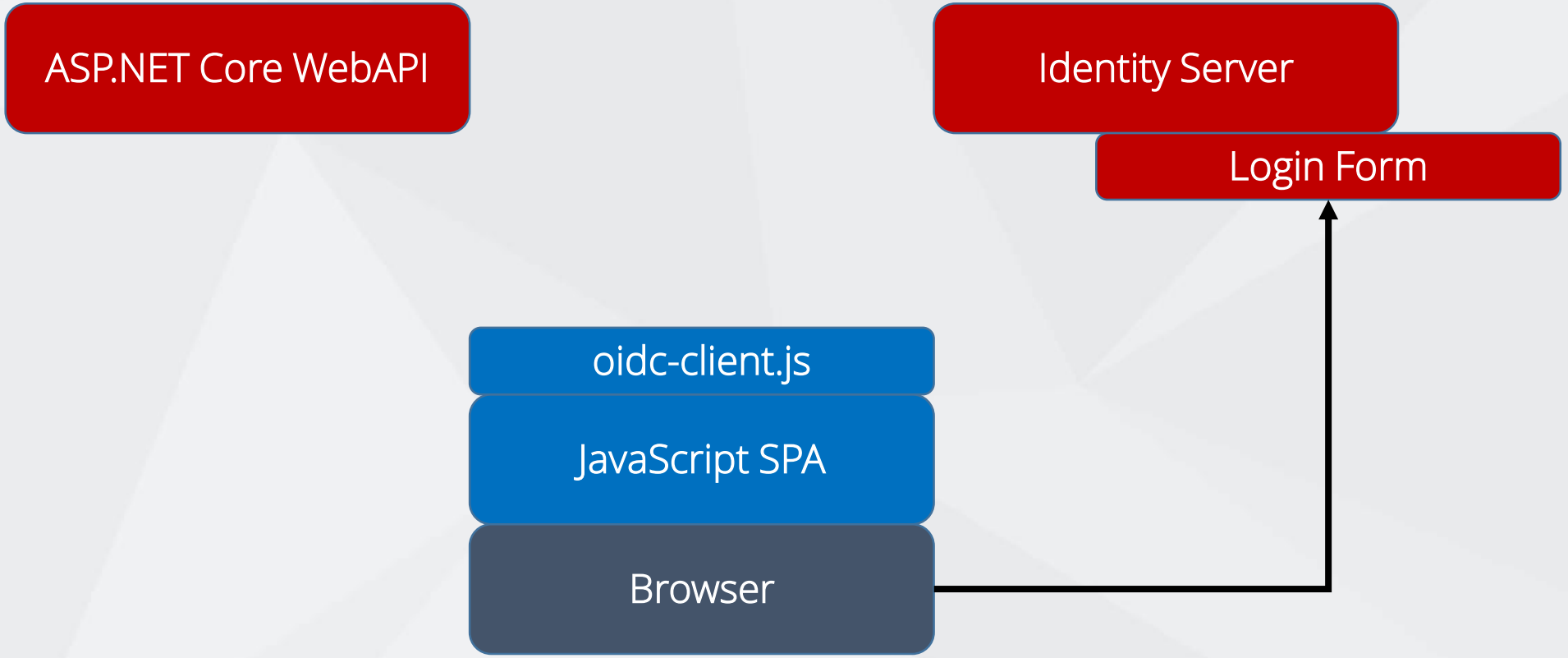
OpenID Connect and OAuth 2.0 framework for ASP.NET Core

Authentication as a service

Single sign-on (and out) over multiple application types

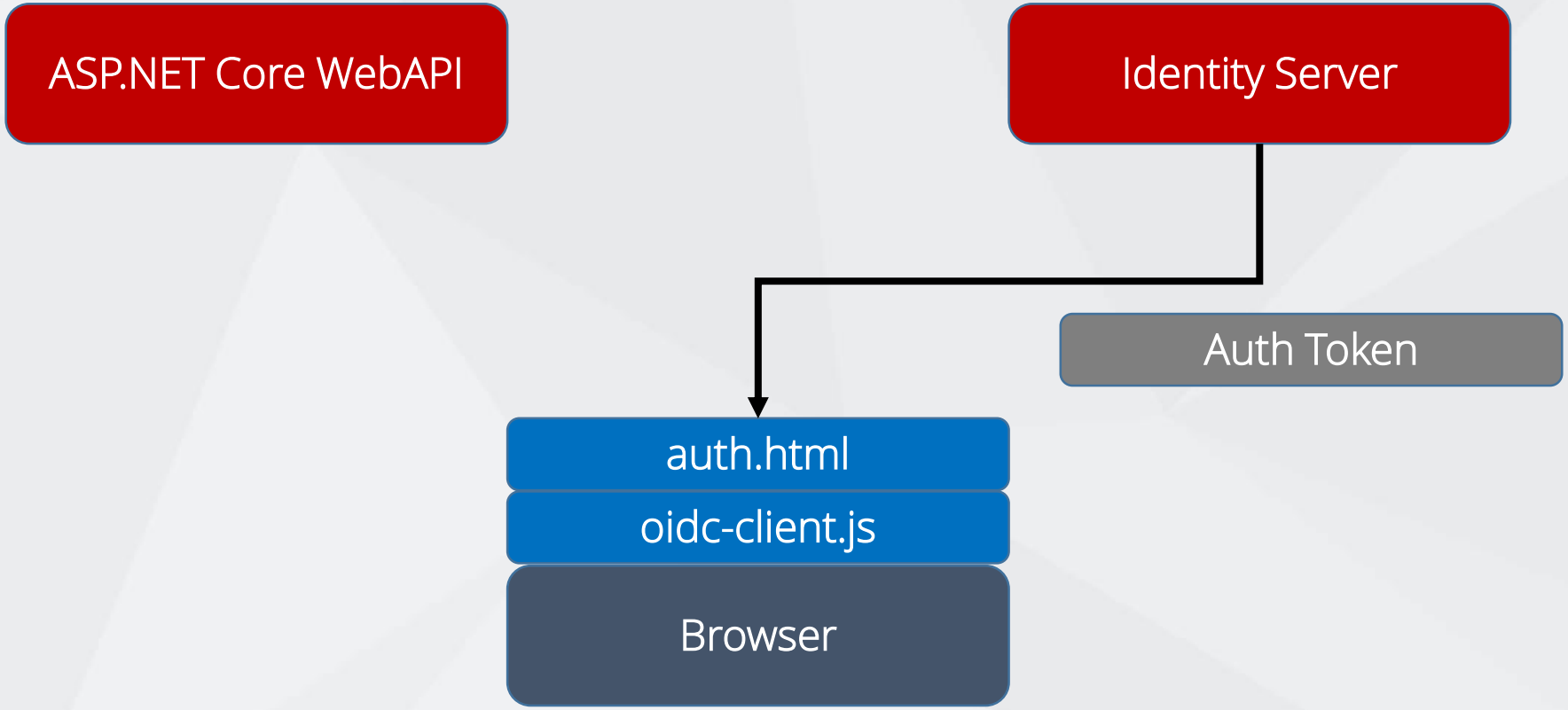
Example: JS application with .NET Core Web API


ASP.NET Core WebAPI
Identity Server
JavaScript SPA
Browser



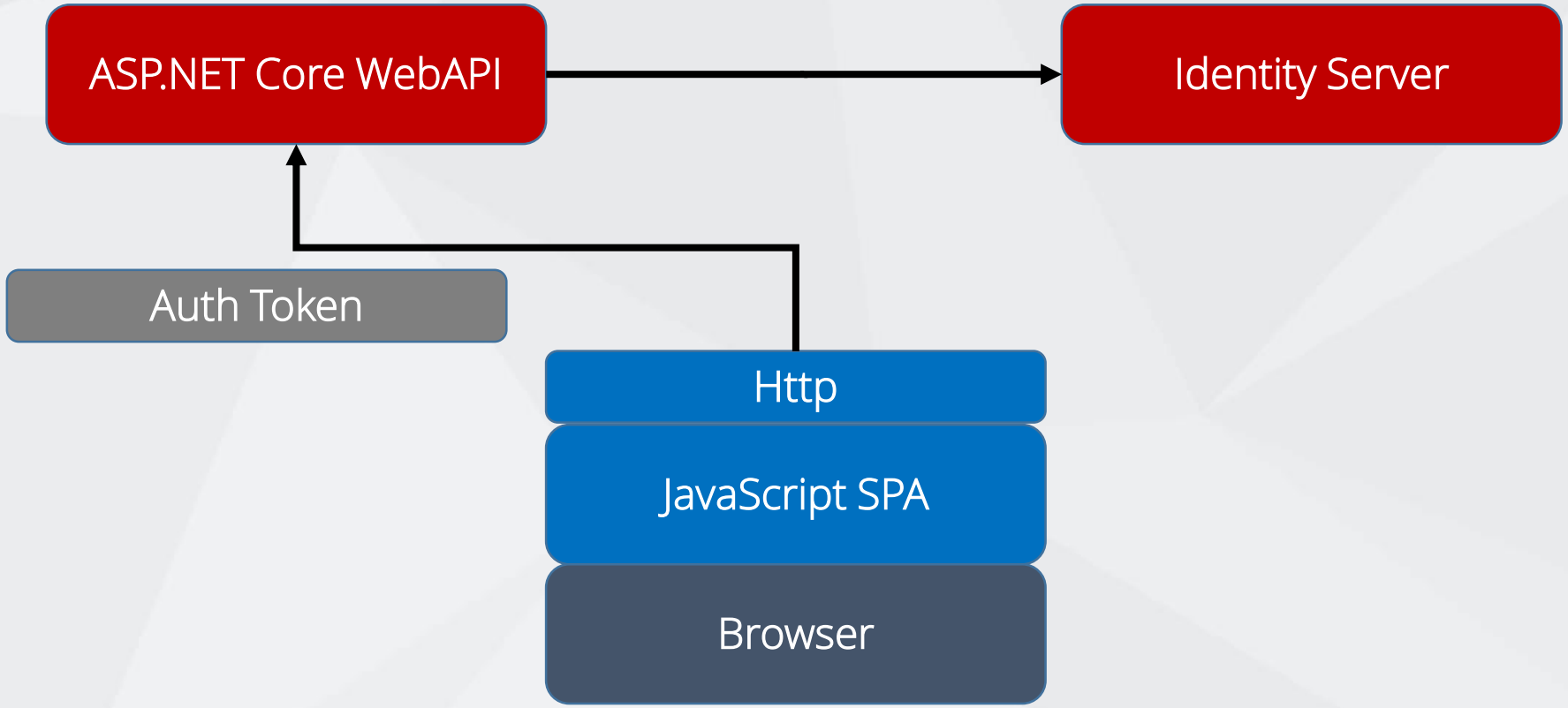
OIDC client redirects browser to a login page on the Identity Server.

User Logs in.



Identity Server redirects back with an Authorization Token

Token is saved on the browser



Send the Auth Token as a HTTP header with all API requests

The API server checks the token against the Identity Server

```
app.UseIdentityServerAuthentication(new IdentityServerAuthenticationOptions
{
    Authority = "https://[redacted]",//Configurati
    RequireHttpsMetadata = false,
    ApiName = "https://[redacted]",//"gr
    NameClaimType = "name",
    RoleClaimType = "role",
    AllowedScopes = { "offline_access", "profile", "openid" },
});
```

Demo: Identity with SPA

Create an Angular SPA with support for user authentication and authorization

Key Points

- Know the options – windows auth, identity, and identity server
- Take a look at the new templates, support ASP.NET Core Identity + IdentityServer4
- For SPA, scaffold the views and recreate for SPA

blazor

- Security
- Blazor
- SignalR & gRPC
- Deployment
- Clean Architecture

Microsoft

Steven Sanderson
@StevenSanderson

Tweets	Following	Followers	Likes
2,208	82	13.9K	14

Steve Sanderson

@stevensanderson

Also known as Steven Sanderson

Bristol, UK

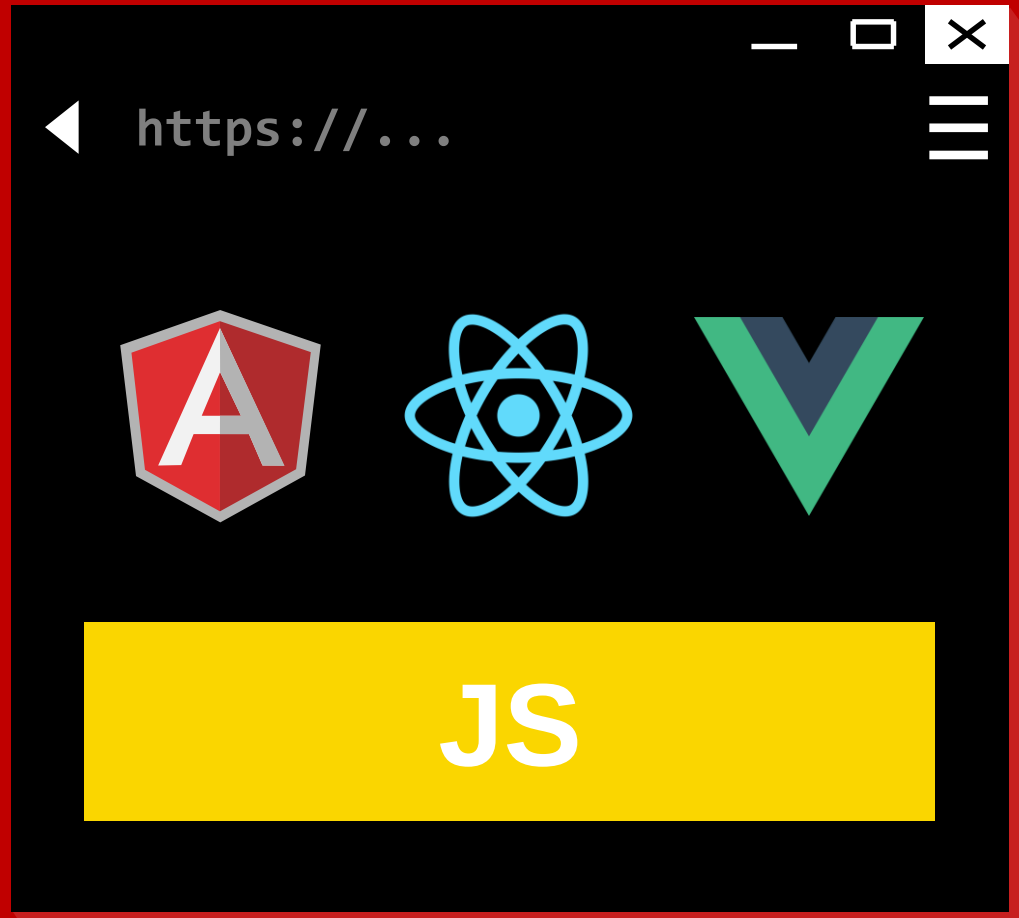
blog.stevensanderson.com

Tweets Tweets & replies Media

Steve Sanderson Retweeted

NDC Conferences @NDC_Conferences · 13h

@aVerySpicyBoi & @stevensanderson 2-day workshop "Blazor and the future of .NET web apps" will get you from the basics of Blazor to building sophisticated UIs using advanced framework features, at #NDCLondon #EarlyBird #NDC #Blazor #London

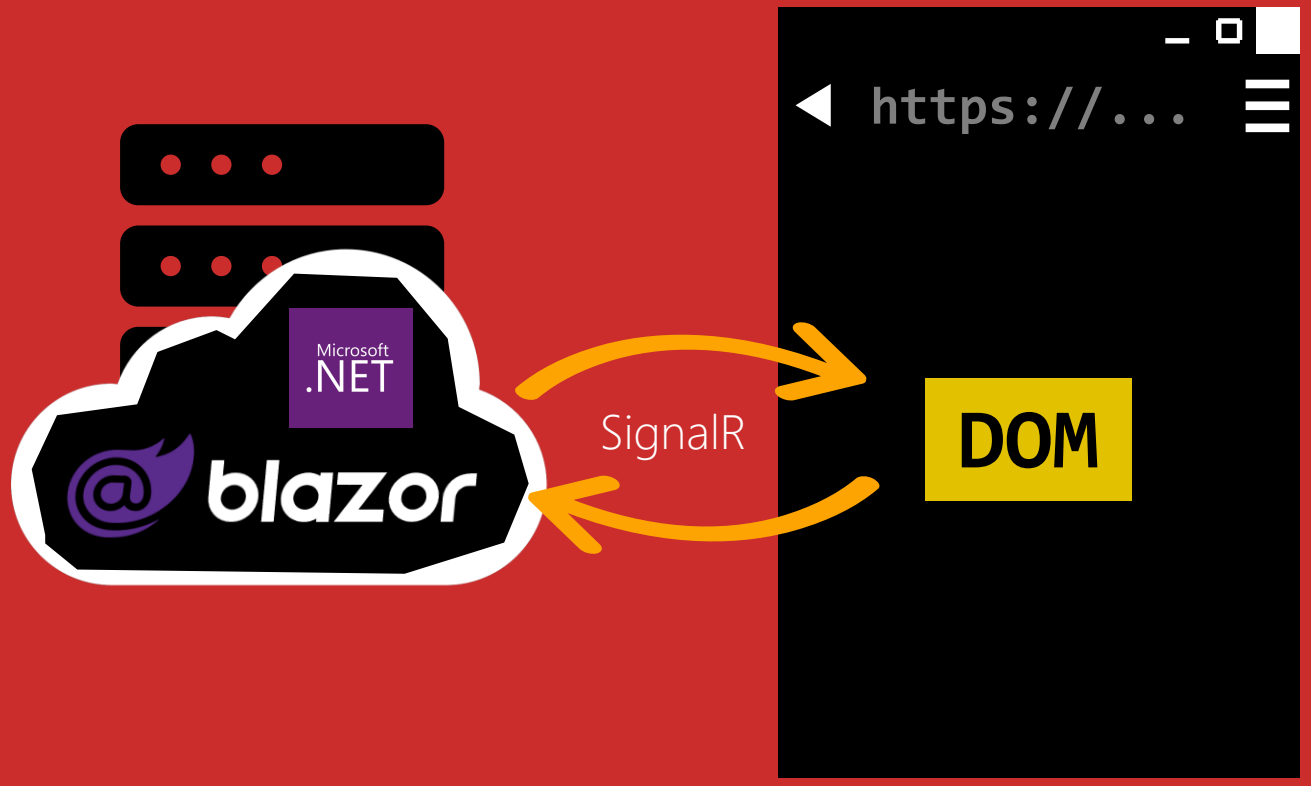

https://...
JS


WA

Client-side

(Blazor WebAssembly)

Client-side
(Blazor WebAssembly)



Server-side
(Blazor Server)

Demo: Client-Side Blazor

Create a Blazor UI

Demo: Server Side Blazor

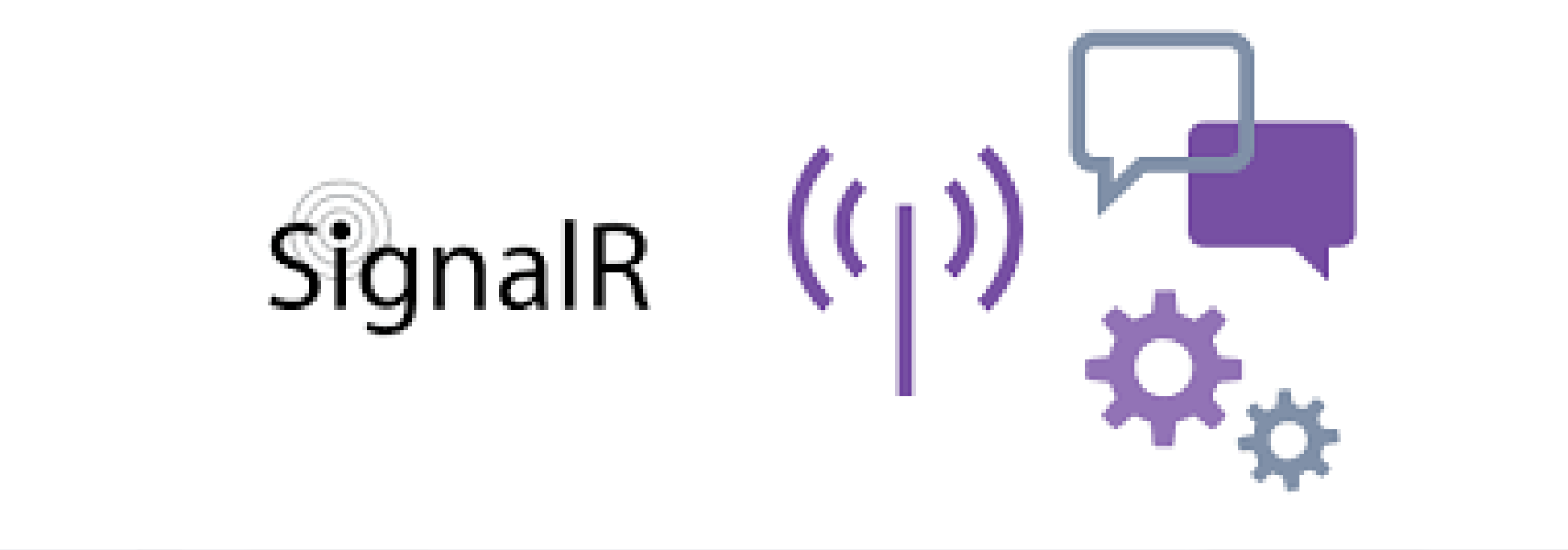
Key Points: Blazor

- Server-side Blazor is released
- Client-Side Blazor getting closer
- Build web UI with little / no javascript
- More on the way!

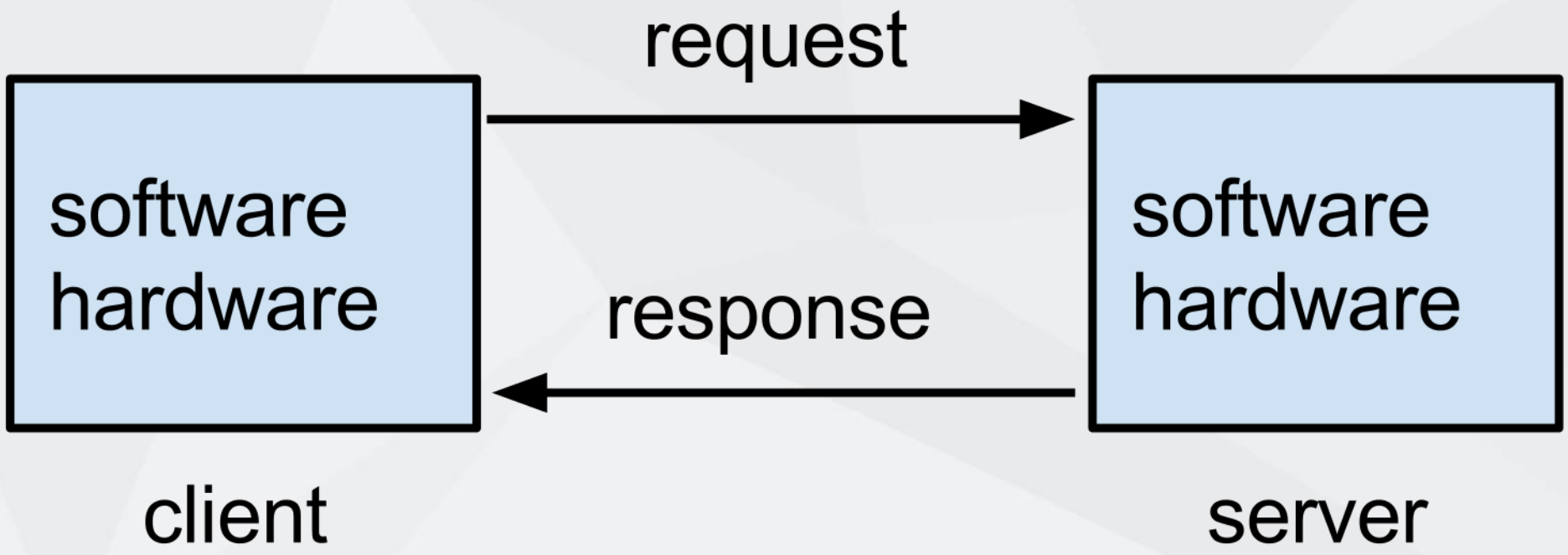
Agenda

- Security
- Blazor
- SignalR & gRPC
- Deployment
- Clean Architecture

Realtime Push Notifications with SignalR



Client- Server Communication



Two – way Communication

Web Sockets

SignalR

```
import { Injectable } from '@angular/core';
import { HubConnection } from '@aspnet/signalr';
import { Store } from '@ngxs/store';
import { CustomerListPushedAction } from '../ngxs/CustomerListActions';
import { Customer } from '../ngxs/Customer';
import { CustomerPushedAction } from '../ngxs/CustomerEditActions';


@Injectable()
export class CustomersHubService {
  public hubConnection: HubConnection;
```

```
openCustomer(customerId: string) {
  console.log('hub invoking OpenCustomer ', customerId);
  this.hubConnection.invoke('OpenCustomer', customerId);
}
```

```
  this.hubConnection.on('PushCustomerList', msg => {
    console.log('PushCustomerList', msg);
    this.store.dispatch(new CustomerListPushedAction(msg));
  });
```

```
using Microsoft.AspNetCore.SignalR;
using System.Threading.Tasks;

namespace SignalRChat.Hubs
{
    public class ChatHub : Hub
    {
        public async Task SendMessage(string user, string message)
        {
            await Clients.All.SendAsync("ReceiveMessage", user, message);
        }
    }
}
```

Key Points: Signal R

- SignalR: realtime async messaging to browsers
- Uses websockets with fall-back to polling
- Hubs on server instead of MVC Controllers
- Scale up with Azure SignalR Service

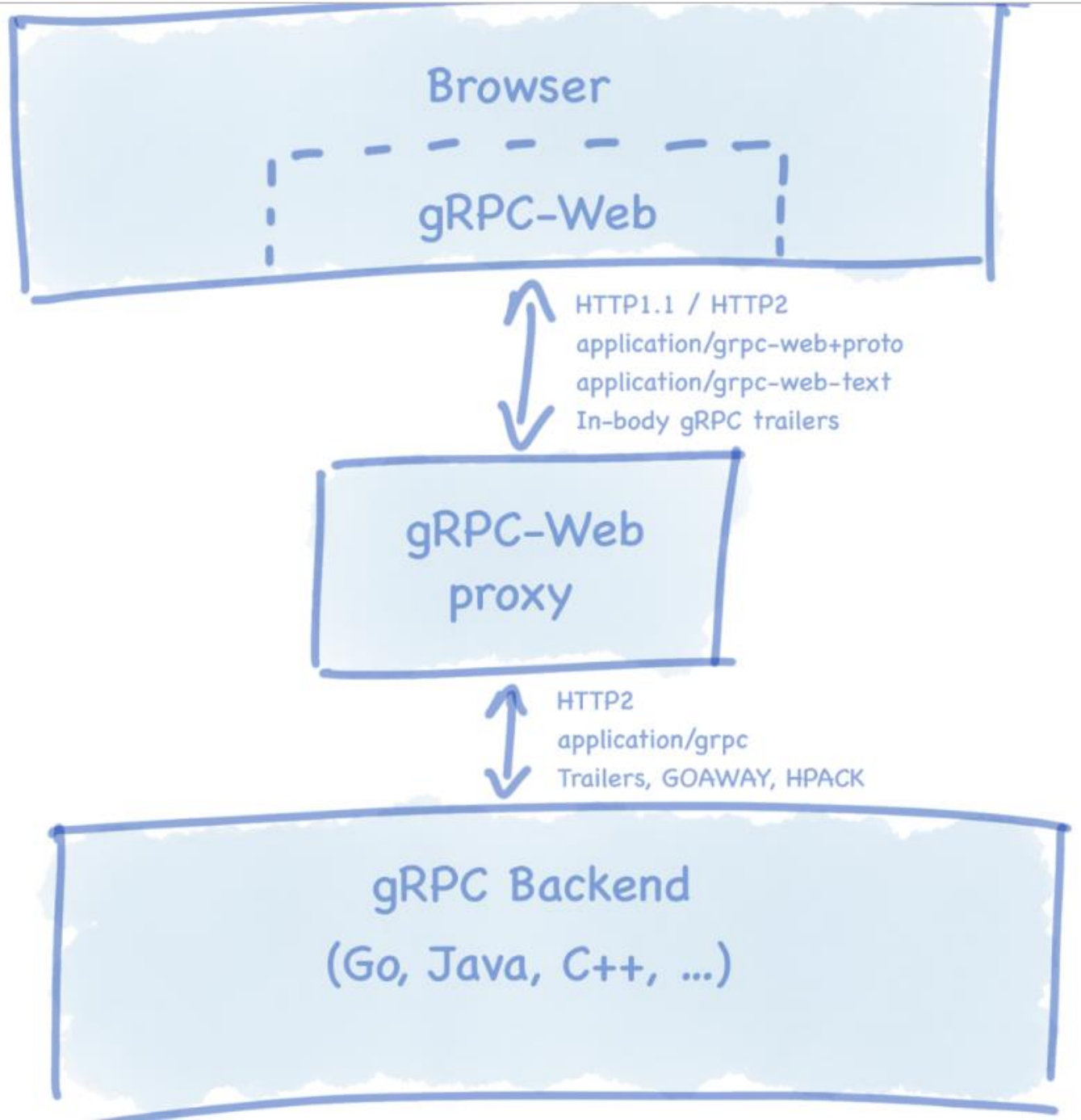
Demo: gRPC

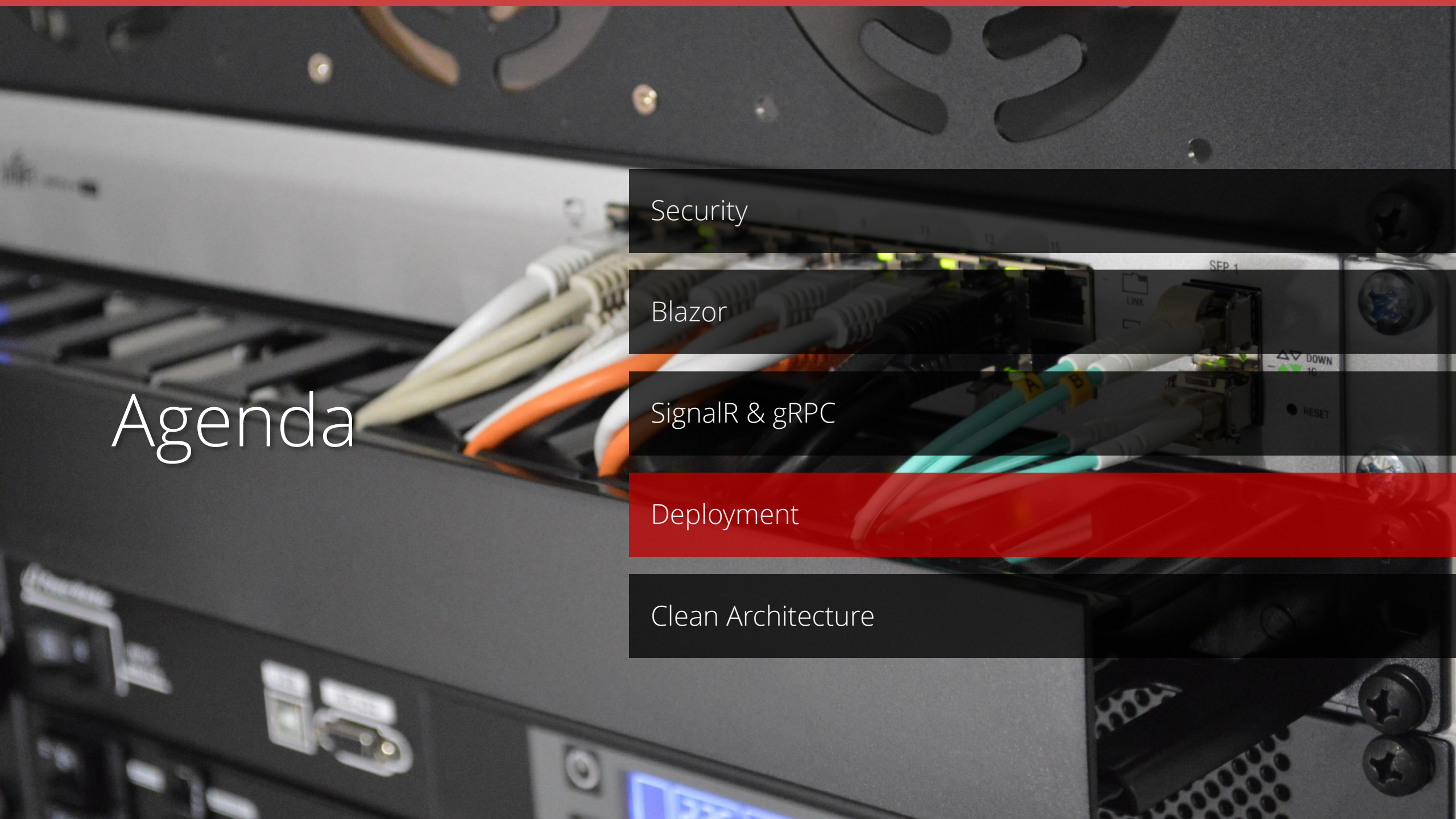
Create a gRPC Server

Create a gRPC Client

Key Points: gRPC

- Remote Procedure Call
- Uses HTTP2
- Protobuf: Faster & more strict than JSON
- Allows streaming connections
- Server-to-server only
- gRPC-Web for browsers – requires a proxy


Browser
gRPC-Web
HTTP1.1 / HTTP2
application/grpc-web+proto
application/grpc-web-text
In-body gRPC trailers
gRPC-Web
proxy
HTTP2
application/grpc
Trailers, GOAWAY, HPACK
gRPC Backend
(Go, Java, C++, ...)



Agenda

- Security
- Blazor
- SignalR & gRPC
- Deployment
- Clean Architecture

Demo: containerize a web app

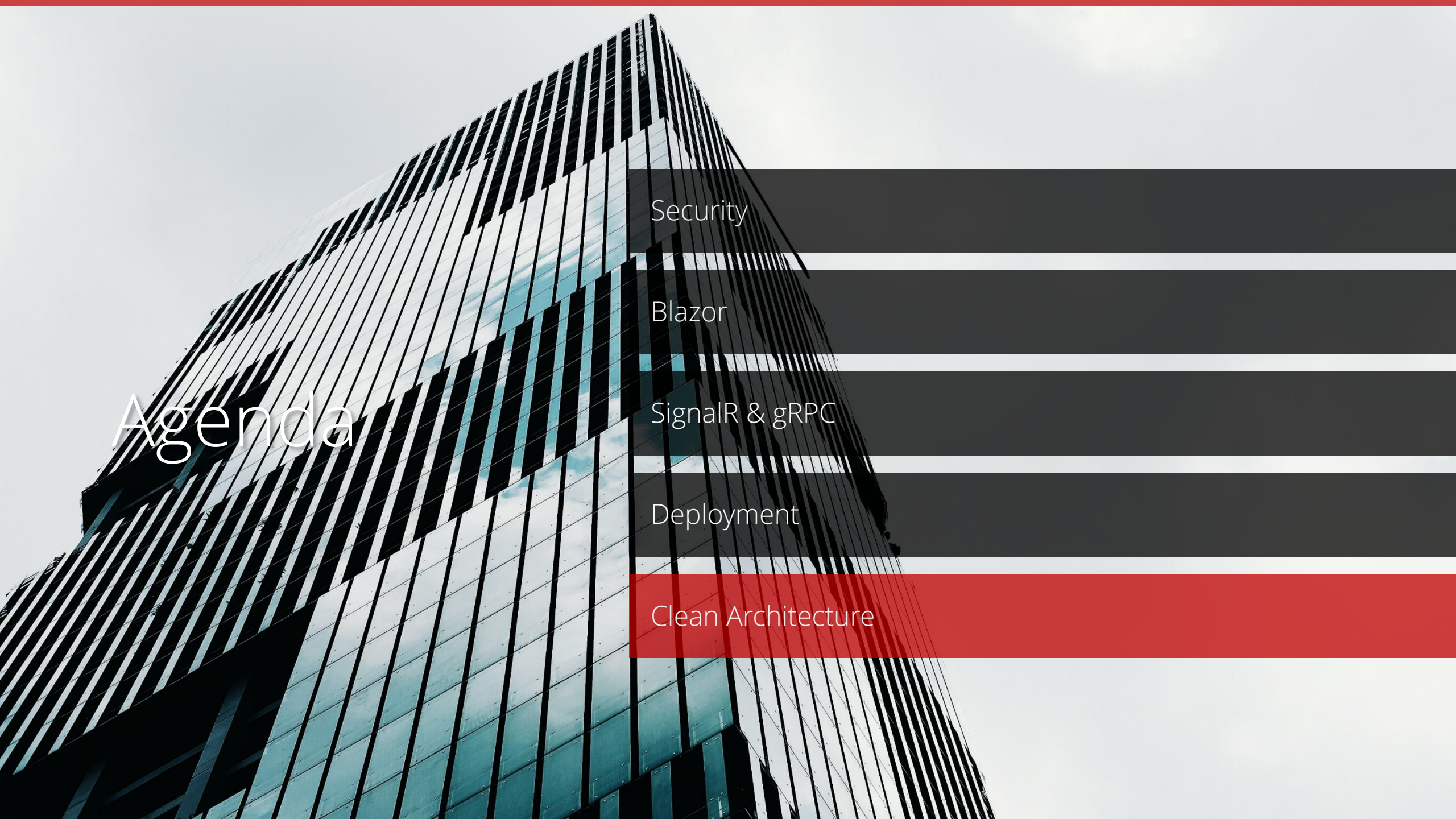
Containerize a web application

Deploy to Azure Webapps for Containers

Key Points: Deployment

- Dotnet cli makes all automated builds easier
- Can deploy to Windows or Linux servers
- Easy to apply per-environment settings.
- Containerize your apps for maximum portability
- Azure Devops has some great tooling to get started

Evaluation Form Reminder



Agenda

- Security
- Blazor
- SignalR & gRPC
- Deployment
- Clean Architecture

Clean Architecture

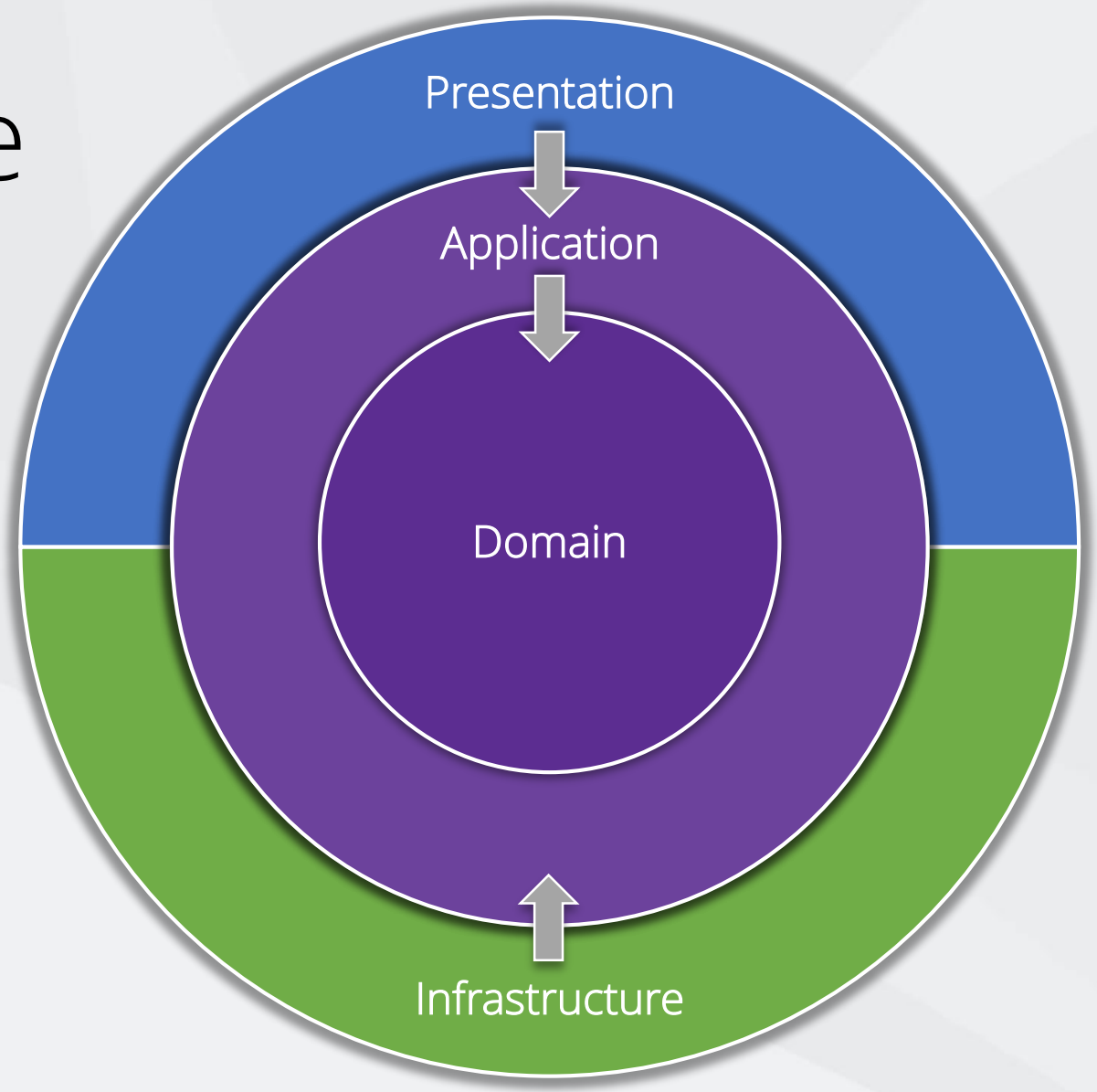
Independent of frameworks

Testable

Independent of UI

Independent of database

Independent anything external



Northwind Traders Sample

Clean Architecture

ASP.NET Core 3.0

Entity Framework Core 3.0

ASP.NET Core Identity 3.0

Repo bit.ly/northwind-traders

Clean Architecture Template

.NET Core Template Package

ASP.NET Core 3.0

Entity Framework Core 3.0

ASP.NET Core Identity 3.0

IdentityServer4

Repo [bit.ly/ca-sln](bit.ly/ca-sln)

Clean Architecture Template

Install the solution template

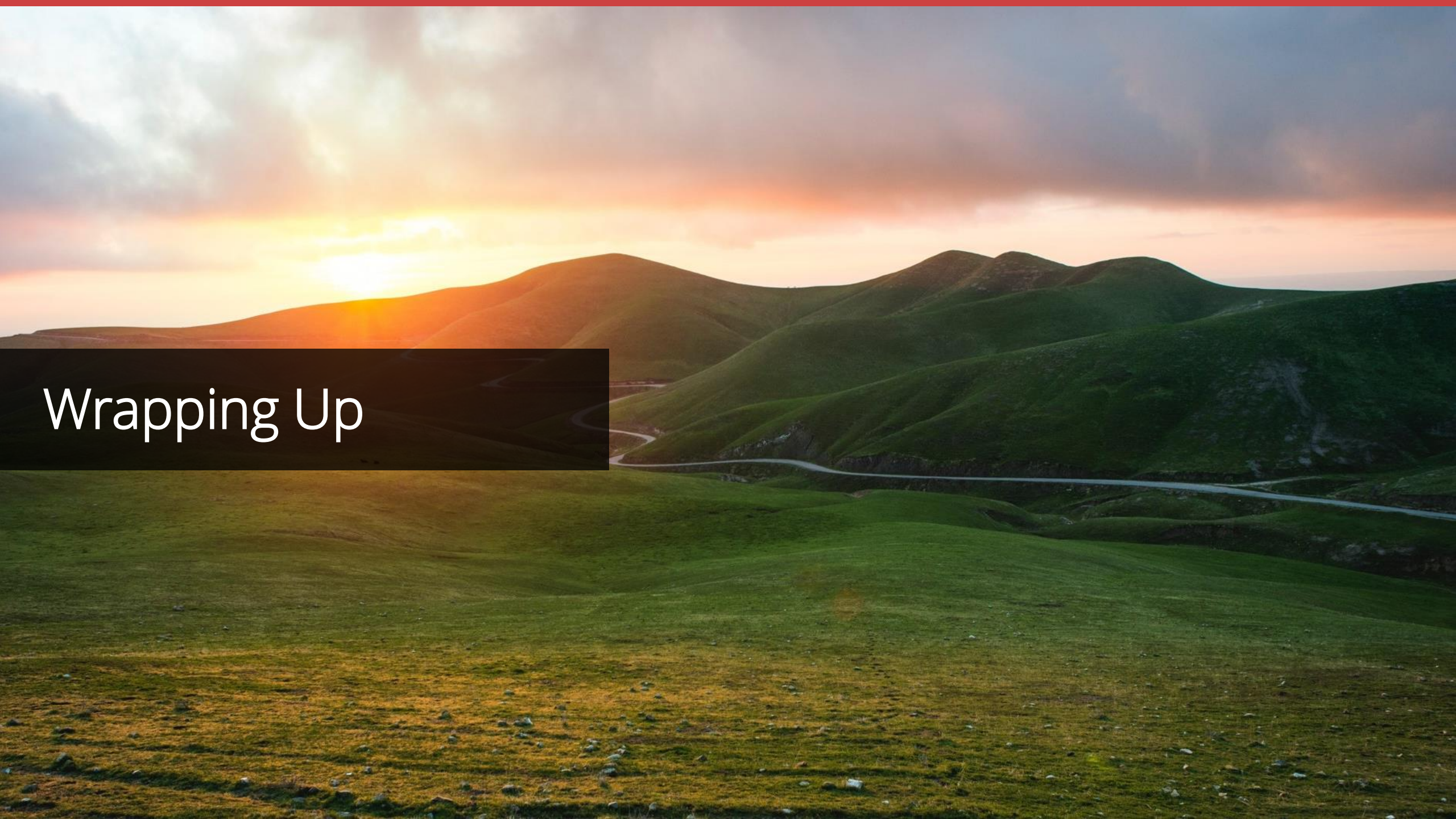
```
dotnet new --install Clean.Architecture.Solution.Template
```

Create a new solution

```
dotnet new ca-sln
```

Key Points

- Clean architecture provides the simplest approach to developing enterprise application development
- Application contains business-logic and types
- Infrastructure contains all external concerns
- Presentation and Infrastructure depend only on Application
- Infrastructure has no dependencies
- Generate new solutions with dotnet new ca-sln



Wrapping Up

The Angular 2-Day Workshop

Now that you've seen what Angular is capable of, it's time to get into learning-by-doing

Wed 27th & Thu 26th Nov 2019 – 2 Day Workshop - $440 - firebootcamp.com/superpower-tours

DURATION
2 Days

PRICE (EARLYBIRD SPECIAL)
$440 inc GST

Subscript

2-Day Workshop

DURATION
1 Day

PRICE
$49 inc GST

Brisbane
MON 21ST MAY 2018
Book Now

Melbourne
THU 24TH MAY 2018
Book Now

Sydney
FRI 25TH MAY 2018
Book Now

Hosted by **SSW**

About the presenter

Thiago Passos

Thiago Passos joined SSW in 2014 as a Senior Software Architect. He's specialized in SharePoint and .NET, preferring C#.NET over VB.NET, working on windows and web applications, including some distributed applications and web services.

Friday, 17th April 2020 – All Day Event - $99 - firebootcamp.com/superpower-tours

SERVICES PRODUCTS TRAINING USER GROUP RULES ABOUT US SSW TV

DURATION
1 Day

PRICE
$49 inc GST

Brisbane
MON 25TH JUN 2018
Book Now

Melbourne
THU 28TH JUN 2018
Book Now

Sydney
FRI 29TH JUN 2018
Book Now

Friday, 15th May 2020 – All Day Event - $99 - firebootcamp.com/superpower-tours

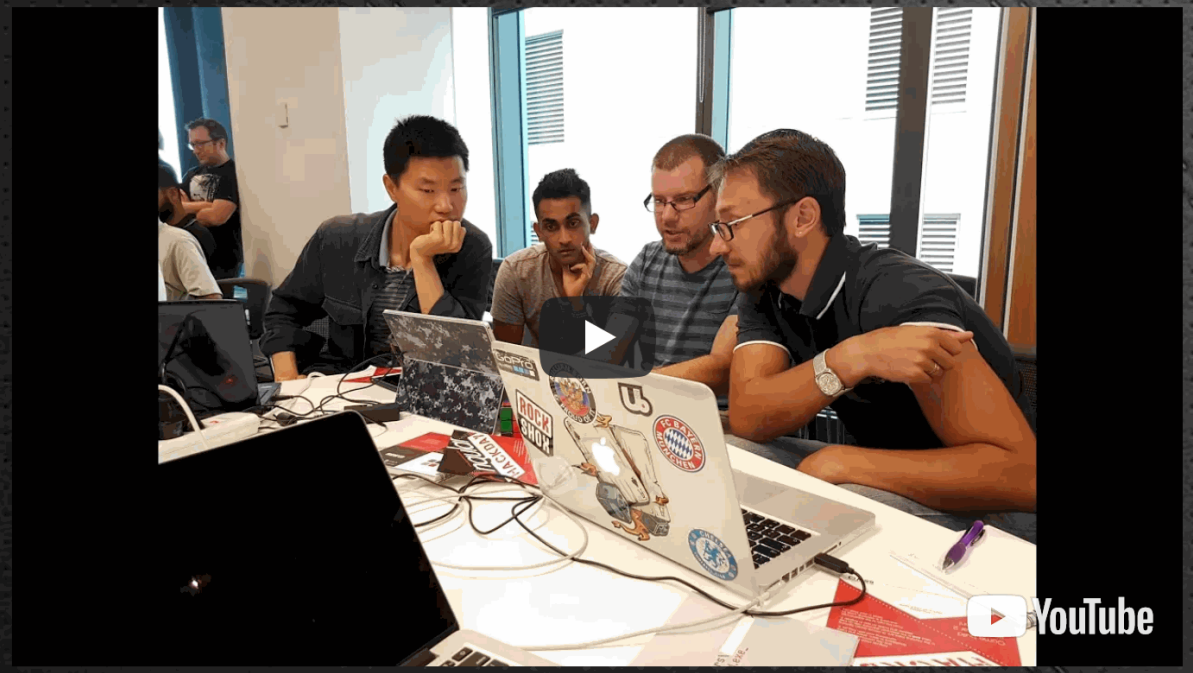
Hosted by **SSW**

About the presenters

Brendan Richards

Throughout his career, Brendan has been a big user and proponent of Open Source software. This has been applied to a broad variety of web-based programming projects spanning the last 12 years.

Brendan has worked with an eclectic mix of languages, platforms and technologies including .Net, Java, PHP, Ruby and Perl



Accelerated learning, knowledge sharing and lots of Angular hacking.

WATCH RELATED VIDEOS

What is Angular Hack Day?

AngularJS Hack Days are community run events for Angular Developers or people who want to learn AngularJS for free.

There will be something for everyone. Experienced AngularJS developers can share ideas with other experienced developers. If you're a beginner then there's plenty to learn on the day.

Saturday, 7th March 2020 – All Day Event - Free - http://angularhackday.com/register

played with AngularJS before it would be good to do a little bit of learning before the day but it's not a requirement as all are welcome.

When and where?

Many cities around the world. Look for dates on a venue near you.

Event Organizers

Adam Stephensen, Duncan Hunter, Ben Cull, Eric Phan and Adam Cogan from **SSW Consulting**. You can contact them via this form.

Coming Soon - Early 2020 – Clean Architecture Superpowers – 2 Day Workshop

Thank you!

Code & Slides available from
github.com/SSWConsulting/Fbc-DotNetCore-201911

info@ssw.com.au

www.ssw.com.au

Sydney | Melbourne | Brisbane