Enterprise Software Development

Join the Conversation #DotNetCoreSuperpowers @SSW_TV

.NET Core Superpowers Tour

Brisbane

Join the Conversation #DotNetCoreSuperpowers @SSW_TV

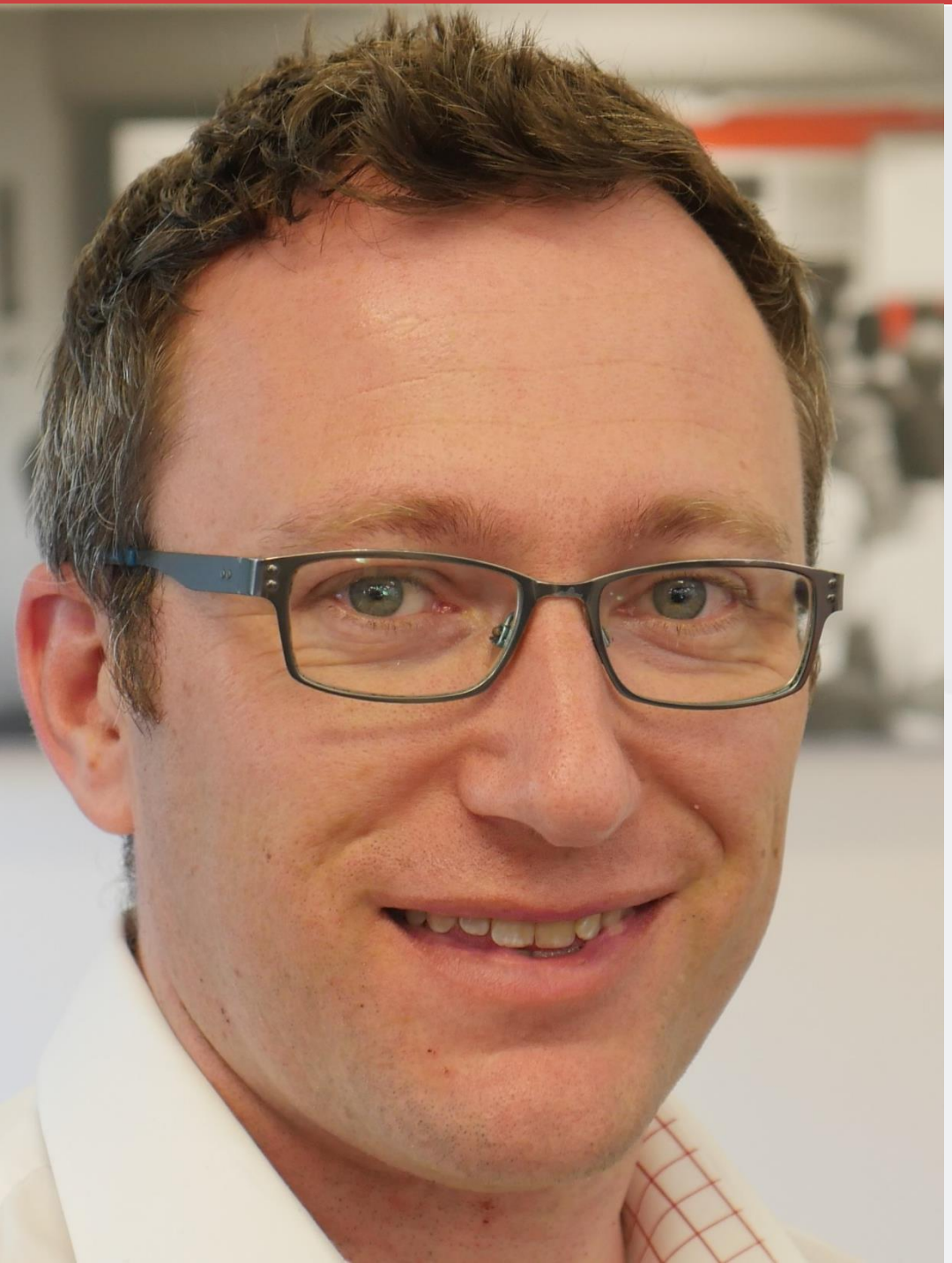
House Keeping

Code & Slides bit.ly/dotnetcoretour

Facilities

Timetable

09:00	Start
10:30 – 10:45	Morning Tea
12:00 – 12:45	Lunch
15:00 – 15:15	Afternoon Tea
16:30 – 17:00	Feedback, Q & A
17:00	Finish



Brendan Richards

@brendanssw @ssw_tv

- Solution Architect @ SSW
- Linux User since 1995
- .NET Developer since 2004
- .NET Core Since Beta 2

Jason Taylor

SSW Solution Architect

Master of Information Technology

Microsoft Certified Solutions Developer

Certified Scrum Master (CSM)

@JasonGtAu

codingflow.net

Agenda

- What's New in Visual Studio 2017 and VS Code
- Getting Started
- Entity Framework Core
- Clean Architecture
- Swashbuckle ...

Agenda

- Validation
- Unit Testing
- Logging & Exceptions
- Security
- Deployment

What's New in Visual Studio 2017 and VS Code

Dev Superpowers Tour

Getting Started with ASP.NET Core

Dev Superpowers Tour

Demo: Getting Started

Creating app using .NET Core in Visual Studio and using the .NET Core CLI

Demo: Returning Views

Returning views within an ASP.NET Core MVC app

Demo: Working with Views

Working with Views in ASP.NET Core MVC

Demo: Returning Data

Returning data within an ASP.NET Core Web API application

Single Page Applications

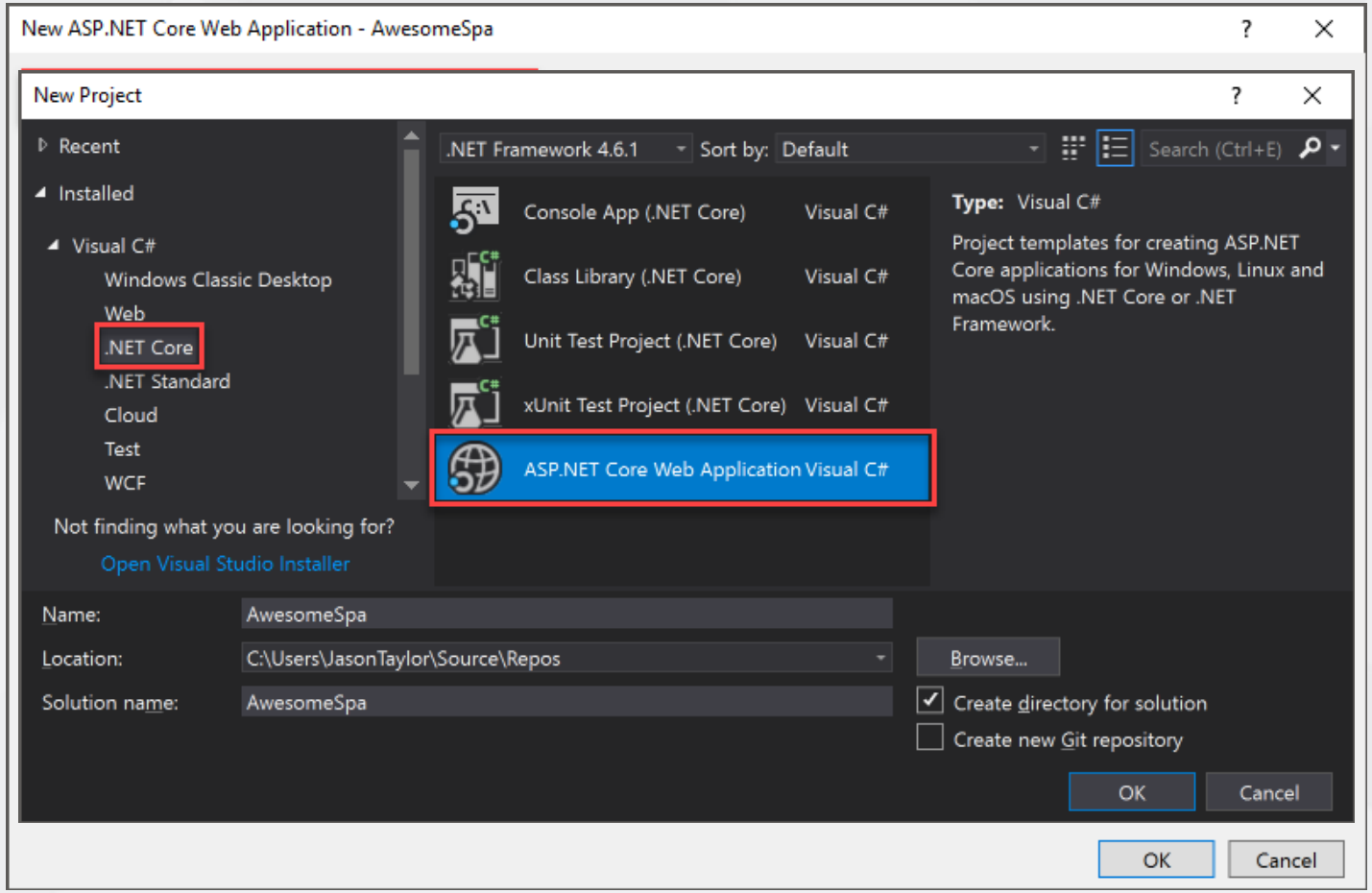
ASP.NET Core has built-in support for SPA

Supports all major client-side frameworks

Cross-platform Windows, Mac, or Linux

Use your preferred IDE or code editor

Using Visual Studio 2017



Using .NET Core CLI

Windows PowerShell

```
> dotnet new --list

Templates                                         Short Name
-------------------------------------------------------------------
ASP.NET Core with Angular                         angular
ASP.NET Core with React.js                        react
ASP.NET Core with React.js and Redux              reactredux
```

Additional Templates

Windows PowerShell

```
> dotnet new --install Microsoft.AspNetCore.SpaTemplates::*
```

Angular + ASP.NET Core

Windows PowerShell

```
> dotnet new angular
```

Aurelia + ASP.NET Core

Windows PowerShell

```
> dotnet new aurelia
```

Knockout + ASP.NET Core

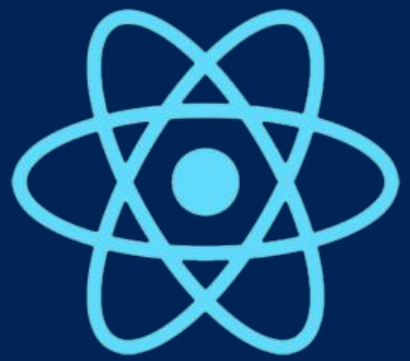
Windows PowerShell

```
> dotnet new knockout
```

React + ASP.NET Core

Windows PowerShell

```
> dotnet new react
```



React + Redux + ASP.NET Core

Windows PowerShell

```
> dotnet new reactredux
```

Vue + ASP.NET Core

Windows PowerShell

```
> dotnet new vue
```

Demo: Single Page Applications

Building single page applications with ASP.NET Core

Demo: Building a Web API

Creating an awesome ASP.NET Core Web API

Entity Framework Core

Dev Superpowers Tour

Entity Framework Core

Lightweight, extensible, and cross-platform

Runs on .NET Core or .NET Framework

Choosing between EF6.x and EF Core (bit.ly/ef6-vs-core)

Porting from EF6.x to EF Core (bit.ly/ef6-to-core)

EF Core Roadmap (bit.ly/efcore-roadmap)

Entity Framework Concepts

DbContext – “Heavyweight” ORM that tracks changes

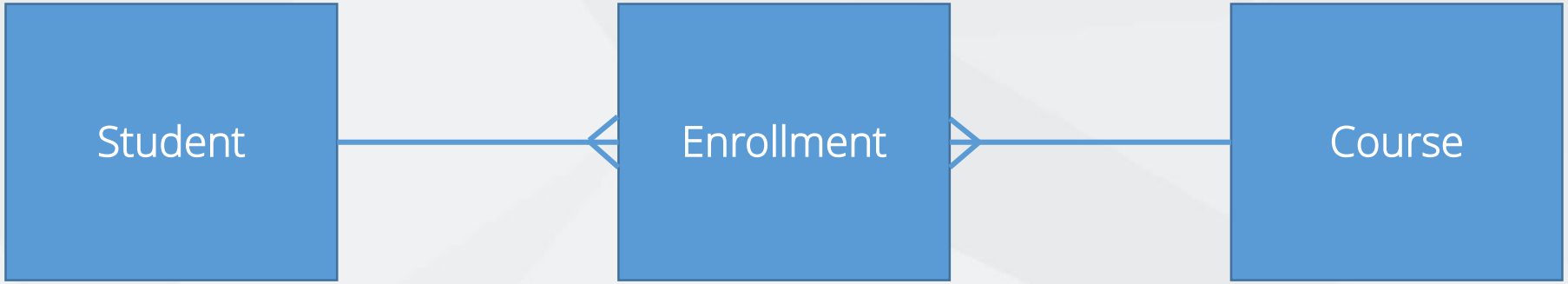
States: Added, Unchanged, Modified, Deleted, Disconnected

“Code First" Only – but you can Scaffold

Migrations – Used to generate and publish schema changes

Lazy Loading – not implemented yet

Simple Model



Demo: DTO / View Models

Managing data with DTOs and ViewModels

Demo: EF Core Scaffolding

Scaffolding a new data model from an existing database

NorthwindTraders

Northwind Traders is a sample application built using ASP.NET Core and Entity Framework Core.

The initial purpose of this solution is to create a code first version of Northwind, using the .NET Core stack.

Getting Started

Northwind Traders EF Core - bit.ly/efcore-northwind

Use these instuctions to get the project up and running.

Clean Architecture

Dev Superpowers Tour

Clean Architecture

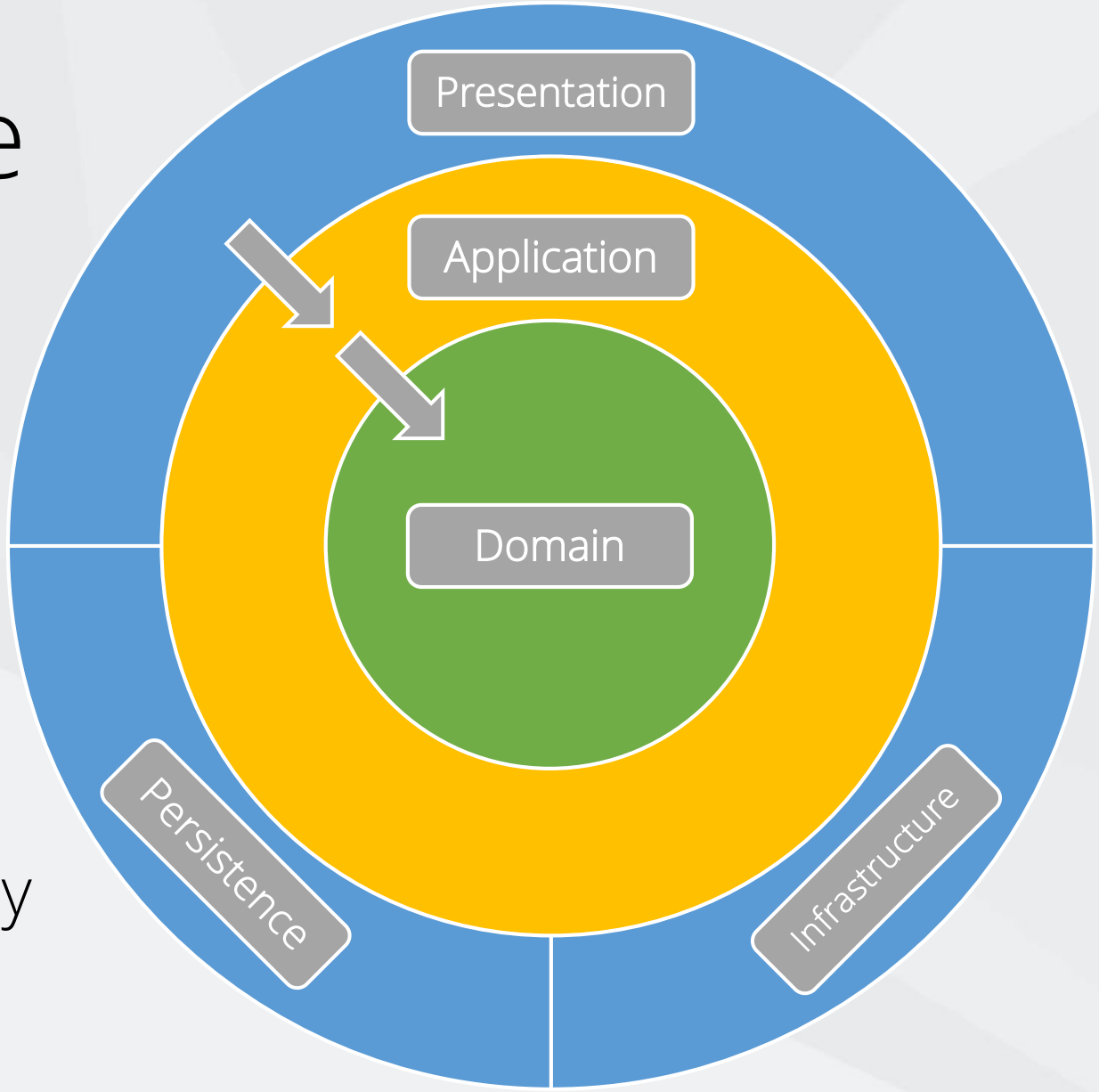
Independent of frameworks

Testable

Independent of UI

Independent of database

Independent of external agency



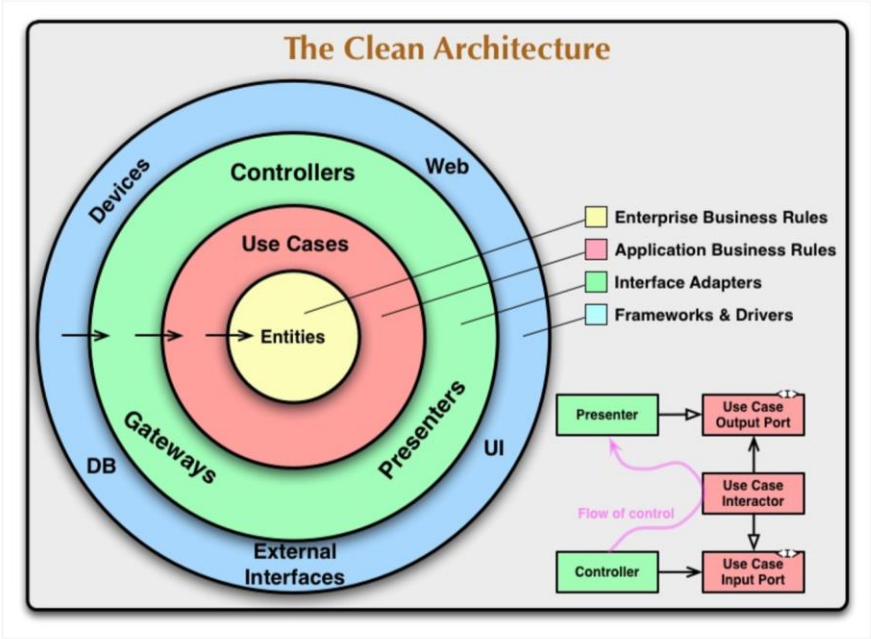
Demo: Clean Architecture

A review of the Northwind solution

The Clean Architecture

Uncle Bob / 13 Aug 2012 Architecture Craftsmanship

Tweet



Over the last several years we've seen a whole range of ideas regarding the architecture of systems. These include:

- Hexagonal Architecture (a.k.a. Ports and Adapters) by Alistair Cockburn and adopted by Steve Freeman, and Nat Pryce in their wonderful book Growing Object Oriented Software
- Onion Architecture by Jeffrey Palermo
- Screaming Architecture from a blog of mine last year
- DCI from James Coplien, and Trygve Reenskaug.
- BCE by Ivar Jacobson from his book *Object Oriented Software Engineering: A Use-Case Driven Approach*

Though these architectures all vary somewhat in their details, they are very similar. They all have the same objective, which is the separation of concerns. They all achieve this separation by dividing the software into layers. Each has at least one

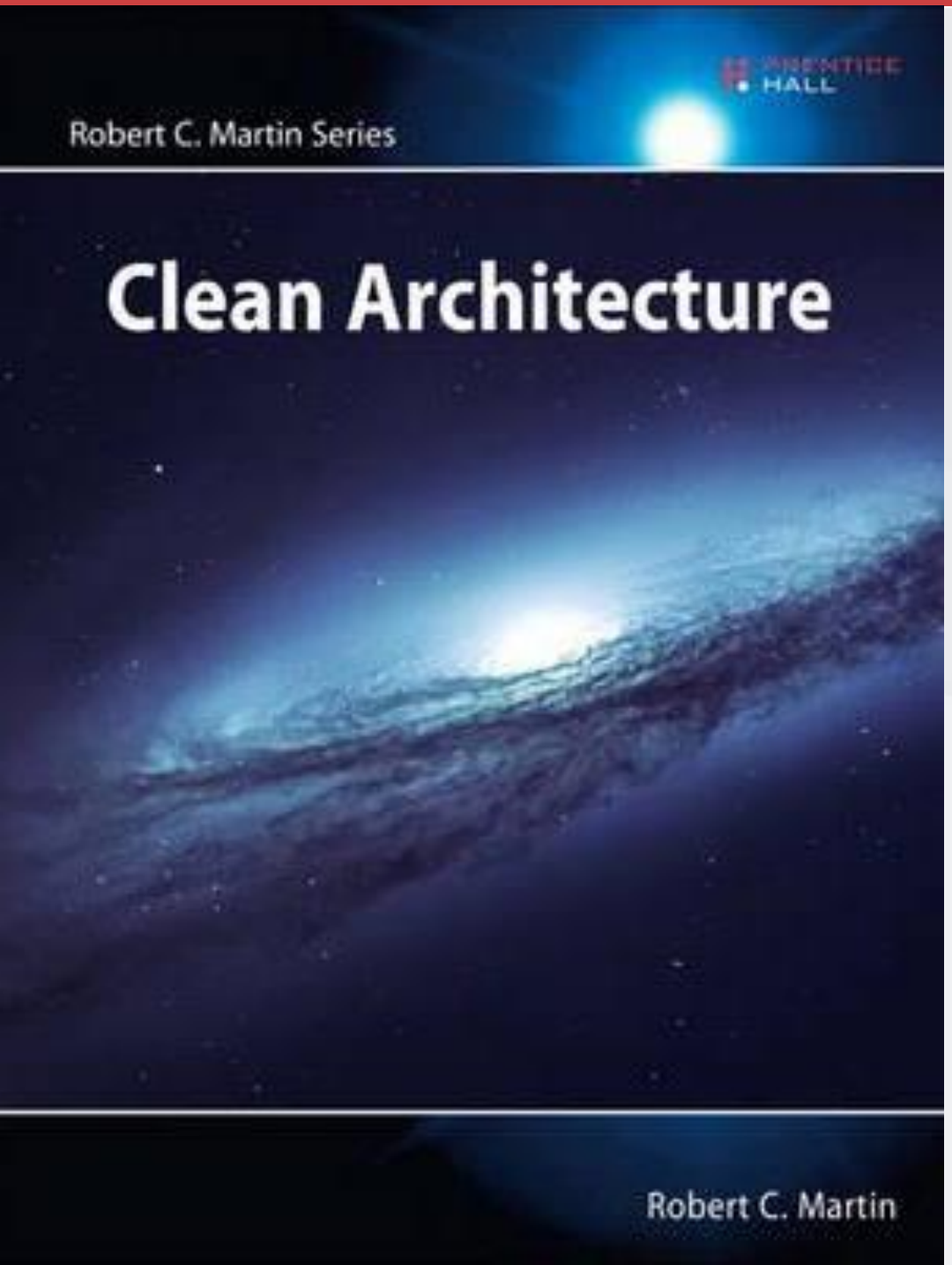
The Clean Architecture

By Robert C. Martin

Original article by Uncle Bob

Published August 2012

bit.ly/clean-architecture



Clean Architecture

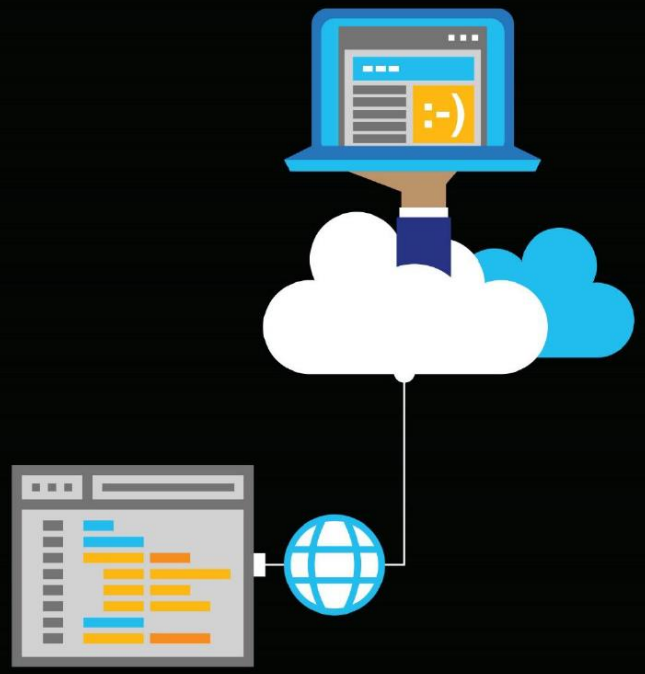
By Robert C. Martin

Author of Clean Code & Clean Coders

bit.ly/clean-architecture-book

Microsoft

Architecting Modern Web Applications with ASP.NET Core and Microsoft Azure



Steve Smith

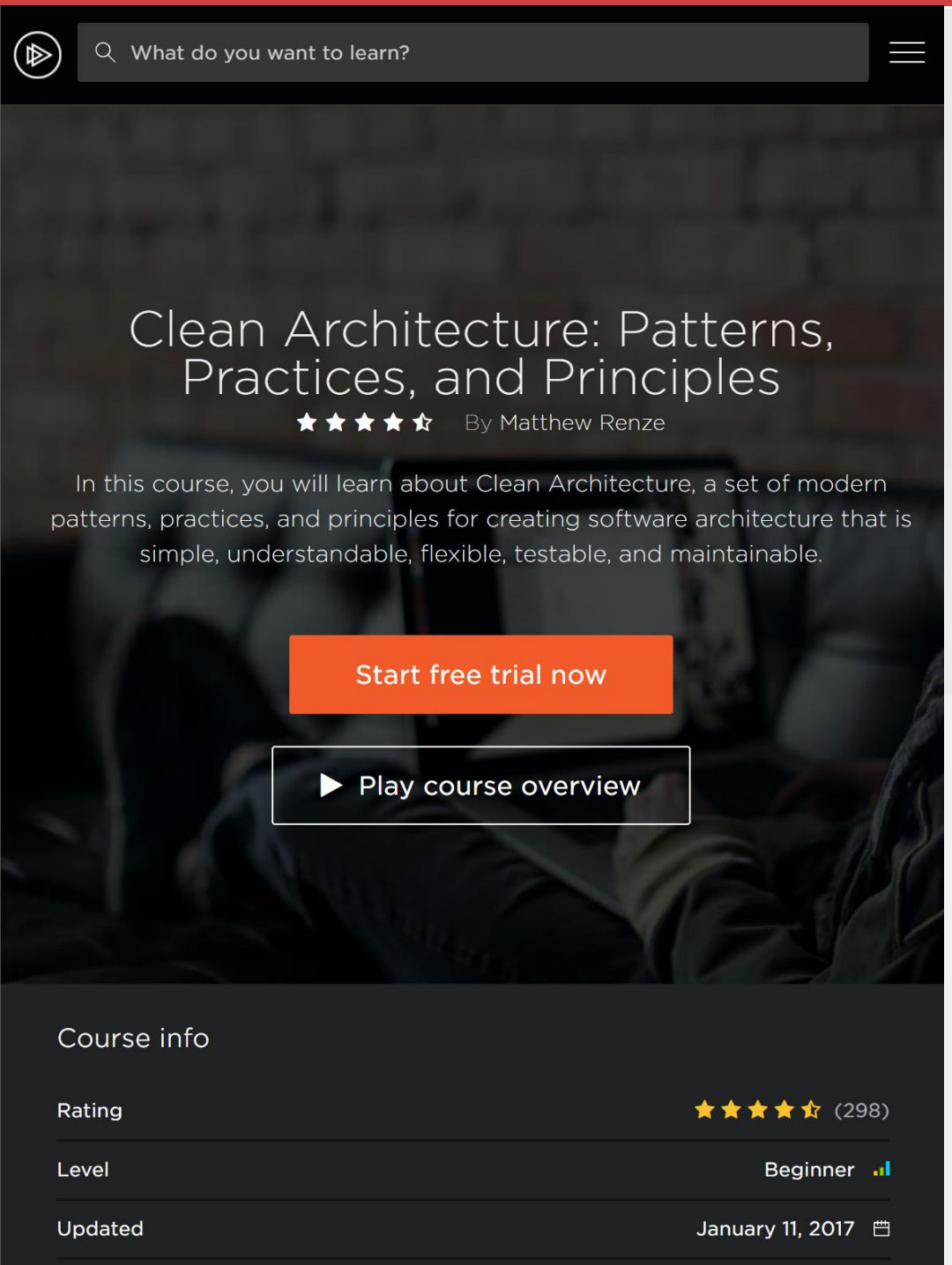
Architecting Modern Web Applications with ASP.NET Core and Azure

By Steve Smith

Microsoft Architecture Guidance

aka.ms/WebAppEbook

github.com/ardalis/CleanArchitecture



Clean Architecture: Patterns, Practices, and Principles

By Matthew Renze

Pluralsight Course

bit.ly/clean-architecture-course

bit.ly/clean-architecture-demo

Swashbuckle

Dev Superpowers Tour

Demo: Installing Swashbuckle

Installing, configuring and using Swagger and Swashbuckle

Unit Testing

Dev Superpowers Tour

Unit Testing

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

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

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

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

Typical Approach

Remove dependencies on EF

Implement abstractions

Create test doubles

Write unit tests

Simplified Approach

~~Remove dependencies on EF~~

~~Implement abstractions~~

~~Create test doubles~~

Write unit tests

Demo: Unit Testing

Writing unit tests for systems that depend on EF Core

EF Core InMemory Provider

Just a database provider

Uses in-memory database

Built for testing purposes

No overhead of I/O operations

Lightweight with minimal dependencies

Limitations

InMemory is not a relational database provider

Constraints won't be enforced, you can violate referential integrity

For unit testing, this does not matter

Concerns

You're writing integration tests, not unit tests

Concerns

Lack of isolation, e.g. tests are exercising code with dependencies on EF

Concerns

Unit of work and repository patterns

are best practice!

Testing with InMemory

2016-10-27 • 3 min to read • Contributors

Note

This documentation is for EF Core. For EF6.x, see Entity Framework 6.

The InMemory provider is useful when you want to test components using something that approximates connecting to the real database, without the overhead of actual database operations.

Tip

You can view this article's sample on GitHub.

InMemory is not a relational database

EF Core database providers do not have to be relational databases. InMemory is designed to be a general purpose database for testing, and is not designed to mimic a relational database.

Some examples of this include:

- InMemory will allow you to save data that would violate referential integrity constraints in a relational database.

Microsoft Docs - Testing with InMemory - bit.ly/efcore-inmemory

NorthwindTraders

Northwind Traders is a sample application built using ASP.NET Core and Entity Framework Core.

The initial purpose of this solution is to create a code first version of Northwind, using the .NET Core stack.

Getting Started

Northwind Traders EF Core - bit.ly/efcore-northwind

Use these instuctions to get the project up and running.

Key Points

- ✗ Don’t remove dependencies on EF
- ✗ No need to implement abstractions
- ✗ Don’t create test doubles
- ✓ Just write unit tests

Validation

Dev Superpowers Tour

Demo: Using Fluent Validation

Installing and configuring Fluent Validation

Logging & Exceptions

Dev Superpowers Tour

Overview

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

Logging with ASP.NET Core

Centralised Logging with Seq

Seq is a powerful log server

Centralised, supports structured logging

Supports ASP.NET Core and others

Easy to install and configure

Demo: Centralised Logging

Adding centralised logging to ASP.NET Core with Seq

Security

Dev Superpowers Tour

Security Choices

Windows Authentication

ASP.NET Core Identity

Identity Server 4

Windows Authentication

Easy to Implement (Hosted on windows)

Good option for Intranet

Can use reverse proxy such as F5 to publish to internet

No System.DirectoryServices (yet)

ADFS can be support OIDC and OAuth2

Startup.cs

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

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 & OpenID

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
```

Startup.cs

```
public void Configure(IApplicationBuilder app,
    IHostingEnvironment env,
    ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole(Configuration.GetSection("Logging"));
    loggerFactory.AddDebug();

    app.UseStaticFiles();

    app.UseIdentity();

    app.UseMvc(routes =>
```

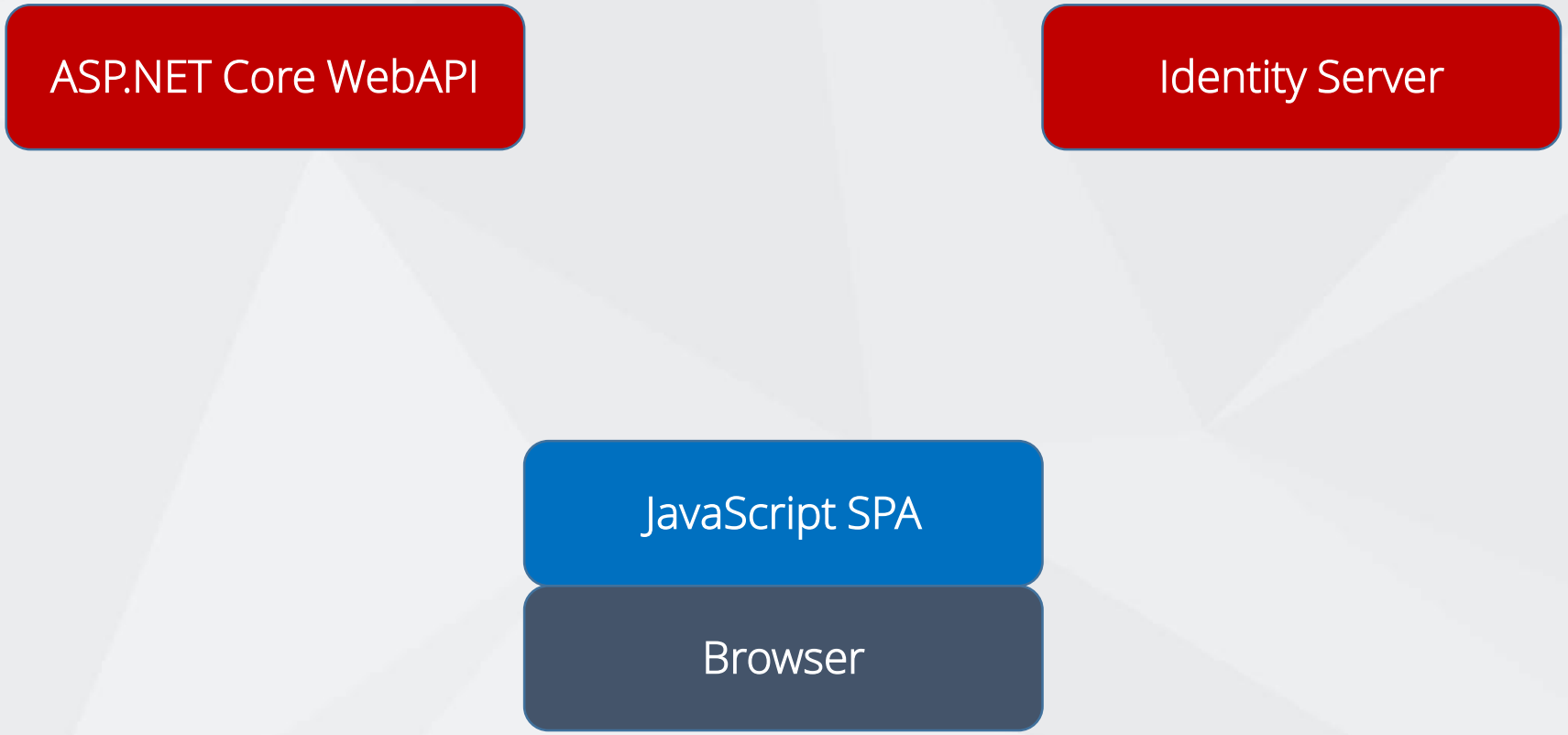
Identity Server 4

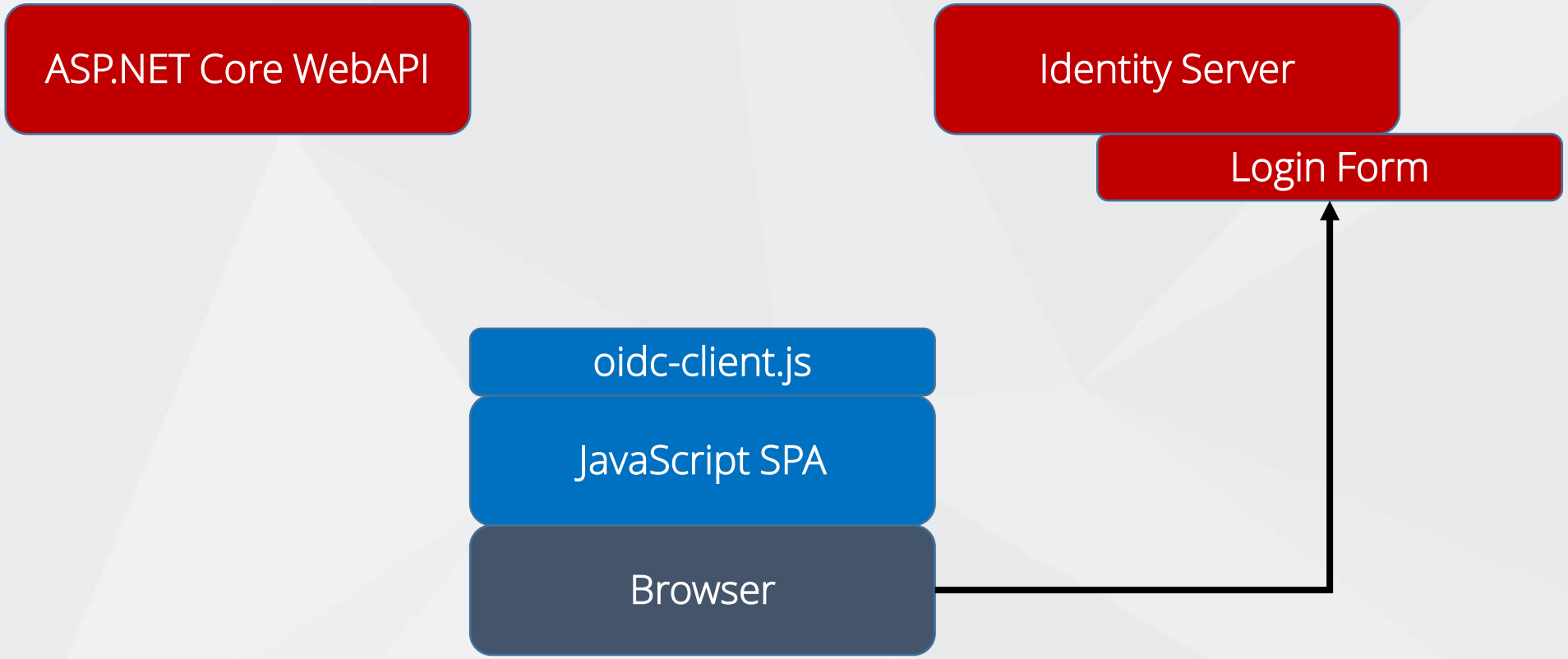
Full Single Sign On solution

Service that provides OpenId Connect and OAuth 2.0

Implemented in .NET Core

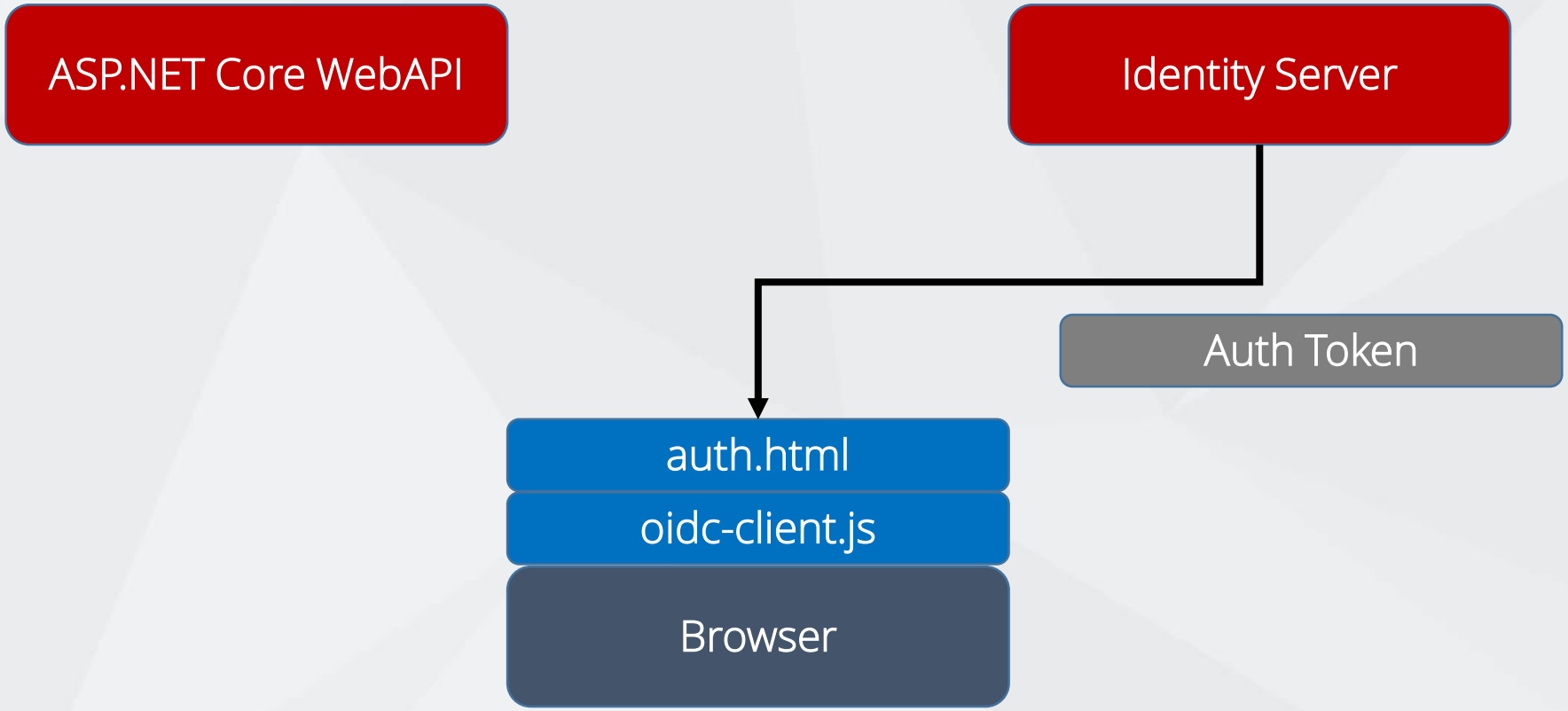
Example: JS application with .NET Core WebAPI


ASP.NET Core WebAPI
Identity Server
JavaScript SPA
Browser



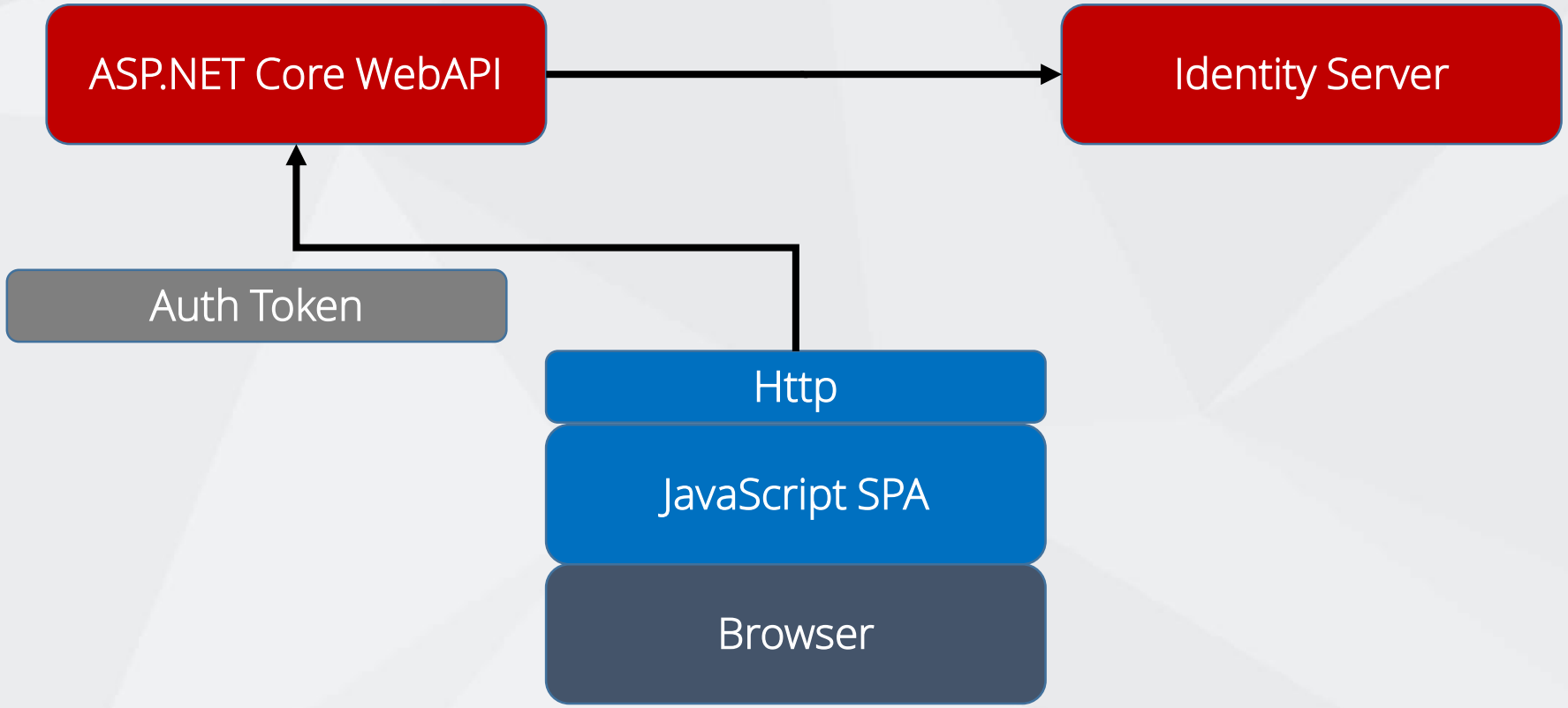
OIDC client redirects browser to a login page on the Identity Server.

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

Deployment

Dev Superpowers Tour

Overview

Framework-Dependent Deployment (FDD)

Self-Contained Deployment (SCD)

Demos

Framework-Dependent Deployment (FDD)

Default deployment model for .NET Core apps

Deploy your app and third-party dependencies

Uses shared system-wide version of .NET Core

Framework-Dependent Deployment (FDD)

- ✅ No need to define target operating systems
- ✅ Small deployment package
- ✅ Multiple apps share same .NET Core installation
- ❌ Requires compatible version of .NET Core on host system
- ❌ .NET Core installation changes could impact your app

Demo: FDD with .NET Core CLI

Creating a framework-dependent deployment with the .NET Core CLI

Self-Contained Deployment (SCD)

Deploy your app, third-party dependencies, and .NET Core

Supports side-by-side versioning of .NET Core

Self-Contained Deployment (SCD)

- ✅ Complete control of .NET Core version
- ✅ Target system can run your .NET Core app
- ❌ Must select target platforms in advance
- ❌ Size of deployment package is large
- ❌ Numerous self-contained apps can consume significant space on target system

Demo: SCD with .NET Core CLI

Creating a self-contained deployment with the .NET Core CLI

Feedback + Q & A

Dev Superpowers Tour

Thank you!

Code & Slides bit.ly/dotnetcoretour

info@ssw.com.au

www.ssw.com.au

Sydney | Melbourne | Brisbane | Adelaide