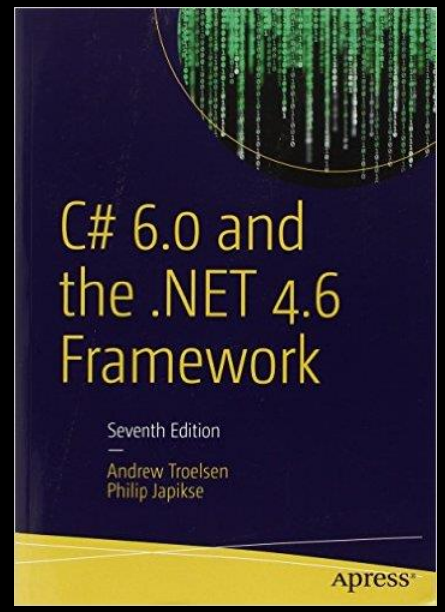
ASP.NET MVC6 SERVICES AND EF CORE 1

Philip Japikse (@skimedic)
skimedic@outlook.com
www.skimedic.com/blog
Microsoft MVP, ASPInsider, MCSD, MCDBA, CSM, CSP
Principal Consultant/Architect, Strategic Data Systems

Phil.About()

- Principal Consultant/Architect, Strategic Data Systems
  - http://www.sds-consulting.com
- Developer, Coach, Author, Teacher
  - http://bit.ly/pro_csharp
- Microsoft MVP, ASPInsider, MCSD, MCDBA, CSM, CSP
- Founder, Agile Conferences, Inc.
  - http://www.dayofagile.org
- President, Cincinnati .NET User's Group



.NET CORE 1

WHAT IS .NET CORE?

- Open source, cross platform development platform for:
  - Windows, macOS, Linux
  - Can be ported to other platforms
- Compatible with
  - .NET f/w, Xamarin, Mono
- Full Command line support
- Consists of (to name a few):
  - .NET Core – console
  - Entity Framework Core
  - ASP.NET Core
  - Windows 10 Universal Windows Platform (UWP)
  - Xamarin.Forms

ENTITY FRAMEWORK CORE 1

WHAT IS ENTITY FRAMEWORK CORE 1

- Newest version of Entity Framework - complete re-write from EF 6.x
- Lightweight, Modularized
- Cross Platform (built on .NET Core)

- Just released as “RTM”
  - Still missing features from EF 6.x
  - Check http://bit.ly/ef_compare to see the current status

NEW FEATURES IN EF CORE 1

- Shadow State Properties
- Alternate Keys
- Client based key generation
- Mixed client/server evaluation - Be careful!
- Creating Raw SQL Queries with LINQ
- New DB Providers – InMemory, Azure (PT), Redis (PT)

(SOME) MISSING FEATURES IN CURRENT VERSION OF EF CORE 1

- EDMX Designer
  - Not coming back!
- Alternate inheritance mapping patterns
  - Implemented: Table Per Hierarchy (TPH)
  - Missing: Table Per Type (TPT), Table Per Concrete Type (TPC)
- Complex/Value types
- Spatial Data Types
- Find
- Lazy/Explicit loading
- Command Interception
- Connection Resiliency

SO, WHICH VERSION?

Entity Framework Core 1

- Faster*
- Support ASP.NET Core/UWP
- Lighter weight install
- Don’t need missing features

Entity Framework 6.x

- Existing EF 6.x Implementation
- Only target Wintel deployment
- Need features missing in Core
  - Connection Resiliency
  - Command Interception (WPF)

CHANGES FROM EF6.1

- No more EDMX models - only “Code First”
- Project.json (for now)
- Context Configuration
- Migrations
- Mixed mode evaluation
- Database Initializers (or lack thereof)

PROJECT.JSON

PROJECT.JSON

- Currently holds configuration information
  - Going away in a future version of VS/.NET Core
- Base settings:

```
{
  "version": "1.0.0-*",
  "dependencies": {
     "Microsoft.NETCore.App": {"type": "platform", "version": "1.0.0" },
  "NETStandard.Library": "1.6.0",
  "System.Linq.Queryable": "4.0.1",
  "SpyStore.Models": {"target": "project", "version": "1.0.0-*“}
  },
  "frameworks": {
    "netcoreapp1.0": {"imports": ["dnxcore50"]}
  },
}
```

ADDING EF DEPENDENCIES

```
{
  "dependencies": {
      "Microsoft.EntityFrameworkCore": "1.0.0",
      "Microsoft.EntityFrameworkCore.Design": {
          "version": "1.0.0-preview2-final", "type": "build"
      },
    "Microsoft.EntityFrameworkCore.Relational": "1.0.0",
    "Microsoft.EntityFrameworkCore.SqlServer": "1.0.0",
    "Microsoft.EntityFrameworkCore.SqlServer.Design": "1.0.0",
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final",
    "System.Linq.Queryable": "4.0.1",
  },
  "tools": {
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final"
  },
}
```

ADDING PROJECT DEPENDENCIES

- Specify project as target (instead of package) to indicate local project

```
{
   "dependencies": {
   "SpyStore.Models": {
      "target": "project",
      "version": "1.0.0-*"
    }
  },
}
```

CONFIGURING EF CONTEXT

CONFIGURATION

- Uses DbContextOptionsBuilder
  - instead of just name of connection string
- Constructor Injection

```
public SpyStoreContext(DbContextOptions<SpyStoreContext> options):base(options) {}
```

- OnConfiguring

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseSqlServer($“{connectionString}");
    }
}
```

MIGRATIONS

EF CORE 1 MIGRATIONS

- No longer stores a hash in the DB
  - <Context>ModelSnapshot.cs

- Run from the command line (or package manager console)
  - dotnet ef migrations [options] [add || list || remove || script]
  - dotnet ef database update [update || drop] [options]

- Reverse Engineer a Database:
  - dotnet ef dbcontext scaffold [arguments] [options]

MIGRATIONS

- Full support of multiple contexts
- <Context>ModelSnapshot holds current DDL
- Migration table only stores class name and version for each applied migration
  - Hash of database was removed

EF CORE 1 MULTI-CONTEXT MIGRATIONS

- Specify:
  - Output directory
  - Context (fully qualified)

- Ignore changes isn't supported!
  - Must edit the Up method

```
dotnet ef migrations add initial
  -o EF\Migrations
  -c SpyStore.DAL.EF.SpyStoreContext

dotnet ef database update initial
  -c SpyStore.DAL.EF.SpyStoreContext
```

MIGRATIONS ISSUE WITH CURRENT VS TOOLING

- Current Version of Tooling Doesn't Support Migrations on class libraries
- Must add "emitEntryPoint in project.json and Main in Program.cs

```
"buildOptions": {
   "preserveCompilationContext": true,
   //set this to true to run migrations
   "emitEntryPoint": true
 },
```

MIXED MODE EVALUATION

CLIENT SIDE EVALUATION

- Allows for .NET functions to be added to LINQ statements
  - Data is pulled into the client to execute the .NET function
- Mixed mode evaluation is enabled by default
  - Applies to the entire context – can't be changed per query
- Can't "disable" – can only cause EF to throw exceptions
  - Automate testing is vital for discovery
  - Single[OrDefault] – Client side
  - First[OrDefault] – Server side

DISABLING CLIENT SIDE EVALUATION

- OnConfiguring

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseSqlServer($“{connectionString}")
          .ConfigureWarnings(warnings =>
              warnings.Throw(RelationalEventId.QueryClientEvaluationWarning));
    }
}
```

DATA INITIALIZATION

DATA INITIALIZERS

- EF Core missing key features for initialization (EF 6 Style)
  - DropCreateDatabase[Always || IfModelChanges]
    - Automatic execution
  - Seed method on migrations
  - AddOrUpdate
- Must roll your own
  - Database.EnsureDeleted
  - Database.EnsureCreated || Migrate

DEMO

Entity Framework Core 1

ASP.NET CORE MVC 6 WEB SERVICES

ASP.NET AND WEB API COMPLETELY RETOOLED

- Based on .NET Core
  - No longer dependent on System.Web.dll or IIS
  - Cross platform capable
  - Side by Side versioning
- MVC and Web API Unified into the same framework
- Built on
  - Dependency Injection
  - Environment based configuration
- Default pipeline == NOTHING!
  - Fully customizable for your application needs

CHANGES FROM PREVIOUS VERSIONS

- Configuration and pipeline management no longer use:
  - global.asax, web.config, app_start
  - Instead uses json and startup.cs
- Controllers derive from Controller (not ApiController)
  - IHttpActionResult is now IActionResult
- Routing
  - Attribute-based used by default
- Verbs
  - Must add verb attribute to actions
    - HttpGet is still default
- JSON handling - camelCased by default

ANATOMY OF AN MVC6 APP

- A console app that creates a web server in Main

```
var host = new WebHostBuilder()
                .UseKestrel()
                .UseUrls("http://*:40001/")
                .UseContentRoot(Directory.GetCurrentDirectory())
                .UseIISIntegration()
                .UseStartup<Startup>()
                .Build();
host.Run();
```

- Startup class configures pipeline and services

THE STARTUP CLASS

GETTING CONFIGURATION INFORMATION

- Configuration uses json files and the Configuration Builder class

```
public Startup(IHostingEnvironment env)
{
    _env = env;
    var builder = new ConfigurationBuilder()
        .SetBasePath(env.ContentRootPath)
        .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
        .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true)
        .AddEnvironmentVariables();
    Configuration = builder.Build();
}
```

CONFIGURING THE PIPELINE

- The Configure method sets up how to respond to HTTP requests
  - Also used to execute data initializer (EF Core)

```
public void Configure(
    IApplicationBuilder app,
    IHostingEnvironment env,
    ILoggerFactory loggerFactory)
{
      loggerFactory.AddConsole(Configuration.GetSection("Logging"));
      loggerFactory.AddDebug();
      // has to go before UseMvc to show Headers in Response as well as preflight.
      app.UseCors("AllowAll");
      app.UseMvc();
      StoreDataInitializer.InitializeData(app.ApplicationServices);
}
```

CONFIGURING FRAMEWORK SERVICES

- Used to configure any services needed by the application

```
public void ConfigureServices(IServiceCollection services)
{
        // Add framework services.
        services.AddMvc(); //Configuration options will be discussed later
        services.AddCors(options =>
        {
                options.AddPolicy("AllowAll", builder =>
                {
                        builder.AllowAnyHeader()
                                .AllowAnyMethod()
                                .AllowAnyOrigin()
                                .AllowCredentials();
                });
        });
}
```

CONFIGURING DI SERVICES

- Used to configure any services needed by the application
  - Transient – created each time they are requested
  - Scoped – created once per request
  - Singleton – created once (use this instead of implementing singleton dp

```
public void ConfigureServices(IServiceCollection services)
{
        services.AddDbContext<SpyStoreContext>(options => options
                .UseSqlServer(Configuration["Data:ConnectionString"]));

        services.AddScoped<ICategoryRepo, CategoryRepo>();
        services.AddScoped<IProductRepo, ProductRepo>();
        services.AddScoped<ICustomerRepo, CustomerRepo>();
        services.AddScoped<IShoppingCartRepo, ShoppingCartRepo>();
        services.AddScoped<IOrderRepo, OrderRepo>();
        services.AddScoped<IOrderDetailRepo, OrderDetailRepo>();
}
```

JSON HANDLING

- JSON results are camel cased by default (new in RTM)

```
[{"categoryName":"Communications","products":[],"id":2,"timeStamp":"AAAAAAAFKFE="}]
```

- Can revert to Pascal casing in Startup

```
services.AddMvc(config => config
  .AddJsonOptions(options =>
  {
      options.SerializerSettings.ContractResolver = new DefaultContractResolver();
  });
```

CONTROLLERS

CONTROLLERS

- Derive from Controller (not APIController)
- Provides key helper methods to return responses
  - View
  - HttpStatusCode (e.g. return BadRequest())
  - Formatted (e.g. return Json(customer))
  - Content negotiated response
    - (e.g. Ok, Created, CreatedAtRoute, CreatedAtAction)
  - Redirect (e.g. LocalRedirect, RedirectToAction, RedirectToRoute)

ROUTING

ROUTING

- Attribute routing by default
  - Applied at Controller and Action levels
  - Can still use RouteMap (and do in MVC6 for default route)
- Route attribute defines base route for controller

```
[Route("api/[controller]/{customerId}")]
public class ShoppingCartController : Controller
{
}
```

ROUTING

- Http Verb can build on default controller route tables

```
[HttpGet("{productId}",Name = "GetShoppingCartRecord")]
public CartRecordWithProductInfo GetOne(int customerId, int productId)

[HttpGet(Name = "GetShoppingCart")]
public IEnumerable<CartRecordWithProductInfo> GetAll(int customerId)

[HttpPost] //required even if method name starts with "Post"
public IActionResult Create(int customerId, [FromBody] ShoppingCartRecord item)

[HttpPut("{shoppingCartRecordId}")] //Required even if method name starts w\Put
public IActionResult Update(int customerId, int shoppingCartRecordId, [FromBody] CartRecord item)

[HttpDelete("{id}/{timeStamp}")] //Required even if method name starts w\Delete
public IActionResult Delete(int customerId, int id, string timeStamp)

[HttpPost("buy")] //required even if method name starts with "Post"
public IActionResult Purchase(int customerId, [FromBody] Customer customer)
```

DEPENDENCY INJECTION

INJECTING SERVICES INTO CONTROLLERS

- Dependencies are configured in Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
    //https://docs.asp.net/en/latest/fundamentals/dependency-injection.html
    services.AddScoped<IShoppingCartRepo, ShoppingCartRepo>();
}
```

- Instances are pulled in with FromServices attribute

```
public ShoppingCartController([FromServices]IShoppingCartRepo repo)
{
   Repo = repo;
}

public IShoppingCartRepo Repo { get; set; }
```

FILTERS

FILTERS

- *Filters* in ASP.NET MVC allow you to run code before or after a particular stage in the execution pipeline.
  - Filters can be configured globally, per-controller, or per-action.
- Many types available
  - Action
  - Exception
  - Authorization
  - Resource
  - Result

EXCEPTION FILTERS

- Come into play on unhandled exceptions

```
public class SpyStoreExceptionFilter : IExceptionFilter
{
  public void OnException(ExceptionContext context)
  {
    var ex = context.Exception;
    var response = new ErrorMessage { Error = "General Error.", Message = ex.Message };
    context.Result = new ObjectResult(response) {StatusCode = 500};
  }
}
```

- Configured with MVC (for whole application filters)

```
services.AddMvc(config => config
  .Filters.Add(new SpyStoreExceptionFilter(_env.IsDevelopment())))
  .AddJsonOptions(options =>
  {
      options.SerializerSettings.ContractResolver = new DefaultContractResolver();
  });
```

DEMO

MVC6 Web Services

Questions?

Contact Me

phil@sds-consulting.com
www.sds-consulting.com

skimedic@outlook.com
www.skimedic.com/blog
www.twitter.com/skimedic
www.hallwayconversations.com

www.about.me/skimedic

Thank You!

Find the code at: https://github.com/skimedic/DotNetCore