HL7 FHIR DevDays 2017

HL7® FHIR®

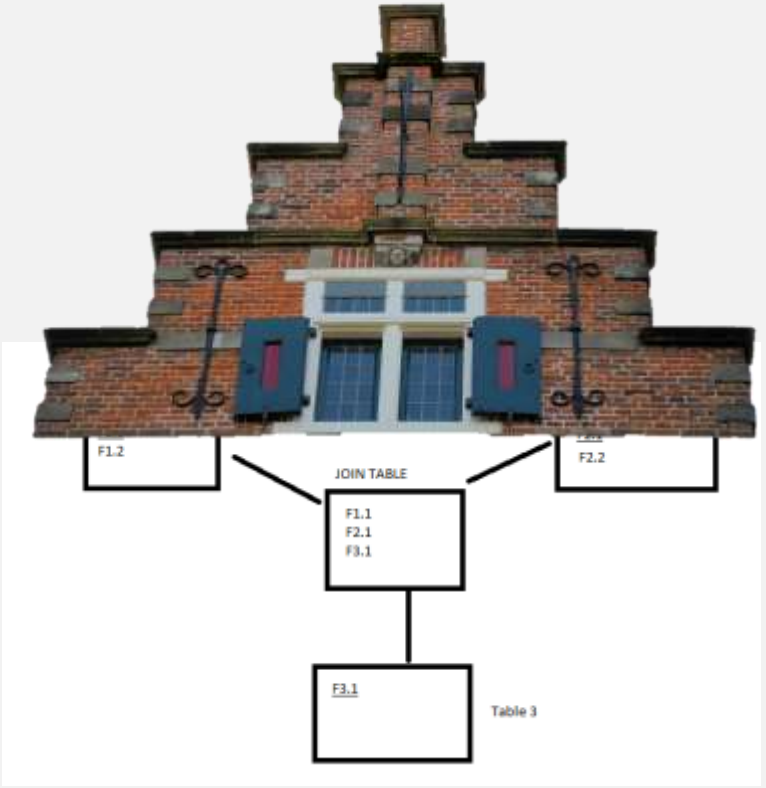
Build your own Vonk FHIR Facade

Christiaan Knaap, Furore

Amsterdam, 15-17 November | @fhir_furore | #fhirdevdays17 | www.fhirdevdays.com

FHIR Facade?

Implementation of a
FHIR RESTful API
as an interface
to an existing
backend system.



Prerequisites

For this session

.NET programming

- ASP.NET Core
- LINQ

FHIR RESTful API

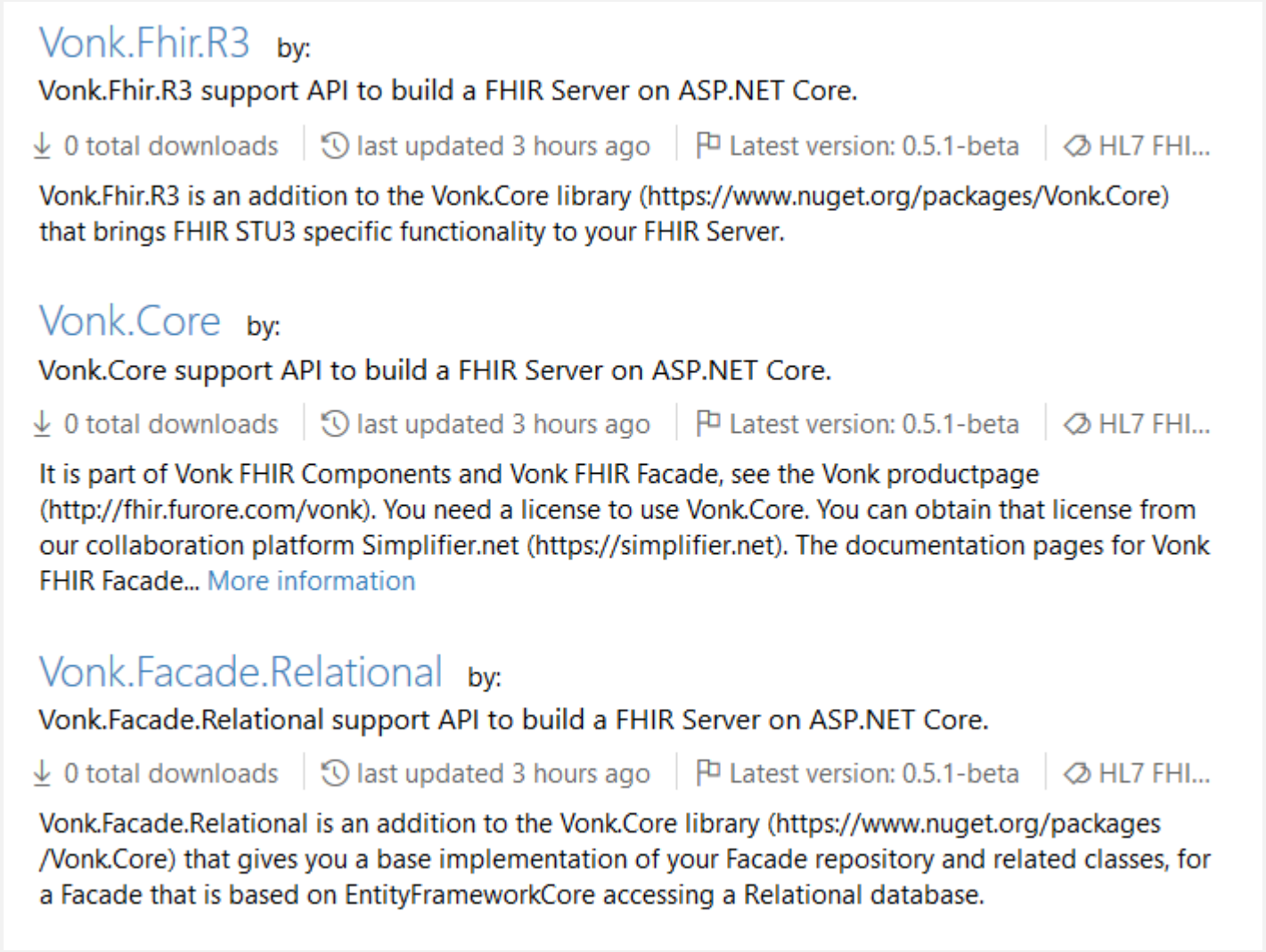
For the exercise

- Git client
- Visual Studio 2017
- .NET Core 2.0 SDK
- SQL Server >= 2012
- Postman / Fiddler

Vonk FHIR Facade?

Set of NuGet packages to build a FHIR Facade with the ASP.NET (Core) web stack

(NuGet package = .NET library)



Programming?

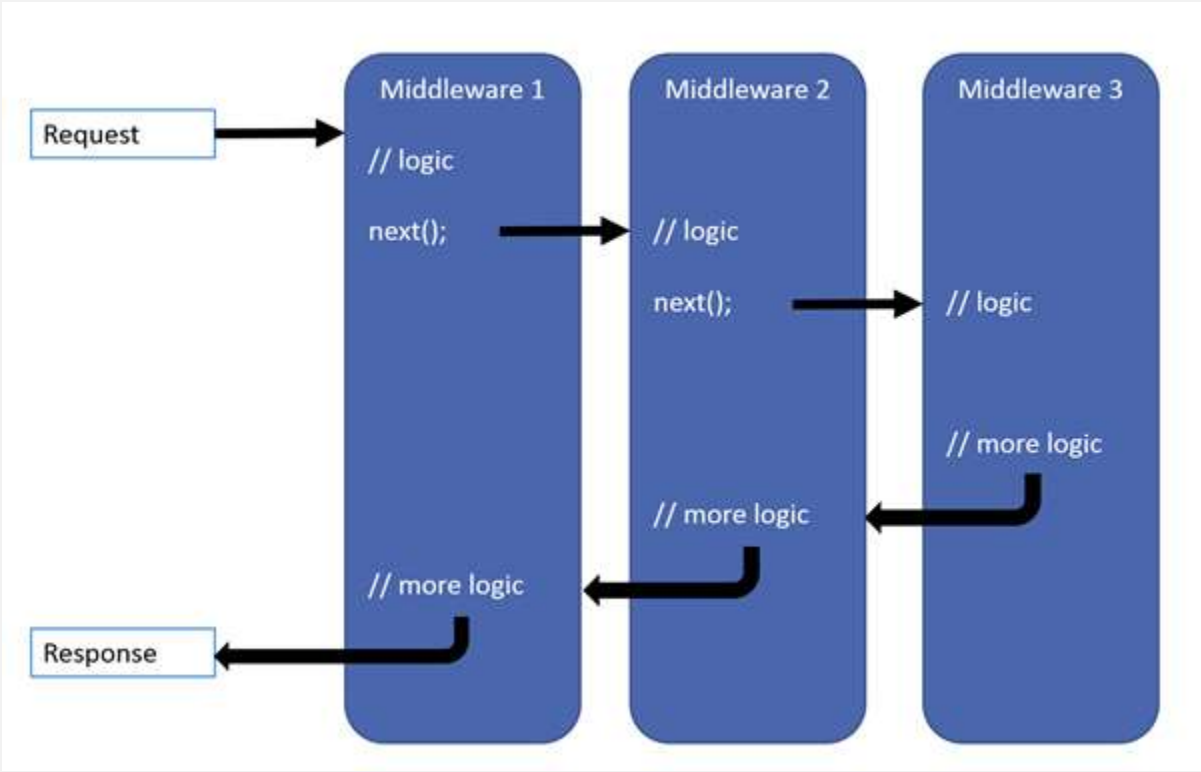
- Maximum flexibility
- Fast
- Collect common patterns
- Provide those in other formats
  - scripting
  - mapping tooling
  - code generation
- A growth model

ASP.NET Core

Pipeline of Middleware

Chain of function calls operating on HttpContext

- Request
- Response



ASP.NET Core

Middleware component

(Class with)

Invoke:

function on

HttpRequest

'next' (call to next middleware)

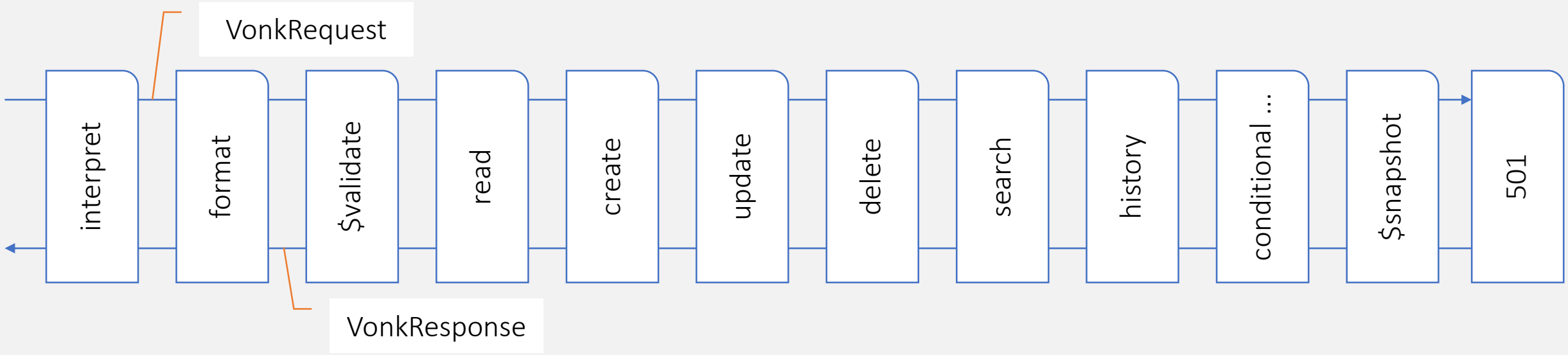
```
public class Middleware
{
  private readonly RequestDelegate _next;

  public Middleware(RequestDelegate next)
  {
    _next = next;
  }

  public async Task Invoke(HttpContext httpContext)
  {
     // do something on the request
     await _next(httpContext);
     // do something on the response
  }
}
```

ASP.NET Core Dependency Injection

- Inversion of Control is key in ASP.NET Core
- Declare your dependencies in your constructor
- Register your services in ConfigureServices()
- Pay attention to scopes
  - Transient
  - Scoped (Request)
  - Singleton

Vonk Pipeline

ASP.NET Core Pipeline with Middleware components implementing parts of the FHIR RESTful API



Vonk Layered architecture

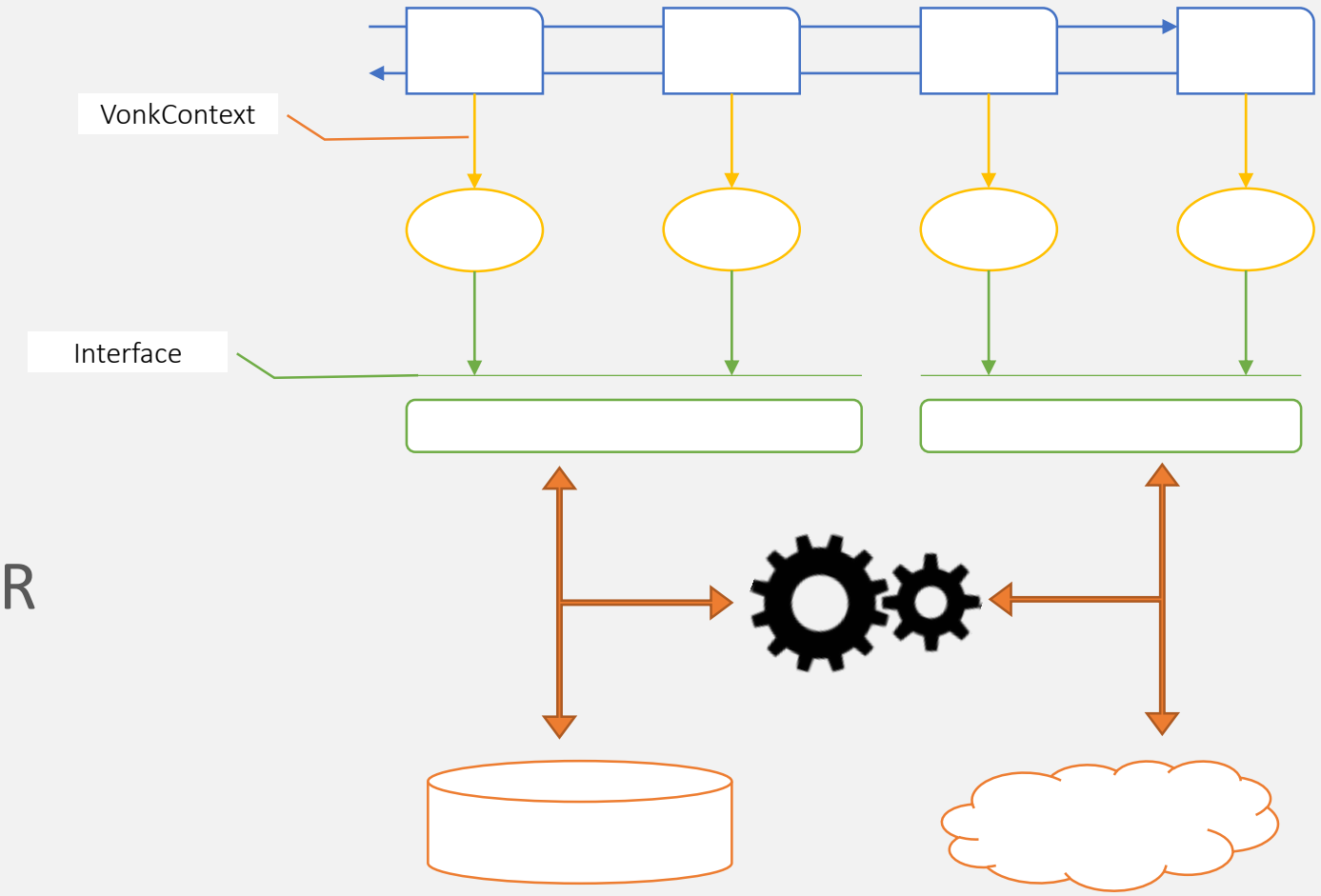
Pipeline of Middleware

calling Services

by using Repositories

that Map data to and from FHIR

and access the backend

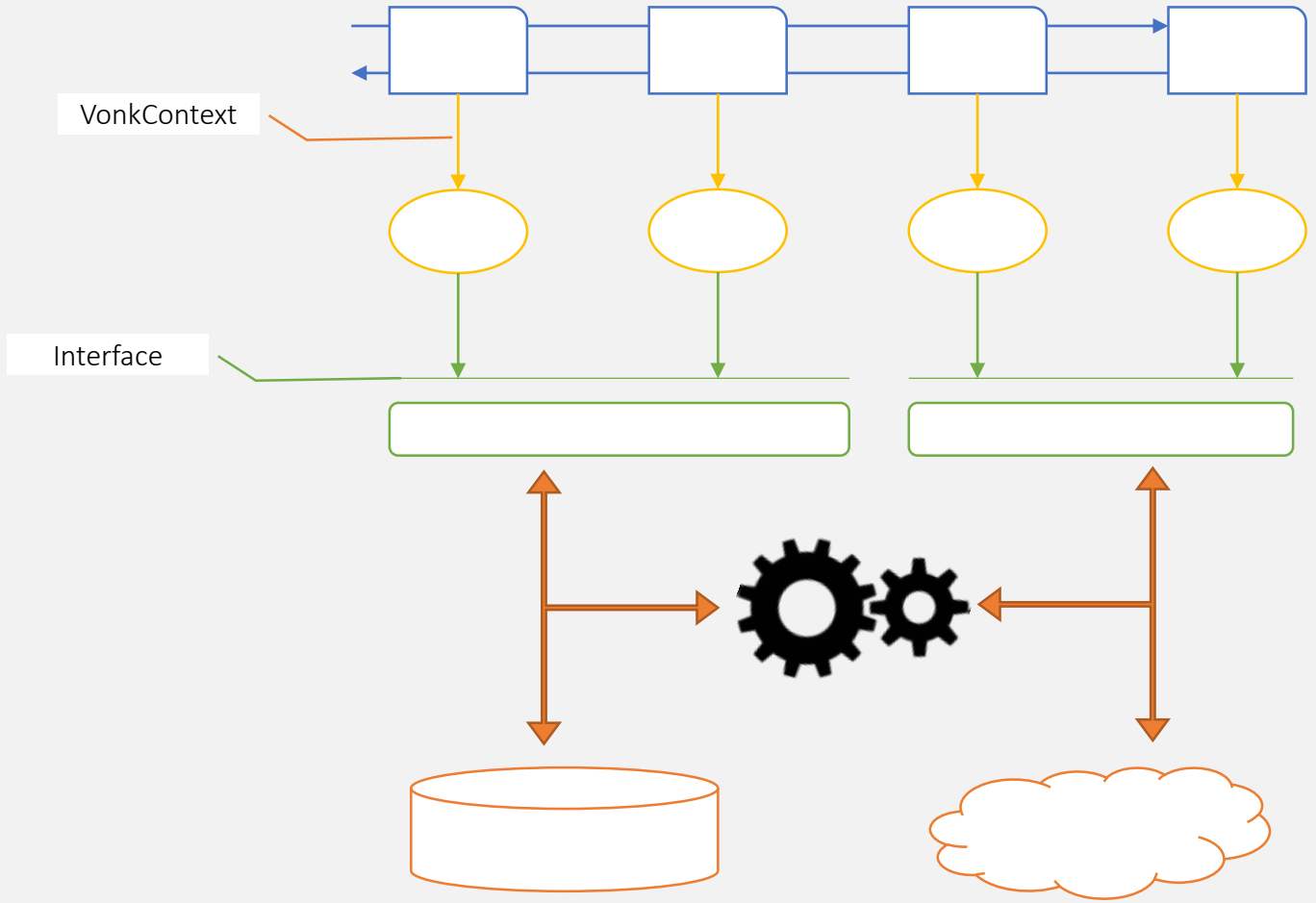


A bit of Vonk and a touch of your own

You get

You build

You have



Tour of components

- For background
  - IVonkContext
  - IResource
  - IConformanceContributor
- You implement
  - ISearchRepository
  - IChangeRepository

IVonkContext

Vonk specific ‘HttpContext’

- VonkRequest
  - interaction
  - resource
- Arguments
- VonkResponse
  - responsecode
  - resource
  - operationoutcome

```
public interface IVonkContext
    {
        IVonkRequest Request { get; }

        IVonkResponse Response { get; }

        IArgumentCollection Arguments { get; }

        Uri ServerBase { get; }

        string Description { get; }
    }
```

Arguments

Arguments from anywhere

- Path
- QueryString
- Form variables

And you can report issues on them

IResource

Abstraction of Resource

Provides metadata and status

IElementNavigator

for FhirPath navigation

Currently based on POCO

Future: cross-version; invalid resources

```
interface IResource
{
  Navigator { get; }
  Type { get; }
  Id { get; set; }
  Version { get; set; }
  LastUpdated { get; set; }
  Currency { get; set; }
  Change { get; set; }
  Contained { get; }
  ContainedResources
}
```

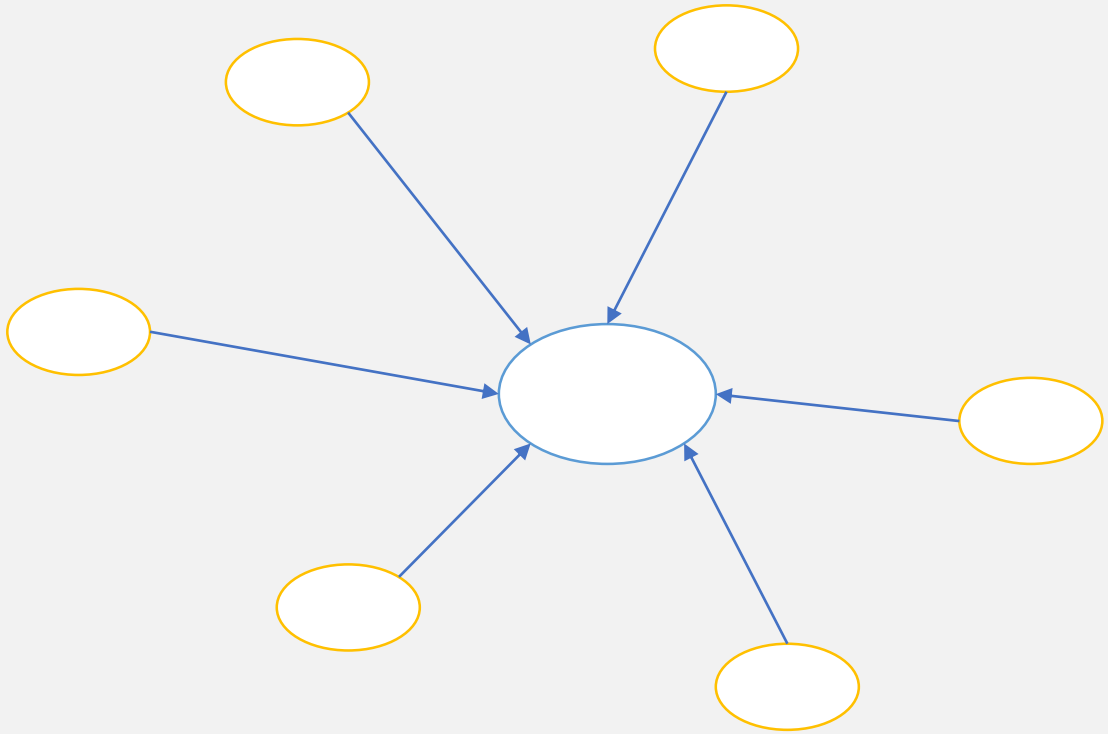
Express your capabilities

Tell what you can do

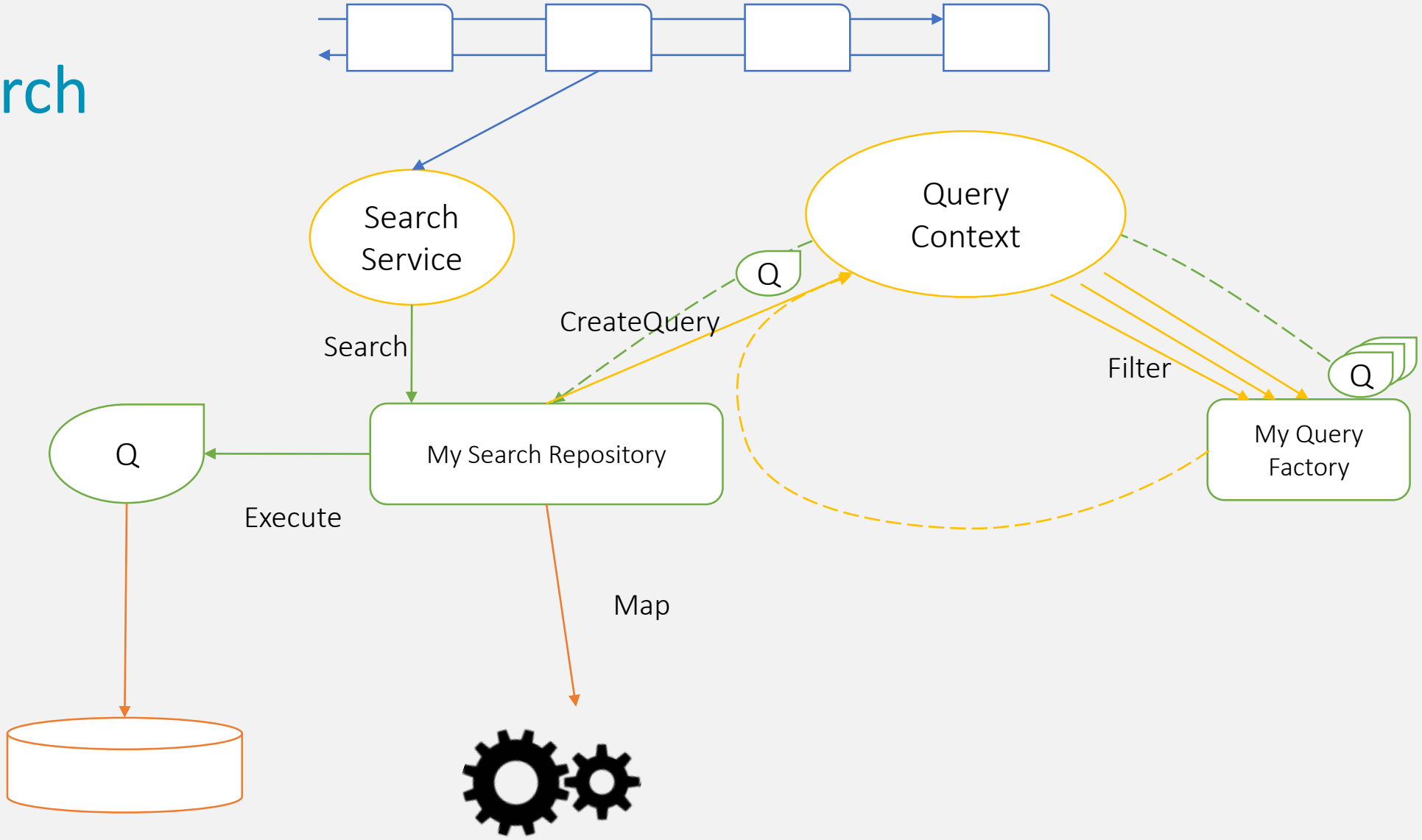
Contribute to the CapabilityStatement

With IConformanceContributor

Most of this is done for you



Search

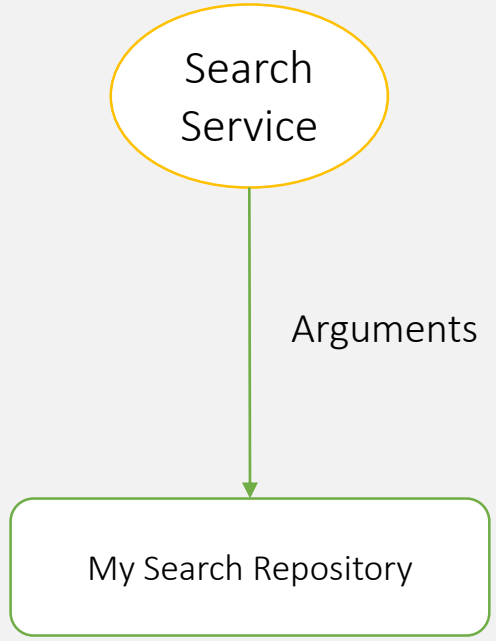

Search Service
Query Context
Q
CreateQuery
Search
Filter
Q
Q
My Search Repository
My Query Factory
Execute
Map

Arguments

You get the ‘raw’ Arguments

You can control them directly if you must

But usually you relay them...



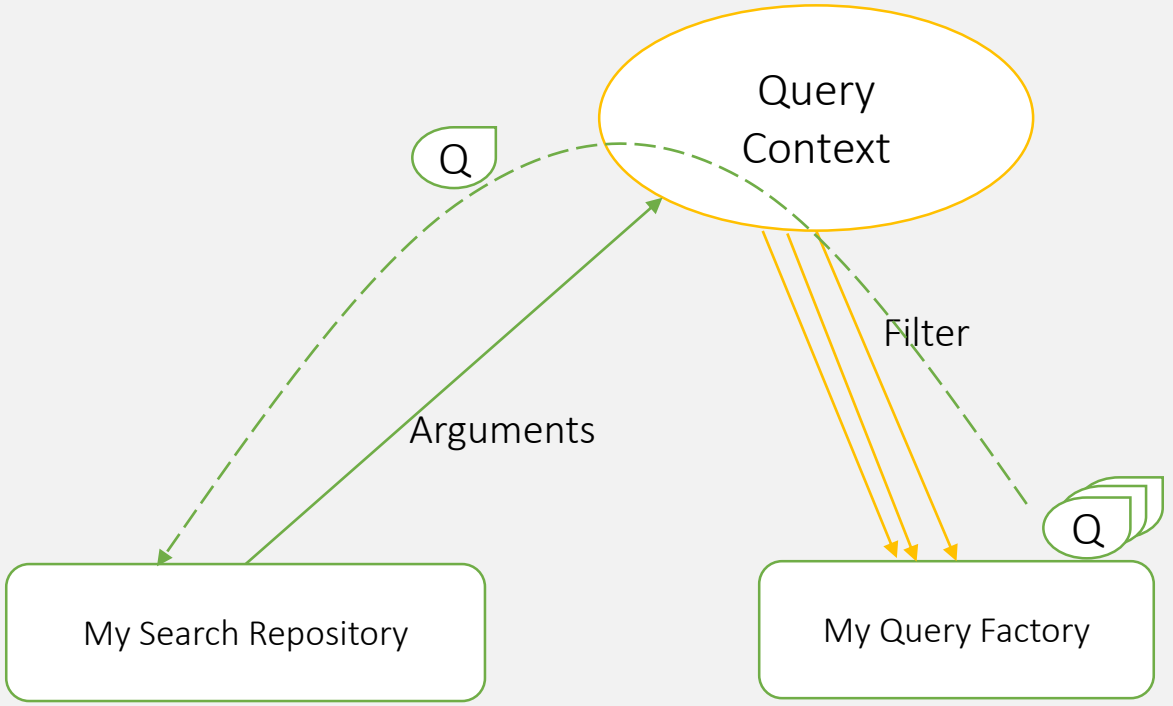
Querying

Vonk finds out WHAT to query for

- id (token)
- value (quantity)
- name (string)
- ... and calls Filter

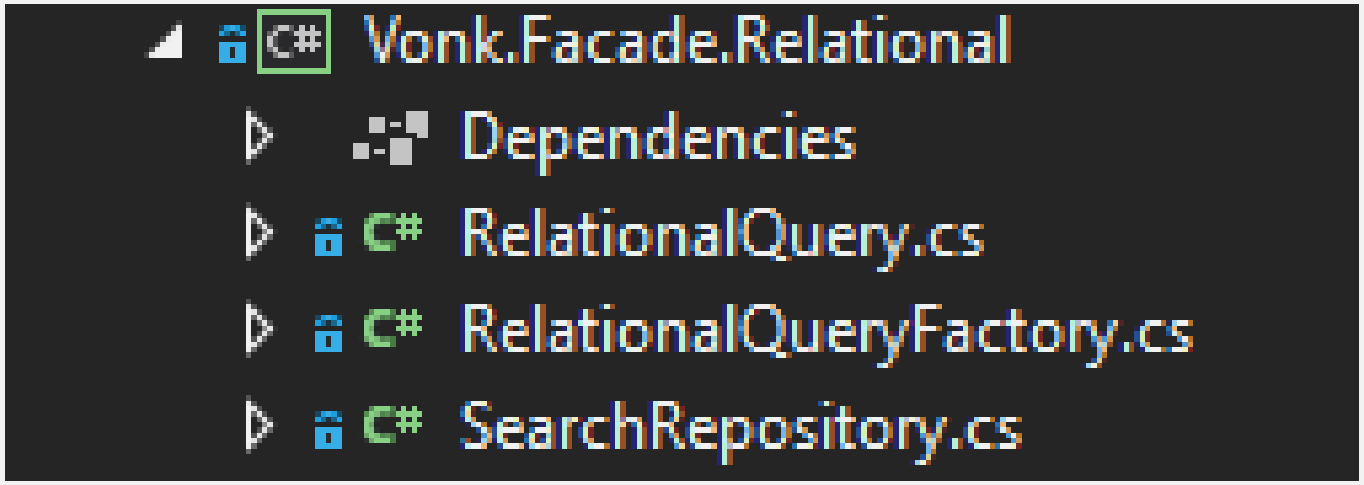
You control HOW you query

- SQL
- json
- http call
- ... and implement that in Q



Relational

Vonk comes with base classes for accessing Relational db with EntityFrameworkCore and LINQ



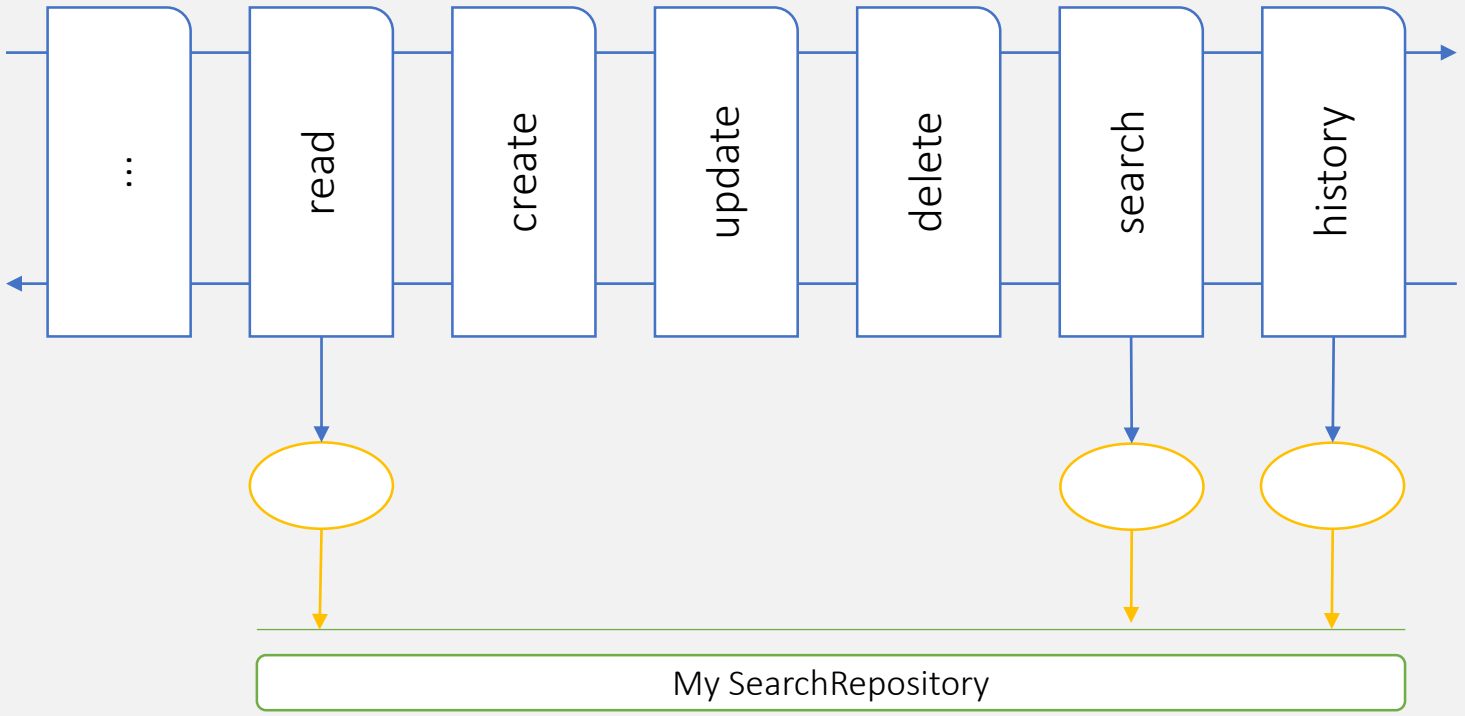
Reuse

ISearchRepository

can enable

all read-only access:

- read
- vread
- search
- history



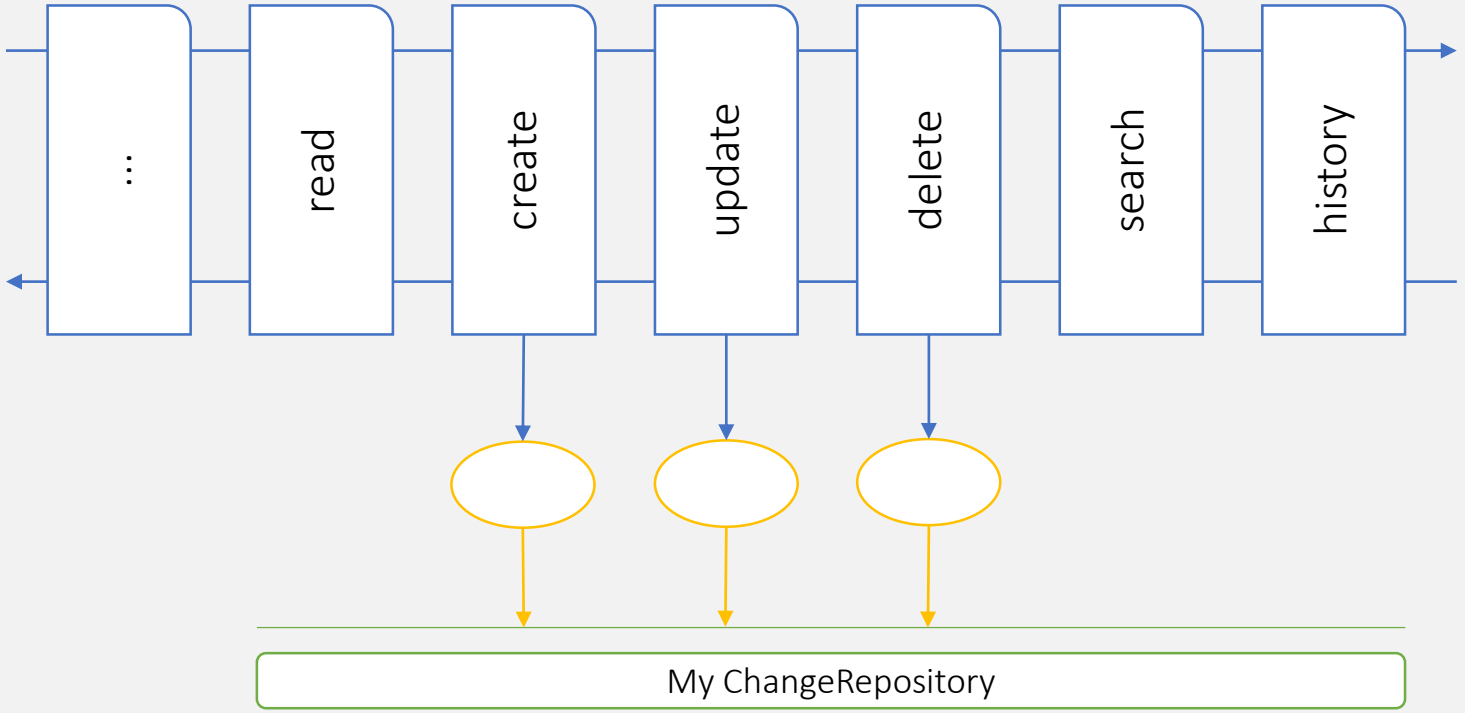
Change

IChangeRepository

can enable

all read-write access:

- create
- update
- delete
- conditional c/u/d
  (with ISearchRepository)



Dependency Injection

Vonk uses ASP.NET Core DI heavily

Register your:

- DbContext
- SearchRepository
- ChangeRepository
- ResourceMapper

A minimal server

Startup

AddFhirServices()

AddVonkMinimalServices()

UseContext()

UseVonkMinimal()

Add Validation

- UseValidationServices()
- AddValidation()

Readonly server

Override SearchRepository

+ RelationalQueryFactory

Write a mapping

your model -> FHIR

Startup

AddReadServices()

AddSearchServices()

UseRead()

UseSearch()

Exercise

- Go to http://docs.simplifier.net/vonk/facade/facadestart.html
- Arrange the prerequisites
- Find the completed example at: https://github.com/furore-fhir/Vonk.Facade.Starter