Pact FHIR Documentation

Release latest

Jan 15, 2019

Contents

FHIR solutions are built from a set of modular components called “Resources”. These resources can easily be assembled into working systems that solve real world clinical and administrative problems at a fraction of the price of existing alternatives. FHIR is suitable for use in a wide variety of contexts – mobile phone apps, cloud communications, EHR-based data sharing, server communication in large institutional healthcare providers, and much more.

CHAPTER 1

Getting Started

1.1 Installation

Pact.Fhir is compatible with .NET Standard 2.0.

You can install the packages via nuget

```
https://www.nuget.org/packages/Pact.Fhir.Core
https://www.nuget.org/packages/Pact.Fhir.Iota
```

1.2 UML

To get an abstract idea how the FHIR project is organized you might want to take a look at the design UML. It can be opened on draw.io https://github.com/PACTCare/Pact.Fhir/blob/master/docs/FHIR_Core_Design.xml

CHAPTER 2

Working with the Core

Right now the core is more or less a code skeleton, that will grow over time. The documentation will be expanded as needed.

The intend is, that interactors know all needed business rules to handle a FHIR operation. Therefore they will orchestrate all actions needed. Given that they are not tightly coupled to a data source, interactors are able to work with different ones. It is up to you to either use the IOTA implementation or implement your own FhirRepository.

2.1 Implementing your own FhirRepository

If you decide to start using the FHIR core with your own data source, what you need to do, is to implement the FhirRepository. For the sake of simplicity the documentation will use the in memory solution below:

```
public class InMemoryFhirRepository : FhirRepository
{
   public InMemoryFhirRepository()
   {
     this.Resources = new List<DomainResource>();
   }

   public List<DomainResource> Resources { get; }

   /// <inheritdoc />
   public override async Task<DomainResource> CreateResourceAsync(DomainResource␣
↪resource)
   {
     var resourceId = Guid.NewGuid().ToString("N");

     this.PopulateMetadata(resource, resourceId, resourceId);
     this.Resources.Add(resource);

     return resource;
   }
```

(continues on next page)

(continued from previous page)

```
    /// <inheritdoc />
    public override async Task<DomainResource> ReadResourceAsync(string id)
    {
      var resource = this.Resources.FirstOrDefault(r => r.Id == id);

      if (resource == null)
      {
        throw new ResourceNotFoundException(id);
      }

      return resource;
    }
}
```

2.2 Creating a resource

```
var interactor = new CreateResourceInteractor(new InMemoryFhirRepository());
var response = await interactor.ExecuteAsync(new CreateResourceRequest { Resource =␣
↪resource });
```

2.3 Reading a resource

```
var interactor = new ReadResourceInteractor(new InMemoryFhirRepository());
var response = await interactor.ExecuteAsync(new ReadResourceRequest { ResourceId =
↪"yourfhirresourcelogicalid" });
```

CHAPTER 3

Working with IOTA and FHIR

If you are not familiar with tangle.net (https://github.com/Felandil/tangle-.net) and IOTA you might want to read a little about it before starting here.

3.1 Working with resources

In the IOTA context, all resources are stored as restricted MAM streams where each resource has its own stream. Resources are assigned the first 64 chars of the corresponding MAM root as a logical/version id upon creation or alteration.

3.2 Keys and Channels

Since access to the MAM streams is restricted, the repository has to keep track of all channels. To get you started the below linked in memory solution should be enough. Anyway you will want to write your own implementation that stores such information in a trusted data storage.

See: https://github.com/PACTCare/Pact.Fhir/blob/develop/Pact.Fhir.Iota.Tests/Services/InMemoryResourceTracker.cs

3.3 Serialization

The resource payload is stored as a serialized JSON or XML depending which serializer implementation you favour. If JSON/XML is not what you are looking for, you can implement your own serializer by impelemting the IFhirTryteSerializer interface.

3.4 Code example

The IOTA FHIR repository is coupled to the tangle.net IOTA REST implementation.

See: https://github.com/PACTCare/Pact.Fhir/blob/develop/Pact.Fhir.Iota.Tests/Utils/IotaResourceProvider.cs

```
var repository = new IotaFhirRepository(IotaResourceProvider.Repository, new␣
↪FhirJsonTryteSerializer(), new InMemoryResourceTracker());
var creationInteractor = new CreateResourceInteractor(repository);
var response = await creationInteractor.ExecuteAsync(new CreateResourceRequest {␣
↪Resource = resource });

var readInteractor = new ReadResourceInteractor(repository);
var response = await readInteractor.ExecuteAsync(new ReadResourceRequest { ResourceId␣
↪= createdResource.Id });
```