UNIVERSITÀ DEGLI STUDI DI TRIESTE

Introduzione all’integrazione di sistemi in sanità

Parte II – REST API - FHIR

Ing. Mario Damiano

TRIESTE, 17 DICEMBRE 2025

PRESENTAZIONI

ING. **MARIO DAMIANO**

LAUREA SPECIALISTICA IN INGEGNERIA CLINICA (2011)

MASTER DI II LIVELLO IN MANAGEMENT OF CLINICAL ENGINEERING (2014)

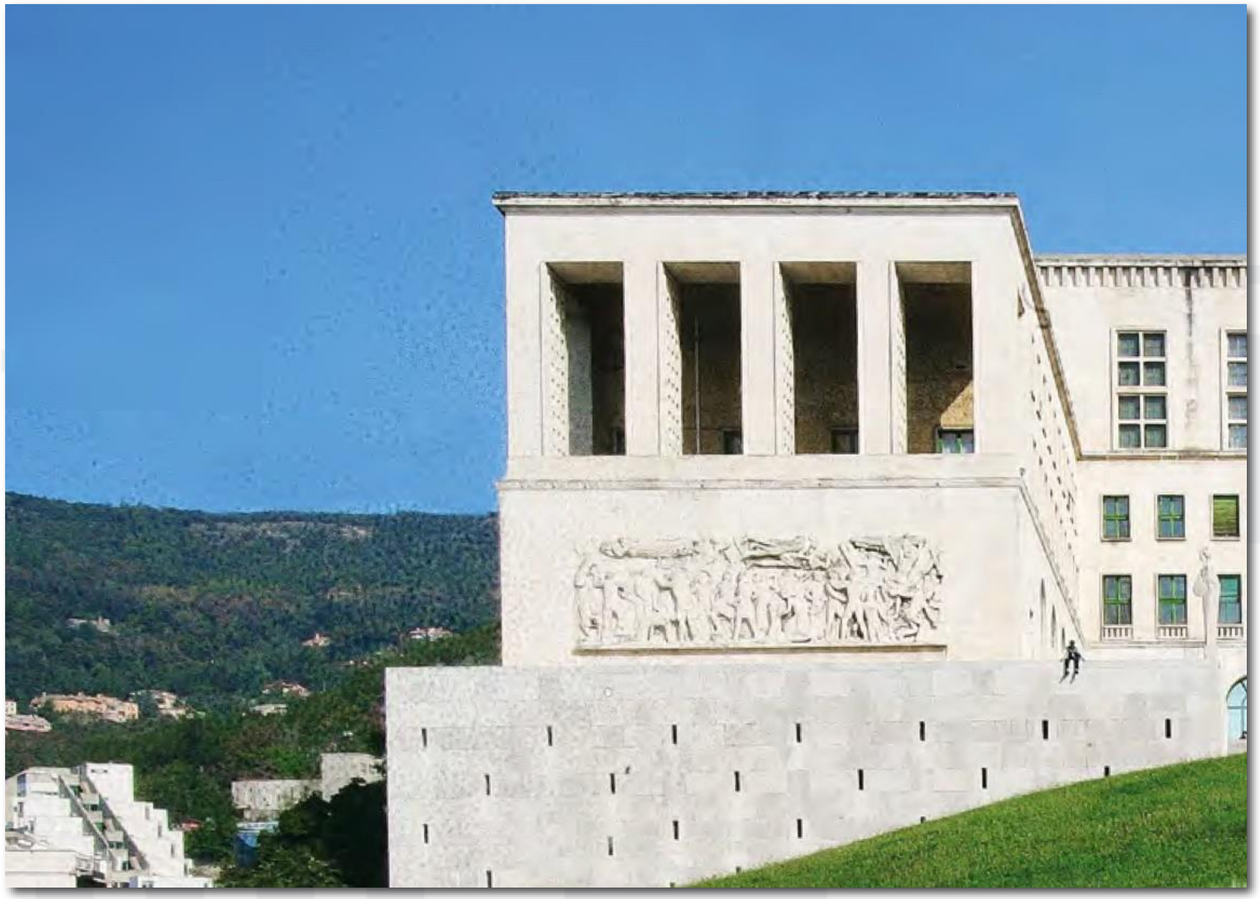
ESPERIENZA PREGRESSA NEL CAMPO DELLA MANUTENZIONE DI APPARECCHIATURE ELETTROMEDICALI CON TBS GROUP (ORA ALTHEA)

VIVO E LAVORO DAL 2014 NEL REGNO UNITO DOVE MI OCCUPO DI INTEGRAZIONE DI SISTEMI IN SANITÀ

CONTATTI:

WWW.LINKEDIN.COM/IN/MARIO-DAMIANO-26503126

EMAIL: *MARIO.DAMIANO@LIVE.COM*



AGENDA

- Un cenno su REST API
- Implementazione di una semplice REST API in Mirth Connect
- Lo standard FHIR
- Un esempio pratico di integrazione FHIR

UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

UN CENNO SU REST API

REST API

Introduzione

REST API è l'acronimo di R*epresentational State Transfer Application Programming Interface* e rappresenta una metodologia utilizzata frequentemente nello sviluppo di servizi internet che permettono l'integrazione strutturata tra sistemi e lo scambio di informazioni tipicamente in formato JSON, XML o HTML.

REST API

Vantaggi

- ✓**Semplicità d'uso e Scalabilità**: Le REST API sono facili da comprendere e utilizzare, seguendo i metodi HTTP standard (GET, POST, PUT, DELETE) e permettendo una scalabilità orizzontale grazie alla loro natura *stateless*.
- ✓**Flessibilità e Interoperabilità**: Supportano vari formati dati, soprattutto JSON, rendendole adatte a diversi tipi di client e piattaforme.
- ✓**Cacheabilità e Riduzione della latenza**: Consentono il caching delle risposte per migliorare le prestazioni e ridurre il carico sul server, eliminando la necessità di memorizzare informazioni sui client.
- ✓**Sicurezza e Facilità di Integrazione**: Garantiscono elevata interoperabilità e sicurezza tramite HTTPS, oltre a supportare meccanismi di autenticazione e autorizzazione per il controllo dell'accesso ai dati come ad esempio Oauth 2.0 e la possibilità di utilizzare criptazione SSL TLS 1.2/1.3
- ✓**Utilizzo orizzontale**: Le REST API sono utilizzate in qualsiasi ambito dove vi sia la necessità di integrare sistemi o realizzare applicazioni che interagiscono con tali sistemi (si pensi ai social media). Quindi ovviamente non soltanto in ambito healthcare.

REST API

Funzionamento

REST definisce la struttura di un'API. Gli sviluppatori devono seguire regole precise nella progettazione di un'API. Una di queste regole stabilisce, ad esempio, che accedere a un URL dovrebbe restituire delle informazioni.

Il sistema identifica ogni URL come una richiesta (request) e i dati restituiti come una risposta (response).

Una REST API scompone una transazione in una sequenza di componenti modulari, ognuno dei quali risponde a un aspetto specifico della transazione. Questa modularità rende REST un approccio di sviluppo flessibile.

I tipi di richieste HTTP sono:

- ✓ **GET**: per ottenere dati.
- ✓ **PUT/PATCH**: per modificare lo stato dei dati
- ✓ **POST**: per creare nuovi dati.
- ✓ **DELETE**: per eliminare dati.

Codici di Risposta HTTP

Ogni richiesta può generare diversi codici di risposta, che indicano lo stato della richiesta:

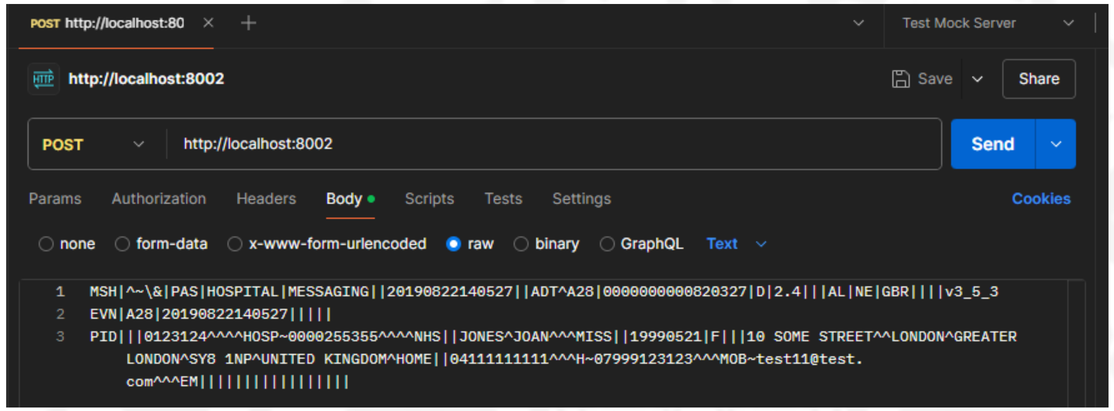
- ✓ **200 OK**: la richiesta è andata a buon fine e i dati sono stati restituiti o aggiornati.
- ✓ **201 Created**: risorsa creata con successo (tipico di POST).
- ✓ **204 No Content**: la richiesta è andata a buon fine ma non ci sono dati da restituire (tipico di DELETE).
- ✓ **404 Not Found**: la risorsa richiesta non esiste.
- ✓ **500 Internal Server Error**: errore lato server durante l'elaborazione della richiesta.

Questi codici di stato permettono al client di sapere l’esito delle operazioni eseguite tramite la REST API.

REST API

Funzionamento

Abbiamo già visto con **Postman** ad esempio come inviare un messaggio HL7 con una richiesta di tipo **POST.**



Andiamo adesso a vedere come possiamo utilizzare Mirth Connect come una REST API.

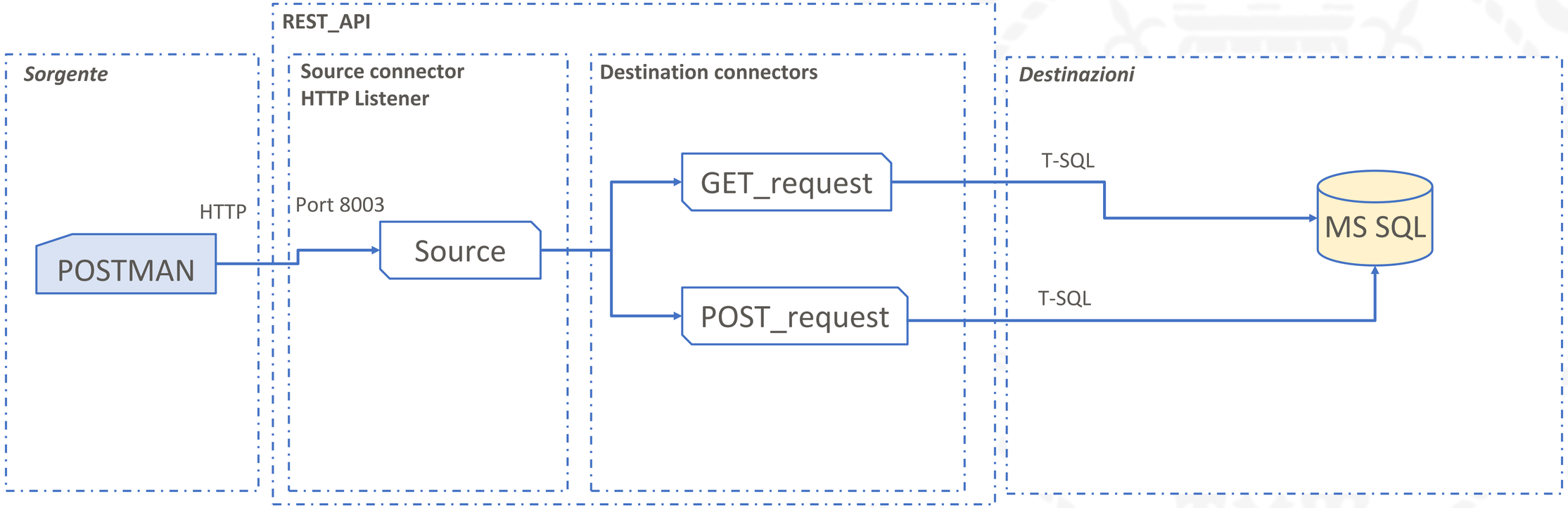
UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

IMPLEMENTAZIONE DI UNA REST API IN MIRTH CONNECT

IMPLEMENTAZIONE DI UNA REST API

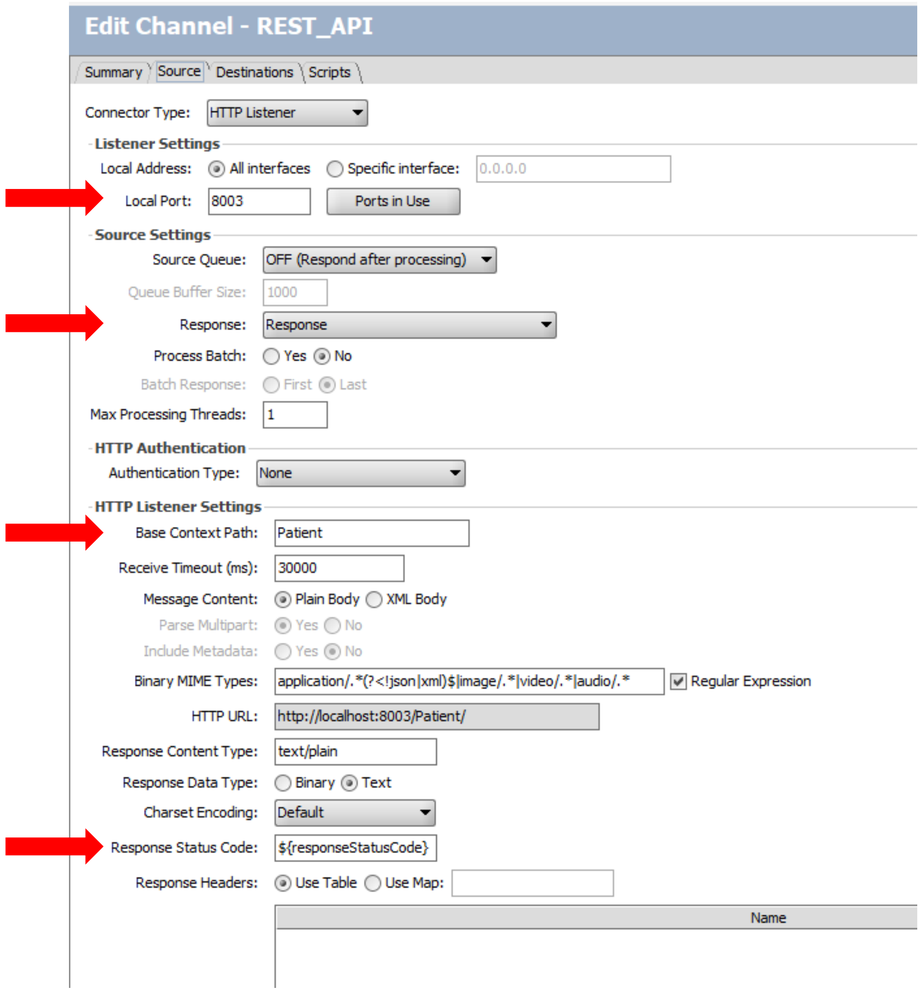
Schema

L'idea è quella di creare un *channel* con un *source connector* di tipo *HTTP listener* che riceverà richieste da **Postman**, mentre le due destinazioni, **GET_request** e **POST_request**, andranno a leggere o scrivere nel database SQL rispettivamente utilizzando **T-SQL** e *stored procedures*.



IMPLEMENTAZIONE DI UNA REST API

Source connector



Il *source connector* come detto è un *HTTP Listener*.

Decidiamo di utilizzare la porta **8003**.

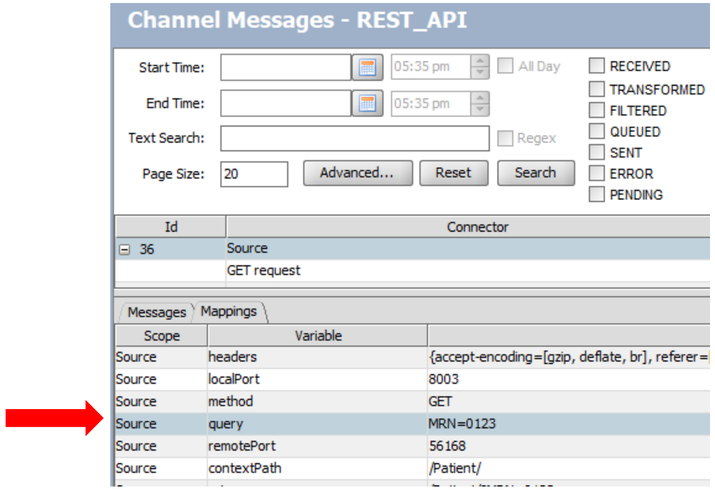
Mirth Connect consente di inviare una risposta dinamica verso il sistema richiedente (**Postman**), utilizziamo per tanto una variabile *Response* che andremo a definire più avanti.

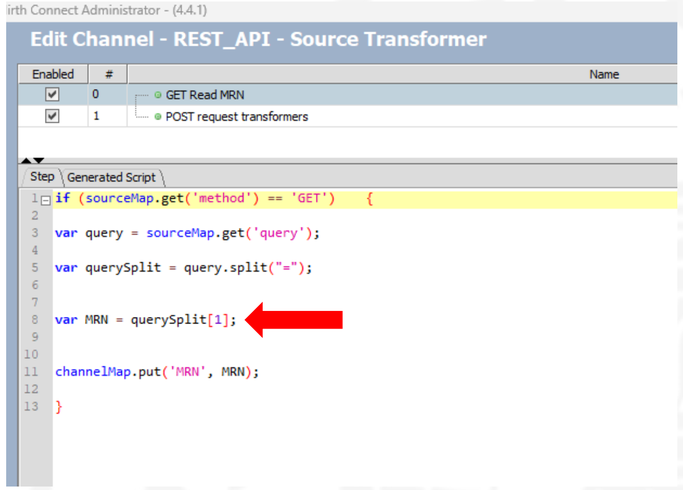
Andiamo poi a definire la *context path*, ovvero la parte dell'URL che segue immediatamente il dominio e precede il percorso della risorsa.

Infine andremo a gestire il codice della risposta HTTP con una seconda variabile, che inizializzeremo come *responseStatusCode.*

IMPLEMENTAZIONE DI UNA REST API

Source connector transformer per la richiesta GET





Il *source connector* ha due transformer, uno dei quali utilizziamo per leggere l'identificativo del paziente per la richiesta GET che andremo poi a utilizzare per la query nel database SQL.

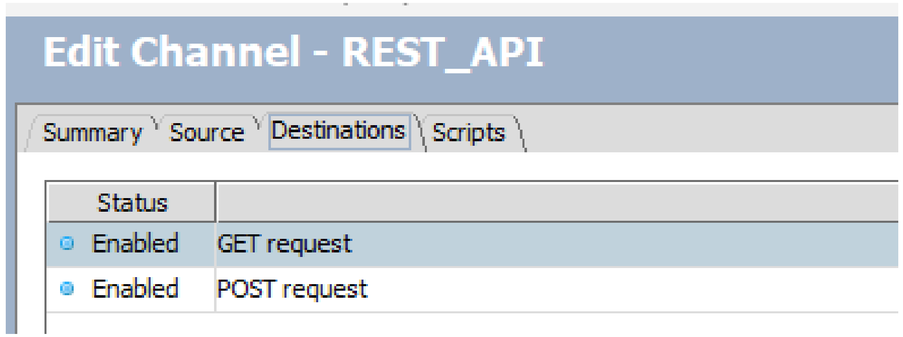
Una richiesta di tipo *GET* infatti, dopo aver definito il *context path*, sarà del tipo:

http://localhost:8003/Patient?MRN=0123

Con una semplice istruzione in JavaScript, leggiamo dalla *source map* la variabile query ed andiamo poi ad estrarre l'identificativo, associato ad una variabile chiamata **MRN**.

IMPLEMENTAZIONE DI UNA REST API

Destination connector



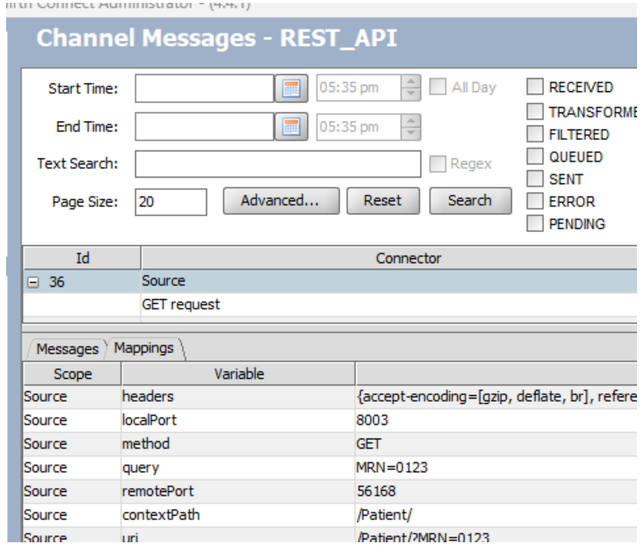
Il *destination connector* contiene due destinazioni separate di tipo *JavaScript writer*.

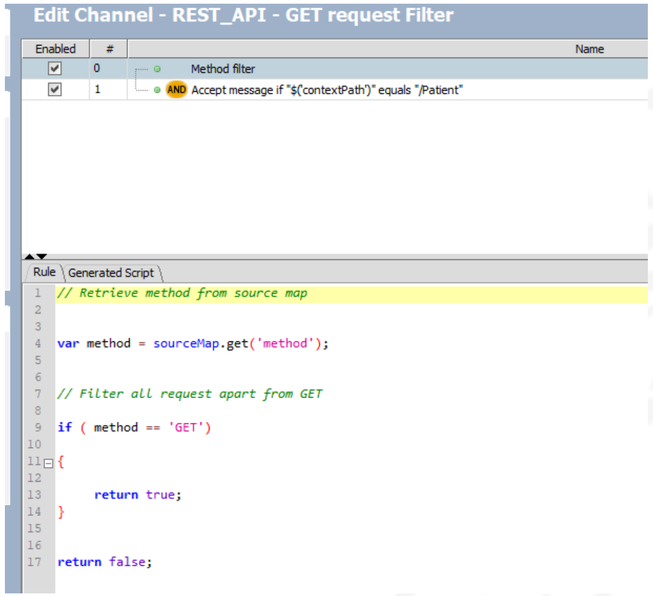
Le due destinazioni sono mutualmente esclusive, quindi solo una delle due sarà eseguita in base al tipo di richiesta proveniente da **Postman**.

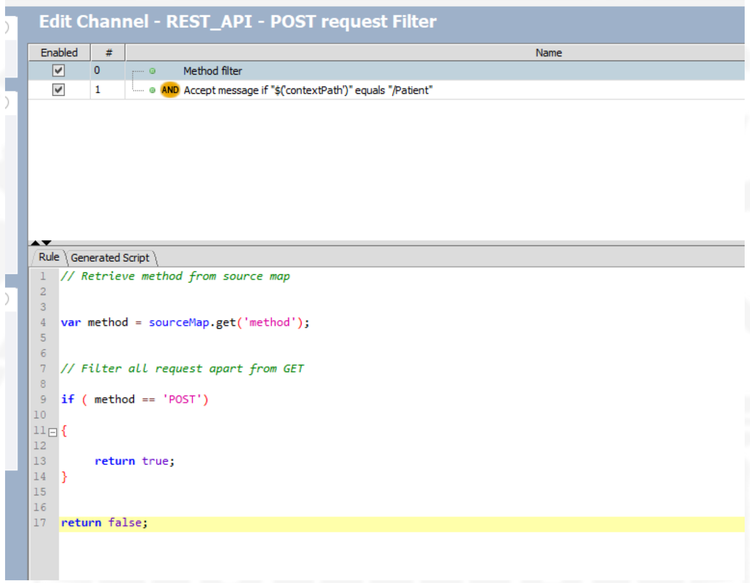
Il comportamento può essere controllato da due semplici filtri.

IMPLEMENTAZIONE DI UNA REST API

Destination connector filters





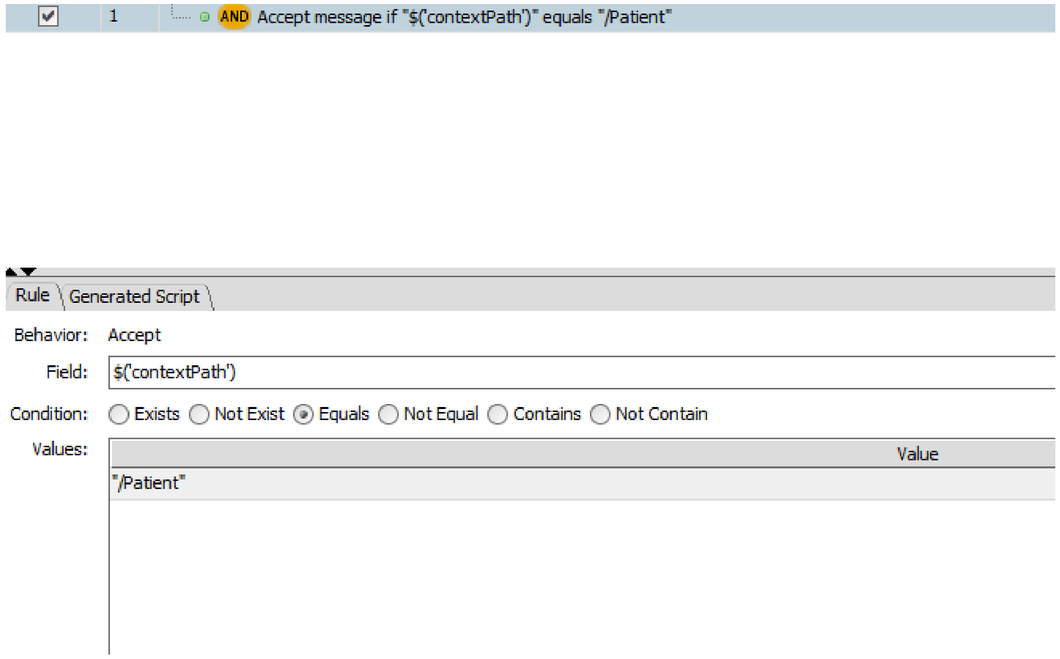


Nella *source map* viene inizializzata una variabile chiamata *method* che andrà a dirci che tipo di operazione andremo ad effettuare.

Nei due filtri, estraiamo il valore della variabile *method* dalla *source map* e la utilizziamo per impostare una condizione booleana con un semplice *IF*. Se la condizione è verificata, eseguiremo il codice del *JavaScript Writer*, altrimenti la richiesta verrà rigettata.

IMPLEMENTAZIONE DI UNA REST API

Destination connector filters

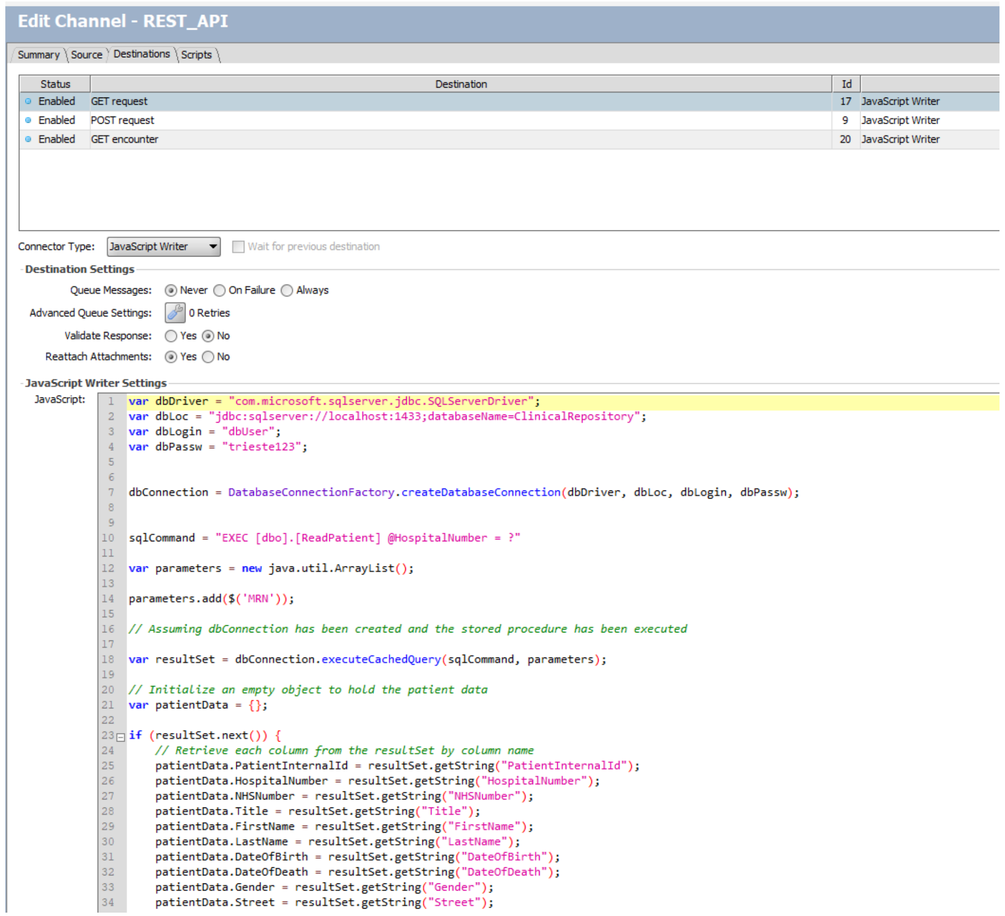


Per entrambe le destinazioni facciamo un check anche sul *context path* utilizzato, ovvero la porzione di **url** che ci da usualmente un'indicazione del tipo di risorsa che vogliamo recuperare o creare.

Nel caso specifico vogliamo filtrare solo sulle richieste con *context path* **/Patient**.

IMPLEMENTAZIONE DI UNA REST API

GET_request JavaScript Writer (1)



Il codice del *JavaScript Writer* consiste in una serie di istruzioni in JavaScript che andranno a:

- Inizializzare la connessione con il database MS SQL.
- Inizializzare il comando T-SQL per eseguire la *stored procedure* **dbo.ReadPatient.**
- Eseguire la *stored procedure.*
- Se trovato, leggere il risultato e inserirlo in un array chiamato *patientData.*
- Gestire l'errore **404** se il record non viene trovato.

IMPLEMENTAZIONE DI UNA REST API

Stored procedure dbo.ReadPatient

```
USE [ClinicalRepository]
GO
/****** Object:  StoredProcedure [dbo].[ReadPatient]    Script Date: 12/11/2024 17:46:23 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER PROCEDURE [dbo].[ReadPatient]
    @HospitalNumber VARCHAR(100)
AS
BEGIN
    SET NOCOUNT ON;

    -- Select the record based on HospitalNumber
    SELECT TOP 1
        PatientInternalId,
        HospitalNumber,
        NHSNumber,
        Title,
        FirstName,
        LastName,
        DateOfBirth,
        DateOfDeath,
        Gender,
        Street,
        City,
        Province,
        Country,
        PostCode,
        MobileNumber,
        SecondaryNumber,
        Email,
        CreatedOn,
        UpdatedOn

    FROM
        dbo.[Patient]
    WHERE
        HospitalNumber = @HospitalNumber;
END
```

La *stored procedure* che andremo ad invocare accetta un unico parametro, ovvero l'identificativo del paziente che abbiamo precedentemente salvato nella variabile **MRN**, e ritornerà la riga della tabella **dbo.Patient** che conterrà il record del paziente per quel preciso valore ricercato.

IMPLEMENTAZIONE DI UNA REST API

GET_request JavaScript Writer (2)

```
// Convert patientData to JSON
var patientDataJson = JSON.stringify(patientData);

// Add JSON to the channel map and set as response
channelMap.put("patientDataJson", patientDataJson);
responseMap.put("Response", patientDataJson);
responseMap.put("responseStatusCode", "200");
```

Nel caso in cui il record è trovato. L'array *patientData* è convertito in formato JSON e inserito nella risposta a cui daremo il nome *Response* (n.b. aggiunta precedentemente nel *source connector*!).

Il codice della risposta viene settato sul valore **200**.

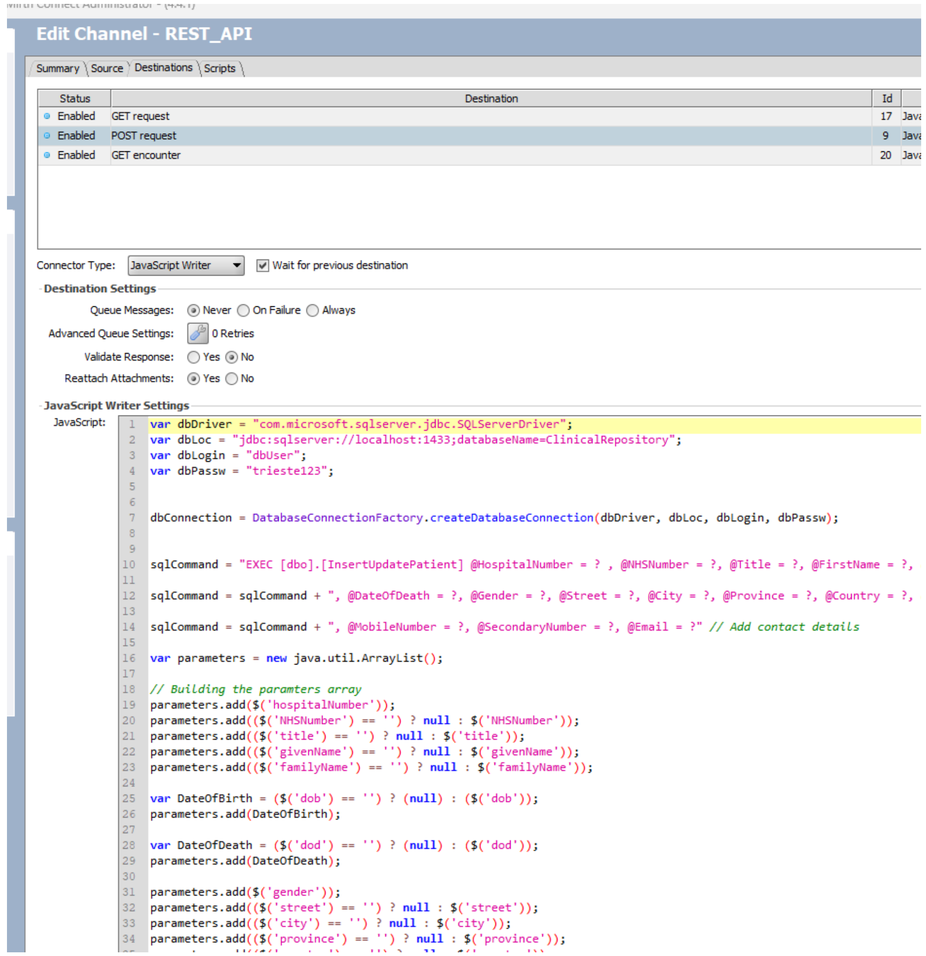
```
// No records found, create a JSON error message
var errorMessage = {
    "error": "Patient record not found",
    "status": 404,
    "message": "No patient found for HospitalNumber " + parameters.get(0)
};

// Convert the error message to JSON
var errorMessageJson = JSON.stringify(errorMessage);

// Set the JSON error message and HTTP 404 status in the response map
responseMap.put("Response", errorMessageJson);
responseMap.put("responseStatusCode", "404");  // Setting HTTP response status code to 404
```

Nel caso in cui il record non è trovato, andiamo a creare una risposta di errore in formato JSON che andremo poi ad associare alla variabile *Response*.

Il codice della risposta viene quindi settato sul valore **404**, seguendo le convenzioni di una REST API.

IMPLEMENTAZIONE DI UNA REST API

POST_request JavaScript Writer



Se la richiesta è di tipo **POST**, andremo ad eseguire il JavaScript Writer della destinazione *POST_request*.

Questa è la stessa operazione che abbiamo visto precedentemente quando abbiamo processato un messaggio HL7 di tipo **ADT^A31**.

La *stored procedure* **dbo.UpdateInsertPatient** accetta una serie di parametri corrispondenti ai campi della tabella **dbo.Patient**. La differenza è che anziché leggere da un messaggio HL7, questa volta andremo a leggere dal contenuto del body della richiesta *POST http* proveniente da **Postman**.

IMPLEMENTAZIONE DI UNA REST API

La richiesta da POSTMAN

La richiesta sarà del tipo *POST http://localhost:8003/Patient*

N.b. non ci sono parametri!

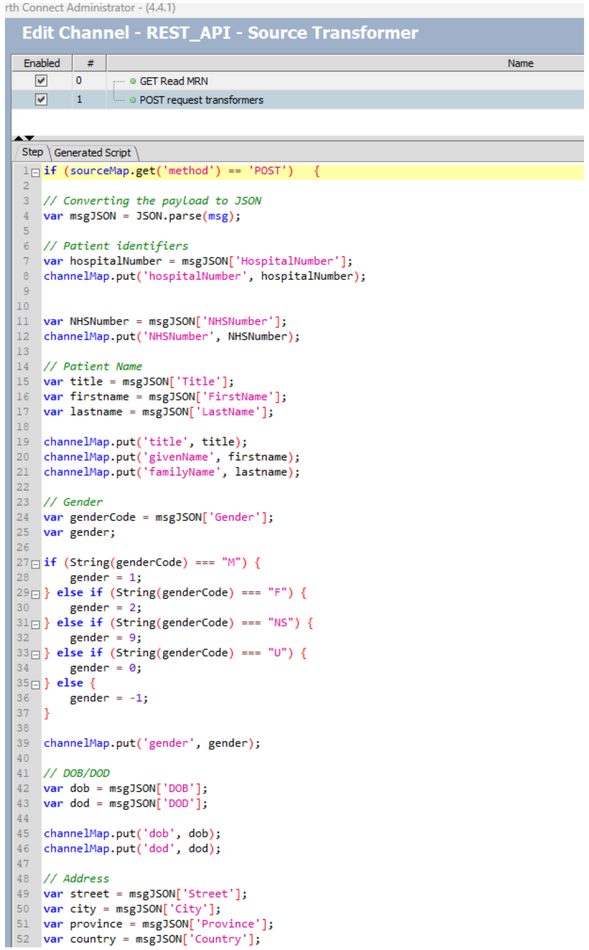
Il contenuto (*body* o *payload*) della richiesta è invece di tipo JSON.

In una richiesta di tipo *GET*, non vi è solitamente alcun contenuto (*body*).

```
{
"HospitalNumber": "22334455",
"NHSNumber" : "1111111111",
"Title": "Mr",
"FirstName": "Jack",
"LastName" : "Rabbit",
"Gender" : "M",
"DOB": "19800210",
"DOD": "",
"Street" : "10 Some Street",
"City" : "London",
"Province": "Greater London",
"Country": "United Kingdom",
"PostCode": "SW1A 1AB",
"ContactDetails": [
{"Mobile": "07777777777",
"Landline": "02011111111",
"Email": "test@test.com"
}]
}
```

IMPLEMENTAZIONE DI UNA REST API

Source connector transformer per la richiesta POST



Il source transformer per la richiesta POST è un *JavaScript writer* con una serie di istruzioni in *JavaScript* che andranno a leggere il payload ricevuto da **Postman** inizializzato in formato JSON e a salvare localmente i valori contenuti nei campi in variabili.

Le variabili saranno poi utilizzate per la chiamata della **stored procedure** per scrivere nel database **SQL**.

IMPLEMENTAZIONE DI UNA REST API

Stored procedure dbo.InsertUpdatePatient

```
ALTER PROCEDURE [dbo].[InsertUpdatePatient]
    @HospitalNumber VARCHAR(100),
    @NHSNumber VARCHAR(50) = NULL,
    @Title VARCHAR(50) = NULL,
    @FirstName VARCHAR(100) = NULL,
    @LastName VARCHAR(100) = NULL,
    @DateOfBirth SMALLDATETIME = NULL,
    @DateOfDeath SMALLDATETIME = NULL,
    @Gender INT,
    @Street VARCHAR(255) = NULL,
    @City VARCHAR(255) = NULL,
    @Province VARCHAR(255) = NULL,
    @Country VARCHAR(255) = NULL,
    @PostCode VARCHAR(20) = NULL,
    @MobileNumber VARCHAR(100) = NULL,
    @SecondaryNumber VARCHAR(100) = NULL,
    @Email VARCHAR(100) = NULL
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @CurrentDateTime DATETIME = GETDATE();
    DECLARE @NewPatientInternalId UNIQUEIDENTIFIER;
```

```
-- Check if a record exists based on HospitalNumber
IF EXISTS (SELECT 1 FROM dbo.Patient WHERE HospitalNumber = @HospitalNumber)
BEGIN
    -- Update the existing record
    UPDATE dbo.Patient
    SET
        NHSNumber = @NHSNumber,
        Title = @Title,
        FirstName = @FirstName,
        LastName = @LastName,
        DateOfBirth = @DateOfBirth,
        DateOfDeath = @DateOfDeath,
        Gender = @Gender,
        Street = @Street,
        City = @City,
        Province = @Province,
        Country = @Country,
        PostCode = @PostCode,
        MobileNumber = @MobileNumber,
        SecondaryNumber = @SecondaryNumber,
        Email = @Email,
        UpdatedOn = @CurrentDateTime
    WHERE HospitalNumber = @HospitalNumber;
END
```

```
ELSE
BEGIN
    -- Generate a new GUID for PatientInternalId
    SET @NewPatientInternalId = NEWID();

    -- Insert a new record
    INSERT INTO dbo.Patient (
        PatientInternalId,
        HospitalNumber,
        NHSNumber,
        Title,
        FirstName,
        LastName,
        DateOfBirth,
        DateOfDeath,
        Gender,
        Street,
        City,
        Province,
        Country,
        PostCode,
        MobileNumber,
        SecondaryNumber,
        Email,
        CreatedOn,
        UpdatedOn
    )
    VALUES (
        @NewPatientInternalId,
        @HospitalNumber,
        @NHSNumber,
        @Title,
        @FirstName,
        @LastName,
        @DateOfBirth,
        @DateOfDeath,
        @Gender,
        @Street,
        @City,
        @Province,
```

La **stored procedure** *dbo.InsertUpdatePatient* contiene le istruzioni in T-SQL per inserire un nuovo record nella tabella per ognuno dei campi previsti. Se il record è stato precedentemente ricevuto, si procede ad un update altrimenti si fa un nuovo inserimento.

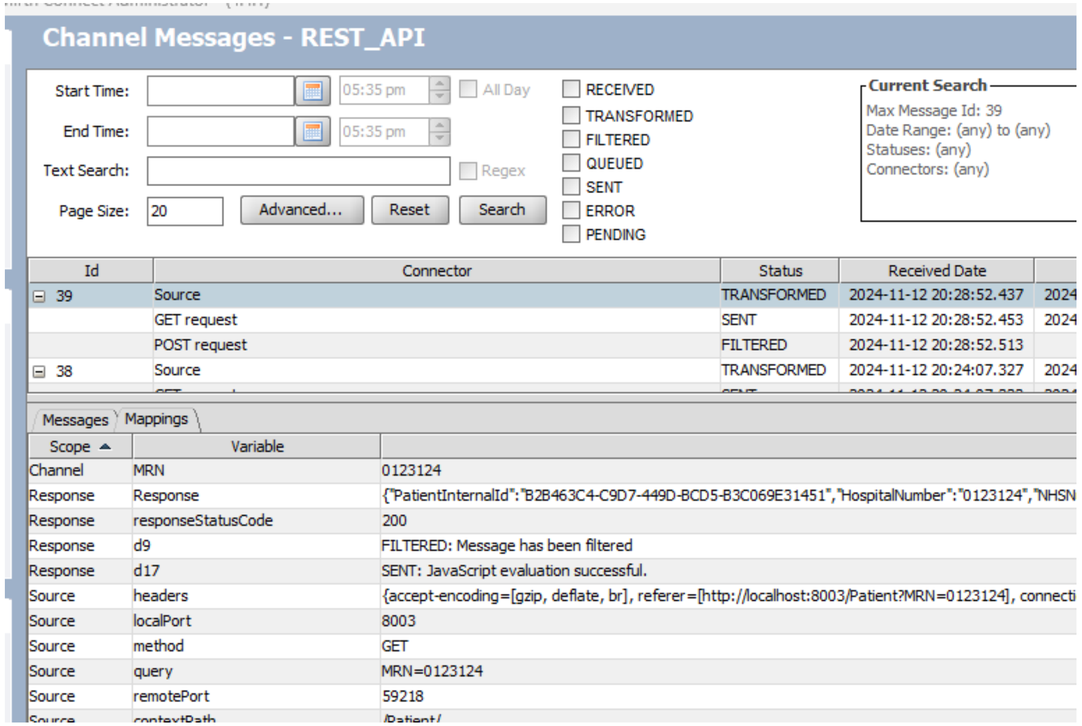
IMPLEMENTAZIONE DI UNA REST API

Esempio pratico end-to-end

Iniziamo da una richiesta *GET HTTP* per cercare il paziente con identificativo **0123123.**

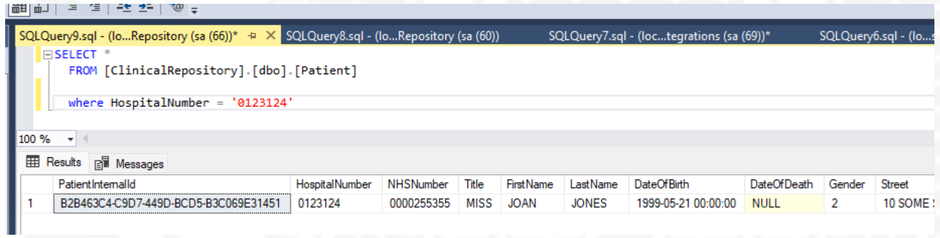
IMPLEMENTAZIONE DI UNA REST API

Esempio pratico end-to-end



Mirth esegue la stored procedure dbo.ReadPatient, individua il record per il paziente **0123124** e lo assegna alla variabile *Response* con codice **200**.

Il record è individuabile nella tabella **dbo.Patient** da una semplice query SQL:



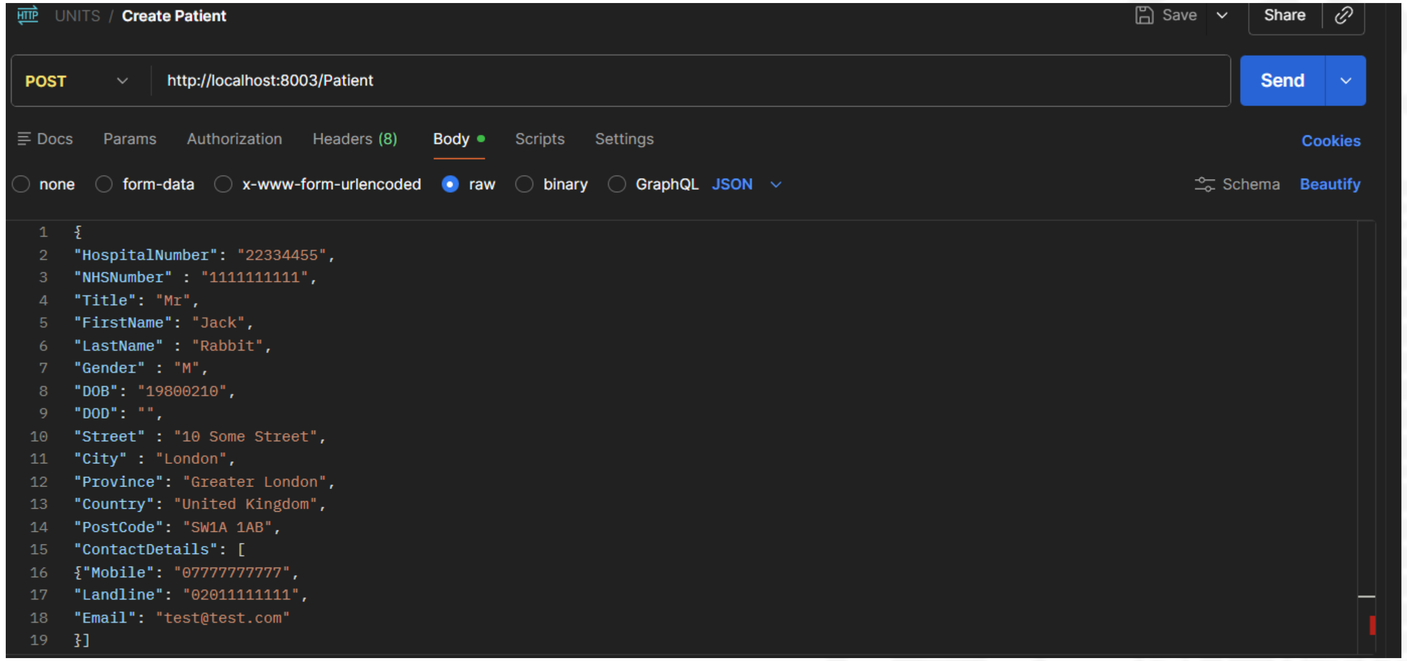
IMPLEMENTAZIONE DI UNA REST API

Esempio pratico end-to-end

Il record è visualizzato in **Postman** nella risposta, si noti il codice **200**.

IMPLEMENTAZIONE DI UNA REST API

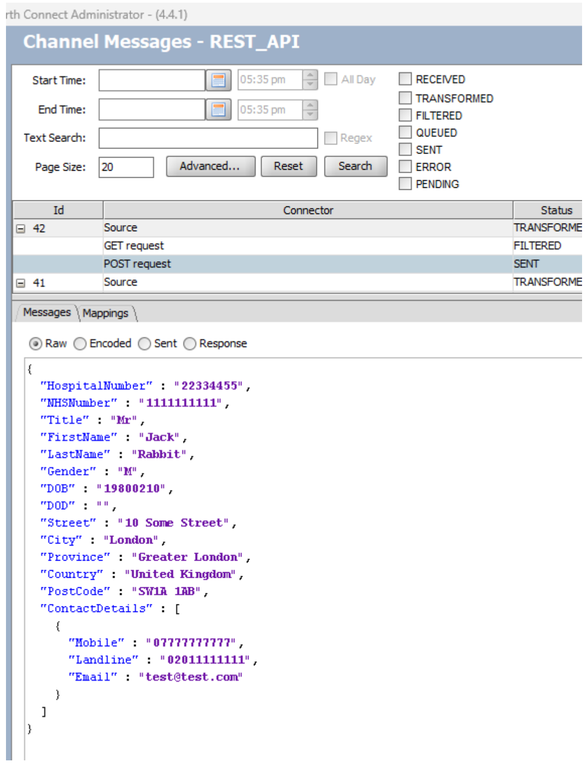
Esempio pratico end-to-end



Supponiamo ora di voler inviare una richiesta di tipo *POST*, dove il contenuto (*body*) della richiesta è il nuovo record che andremo questa volta ad inserire nella tabella **dbo.Patient**

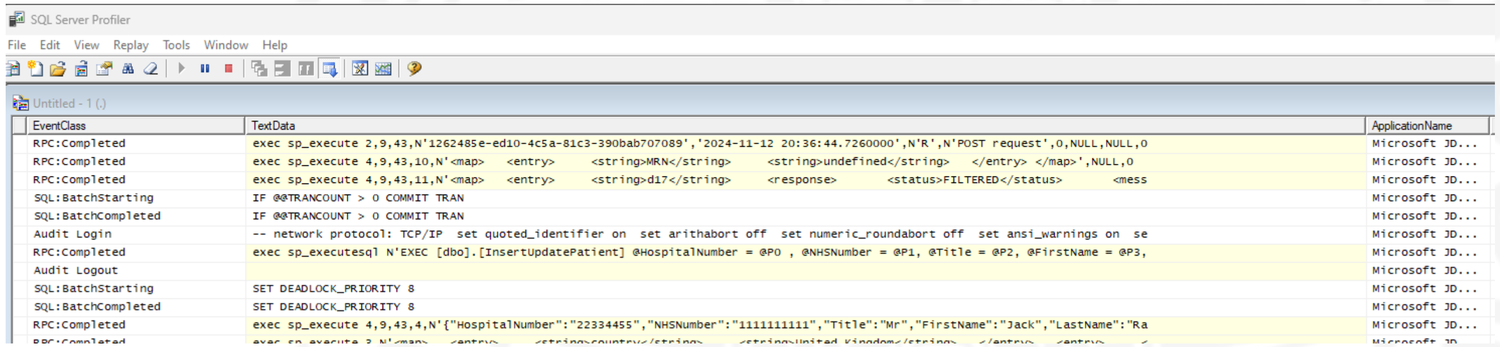
IMPLEMENTAZIONE DI UNA REST API

Esempio pratico end-to-end



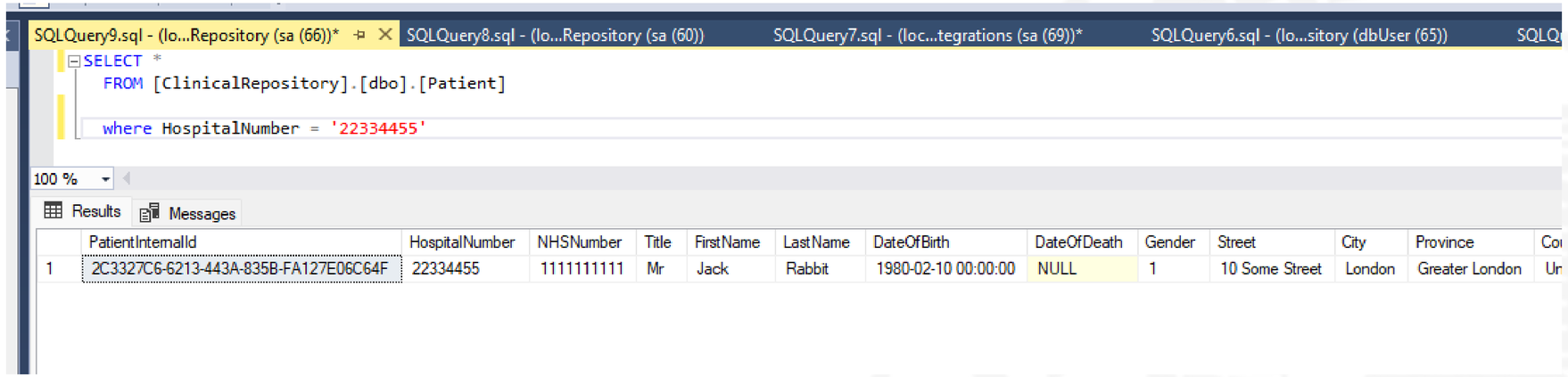
La richiesta è ricevuta da Mirth nella destinazione *POST_request*.

Possiamo osservare con SQL Profiler l'esecuzione della *stored procedure* **dbo.InsertUpdatePatient** e i parametri che vengono utilizzati.



IMPLEMENTAZIONE DI UNA REST API

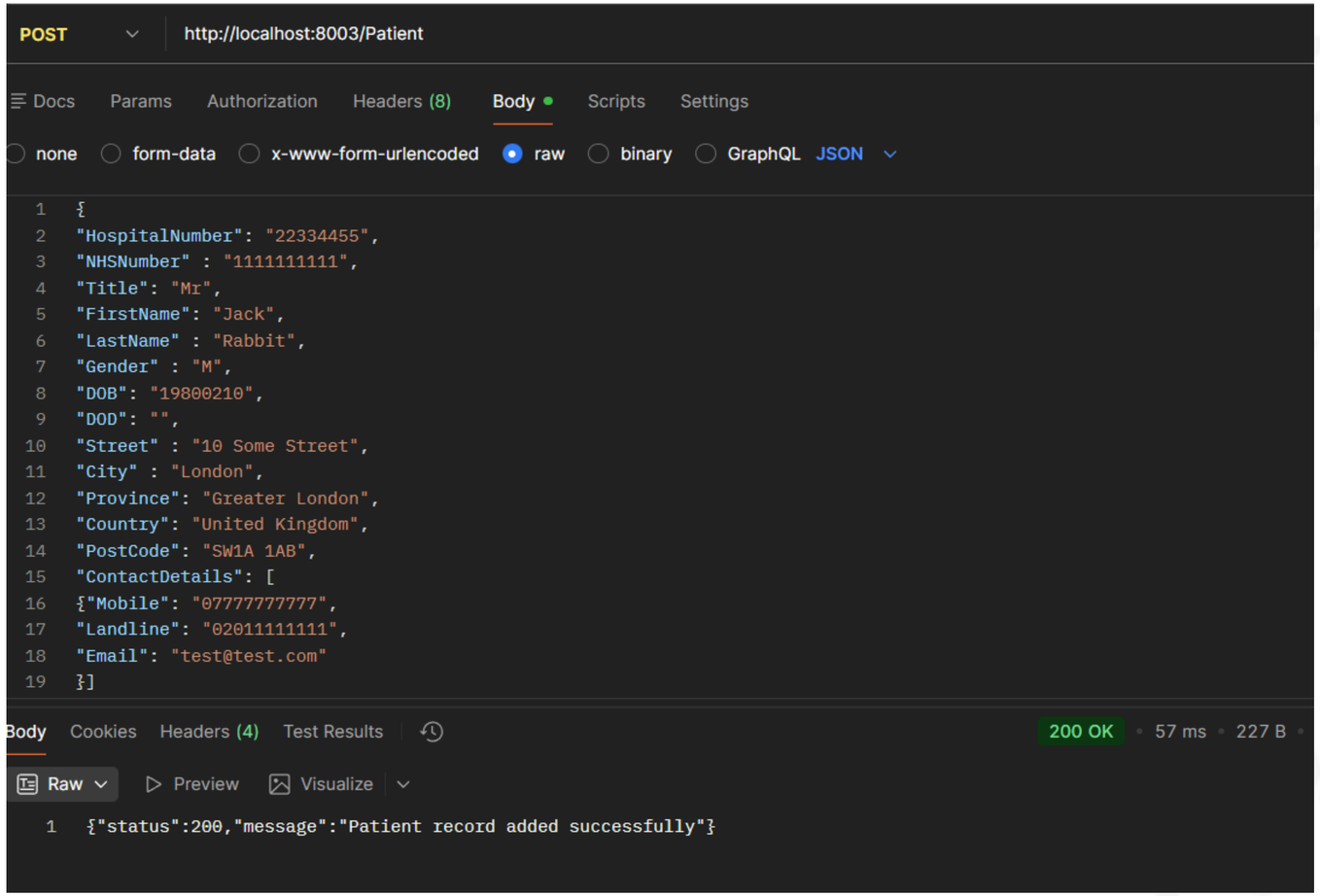
Esempio pratico end-to-end



Il record è scritto correttamente nel database.

IMPLEMENTAZIONE DI UNA REST API

Esempio pratico end-to-end



In **Postman** possiamo osservare il codice di conferma **200** che ci comunica l'avvenuta scrittura del dato.

UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

LO STANDARD FHIR

FHIR

Introduzione

FHIR è l'acronimo di *Fast Healthcare Interoperability Resources.* Nasce su iniziativa di Health Level Seven International per facilitare lo scambio di informazioni tra diversi sistemi sanitari a partire dal 2011.

Nella sua essenza, **FHIR** è uno standard di interoperabilità basato sul modello *REST API* che offre una serie di risorse standardizzate e strutturate in ambito healthcare organizzate tipicamente nel formato JSON o XML.

Attualmente la versione più recente è la release 5 (**R5**)

Ricapitolando, in sostanza **FHIR** si propone di:

1. Utilizzare di una metodologia per l'accesso dei dati (**REST API**)
2. Rappresentare i dati secondo standard specifici (**JSON** o **XML**)
3. Standardizzare l'organizzazione dell'informazione attraverso l'utilizzo di **risorse** e **profili**

Sito ufficiale:

https://hl7.org/fhir/

FHIR

Quali sono i vantaggi?

Interoperabilità: FHIR è stato creato per migliorare ulteriormente l'interoperabilità rispetto a HL7 v2. Grazie all'organizzazione dell'informazione in risorse standardizzate, i dati possono essere facilmente interpretati e scambiati tra vari sistemi di vendors differenti.

Utilizzo di protocolli web: FHIR si basa sul modello REST API utilizzando i formati dati JSON e XML, globalmente riconosciuti e usati in ambito informatico, gestendo l'autenticazione e aspetti legati alla sicurezza informatica secondo standard consolidati nell'industria (es. *OAuth 2.0* e SSL TLS v1.2/1.3).

Flessibilità: l'utilizzo dei profili FHIR permette di adattare le risorse ad esigenze specifiche, di carattere nazionale o della singola organizzazione sanitaria.

Accessibilità: Il modello REST API utilizzato da FHIR consente l'accesso in *real time* dell'informazione utilizzando richieste HTTP.

Controllo dei costi: utilizzare uno standard comune come FHIR riduce la necessità di sviluppare soluzioni personalizzate per interfacciare sistemi diversi che si traduce in un risparmio consistente di tempo e denaro.

Community: l'espansione costante di FHIR permette oggi di avere una comunità di sviluppatori software in ambito internazionale che permette l'evoluzione costante dello standard con aggiornamenti dedicati.

Compatibilità: FHIR è pensato per integrarsi con gli altri standard sanitari, come HL7.

FHIR

Risorse FHIR

Una *risorsa* **FHIR** è un blocco fondamentale di dati strutturati inteso per rappresentare specifiche informazioni. Ogni risorsa è standardizzata e indipendente e permette lo scambio di dati tra sistemi diversi in modo uniforme. Le risorse sono modulari e combinabili per descrivere scenari clinici più complessi.

Esempi di risorse sono:

Patient: contiene dettagli sul paziente, come ad esempio anagrafica, contatti di emergenza ecc.

Practitioner: contiene dettagli su un operatore sanitario, come il codice identificativo, ruolo, nominativo.

Encounter: una delle risorse più importanti. Può rappresentare un ricovero, una visita specialistica, un esame. Diagnostico.

Observation: risorsa utilizzata per contenere i risultati di misurazioni, osservazioni testuali, risultati di analisi di laboratorio.

Medication: descrive un farmaco.

Medication prescription: descrive la prescrizione di un farmaco e indicazioni per il suo utilizzo.

Lista delle risorse FHIR:

https://www.hl7.org/fhir/resourcelist.html

FHIR

L’esempio della patient resource (1)

- Extension References: AuditEvent agent OnBehalfOf, Consent Transcriber, Consent Witness, Contact detail reference, DocumentReference Source Patient... Show 8 more

8.1.3 Resource Content

Structure | UML | XML | JSON | Turtle | R4 Diff | All

Structure

Name	Flags	Card.	Type	Description & Constraints
Patient	N		DomainResource	Information about an individual or animal receiving health care services Elements defined in Ancestors: id, meta, implicitRules, language, text, contained, extension, modifierExtension
identifier	Σ	0..*	Identifier	An identifier for this patient
active	?! Σ	0..1	boolean	Whether this patient's record is in active use
name	Σ	0..*	HumanName	A name associated with the patient
telecom	Σ	0..*	ContactPoint	A contact detail for the individual
gender	Σ	0..1	code	male | female | other | unknown Binding: AdministrativeGender (Required)
birthDate	Σ	0..1	date	The date of birth for the individual
deceased[x]	?! Σ	0..1		Indicates if the individual is deceased or not
deceasedBoolean			boolean	
deceasedDateTime			dateTime	
address	Σ	0..*	Address	An address for the individual
maritalStatus		0..1	CodeableConcept	Marital (civil) status of a patient Binding: Marital Status Codes (Extensible)
multipleBirth[x]		0..1		Whether patient is part of a multiple birth
multipleBirthBoolean			boolean	
multipleBirthInteger			integer	
photo		0..*	Attachment	Image of the patient
contact	C	0..*	BackboneElement	A contact party (e.g. guardian, partner, friend) for the patient *+ Rule: SHALL at least contain a contact's details or a reference to an organization*
relationship		0..*	CodeableConcept	The kind of relationship Binding: Patient Contact Relationship (Extensible)
name	C	0..1	HumanName	A name associated with the contact person
telecom	C	0..*	ContactPoint	A contact detail for the person
address	C	0..1	Address	Address for the contact person
gender		0..1	code	male | female | other | unknown

Se andiamo ad osservare la *patient resource* sul sito ufficiale (https://hl7.org/fhir/patient.html) FHIR ci propone una struttura di base con una serie di campi essenziali e la loro descrizione.

FHIR

L’esempio della patient resource (2)

8.1.3 Resource Content

Structure | UML | XML | JSON | Turtle | R4 Diff | All

JSON Template

```
{
  "resourceType" : "Patient",
  // from Resource: id, meta, implicitRules, and language
  // from DomainResource: text, contained, extension, and modifierExtension
  "identifier" : [{ Identifier }], // An identifier for this patient
  "active" : <boolean>, // Whether this patient's record is in active use
  "name" : [{ HumanName }], // A name associated with the patient
  "telecom" : [{ ContactPoint }], // A contact detail for the individual
  "gender" : "<code>", // male | female | other | unknown
  "birthDate" : "<date>", // The date of birth for the individual
  // deceased[x]: Indicates if the individual is deceased or not. One of these 2:
  "deceasedBoolean" : <boolean>,
  "deceasedDateTime" : "<dateTime>",
  "address" : [{ Address }], // An address for the individual
  "maritalStatus" : { CodeableConcept }, // Marital (civil) status of a patient
  // multipleBirth[x]: Whether patient is part of a multiple birth. One of these 2:
  "multipleBirthBoolean" : <boolean>,
  "multipleBirthInteger" : <integer>,
  "photo" : [{ Attachment }], // Image of the patient
  "contact" : [{ // A contact party (e.g. guardian, partner, friend) for the patient
    "relationship" : [{ CodeableConcept }], // The kind of relationship
    "name" : { HumanName }, // I A name associated with the contact person
    "telecom" : [{ ContactPoint }], // I A contact detail for the person
    "address" : { Address }, // I Address for the contact person
    "gender" : "<code>", // male | female | other | unknown
    "organization" : { Reference(Organization) }, // I Organization that is associated with the contact
    "period" : { Period } // The period during which this contact person or organization is valid to be contacted relating to
this patient
  }],
  "communication" : [{ // A language which may be used to communicate with the patient about his or her health
    "language" : { CodeableConcept }, // R!  The language which can be used to communicate with the patient about his or her h
ealth
    "preferred" : <boolean> // Language preference indicator
  }],
  "generalPractitioner" : [{ Reference(Organization|Practitioner|
   PractitionerRole) }], // Patient's nominated primary care provider
  "managingOrganization" : { Reference(Organization) }, // Organization that is the custodian of the patient record
  "link" : [{ // Link to a Patient or RelatedPerson resource that concerns the same actual individual
    "other" : { Reference(Patient|RelatedPerson) }, // R!  The other patient or related person resource that the link refers t
o
    "type" : "<code>" // R!  replaced-by | replaces | refer | seealso
```

Cliccando sulla JSON tab possiamo vedere la stessa informazione organizzata in formato JSON, ciò significa che se effettuassimo una richiesta http (ad esempio:

GET https://url.com/r4/Patient?_id=Patient/12345)

otterremmo nella risposta il risultato in formato JSON come qui rappresentato

FHIR

La ricerca di una risorsa

8.1.13 Search Parameters

Search parameters for this resource. See also the full list of search parameters for this resource, and check the Extensions registry for search parameters on extensions related to this resource. The common parameters also apply. See Searching for more information about searching in REST, messaging, and services.

Name	Type	Description	Expression	In Common
active	token	Whether the patient record is active	Patient.active	
address	string	A server defined search that may match any of the string fields in the Address, including line, city, district, state, country, postalCode, and/or text	Patient.address	4 Resources
address-city	string	A city specified in an address	Patient.address.city	4 Resources
address-country	string	A country specified in an address	Patient.address.country	4 Resources
address-postalcode	string	A postalCode specified in an address	Patient.address.postalCode	4 Resources
address-state	string	A state specified in an address	Patient.address.state	4 Resources
address-use	token	A use code specified in an address	Patient.address.use	4 Resources
birthdate	date	The patient's date of birth	Patient.birthDate	3 Resources
death-date	date	The date of death has been provided and satisfies this search value	(Patient.deceased.ofType(dateTime))	
deceased	token	This patient has been marked as deceased, or has a death date entered	Patient.deceased.exists() and Patient.deceased != false	
email	token	A value in an email contact	Patient.telecom.where(system='email')	5 Resources
family	string	A portion of the family name of the patient	Patient.name.family	2 Resources
gender	token	Gender of the patient	Patient.gender	4 Resources
general-practitioner	reference	Patient's nominated general practitioner, not the organization that manages the record	Patient.generalPractitioner (Practitioner, Organization, PractitionerRole)	
given	string	A portion of the given name of the patient	Patient.name.given	2 Resources
identifier	token	A patient identifier	Patient.identifier	
language	token	Language code (irrespective of use value)	Patient.communication.language	
link	reference	All patients/related persons linked to the given patient	Patient.link.other (Patient, RelatedPerson)	
name	string	A server defined search that may match any of the string fields in the HumanName, including family, given, prefix, suffix, and/or text	Patient.name	

Non tutti i campi rappresentati in una risorsa sono ricercabili.

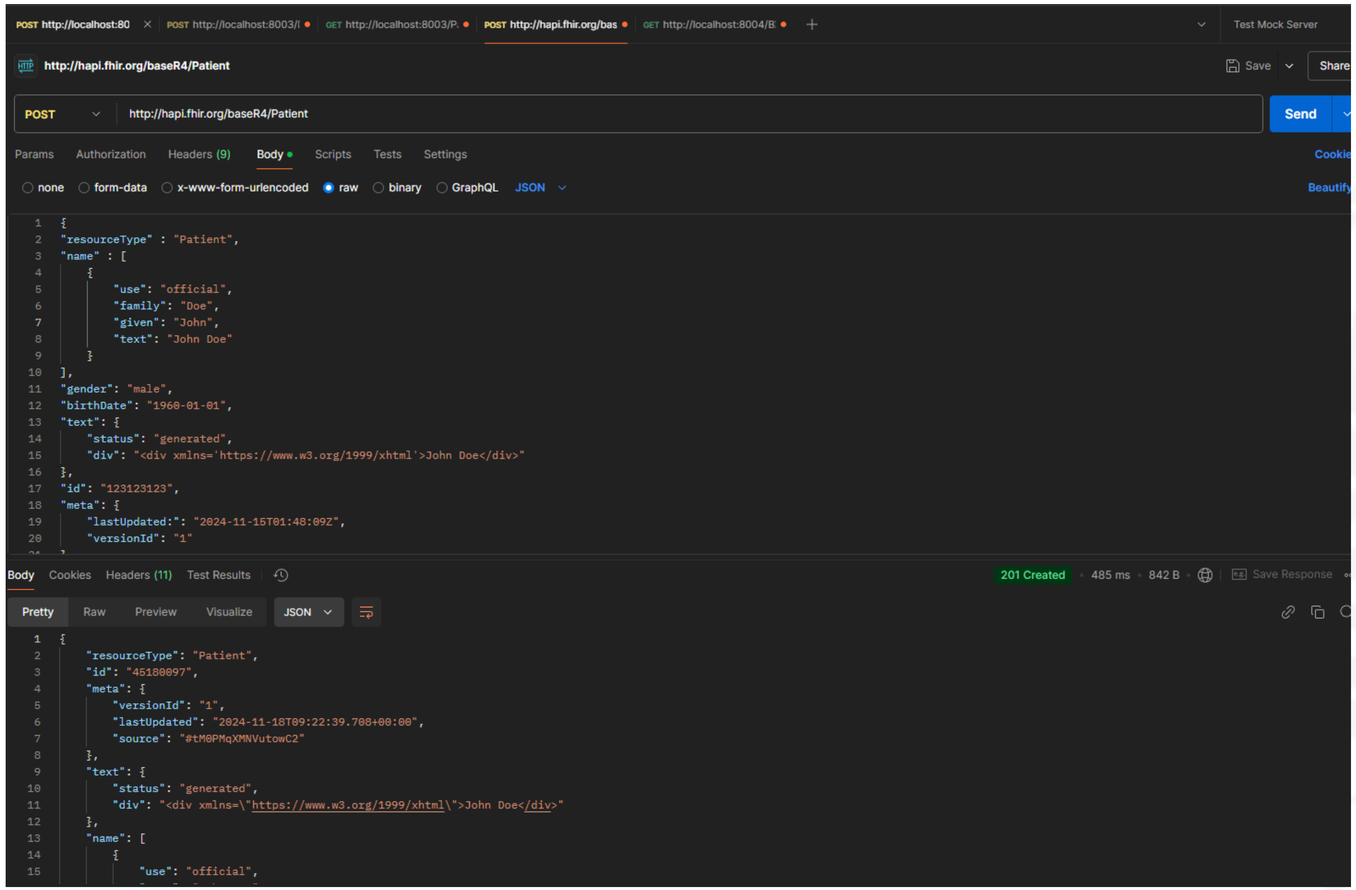
FHIR ci indica quindi quali sono tali campi.

Nell'esempio a fianco, possiamo osservare i parametri di ricerca per la *patient resource* per operazioni di tipo GET.

I parametri possono essere concatenati tra loro per ricerche più o meno specifiche.

FHIR

La scrittura di una risorsa



Possiamo inserire una nuovo record in formato FHIR utilizzando una richiesta di tipo POST.

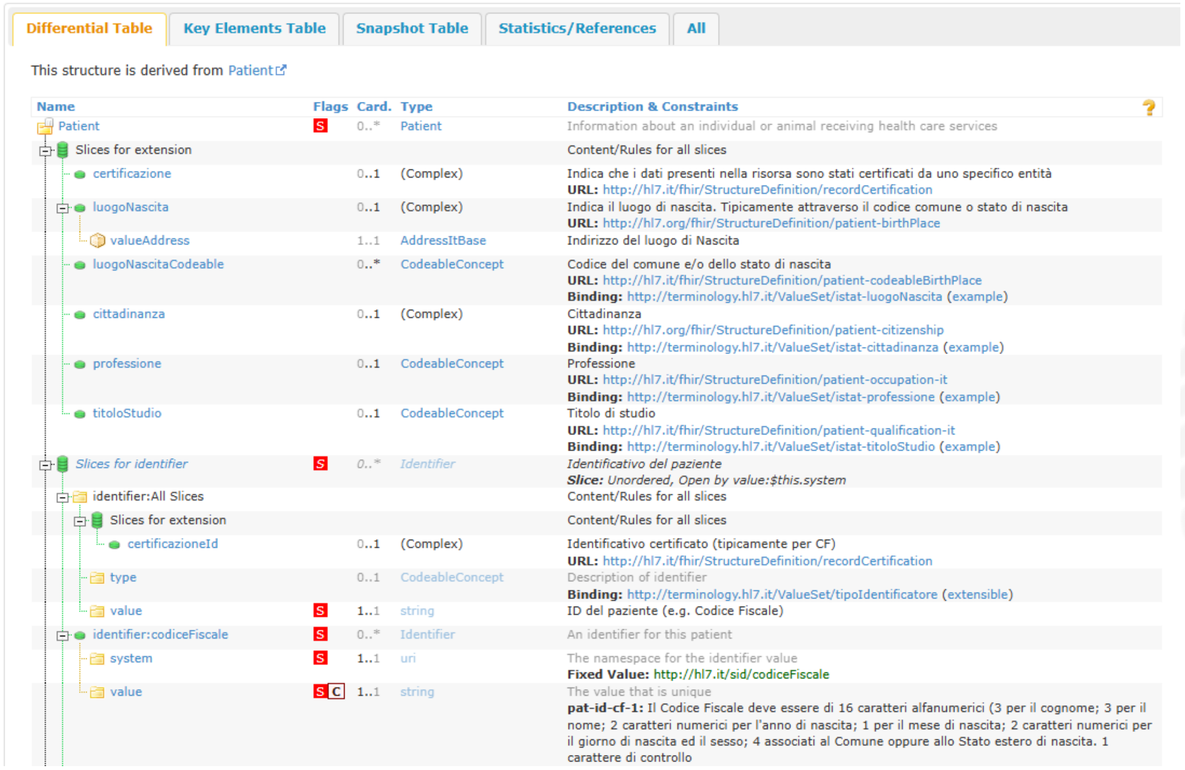
Nell'esempio a fianco, utilizziamo **Postman** mandare una richiesta POST verso un endpoint FHIR (si noti il *context path* **/Patient**). Il payload rappresenta un nuovo paziente inserito seguendo lo schema della risorsa **Patient** e utilizzando il formato JSON.

Il codice **201** (*created*) conferma il corretto inserimento del dato.

FHIR

Profili FHIR

I profili **FHIR** sono invece utilizzati per personalizzare una risorsa sulla base delle esigenze specifiche di un'organizzazione sanitaria o per requisiti nazionali.



Differential Table | Key Elements Table | Snapshot Table | Statistics/References | All

This structure is derived from Patient

Name	Flags	Card.	Type	Description & Constraints
Patient	S	0..*	Patient	Information about an individual or animal receiving health care services
Slices for extension				Content/Rules for all slices
certificazione		0..1	(Complex)	Indica che i dati presenti nella risorsa sono stati certificati da uno specifico entità **URL:** http://hl7.it/fhir/StructureDefinition/recordCertification
luogoNascita		0..1	(Complex)	Indica il luogo di nascita. Tipicamente attraverso il codice comune o stato di nascita **URL:** http://hl7.org/fhir/StructureDefinition/patient-birthPlace
valueAddress		1..1	AddressItBase	Indirizzo del luogo di Nascita
luogoNascitaCodeable		0..*	CodeableConcept	Codice del comune e/o dello stato di nascita **URL:** http://hl7.it/fhir/StructureDefinition/patient-codeableBirthPlace **Binding:** http://terminology.hl7.it/ValueSet/istat-luogoNascita (example)
cittadinanza		0..1	(Complex)	Cittadinanza **URL:** http://hl7.org/fhir/StructureDefinition/patient-citizenship **Binding:** http://terminology.hl7.it/ValueSet/istat-cittadinanza (example)
professione		0..1	CodeableConcept	Professione **URL:** http://hl7.it/fhir/StructureDefinition/patient-occupation-it **Binding:** http://terminology.hl7.it/ValueSet/istat-professione (example)
titoloStudio		0..1	CodeableConcept	Titolo di studio **URL:** http://hl7.it/fhir/StructureDefinition/patient-qualification-it **Binding:** http://terminology.hl7.it/ValueSet/istat-titoloStudio (example)
Slices for identifier	S	0..*	*Identifier*	*Identificativo del paziente* ***Slice:*** *Unordered, Open by value:$this.system*
identifier:All Slices				Content/Rules for all slices
Slices for extension				Content/Rules for all slices
certificazioneId		0..1	(Complex)	Identificativo certificato (tipicamente per CF) **URL:** http://hl7.it/fhir/StructureDefinition/recordCertification
type		0..1	CodeableConcept	Description of identifier **Binding:** http://terminology.hl7.it/ValueSet/tipoIdentificatore (extensible)
value	S	1..1	string	ID del paziente (e.g. Codice Fiscale)
identifier:codiceFiscale	S	0..*	Identifier	An identifier for this patient
system	S	1..1	uri	The namespace for the identifier value **Fixed Value:** http://hl7.it/sid/codiceFiscale
value	S C	1..1	string	The value that is unique **pat-id-cf-1:** Il Codice Fiscale deve essere di 16 caratteri alfanumerici (3 per il cognome; 3 per il nome; 2 caratteri numerici per l'anno di nascita; 1 per il mese di nascita; 2 caratteri numerici per il giorno di nascita ed il sesso; 4 associati al Comune oppure allo Stato estero di nascita. 1 carattere di controllo

Ad esempio possiamo vedere come il profilo generico della risorsa *patient* per l'Italia contiene alcuni campi specifici come il codice fiscale, non presenti nell'esempio generico.

Per approfondire:

https://build.fhir.org/ig/hl7-it/base//StructureDefinition-Patient-it-base.html

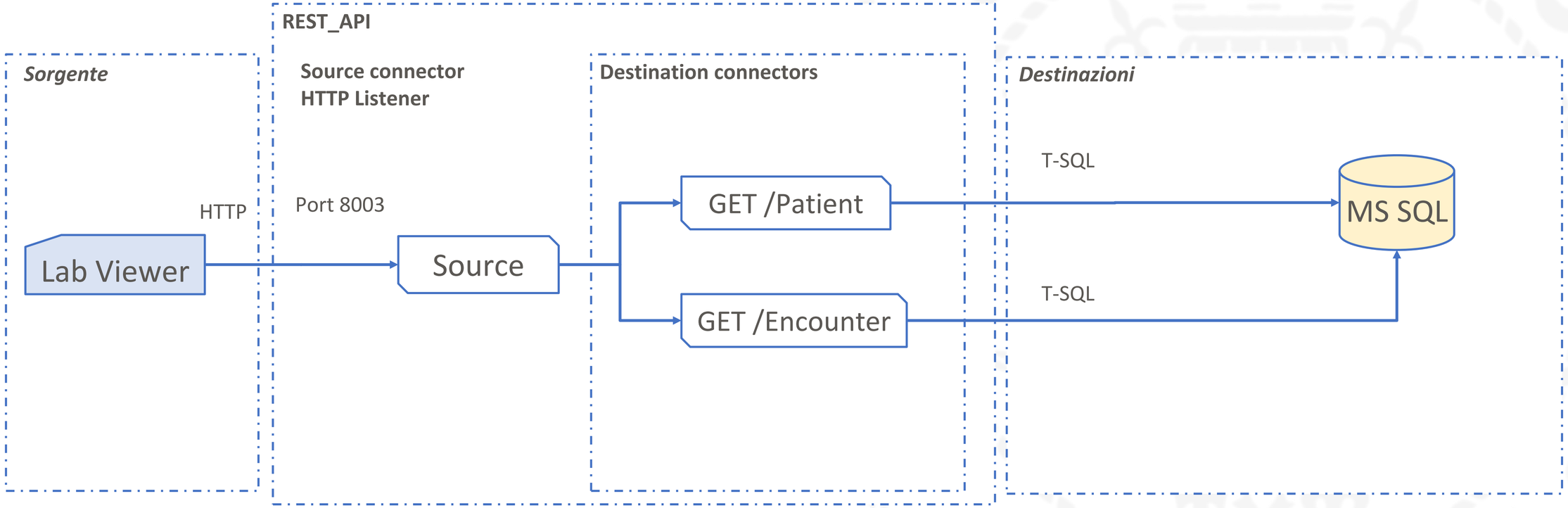
UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

UN ESEMPIO PRATICO DI INTEGRAZIONE FHIR

IMPLEMENTAZIONE FHIR

Schema REST API

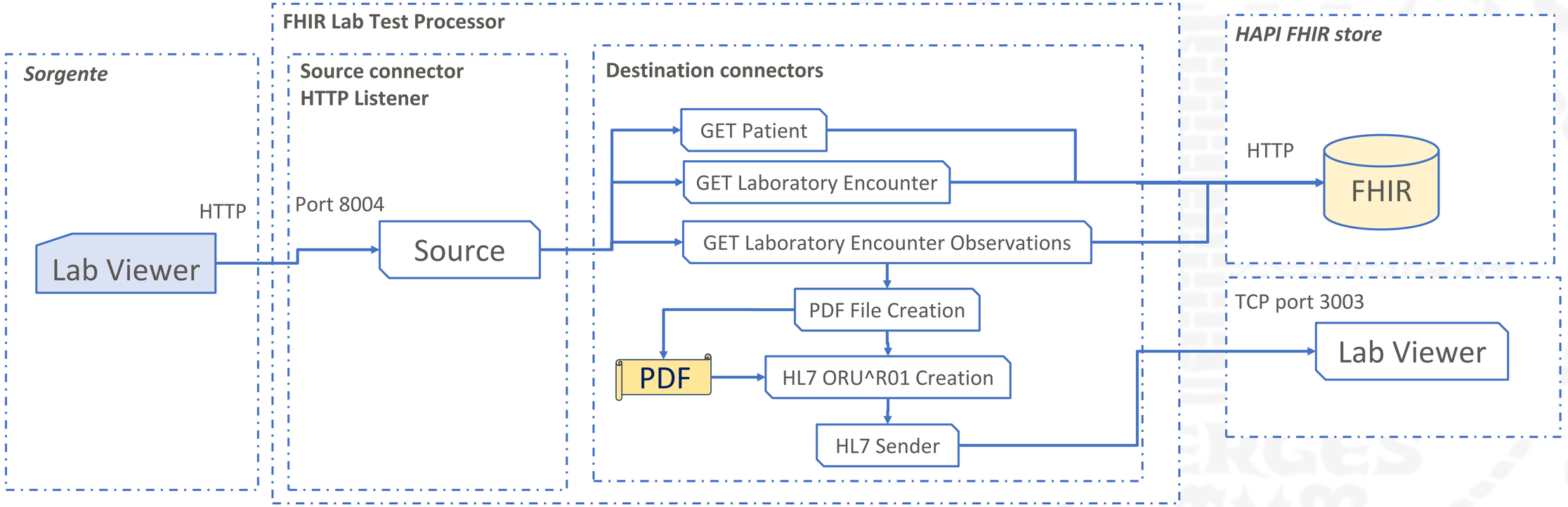
Un applicativo web chiamato Lab Viewer (implementato in *node.js*) permette la possibilità di ricerca paziente e di ricerca di encounters di laboratorio tramite REST API implementata in Mirth e interfacciata con un database MS SQL che si può assumere essere alimentato da un feed HL7 di tipo ADT^A28/A31 e ORM^O01



IMPLEMENTAZIONE FHIR

Schema FHIR processor

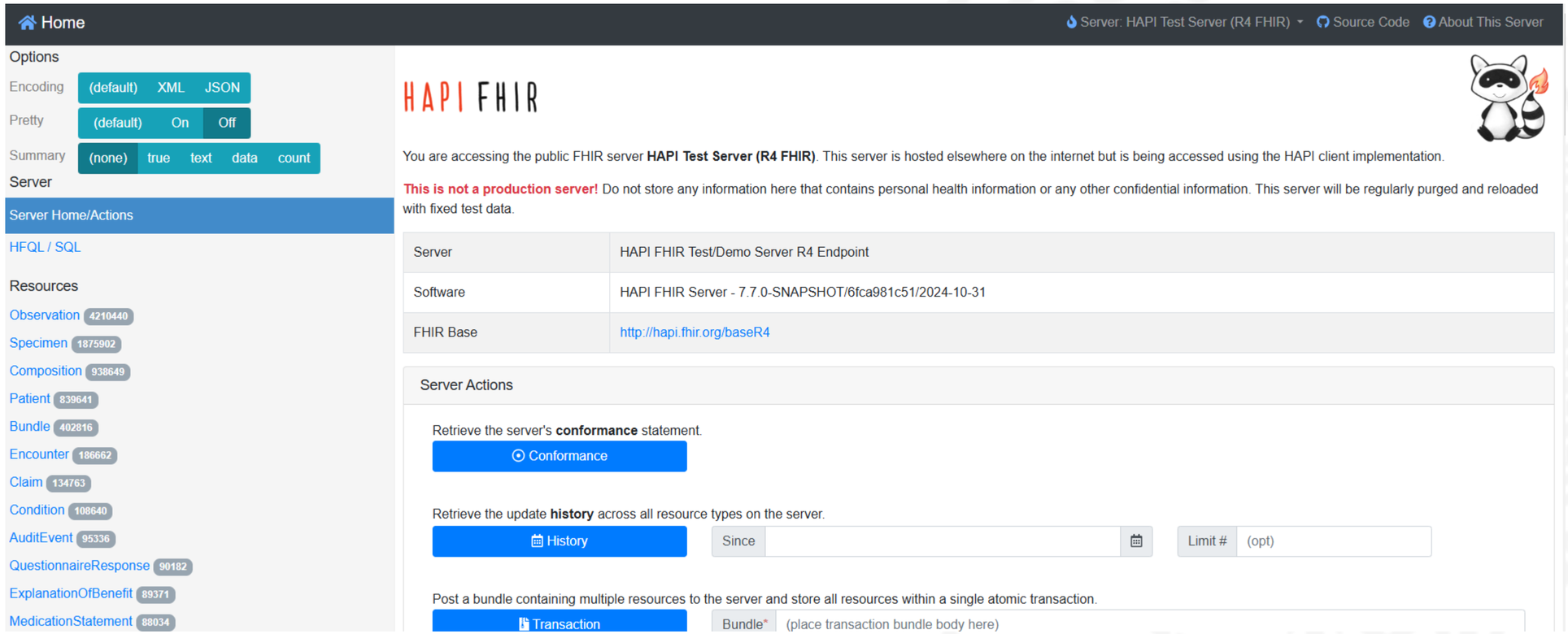
Andiamo a creare una seconda interfaccia in Mirth rappresentata da un *channel* chiamato **FHIR Lab Test Processor** che ci permette di ricevere richieste HTTP dal lab viewer per andare a cercare, per un determinato encounter di laboratorio, il risultato di un'analisi del sangue salvato in un repository **FHIR** per quel determinato paziente, produrre un **file PDF** con i valori da salvare localmente in un archivio e infine inviare tale risultato verso **lab viewer** per la successiva visualizzazione con un messaggio **HL7 ORU^R01** utilizzando il protocollo TCP/IP.



IMPLEMENTAZIONE FHIR

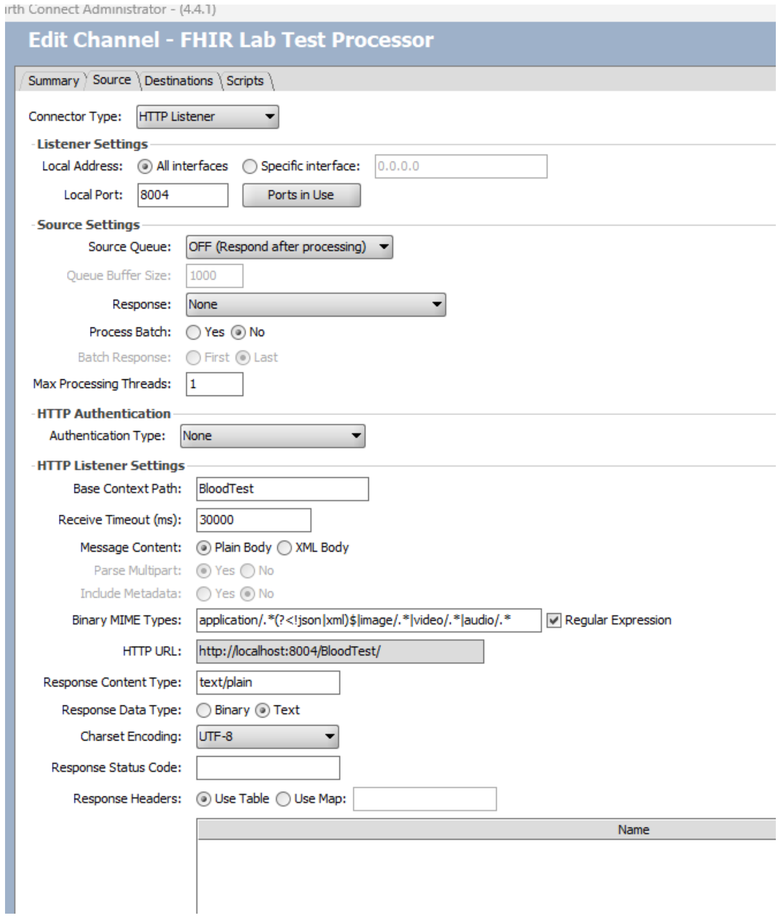
HAPI FHIR test server

Per leggere i dati utilizzeremo una implementazione open source dello standard FHIR. Il progetto **HAPI** offre infatti l'accesso ad una **FHIR API** con un gran numero di risorse disponibili. L'accesso è disponibile sia in lettura che in scrittura, non contiene dati di pazienti reali ed è di libero accesso (https://hapi.fhir.org/).



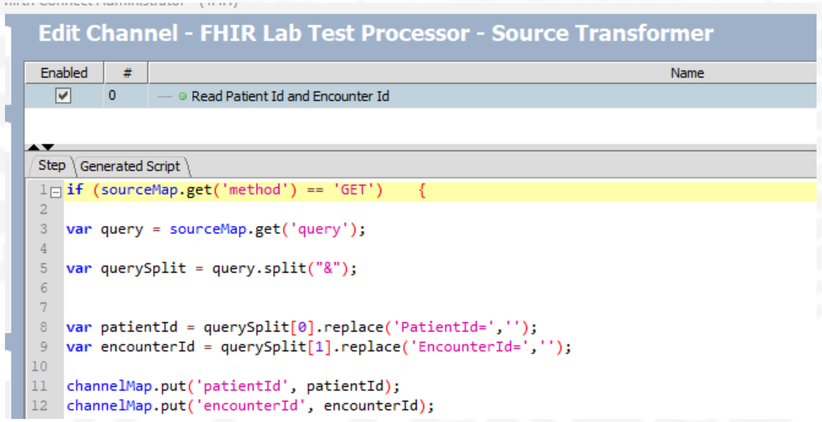
IMPLEMENTAZIONE FHIR

Source connector FHIR processor



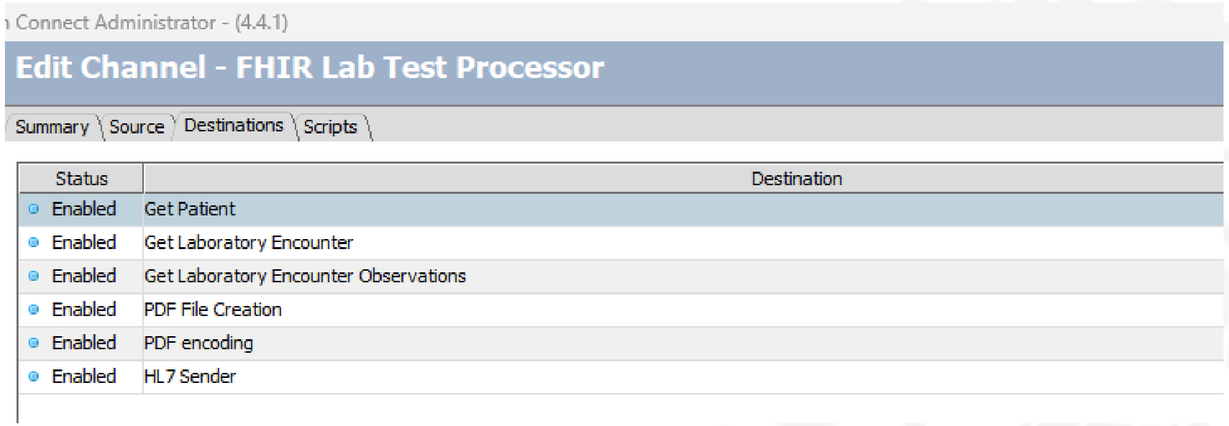
Il *source connector* è il solito *HTTP Listener*, questa volta allocando la porta **8004**. Si noti il *context path* **BloodTest** che sarà utilizzato nell'URL per la richiesta GET HTTP.

Il *source transformer* contiene l'istruzione in *JavaScript* per leggere il **patient ID** e l'**encounter ID** (identificativo dell'analisi) dall'URL.



IMPLEMENTAZIONE FHIR

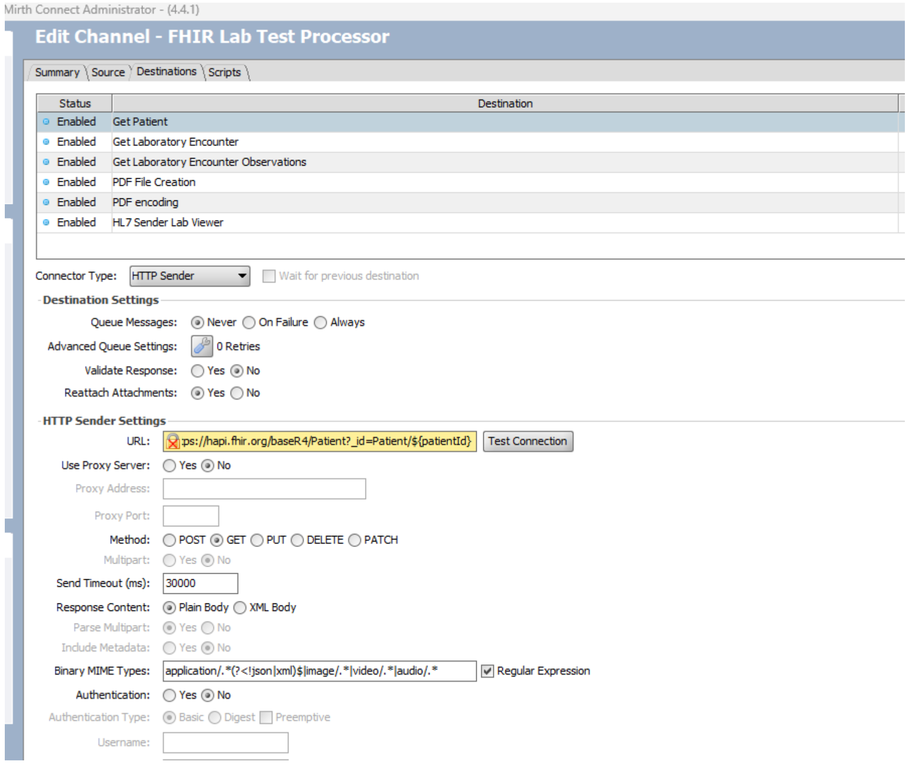
Destination connector FHIR processor



Il *destination connector* contiene 6 destinazioni che andremo ora a vedere nel dettaglio. Le destinazioni sono «eseguite» sequenzialmente.

IMPLEMENTAZIONE FHIR

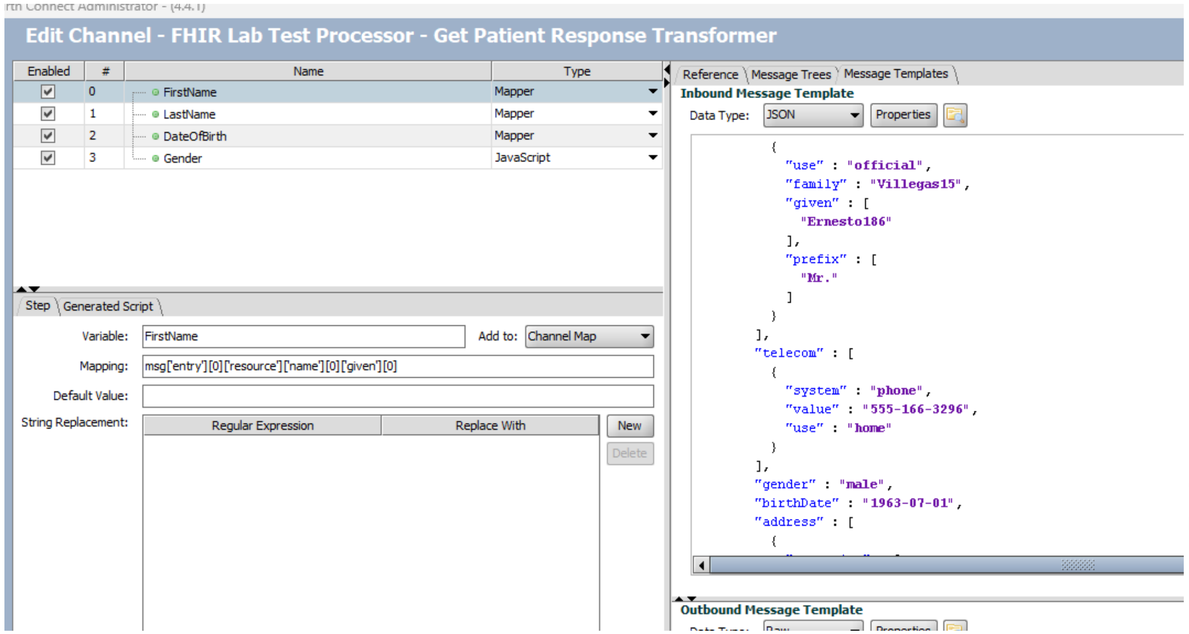
Get Patient (1)



Si tratta di un HTTP Sender che invia una GET request verso l'endpoint HAPI per la risorsa *patient* (identificata da **/Patient**) con un solo parametro di ricerca specificato (la variabile locale *${patientId}* estratta dall'URL della richiesta iniziale).

IMPLEMENTAZIONE FHIR

Get Patient (2)

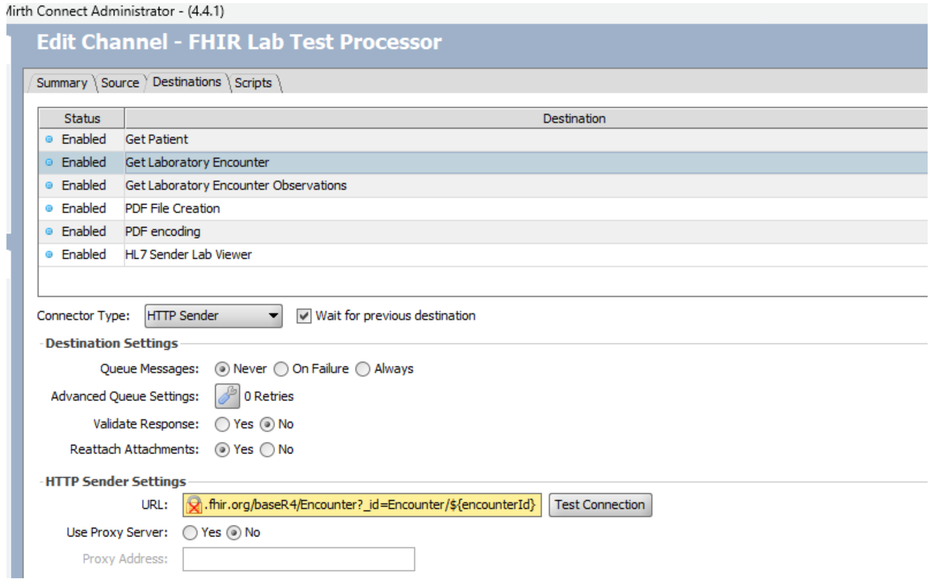


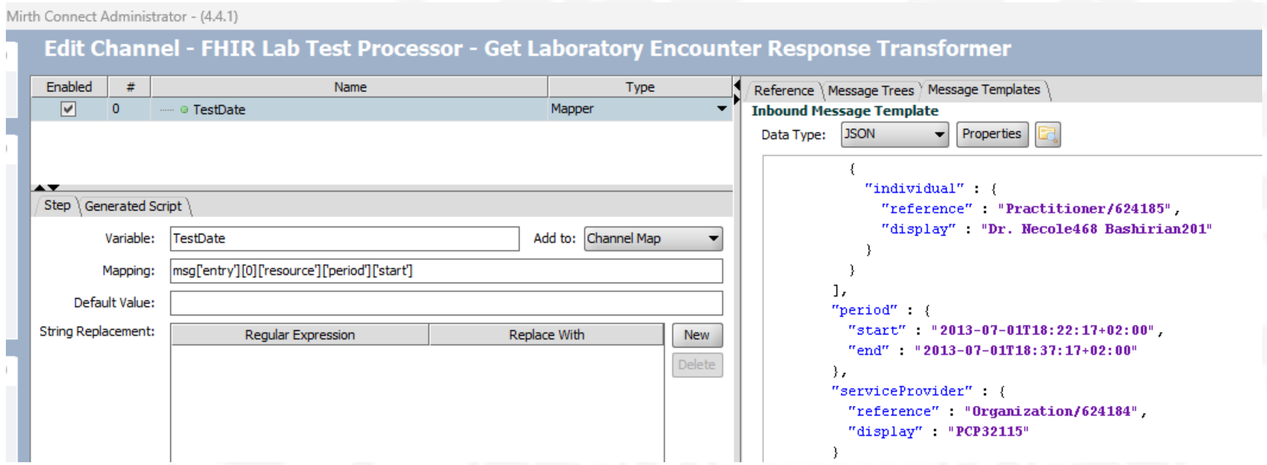
La risposta sarà un payload in formato JSON contenente la risorsa *patient* per l'ID ricercato.

Da essa estraiamo alcuni dati anagrafici che andremo a inserire nel PDF, come nome/cognome/data di nascita/sesso).

IMPLEMENTAZIONE FHIR

Get Laboratory Encounter

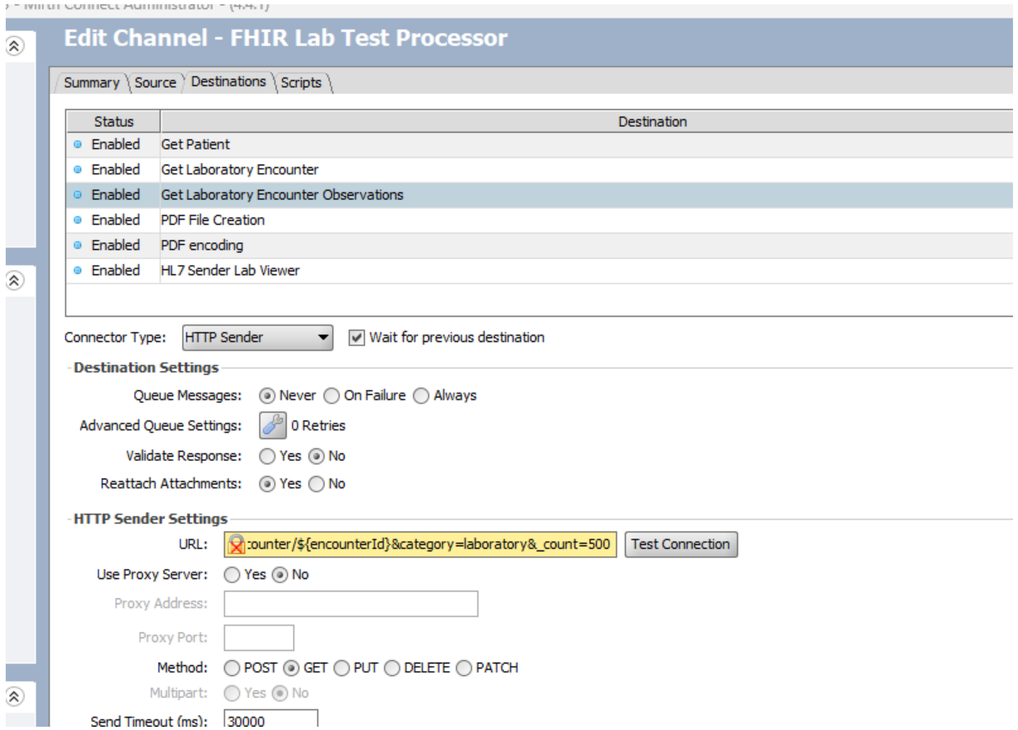




Similmente andiamo adesso a cercare la risorsa *encounter* per l'esame del sangue dalla quale andremo a leggere la data in cui l'esame è stato fatto e che inseriremo anch'essa nel PDF.

IMPLEMENTAZIONE FHIR

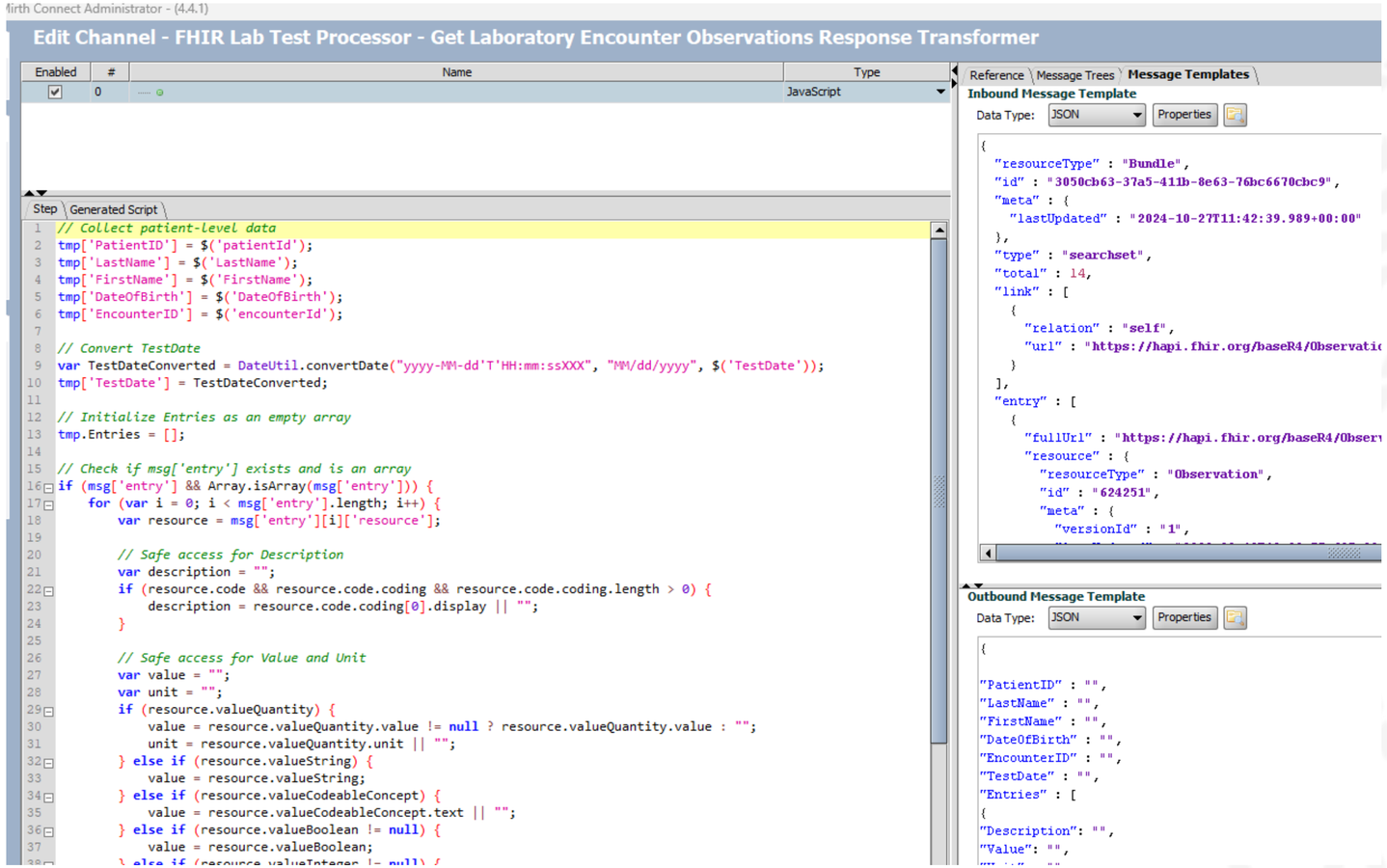
Get Laboratory Encounter Observations (1)



Utilizziamo ora l'endpoint per la risorsa *observation* per andare a leggere i valori individuali.

IMPLEMENTAZIONE FHIR

Get Laboratory Encounter Observations (2)



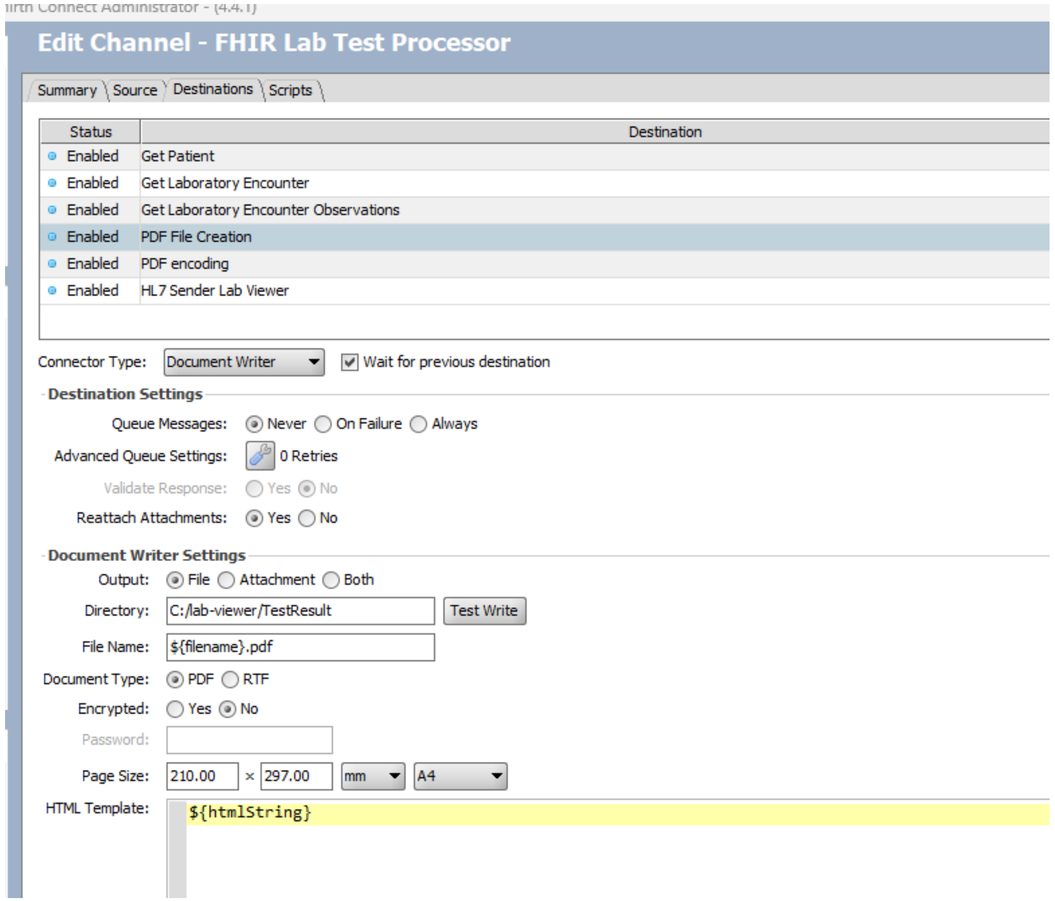
La risposta è un array (FHIR *bundle*) contente diverse risorse di tipo *observation*, ognuna indicante il tipo di sostanza ricercata (es. **glucosio**) e il valore.

Per rendere il documento finale in un formato leggibile, andiamo quindi a leggere nuovamente le variabili del paziente e la data del risultato assieme a tutti i singoli valori utilizzando *JavaScript*.

Il tutto viene formattato come una HTML string che andremo poi a convertire in un file **PDF.**

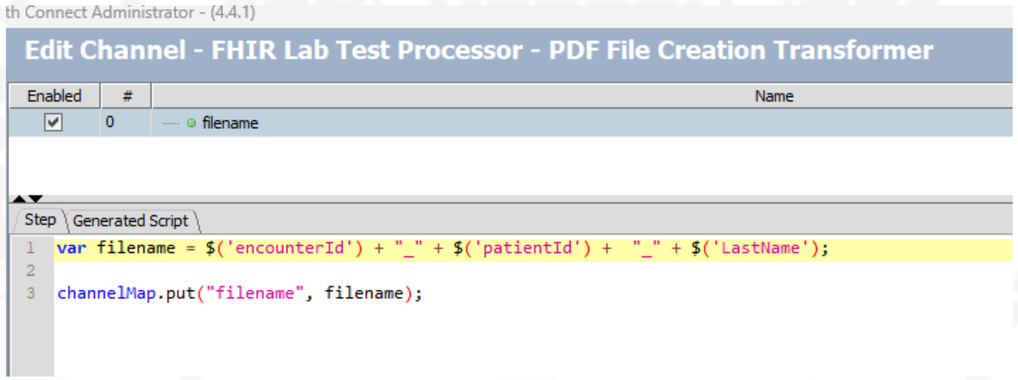
IMPLEMENTAZIONE FHIR

PDF File Creation



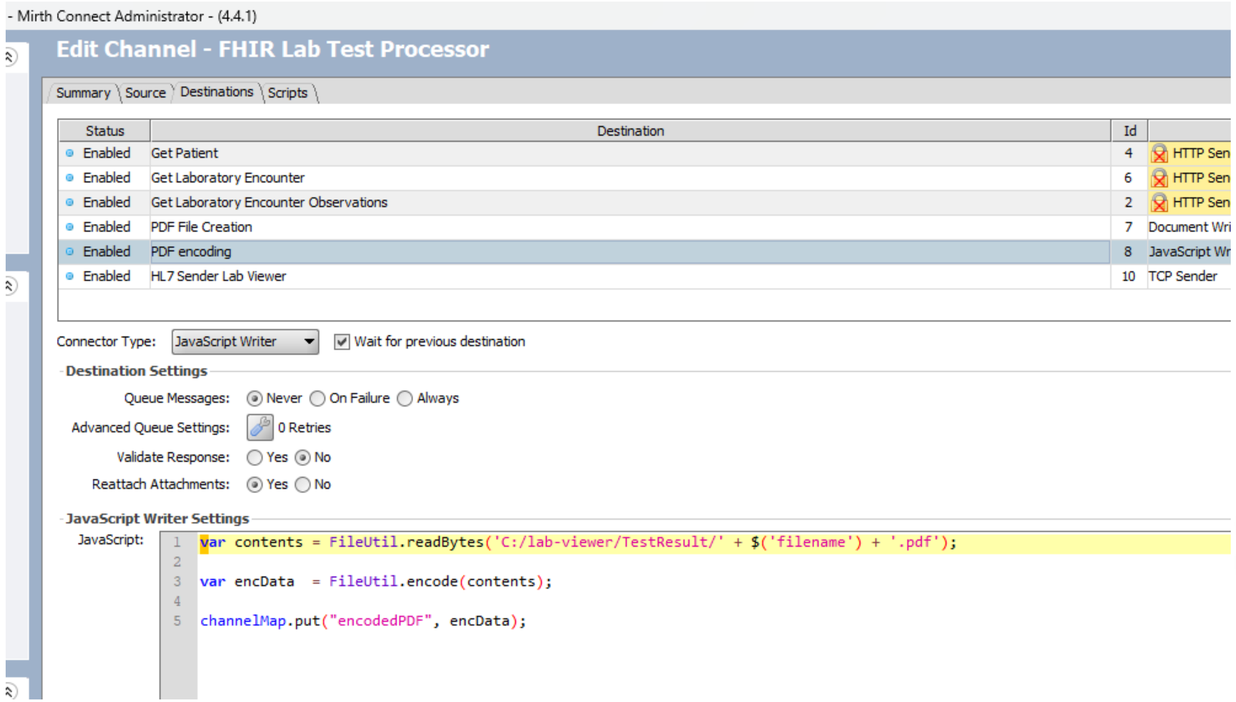
L'HTML string viene convertita in un file PDF utilizzando un connector type di tipo *Document Writer*. Il nome del file è ottenuto con un semplice destination transformer concatenando *l'encounter ID*, il *patient ID* e il cognome.

Indichiamo l'estensione (**.pdf**) e il percorso della cartella dove andremo a scrivere il file (**C:/lab-viewer/TestResult**).



IMPLEMENTAZIONE FHIR

PDF Encoding



Andiamo a leggere il file PDF appena creato utilizzando un *JavaScript writer* dove usiamo la *Java class* **FileUtil.** Questa classe offre una serie di *methods* utilizzabili direttamente in *JavaScript* per leggere o scrivere file.

In questo caso, leggiamo dalla cartella precedentemente utilizzata il file PDF appena creato e lo encodiamo utilizzando la codifica **base64.**

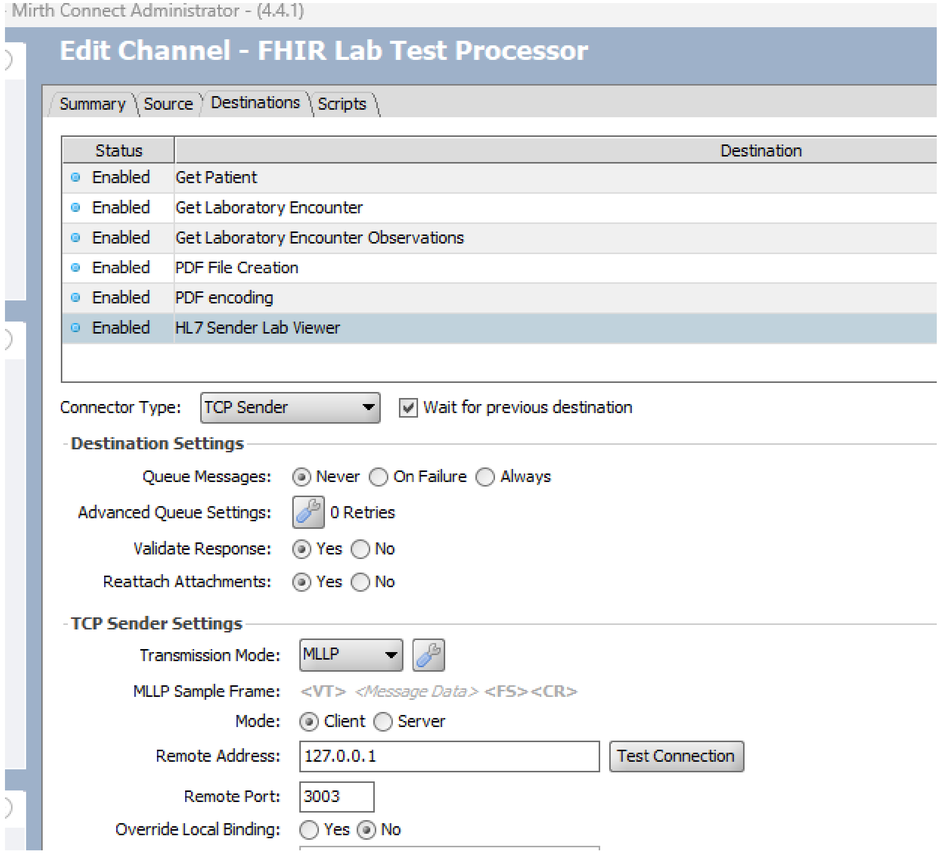
Per approfondire:

http://javadocs.mirthcorp.com/connect/3.2.2/user-api/com/mirth/connect/server/userutil/FileUtil.html

https://en.wikipedia.org/wiki/Base64

IMPLEMENTAZIONE FHIR

HL7 Sender

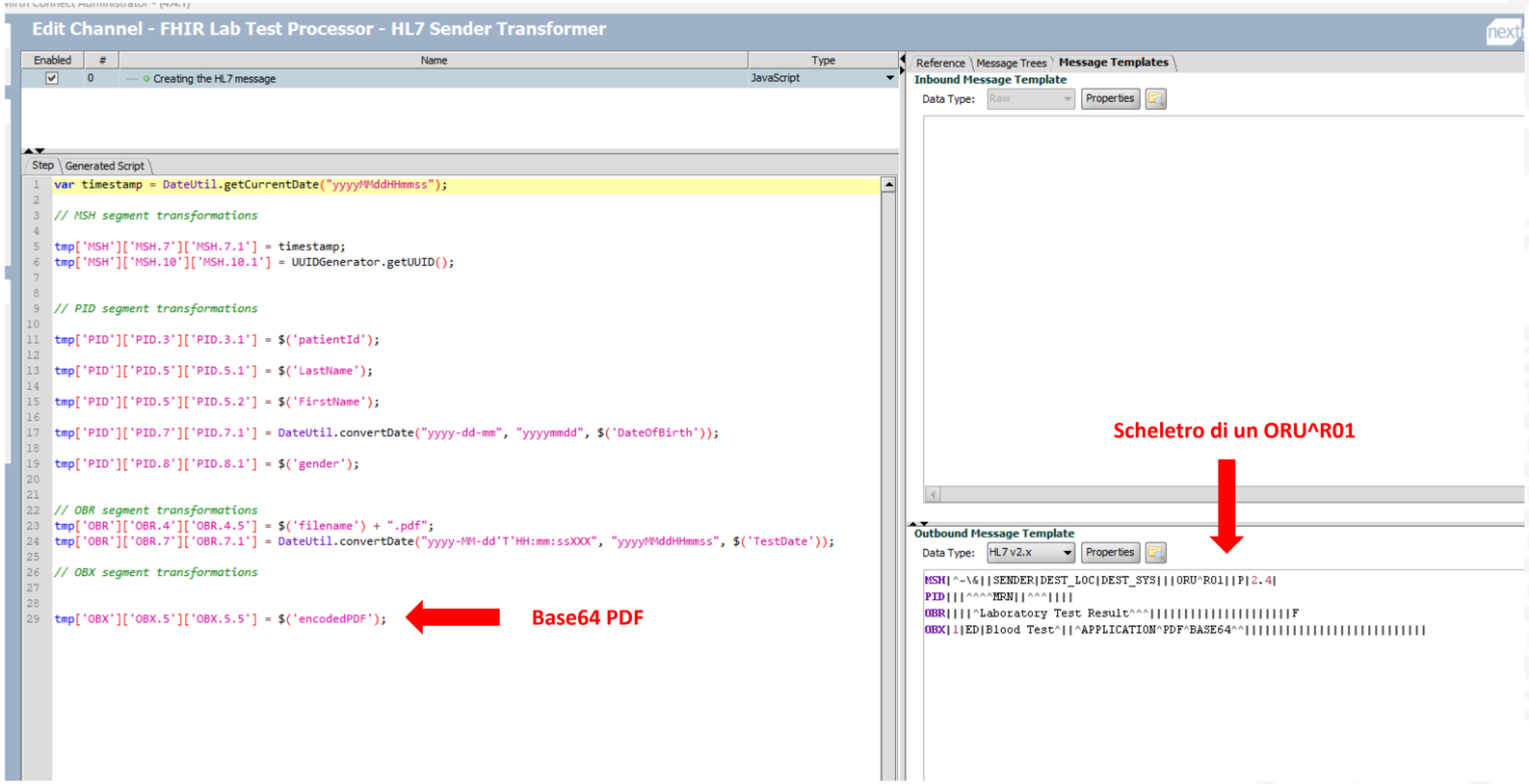


L'ultima destinazione è un *TCP Sender* che invierà un messaggio **HL7 ORU^R01** di ritorno all'applicazione labviewer, per la visualizzazione del PDF nell'applicativo.

Il file salvato localmente ai fini di archiviazione può essere ovviamente aperto e visualizzato con un qualsiasi PDF viewer

IMPLEMENTAZIONE FHIR

HL7 Sender Transformer



Il messaggio HL7 outbound viene creato attraverso un *destination transformer* di tipo *JavaScript* dove utilizziamo un outbound template che definisce lo «scheletro» del messaggio.

Il messaggio è un **ORU^R01** comunemente utilizzato per inviare risultati e report.

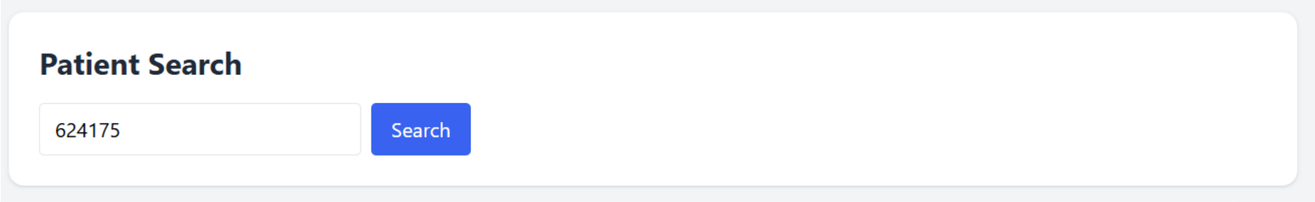
Lo «scheletro» viene riempito con le variabili per ciascuna transazione, ovvero i dati demografici del paziente, l'identificativo del messaggio con il timestamp e ovviamente il documento PDF in formato **base64** che andremo ad aggiungere in un segmento **OBX** (*observation*).

UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

DIMOSTRAZIONE END TO END

IMPLEMENTAZIONE FHIR

Dimostrazione end-to-end



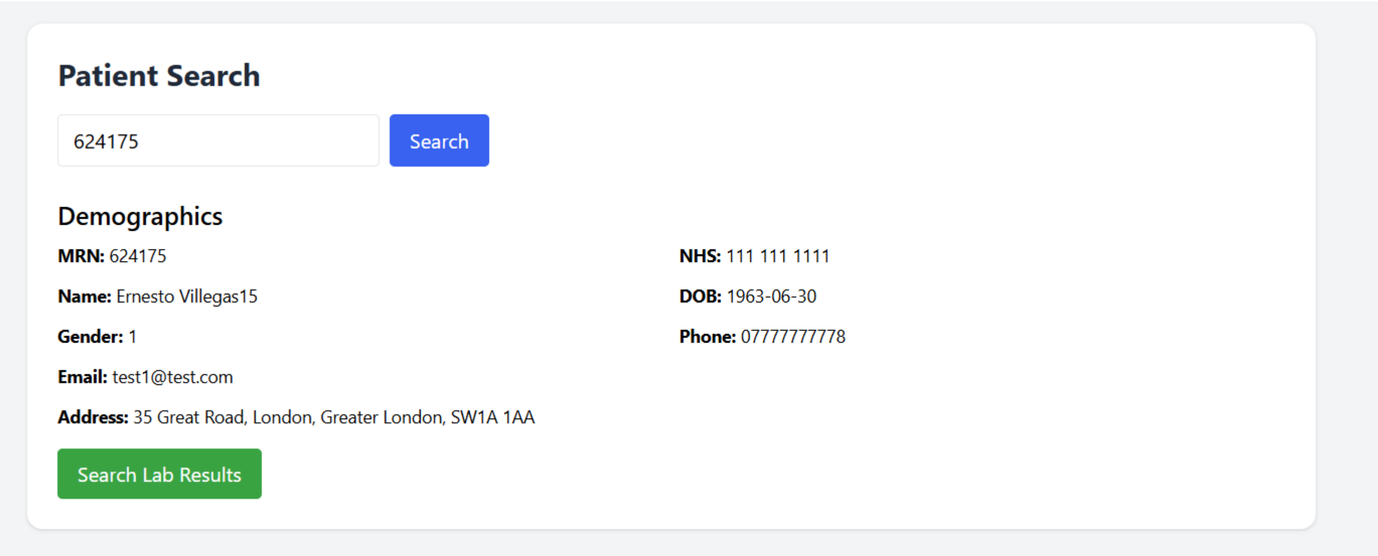
Iniziamo la nostra ricerca paziente dall'applicativo lab view, inserendo l'identificativo paziente (**MRN**, *medical record number*) e cliccando sul tasto *Search*.

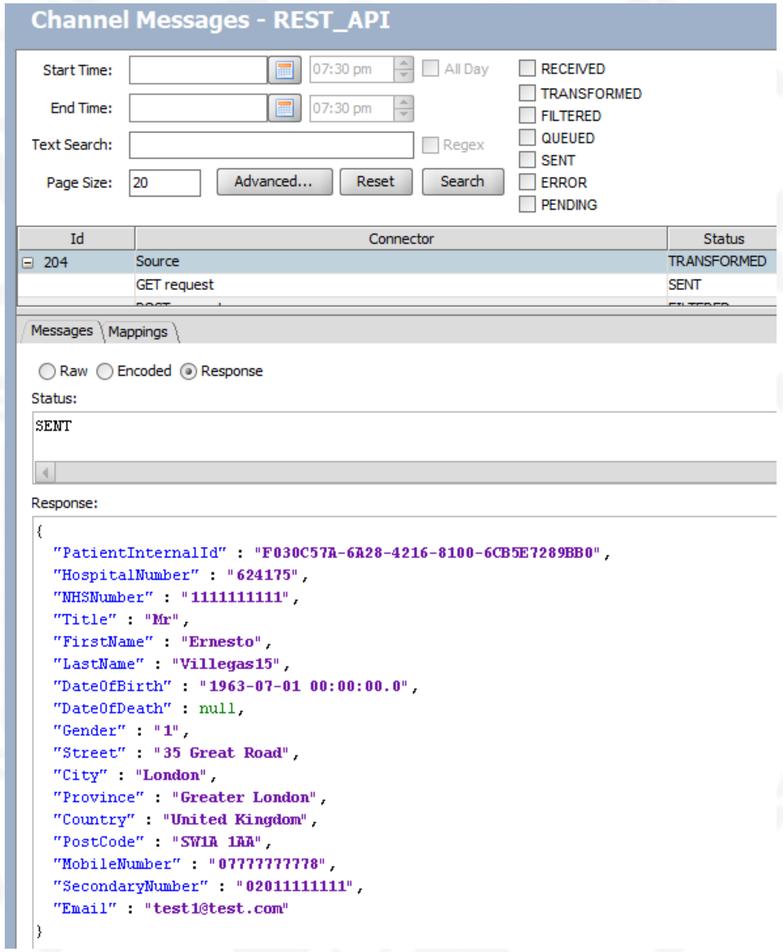
Questa azione produrra una richiesta http verso la porta 8003 del tipo:

GET http://localhost:8003/Patient?MRN=624175

IMPLEMENTAZIONE FHIR

Dimostrazione end-to-end



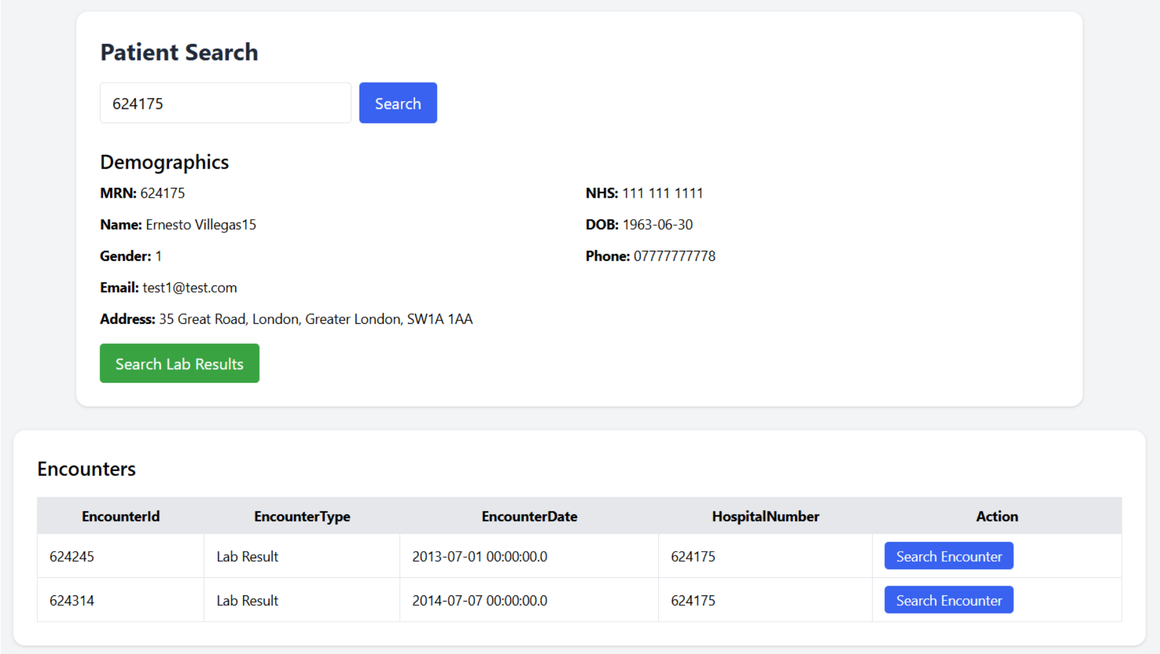


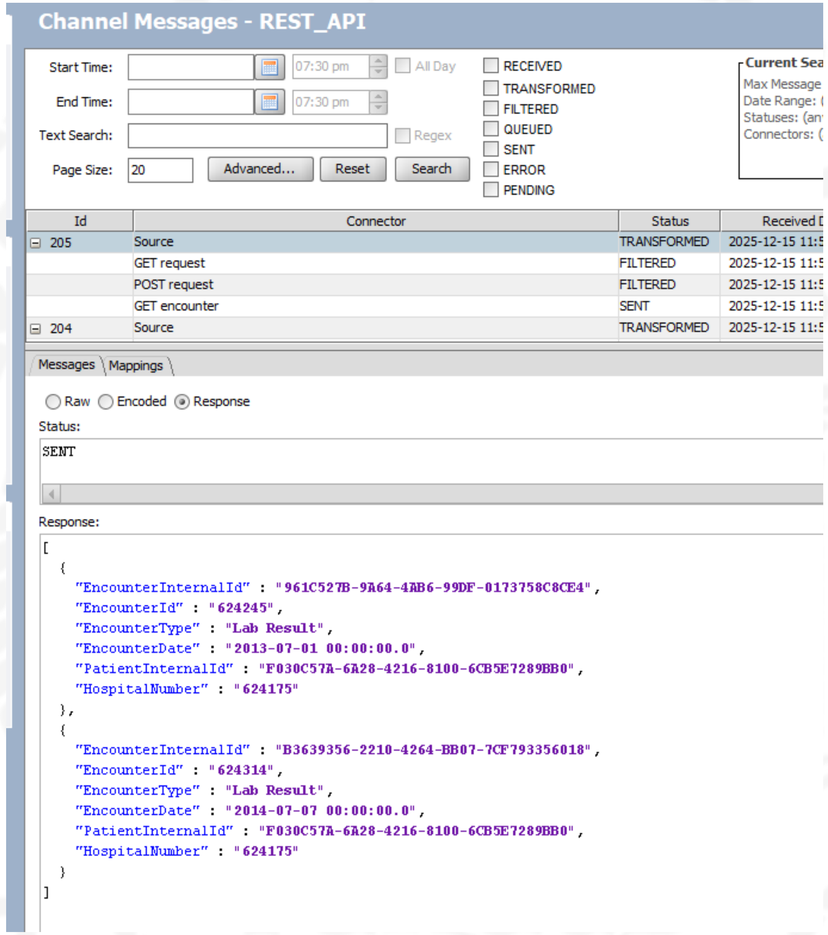
Otteniamo subito il codice http **200** internamente, ovvero la richiesta è ricevuta da Mirth correttamente e il record viene quindi visualizzato sullo schermo istantaneamente.

La risposta ottenuta da Mirth è un payload JSON che viene utilizzato dall'applicazione per la visualizzazione dei dati paziente. Il record è recuperato da un database MS SQL.

IMPLEMENTAZIONE FHIR

Dimostrazione end-to-end





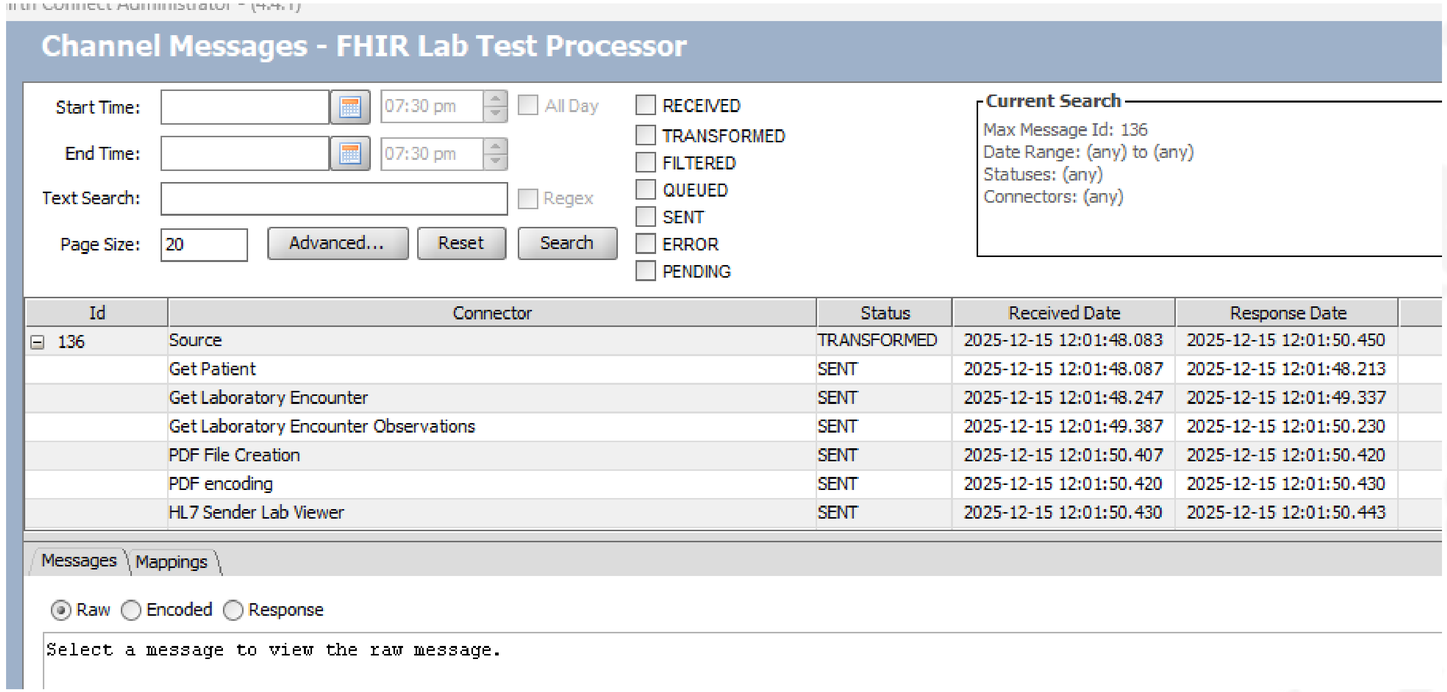
Successivamente, cliccando su *Search Lab Results* inviamo una seconda richiesta HTTP del tipo:

GET ***http://localhost:8003/Encounter?MRN=624175***

Che ci permette di recuperare tutti gli *encounter* per quel determinato paziente e visualizzarli sullo schermo.

IMPLEMENTAZIONE FHIR

Dimostrazione end-to-end



Se clicchiamo adesso sul tasto *Search Encounter*, invieremo la seguente richiesta HTTP verso Mirth:

GET ***http://localhost:8004/BloodTest?PatientId=624175&EncounterId=624245***

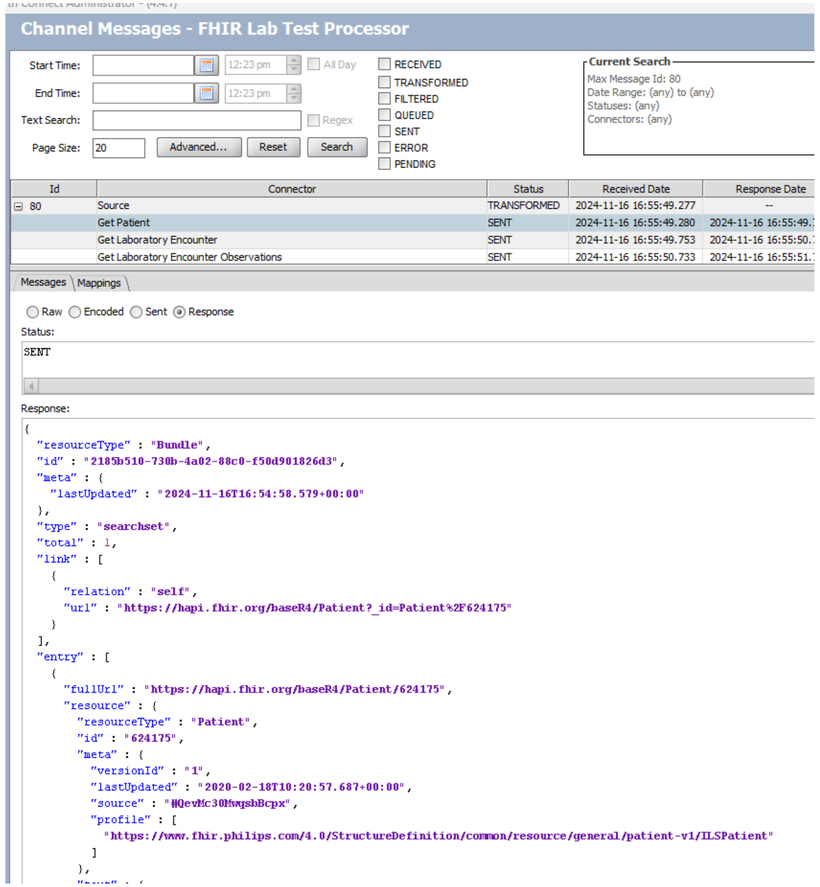
Si noti come la porta ora è la 8004 perchè

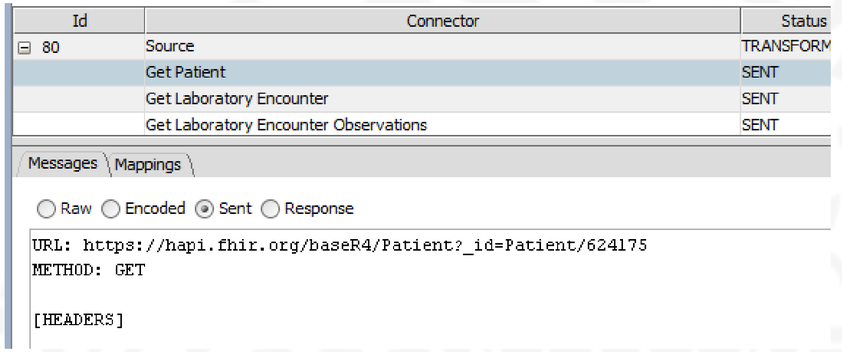
utilizziamo il canale Mirth **FHIR Lab Test Processor**.

Nel *dashboard screen* in Mirth possiamo osservare che tutte le destinazioni sono state eseguite correttamente senza errori.

IMPLEMENTAZIONE FHIR

Dimostrazione end-to-end

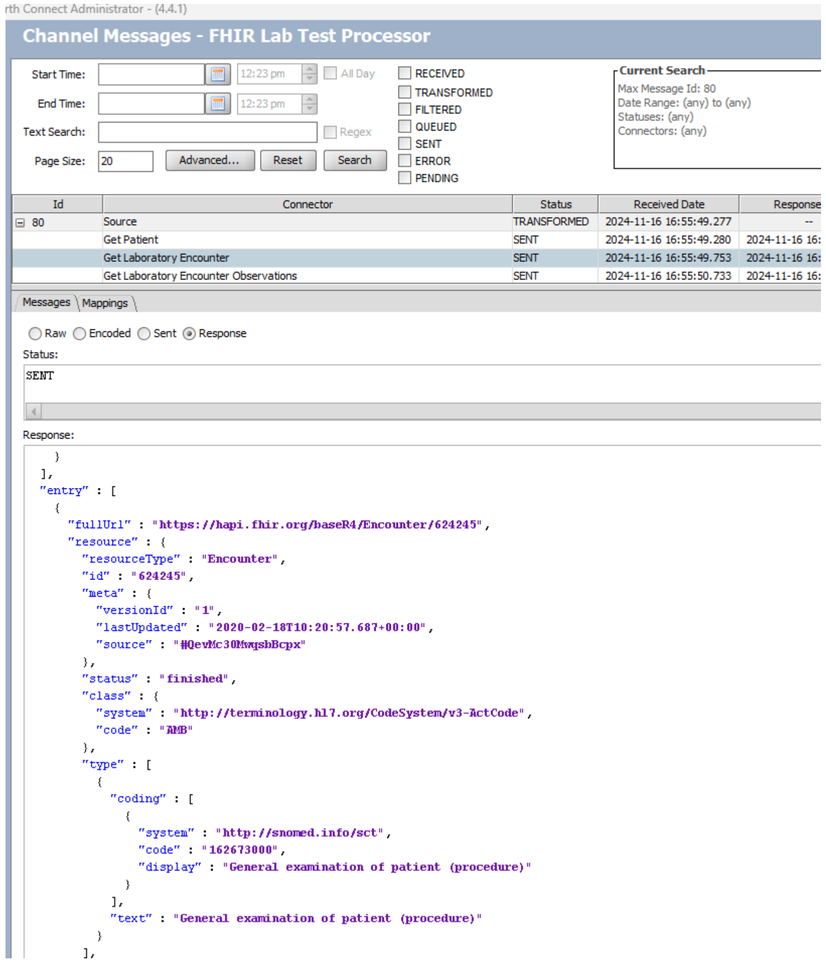


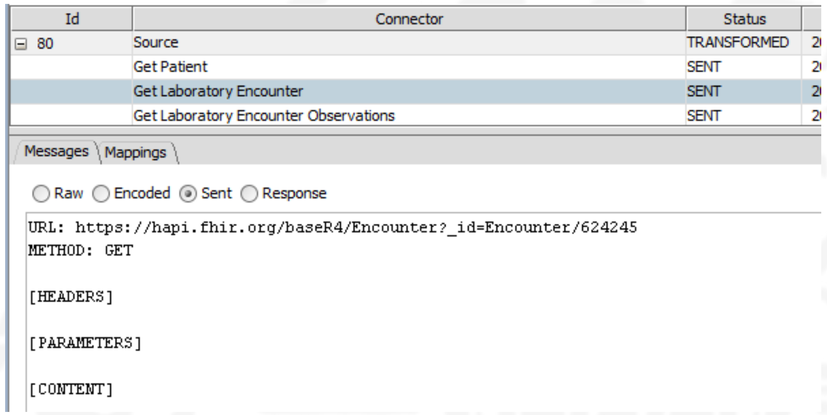


La richiesta GET per la risorsa *patient* da come risposta i dettagli del paziente ricercato in formato JSON secondo lo schema FHIR.

IMPLEMENTAZIONE FHIR

Dimostrazione end-to-end

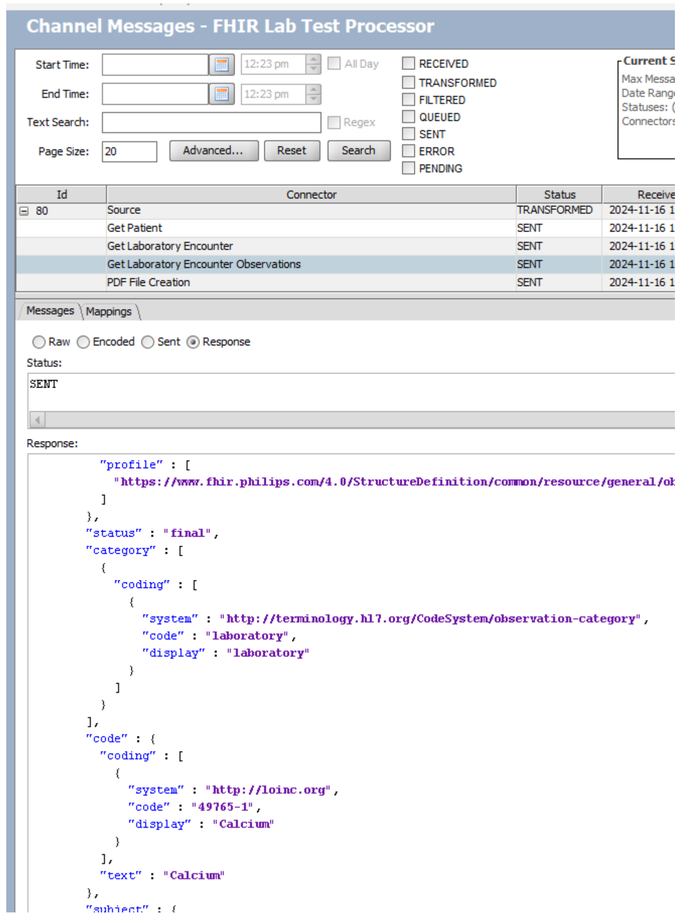


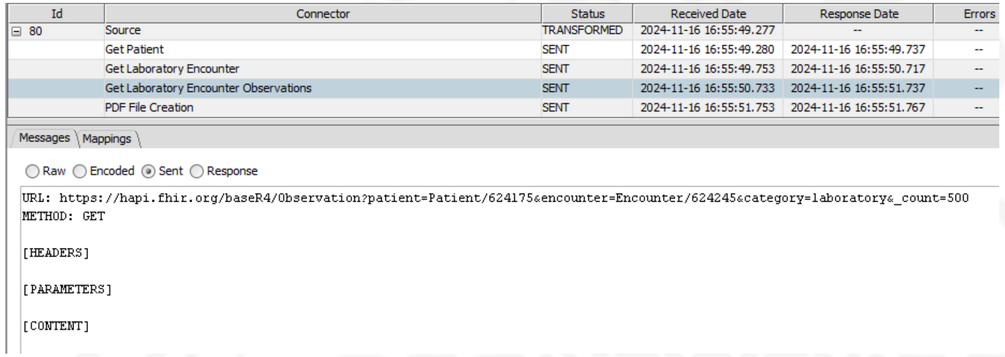


Similmente la richiesta GET per la risorsa *encounter* ritorna il payload in JSON dell'esame del sangue.

IMPLEMENTAZIONE FHIR

Dimostrazione end-to-end

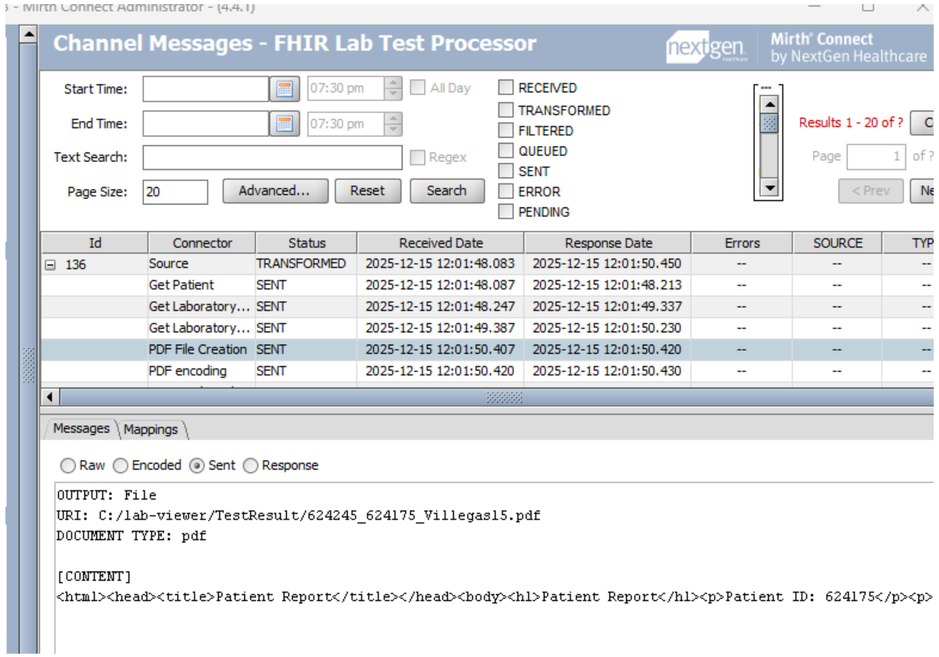


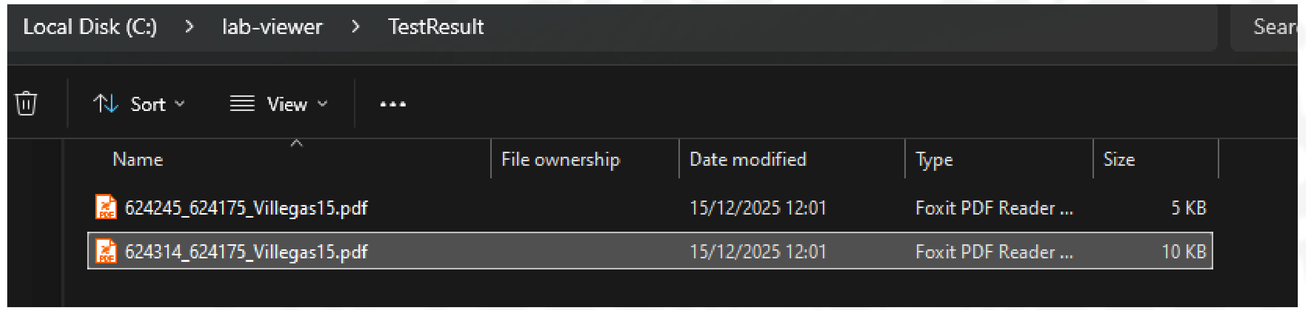


Stessa cosa per le singole *observation* per i valori che andiamo a leggere.

IMPLEMENTAZIONE FHIR

Dimostrazione end-to-end

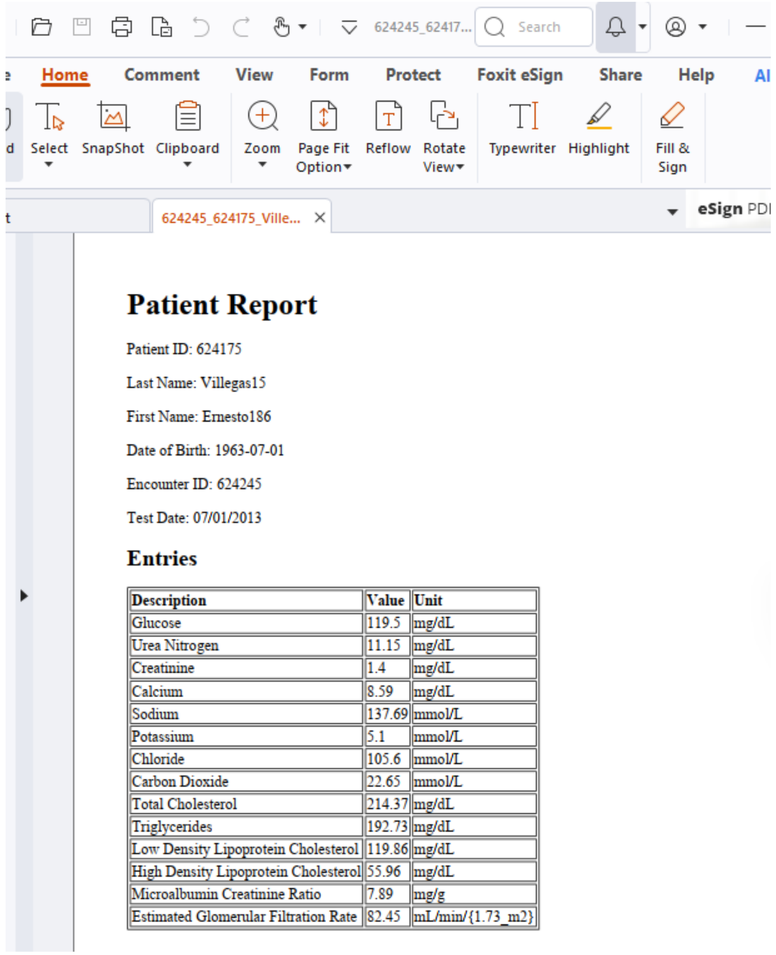




Il file PDF viene creato correttamente nella cartella di destinazione.

IMPLEMENTAZIONE FHIR

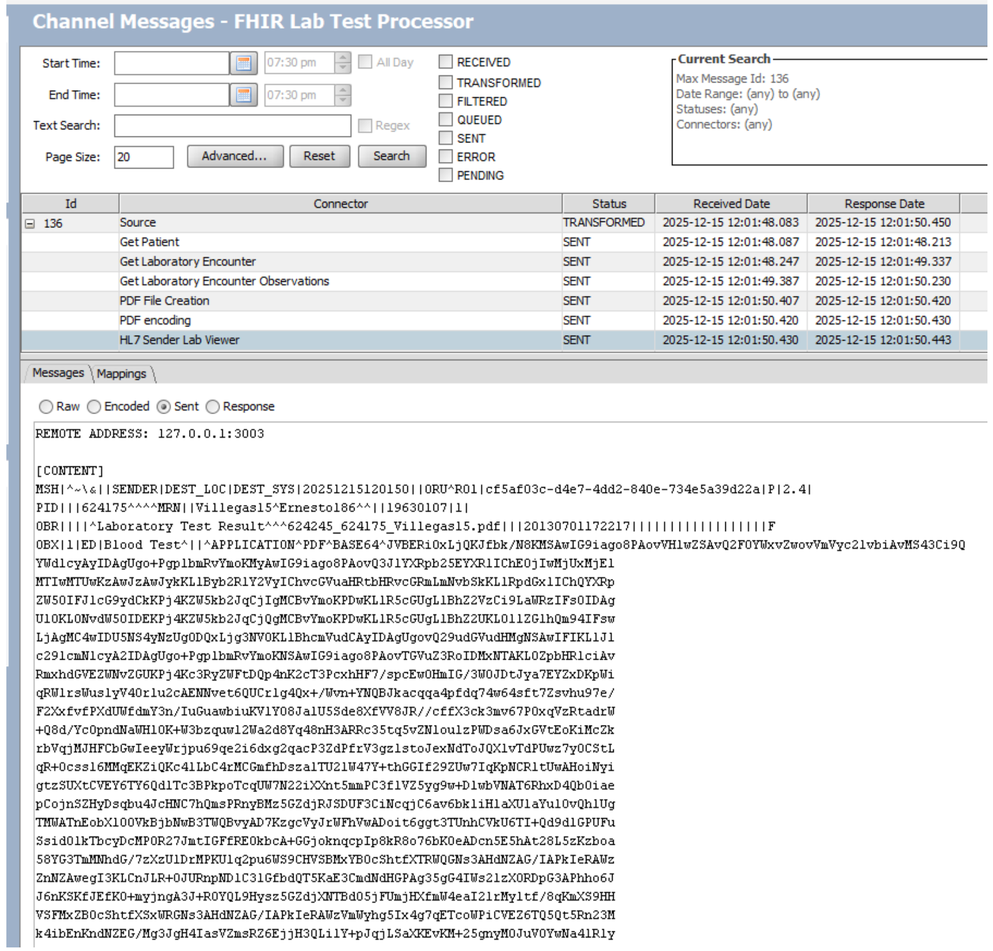
Dimostrazione end-to-end



Il file può quindi essere aperto e visualizzato con un qualsiasi PDF viewer.

IMPLEMENTAZIONE FHIR

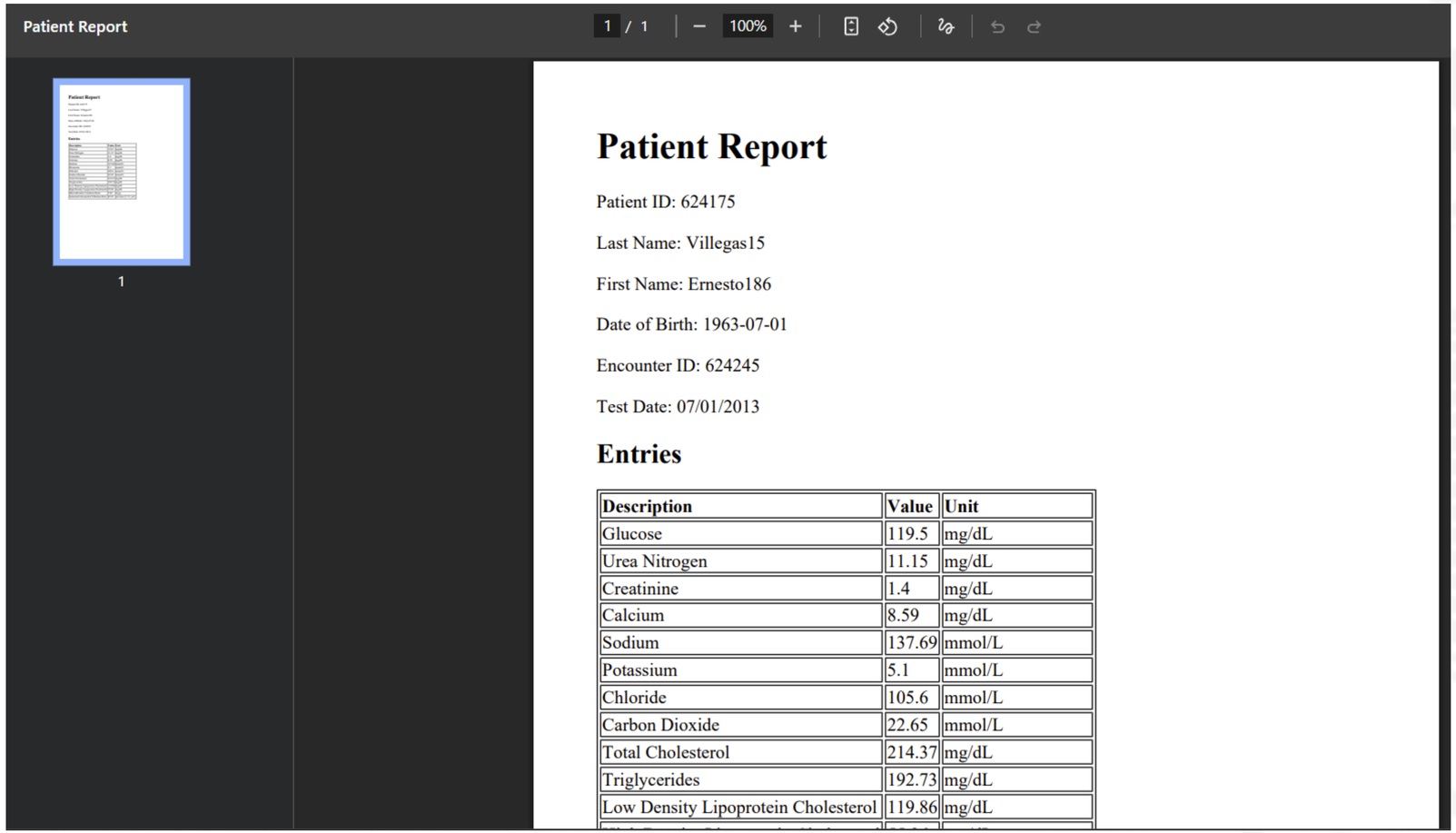
Dimostrazione end-to-end



Infine nell'ultimo step dopo essere letto dalla cartella e encodato in Base64, il file PDF è incapsulato nel messaggio HL7 di outbound **ORU^R01** e inviato verso la destinazione TCP, rappresentata localmente da **127.0.0.1** (*localhost*) su porta **3003** e utilizzata dall'applicativo lab viewer per visualizzare il PDF sullo schermo.

IMPLEMENTAZIONE FHIR

Dimostrazione end-to-end



Il messaggio **ORU^R01** è ricevuto e il PDF è visualizzato sullo schermo tramite un semplice pop up.

IMPLEMENTAZIONE FHIR

Conclusioni

Abbiamo visto come FHIR possa aiutarci a leggere informazioni sanitarie complesse, come ad esempio risultati di laboratorio, combinando diverse risorse FHIR tra loro (*patient, encounter, encounter observations*), transformarle in un file PDF facilmente fruibile da pazienti e operatori sanitari e condividendo ulteriormente i dati dell'esame con un sistema terzo utilizzando HL7.

La piena sinergia tra FHIR e HL7 permette di ottenere ottimi risultati in termini di interoperabilità tra sistemi con il minimo sforzo da un punto prettamente tecnico per l'implementazione della soluzione grazie all'utilizzo di Mirth Connect.