

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 ELETTRONICHE MEDICALI CON TBS GROUP (ORA ALTHEA)

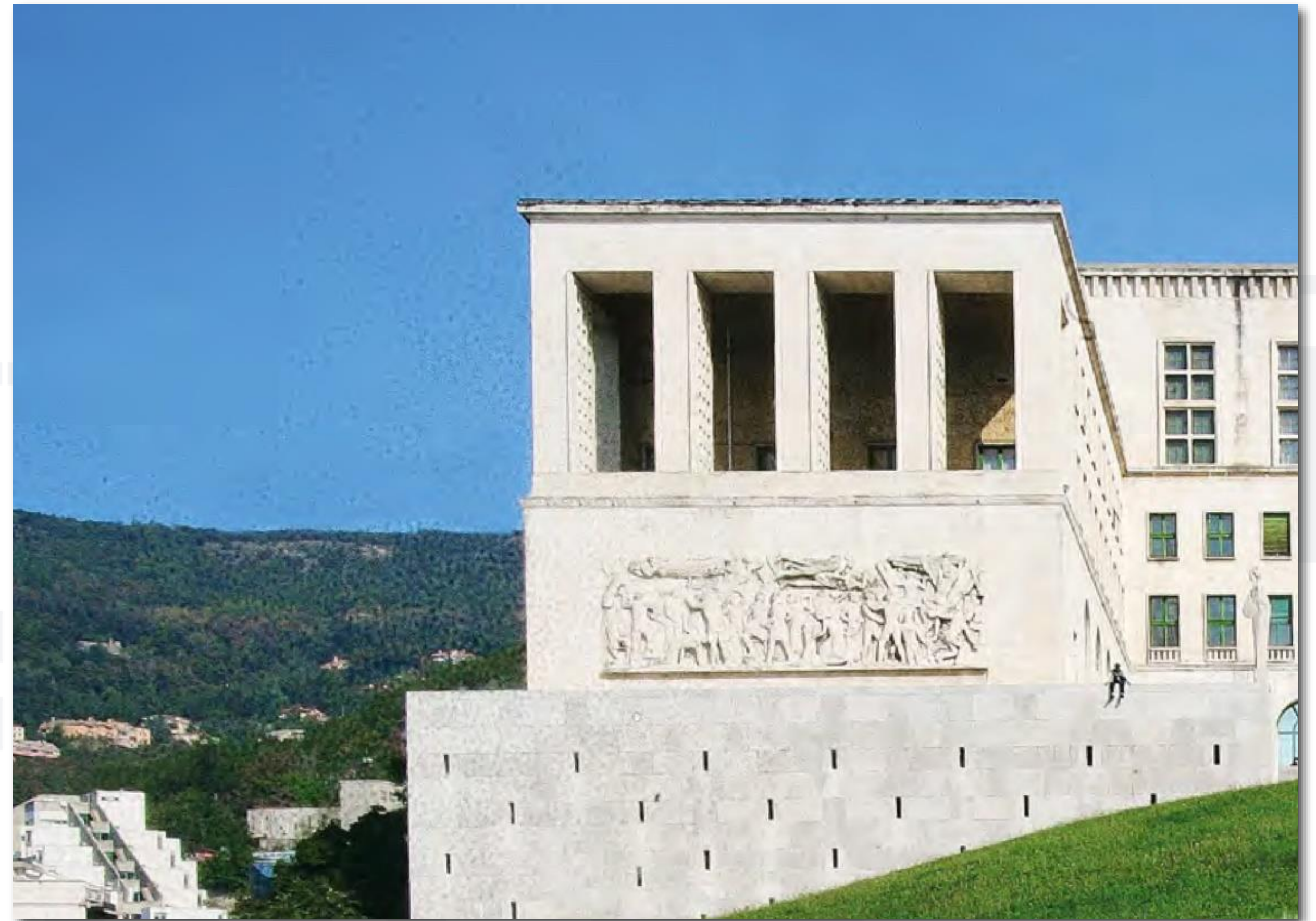
VIVO E LAVORO DAL 2014 NEL REGNO UNITO DOVE MI OCCUPO DI INTEGRAZIONE DI SISTEMI IN SANITÀ



CONTATTI:

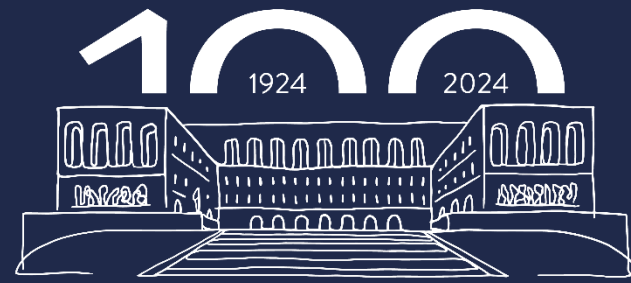
[WWW.LINKEDIN.COM/IN/MARIO-DAMIANO-26503126](https://www.linkedin.com/in/mario-damiano-26503126)

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 *Representational 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 crittazione 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:

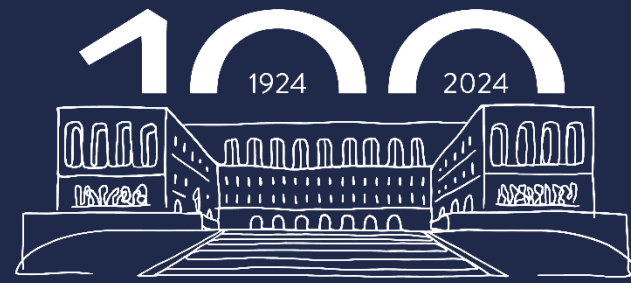
- ✓ **GET**: per ottenere dati.
- ✓ **PUT/PATCH**: per modificare lo stato dei dati
- ✓ **POST**: per creare nuovi dati.
- ✓ **DELETE**: per eliminare dati.

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.



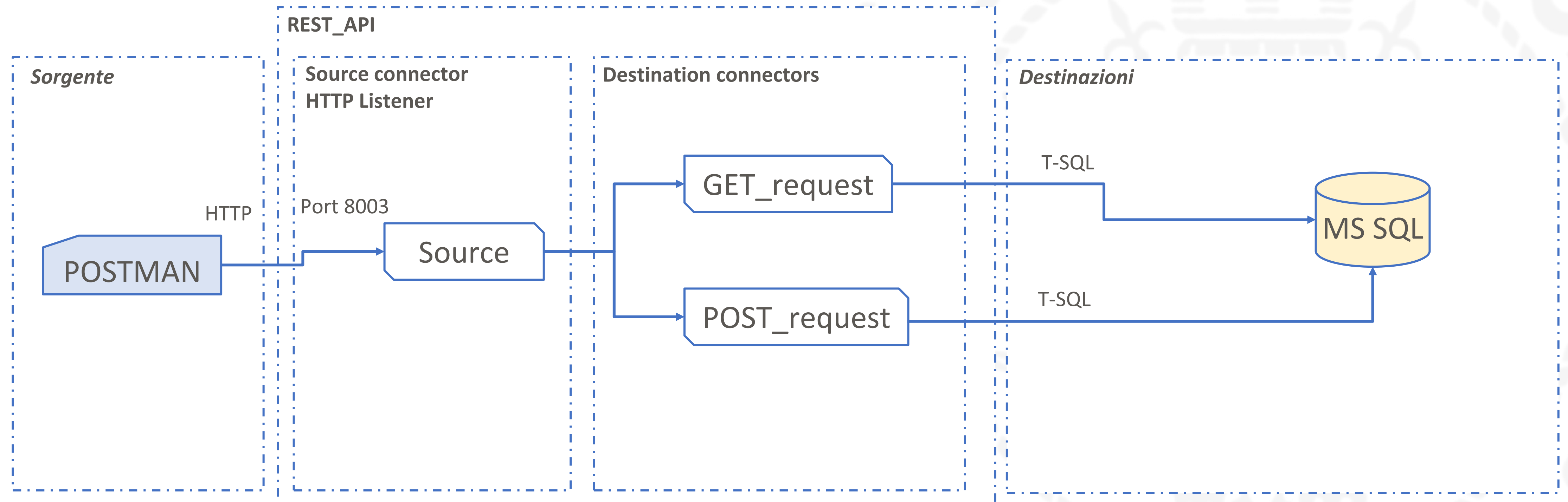
UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

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

Edit Channel - REST_API

Summary | Source | Destinations | Scripts

Connector Type: HTTP Listener

Listener Settings

Local Address: All interfaces Specific interface: 0.0.0.0

Local Port: 8003

Source Settings

Source Queue: OFF (Respond after processing)

Queue Buffer Size: 1000

Response: Response

Process Batch: Yes No

Batch Response: First Last

Max Processing Threads: 1

HTTP Authentication

Authentication Type: None

HTTP Listener Settings

Base Context Path: Patient

Receive Timeout (ms): 30000

Message Content: Plain Body XML Body

Parse Multipart: Yes No

Include Metadata: Yes No

Binary MIME Types: application/*, image/*, video/*, audio/* Regular Expression

HTTP URL: http://localhost:8003/Patient/

Response Content Type: text/plain

Response Data Type: Binary Text

Charset Encoding: Default

Response Status Code: responseStatusCode

Response Headers: Use Table Use Map

Name

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

Decidiamo di utilizzare la porta **8003**.

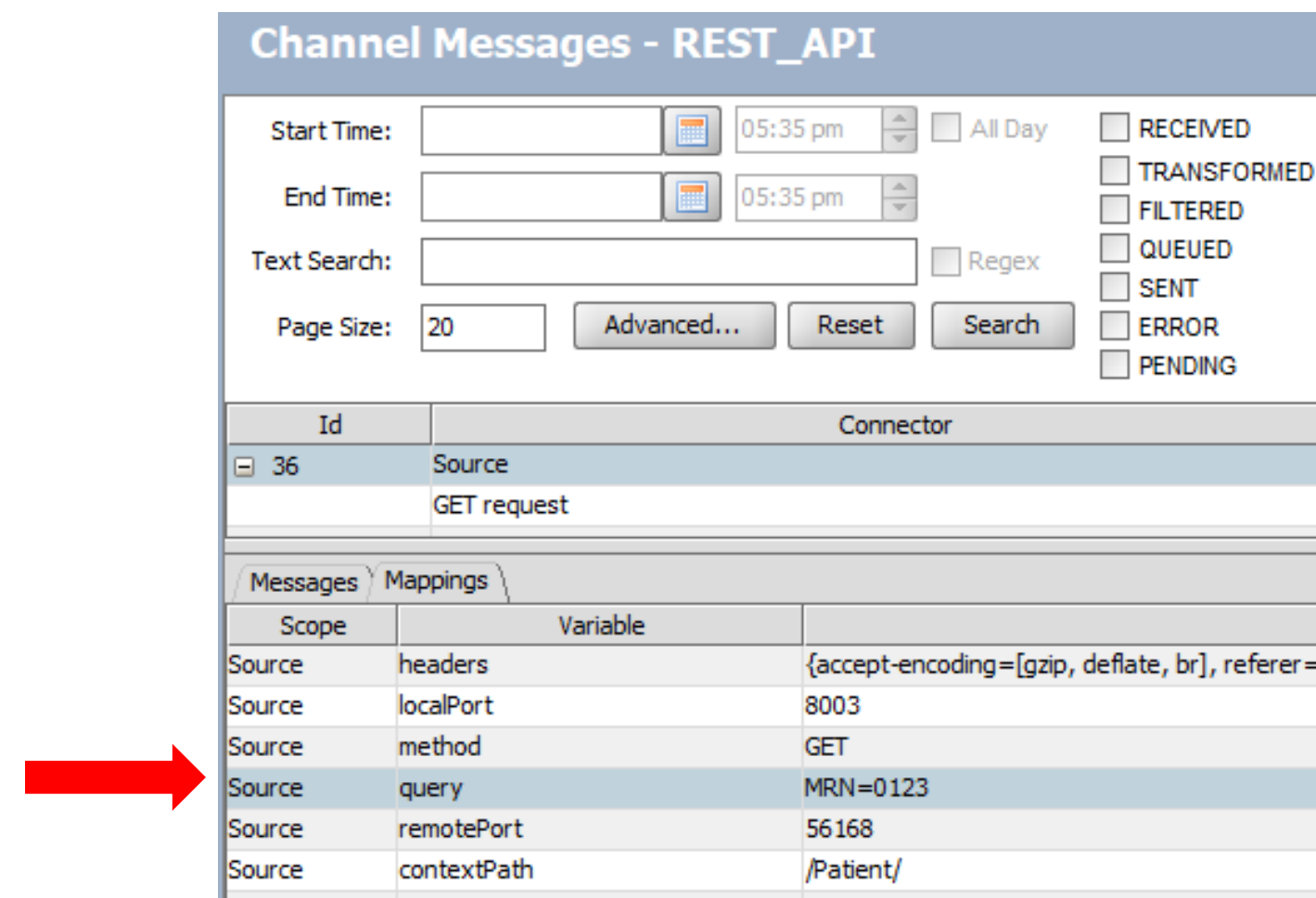
Mirth Connect consente di inviare una risposta dinamica verso il sistema richiedente (**Postman**), utilizziamo per tanto una variabile *Response* che andremo a definire più avanti.

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



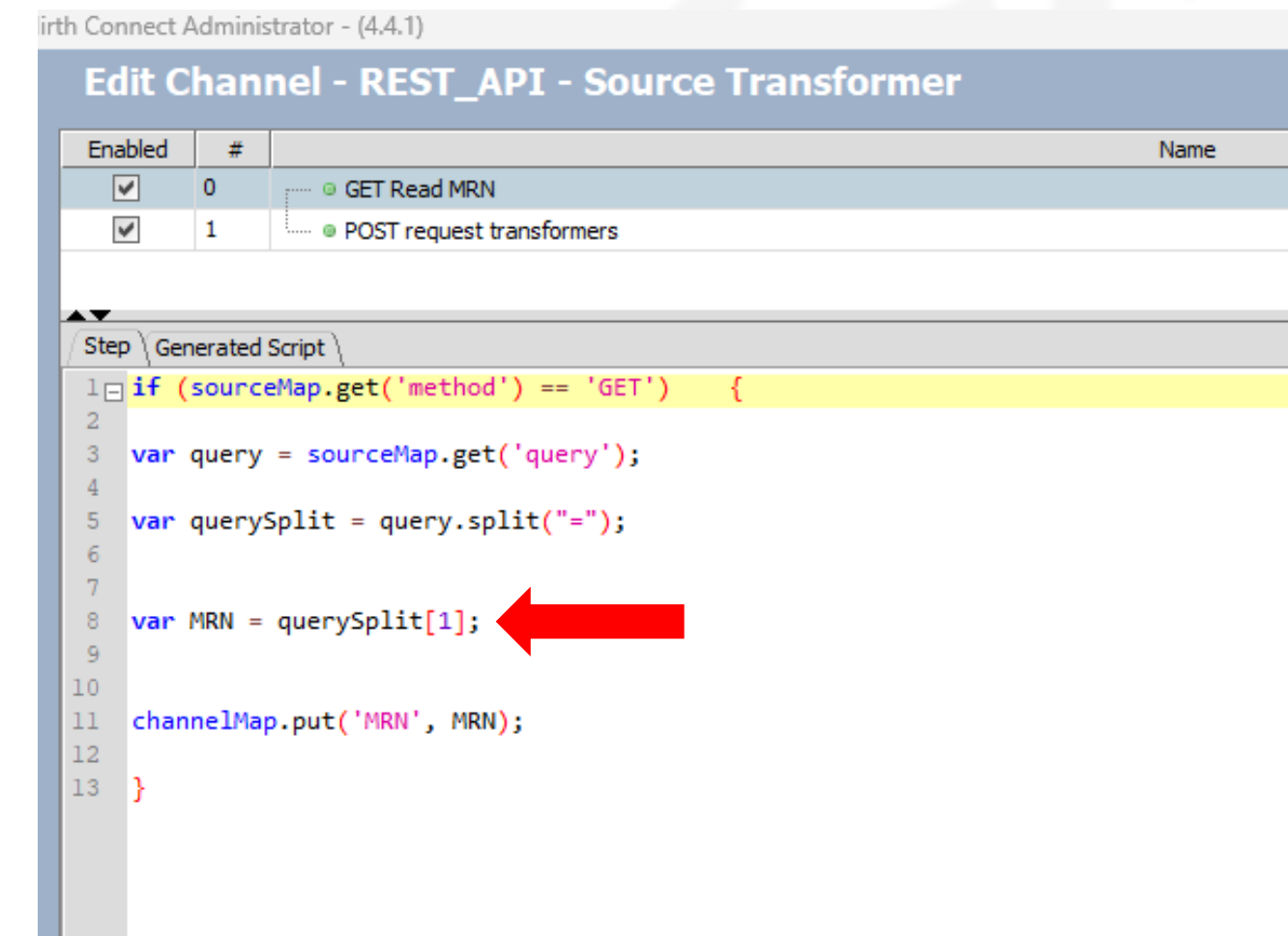
Channel Messages - REST_API

Start Time: 05:35 pm All Day RECEIVED
End Time: 05:35 pm TRANSFORMED
Text Search: Regex FILTERED
Page Size: 20 Advanced... Reset Search QUEUED
SENT
ERROR
PENDING

Id	Connector
36	Source
	GET request

Messages Mappings

Scope	Variable	
Source	headers	{accept-encoding=[gzip, deflate, br], referer=
Source	localPort	8003
Source	method	GET
Source	query	MRN=0123
Source	remotePort	56168
Source	contextPath	/Patient/



irth Connect Administrator - (4.4.1)

Edit Channel - REST_API - Source Transformer

Enabled	#	Name
<input checked="" type="checkbox"/>	0	GET Read MRN
<input checked="" type="checkbox"/>	1	POST request transformers

Step Generated Script

```
1 if (sourceMap.get('method') == 'GET') {  
2  
3 var query = sourceMap.get('query');  
4  
5 var querySplit = query.split("=");  
6  
7  
8 var MRN = querySplit[1];  
9  
10  
11 channelMap.put('MRN', MRN);  
12  
13 }
```

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.

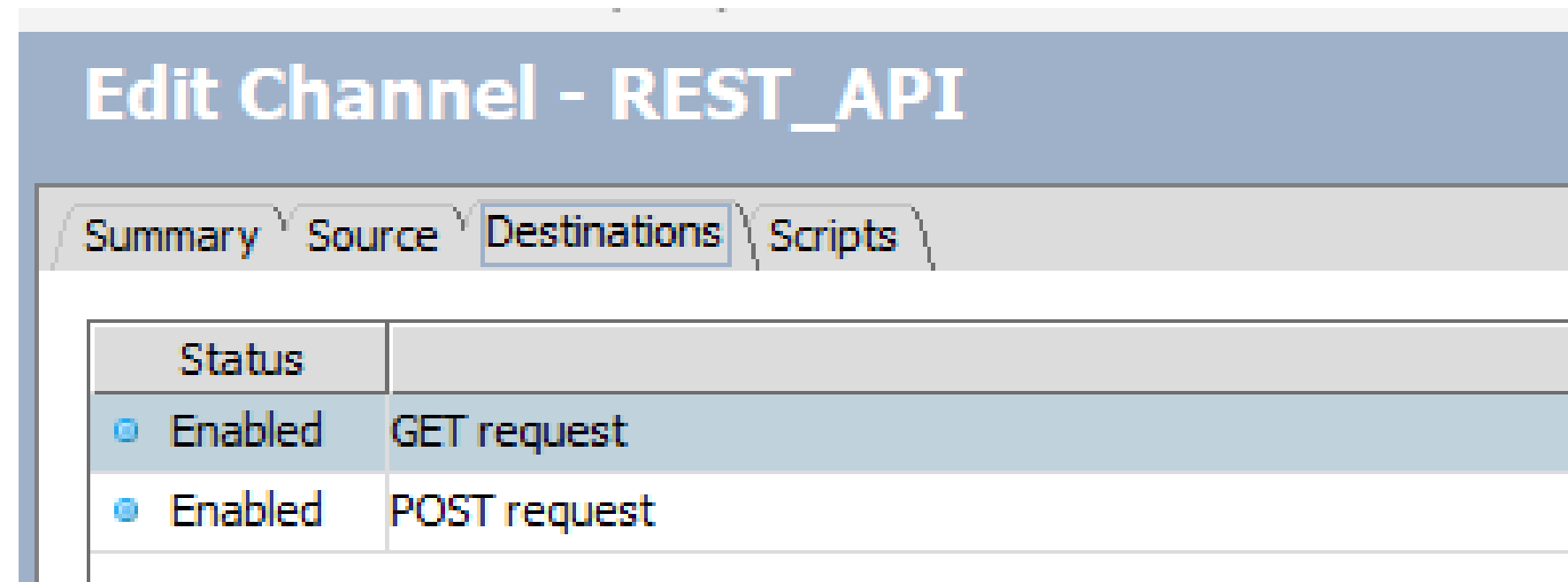
Una richiesta di tipo *GET* infatti, dopo aver definito il *context path*, sarà del tipo:

<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



Status	
<input checked="" type="radio"/> Enabled	GET request
<input checked="" type="radio"/> Enabled	POST request

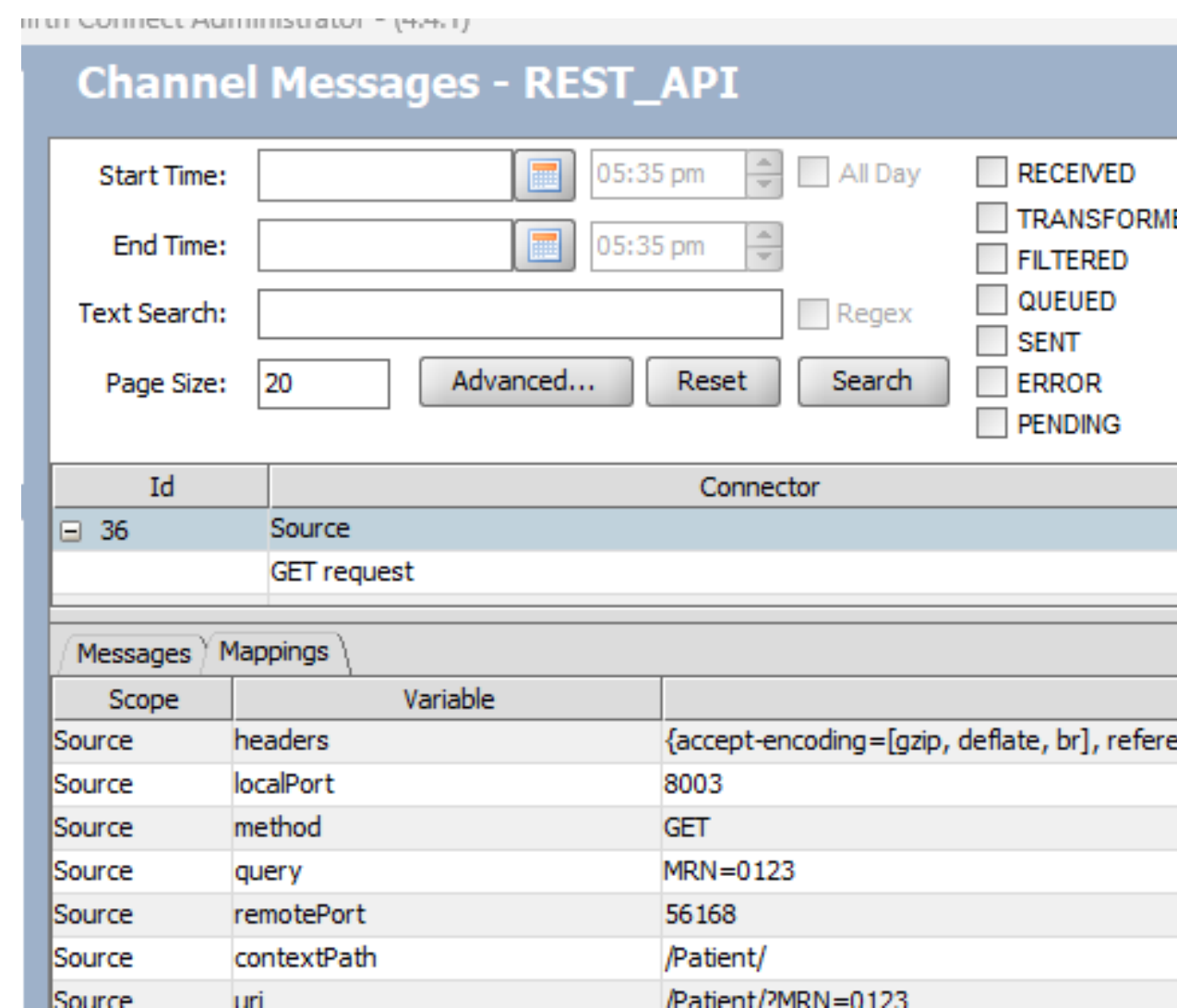
Il *destination connector* contiene due destinazioni separate di tipo *JavaScript writer*.

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

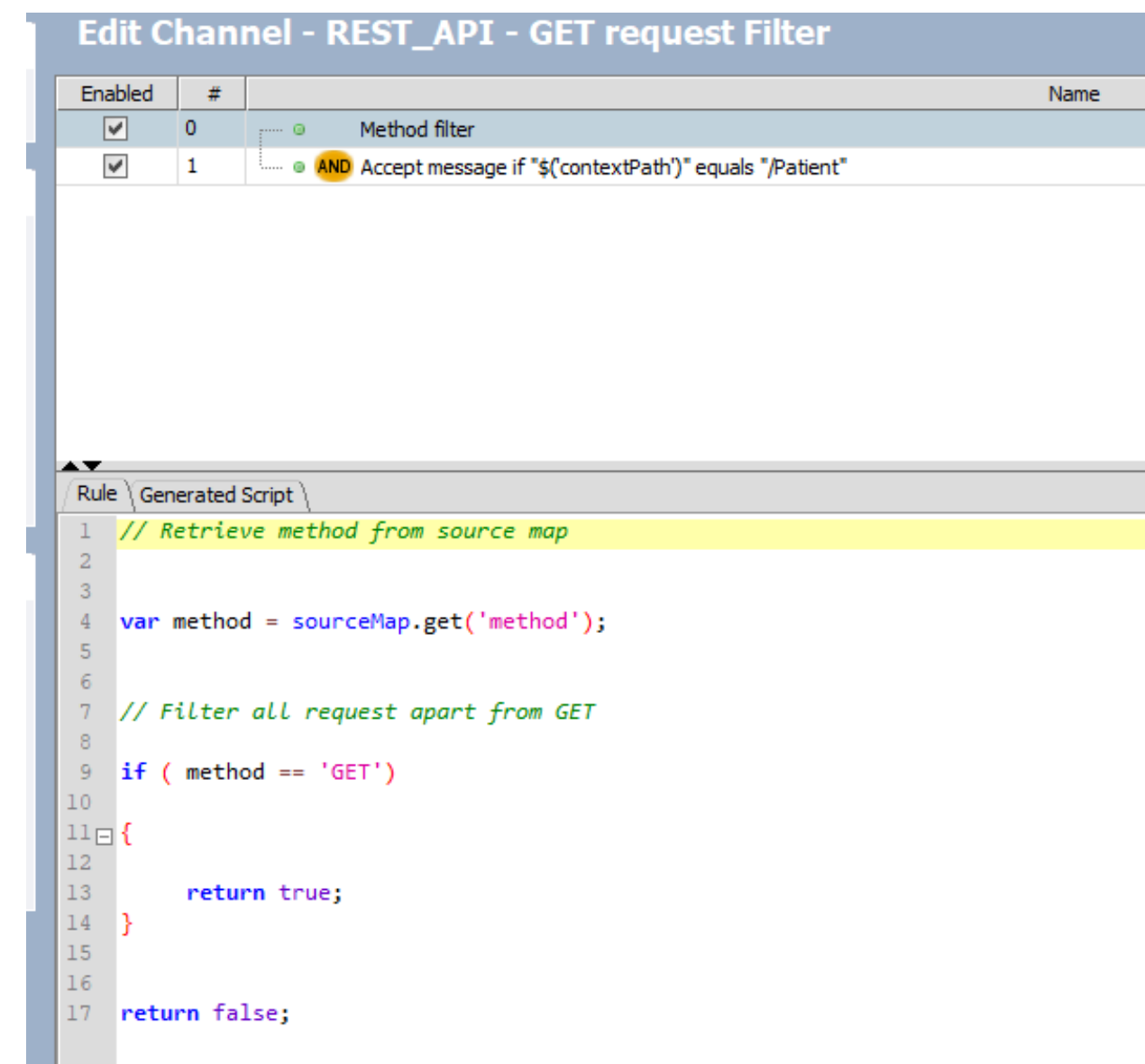


Channel Messages - REST_API

Start Time: 05:35 pm All Day RECEIVED
End Time: 05:35 pm TRANSFORME
Text Search: Regex FILTERED
Page Size: 20 Advanced... Reset Search QUEUED
SENT
ERROR
PENDING

Id	Connector
36	Source
	GET request

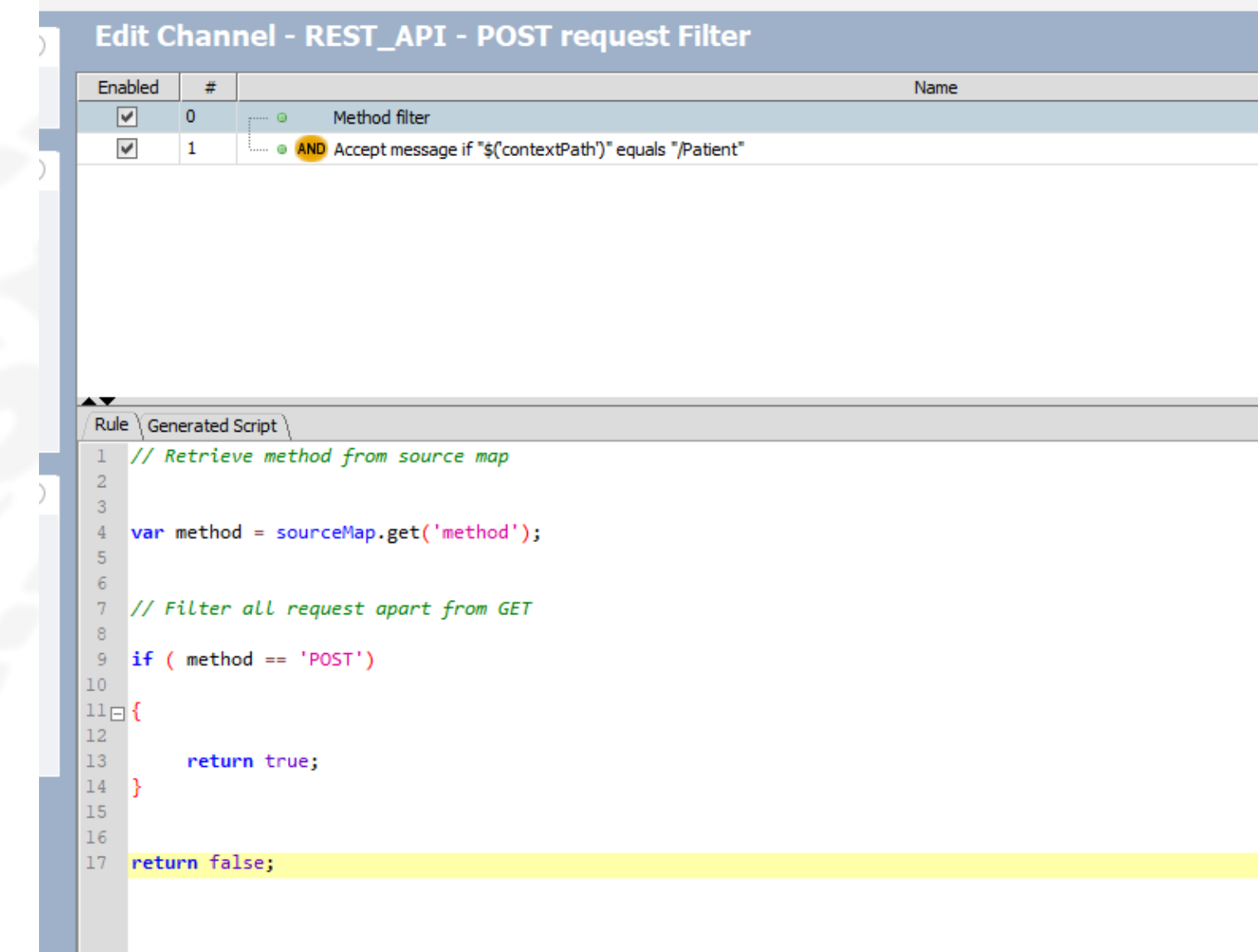
Scope	Variable	
Source	headers	{accept-encoding=[gzip, deflate, br], refere
Source	localPort	8003
Source	method	GET
Source	query	MRN=0123
Source	remotePort	56168
Source	contextPath	/Patient/
Source	uri	/Patient/MRN=0123



Edit Channel - REST_API - GET request Filter

Enabled	#	Name
<input checked="" type="checkbox"/>	0	Method filter
<input checked="" type="checkbox"/>	1	AND Accept message if "\${contextPath}" equals "/Patient"

```
1 // Retrieve method from source map
2
3
4 var method = sourceMap.get('method');
5
6
7 // Filter all request apart from GET
8
9 if ( method == 'GET')
10 {
11     return true;
12 }
13
14
15
16
17 return false;
```



Edit Channel - REST_API - POST request Filter

Enabled	#	Name
<input checked="" type="checkbox"/>	0	Method filter
<input checked="" type="checkbox"/>	1	AND Accept message if "\${contextPath}" equals "/Patient"

```
1 // Retrieve method from source map
2
3
4 var method = sourceMap.get('method');
5
6
7 // Filter all request apart from GET
8
9 if ( method == 'POST')
10 {
11     return true;
12 }
13
14
15
16
17 return false;
```

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

1 **AND** Accept message if "\${contextPath}" equals "/Patient"

Rule | Generated Script

Behavior: **Accept**

Field:

Condition: Exists Not Exist Equals Not Equal Contains Not Contain

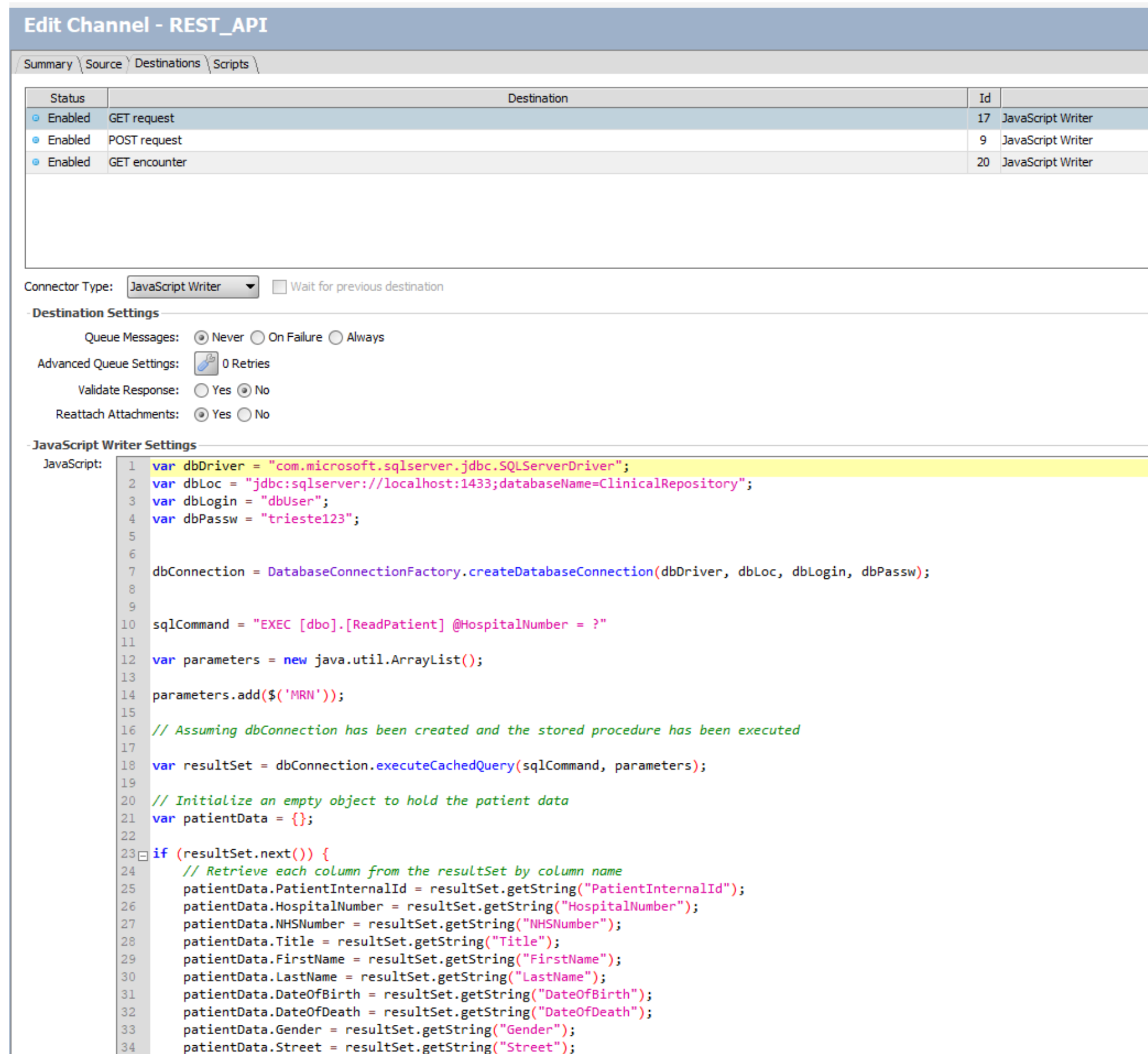
Values:	Value
<input style="width: 95%;" type="text" value="/Patient"/>	

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)



Status	Destination	Id
Enabled	GET request	17 JavaScript Writer
Enabled	POST request	9 JavaScript Writer
Enabled	GET encounter	20 JavaScript Writer

Connector Type: JavaScript Writer Wait for previous destination

Destination Settings

Queue Messages: Never On Failure Always

Advanced Queue Settings: 0 Retries

Validate Response: Yes No

Reattach Attachments: Yes No

JavaScript Writer Settings

```
JavaScript:
1 var dbDriver = "com.microsoft.sqlserver.jdbc.SQLServerDriver";
2 var dbLoc = "jdbc:sqlserver://localhost:1433;databaseName=ClinicalRepository";
3 var dbLogin = "dbUser";
4 var dbPassw = "triestel23";
5
6
7 dbConnection = DatabaseConnectionFactory.createDatabaseConnection(dbDriver, dbLoc, dbLogin, dbPassw);
8
9
10 sqlCommand = "EXEC [dbo].[ReadPatient] @HospitalNumber = ?"
11
12 var parameters = new java.util.ArrayList();
13 parameters.add($('MRN'));
14
15
16 // Assuming dbConnection has been created and the stored procedure has been executed
17
18 var resultSet = dbConnection.executeCachedQuery(sqlCommand, parameters);
19
20 // Initialize an empty object to hold the patient data
21 var patientData = {};
22
23 if (resultSet.next()) {
24 // Retrieve each column from the resultSet by column name
25 patientData.PatientInternalId = resultSet.getString("PatientInternalId");
26 patientData.HospitalNumber = resultSet.getString("HospitalNumber");
27 patientData.NHSNumber = resultSet.getString("NHSNumber");
28 patientData.Title = resultSet.getString("Title");
29 patientData.FirstName = resultSet.getString("FirstName");
30 patientData.LastName = resultSet.getString("LastName");
31 patientData.DateOfBirth = resultSet.getString("DateOfBirth");
32 patientData.DateOfDeath = resultSet.getString("DateOfDeath");
33 patientData.Gender = resultSet.getString("Gender");
34 patientData.Street = resultSet.getString("Street");
```

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");

// 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 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**.

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

Edit Channel - REST_API

Summary | Source | Destinations | Scripts

Status	Destination	Id	Java
Enabled	GET request	17	Java
Enabled	POST request	9	Java
Enabled	GET encounter	20	Java

Connector Type: JavaScript Writer Wait for previous destination

Destination Settings

Queue Messages: Never On Failure Always

Advanced Queue Settings: Retries

Validate Response: Yes No

Reattach Attachments: Yes No

JavaScript Writer Settings

JavaScript:

```
1 var dbDriver = "com.microsoft.sqlserver.jdbc.SQLServerDriver";
2 var dbLoc = "jdbc:sqlserver://localhost:1433;databaseName=ClinicalRepository";
3 var dbLogin = "dbUser";
4 var dbPassw = "trieste123";
5
6
7 dbConnection = DatabaseConnectionFactory.createDatabaseConnection(dbDriver, dbLoc, dbLogin, dbPassw);
8
9
10 sqlCommand = "EXEC [dbo].[InsertUpdatePatient] @HospitalNumber = ?, @NHSNumber = ?, @Title = ?, @FirstName = ?,
11
12 sqlCommand = sqlCommand + ", @DateOfDeath = ?, @Gender = ?, @Street = ?, @City = ?, @Province = ?, @Country = ?,
13
14 sqlCommand = sqlCommand + ", @MobileNumber = ?, @SecondaryNumber = ?, @Email = ?" // Add contact details
15
16 var parameters = new java.util.ArrayList();
17
18 // Building the paramters array
19 parameters.add('${hospitalNumber}');
20 parameters.add((${'NHSNumber'} == '') ? null : ${'NHSNumber'});
21 parameters.add((${'title'} == '') ? null : ${'title'});
22 parameters.add((${'givenName'} == '') ? null : ${'givenName'});
23 parameters.add((${'familyName'} == '') ? null : ${'familyName'});
24
25 var DateOfBirth = (${'dob'} == '') ? (null) : (${'dob'});
26 parameters.add(DateOfBirth);
27
28 var DateOfDeath = (${'dod'} == '') ? (null) : (${'dod'});
29 parameters.add(DateOfDeath);
30
31 parameters.add('${gender}');
32 parameters.add((${'street'} == '') ? null : ${'street'});
33 parameters.add((${'city'} == '') ? null : ${'city'});
34 parameters.add((${'province'} == '') ? null : ${'province'});
```

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

```
rh Connect Administrator - (4.4.1)
Edit Channel - REST_API - Source Transformer

Enabled # Name
[checked] 0 GET Read MRN
[checked] 1 POST request transformers

Step \ Generated Script
1 if (sourceMap.get('method') == 'POST') {
2
3 // Converting the payload to JSON
4 var msgJSON = JSON.parse(msg);
5
6 // Patient identifiers
7 var hospitalNumber = msgJSON['HospitalNumber'];
8 channelMap.put('hospitalNumber', hospitalNumber);
9
10
11 var NHSNumber = msgJSON['NHSNumber'];
12 channelMap.put('NHSNumber', NHSNumber);
13
14 // Patient Name
15 var title = msgJSON['Title'];
16 var firstname = msgJSON['FirstName'];
17 var lastname = msgJSON['LastName'];
18
19 channelMap.put('title', title);
20 channelMap.put('givenName', firstname);
21 channelMap.put('familyName', lastname);
22
23 // Gender
24 var genderCode = msgJSON['Gender'];
25 var gender;
26
27 if (String(genderCode) === "M") {
28     gender = 1;
29 } else if (String(genderCode) === "F") {
30     gender = 2;
31 } else if (String(genderCode) === "NS") {
32     gender = 9;
33 } else if (String(genderCode) === "U") {
34     gender = 0;
35 } else {
36     gender = -1;
37 }
38
39 channelMap.put('gender', gender);
40
41 // DOB/DOD
42 var dob = msgJSON['DOB'];
43 var dod = msgJSON['DOD'];
44
45 channelMap.put('dob', dob);
46 channelMap.put('dod', dod);
47
48 // Address
49 var street = msgJSON['Street'];
50 var city = msgJSON['City'];
51 var province = msgJSON['Province'];
52 var country = msgJSON['Country'];
```

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
```

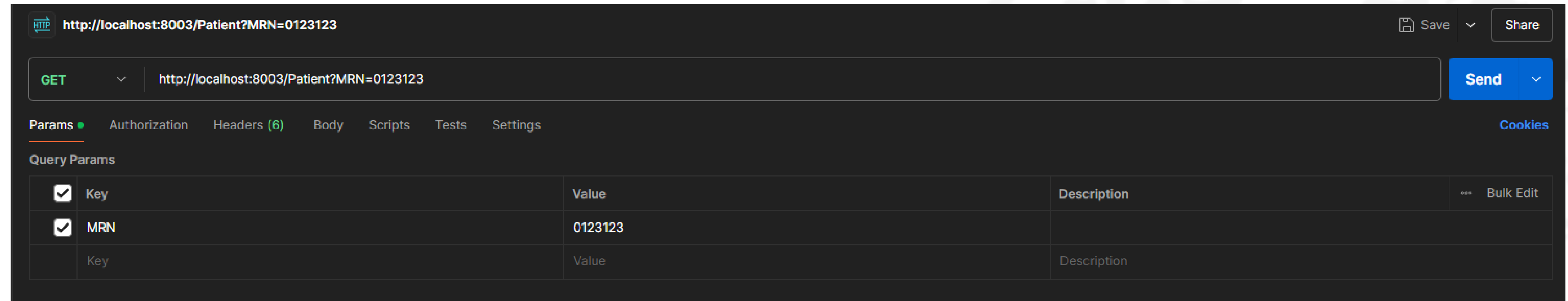
```
ELSE
BEGIN
    -- Generate a new GUID for PatientInternalId
    SET @NewPatientInternalId = NEWID();

    -- 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

Channel Messages - REST_API

Start Time: 05:35 pm All Day RECEIVED
End Time: 05:35 pm TRANSFORMED
Text Search: FILTERED
Page Size: 20 Regex QUEUED
 SENT
 ERROR
 PENDING

Current Search
Max Message Id: 39
Date Range: (any) to (any)
Statuses: (any)
Connectors: (any)

Id	Connector	Status	Received Date
39	Source	TRANSFORMED	2024-11-12 20:28:52.437
	GET request	SENT	2024-11-12 20:28:52.453
	POST request	FILTERED	2024-11-12 20:28:52.513
38	Source	TRANSFORMED	2024-11-12 20:24:07.327

Messages | **Mappings**

Scope	Variable	Value
Channel	MRN	0123124
Response	Response	{"PatientInternalId": "B2B463C4-C9D7-449D-BCD5-B3C069E31451", "HospitalNumber": "0123124", "NHSN": "0000255355"}
Response	responseStatusCode	200
Response	d9	FILTERED: Message has been filtered
Response	d17	SENT: JavaScript evaluation successful.
Source	headers	{accept-encoding=[gzip, deflate, br], referer=[http://localhost:8003/Patient?MRN=0123124], connecti
Source	localPort	8003
Source	method	GET
Source	query	MRN=0123124
Source	remotePort	59218
Source	contextPath	/Patient/

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

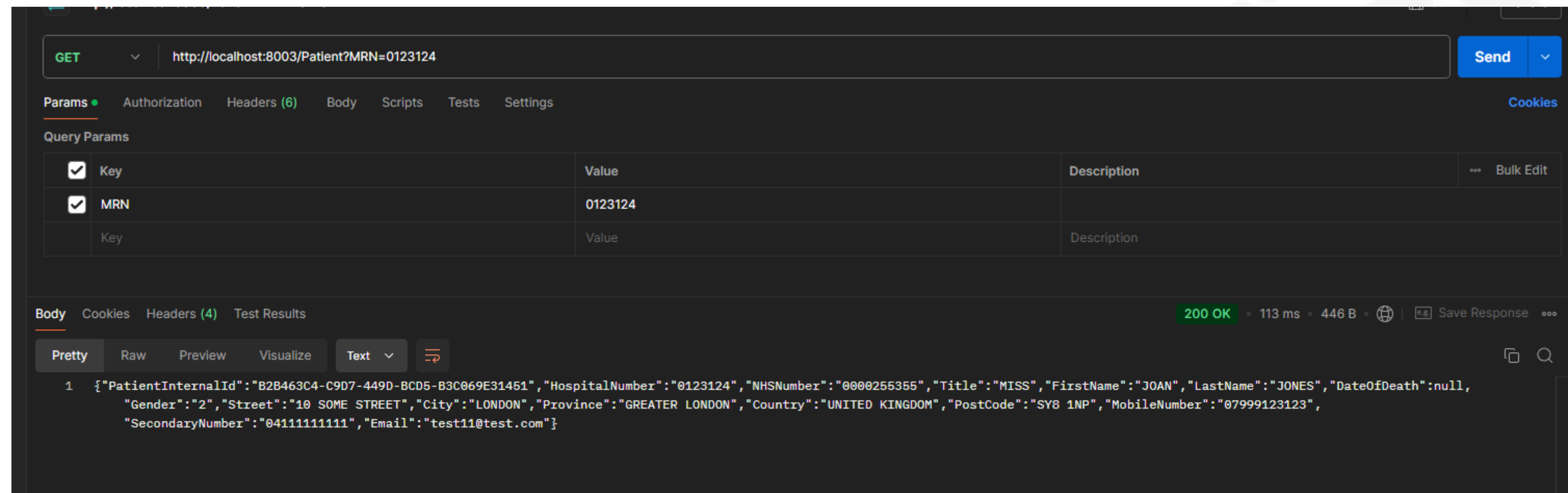
Il record è individuabile nella tabella **dbo.Patient** da una semplice query SQL:

```
SELECT *  
FROM [ClinicalRepository].[dbo].[Patient]  
where HospitalNumber = '0123124'
```

PatientInternalId	HospitalNumber	NHSNumber	Title	FirstName	LastName	DateOfBirth	DateOfDeath	Gender	Street
B2B463C4-C9D7-449D-BCD5-B3C069E31451	0123124	0000255355	MISS	JOAN	JONES	1999-05-21 00:00:00	NULL	2	10 SOME :

IMPLEMENTAZIONE DI UNA REST API

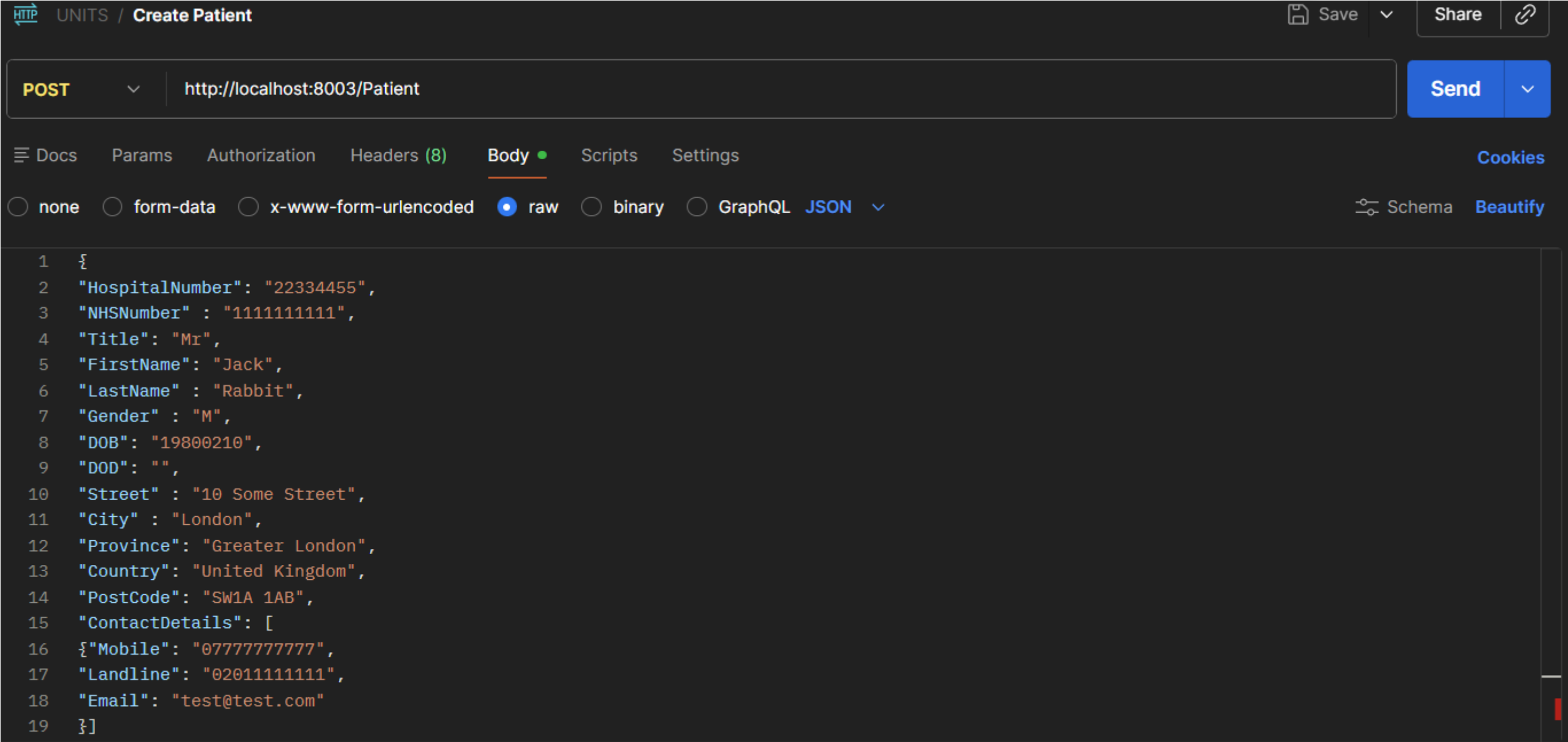
Esempio pratico end-to-end



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

IMPLEMENTAZIONE DI UNA REST API

Esempio pratico end-to-end



The screenshot shows a REST client interface for a POST request to `http://localhost:8003/Patient`. The request body is a JSON object representing a patient record. The interface includes tabs for Docs, Params, Authorization, Headers (8), Body (selected), Scripts, and Settings. The Body tab is set to 'raw' and 'JSON'. The JSON body is as follows:

```
1 {
2   "HospitalNumber": "22334455",
3   "NHSNumber": "1111111111",
4   "Title": "Mr",
5   "FirstName": "Jack",
6   "LastName": "Rabbit",
7   "Gender": "M",
8   "DOB": "19800210",
9   "DOD": "",
10  "Street": "10 Some Street",
11  "City": "London",
12  "Province": "Greater London",
13  "Country": "United Kingdom",
14  "PostCode": "SW1A 1AB",
15  "ContactDetails": [
16    { "Mobile": "0777777777",
17      "Landline": "0201111111",
18      "Email": "test@test.com"
19    }
20  ]
21 }
```

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

Channel Messages - REST_API

Start Time: 05:35 pm All Day RECEIVED
End Time: 05:35 pm TRANSFORMED
Text Search: Regex FILTERED
Page Size: 20 Advanced... Reset Search QUEUED
SENT
ERROR
PENDING

Id	Connector	Status
42	Source	TRANSFORME
	GET request	FILTERED
	POST request	SENT
41	Source	TRANSFORME

Messages | Mappings

Raw Encoded Sent Response

```
{
  "HospitalNumber": "22334455",
  "MHSNumber": "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 1AA",
  "ContactDetails": [
    {
      "Mobile": "07777777777",
      "Landline": "02011111111",
      "Email": "test@test.com"
    }
  ]
}
```

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.

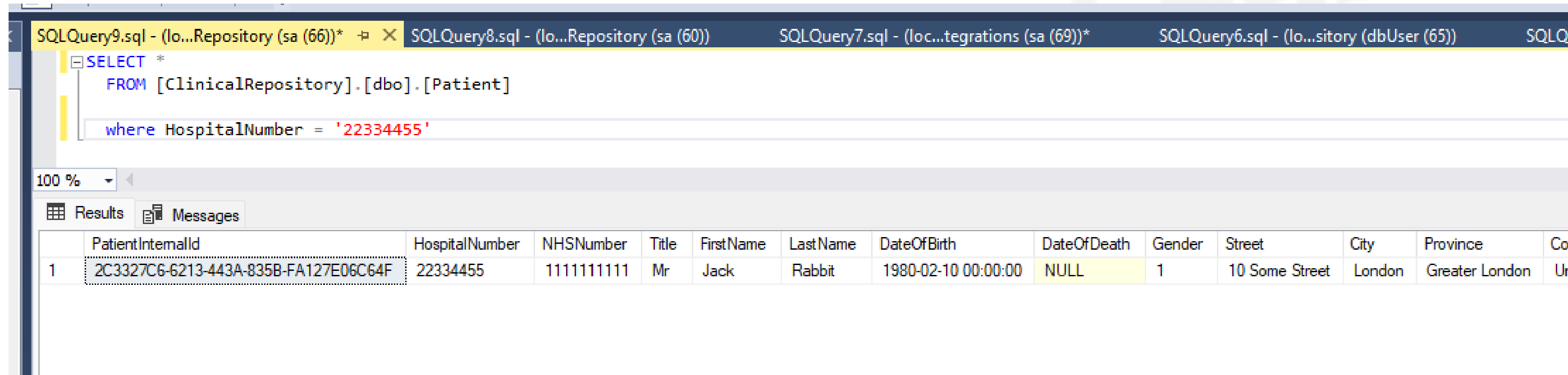
SQL Server Profiler

File Edit View Replay Tools Window Help

EventClass	TextData	ApplicationName
RPC:Completed	exec sp_execute 2,9,43,N'1262485e-ed10-4c5a-81c3-390bab707089','2024-11-12 20:36:44.7260000',N'R',N'POST request',0,NULL,NULL,0	Microsoft JD...
RPC:Completed	exec sp_execute 4,9,43,10,N'<map> <entry> <string>MRN</string> <string>undefined</string> </entry> </map>',NULL,0	Microsoft JD...
RPC:Completed	exec sp_execute 4,9,43,11,N'<map> <entry> <string>d17</string> <response> <status>FILTERED</status> <mess	Microsoft JD...
SQL:BatchStarting	IF @@TRANCOUNT > 0 COMMIT TRAN	Microsoft JD...
SQL:BatchCompleted	IF @@TRANCOUNT > 0 COMMIT TRAN	Microsoft JD...
Audit Login	-- network protocol: TCP/IP set quoted_identifier on set arithabort off set numeric_roundabort off set ansi_warnings on se	Microsoft JD...
RPC:Completed	exec sp_executesql N'EXEC [dbo].[InsertUpdatePatient] @HospitalNumber = @P0 , @NHSNumber = @P1, @Title = @P2, @FirstName = @P3,	Microsoft JD...
Audit Logout		Microsoft JD...
SQL:BatchStarting	SET DEADLOCK_PRIORITY 8	Microsoft JD...
SQL:BatchCompleted	SET DEADLOCK_PRIORITY 8	Microsoft JD...
RPC:Completed	exec sp_execute 4,9,43,4,N'{"HospitalNumber":"22334455","NHSNumber":"1111111111","Title":"Mr","FirstName":"Jack","LastName":"Ra	Microsoft JD...
RPC:Completed	exec sp_execute 3,N'<map> <entry> <string>country</string> <string>United Kingdom</string> </entry> <entry>	Microsoft JD...

IMPLEMENTAZIONE DI UNA REST API

Esempio pratico end-to-end



The screenshot shows a SQL Server Enterprise Manager window with several query tabs. The active tab is 'SQLQuery9.sql - (lo...Repository (sa (66)))'. The query text is:

```
SELECT *  
FROM [ClinicalRepository].[dbo].[Patient]  
where HospitalNumber = '22334455'
```

The results pane shows a single record with the following data:

	PatientInternalId	HospitalNumber	NHSNumber	Title	FirstName	LastName	DateOfBirth	DateOfDeath	Gender	Street	City	Province	Country
1	2C3327C6-6213-443A-835B-FA127E06C64F	22334455	1111111111	Mr	Jack	Rabbit	1980-02-10 00:00:00	NULL	1	10 Some Street	London	Greater London	United Kingdom

Il record è scritto correttamente nel database.

IMPLEMENTAZIONE DI UNA REST API

Esempio pratico end-to-end

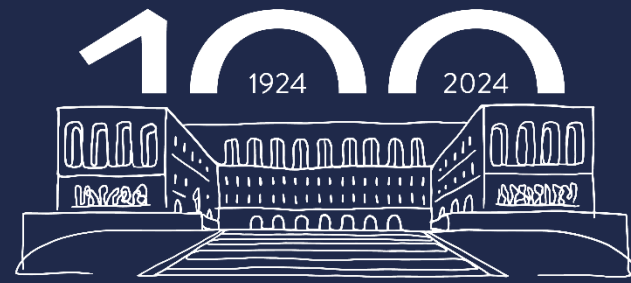
```
POST http://localhost:8003/Patient

{
  "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": "0777777777",
      "Landline": "0201111111",
      "Email": "test@test.com"
    }
  ]
}

200 OK • 57 ms • 227 B

{"status":200,"message":"Patient record added successfully"}
```

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

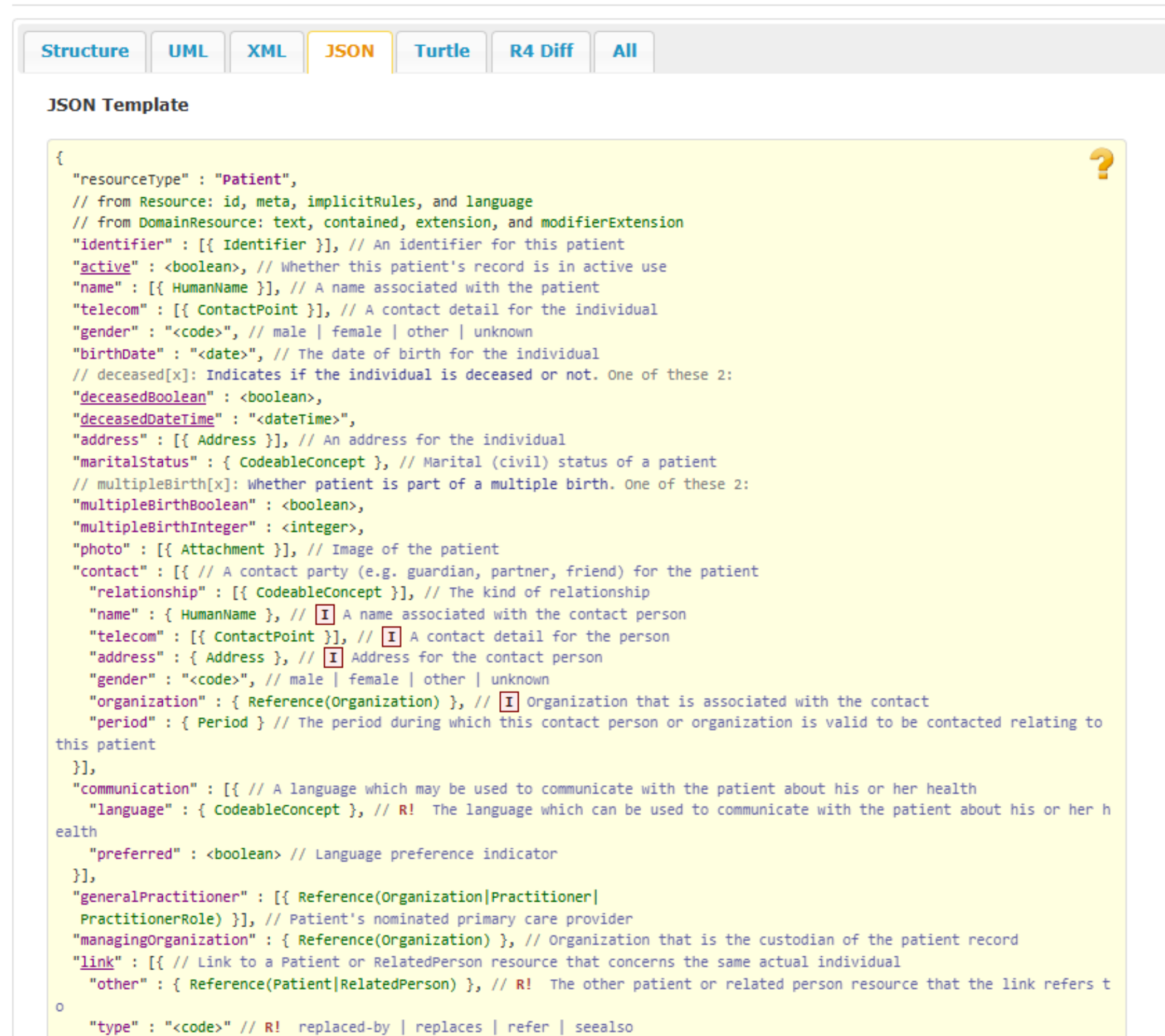
Name	Flags	Card.	Type	Description & Constraints
Patient	N		DomainResource	Information about an individual or animal receiving health care services
identifier	Σ	0..*	Identifier	Elements defined in Ancestors: id, meta, implicitRules, language, text, contained, extension, modifierExtension An identifier 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



The screenshot shows the JSON Template for the Patient resource in FHIR. The interface includes tabs for Structure, UML, XML, JSON (selected), Turtle, R4 Diff, and All. The JSON template is displayed in a yellow-highlighted code editor with a question mark icon in the top right corner. The template defines the structure of a Patient resource, including fields like identifier, active, name, telecom, gender, birthDate, deceasedBoolean, address, maritalStatus, multipleBirthBoolean, photo, contact, communication, and link. Comments provide detailed information about each field, such as data types, constraints, and relationships.

```
{
  "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 health
    "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 to
  }],
  "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

```
POST http://hapi.fhir.org/baseR4/Patient

{
  "resourceType": "Patient",
  "name": [
    {
      "use": "official",
      "family": "Doe",
      "given": "John",
      "text": "John Doe"
    }
  ],
  "gender": "male",
  "birthDate": "1960-01-01",
  "text": {
    "status": "generated",
    "div": "<div xmlns='https://www.w3.org/1999/xhtml'>John Doe</div>"
  },
  "id": "123123123",
  "meta": {
    "lastUpdated": "2024-11-15T01:48:09Z",
    "versionId": "1"
  }
}

201 Created - 485 ms - 842 B
{
  "resourceType": "Patient",
  "id": "45180097",
  "meta": {
    "versionId": "1",
    "lastUpdated": "2024-11-18T09:22:39.708+00:00",
    "source": "#tM0PMqXMNVutowC2"
  },
  "text": {
    "status": "generated",
    "div": "<div xmlns='https://www.w3.org/1999/xhtml'>John Doe</div>"
  },
  "name": [
    {
      "use": "official",
```

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

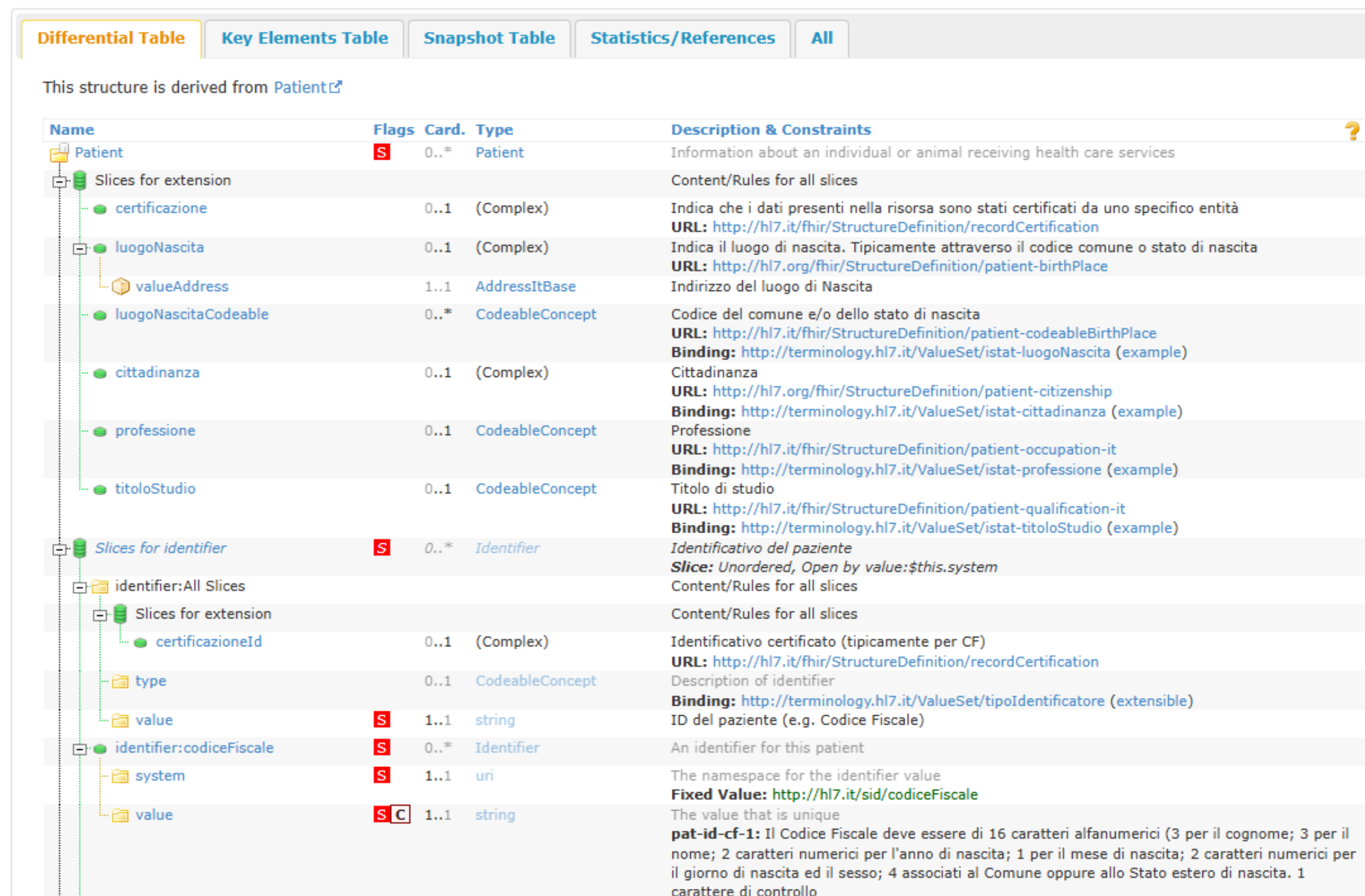
Nell'esempio a fianco, utilizziamo **Postman** per mandare una richiesta POST verso un endpoint FHIR (si noti il *context path /Patient*). Il payload rappresenta un nuovo paziente inserito seguendo lo schema della risorsa **Patient** e utilizzando il formato JSON.

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.



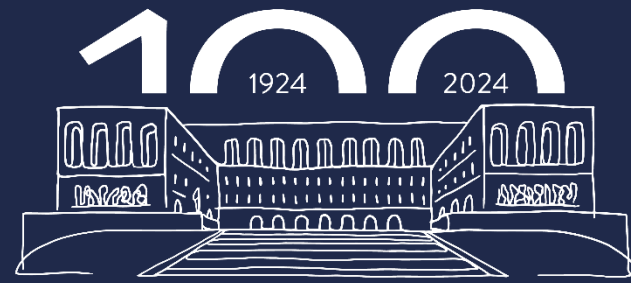
This structure is derived from Patient

Name	Flags	Card.	Type	Description & Constraints
Patient	S	0..*	Patient	Information about an individual or animal receiving health care services
Slices for extension				
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				
identifier:All Slices	S	0..*	Identifier	Identificativo del paziente Slice: Unordered, Open by value:\$this.system
Slices for extension				
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>



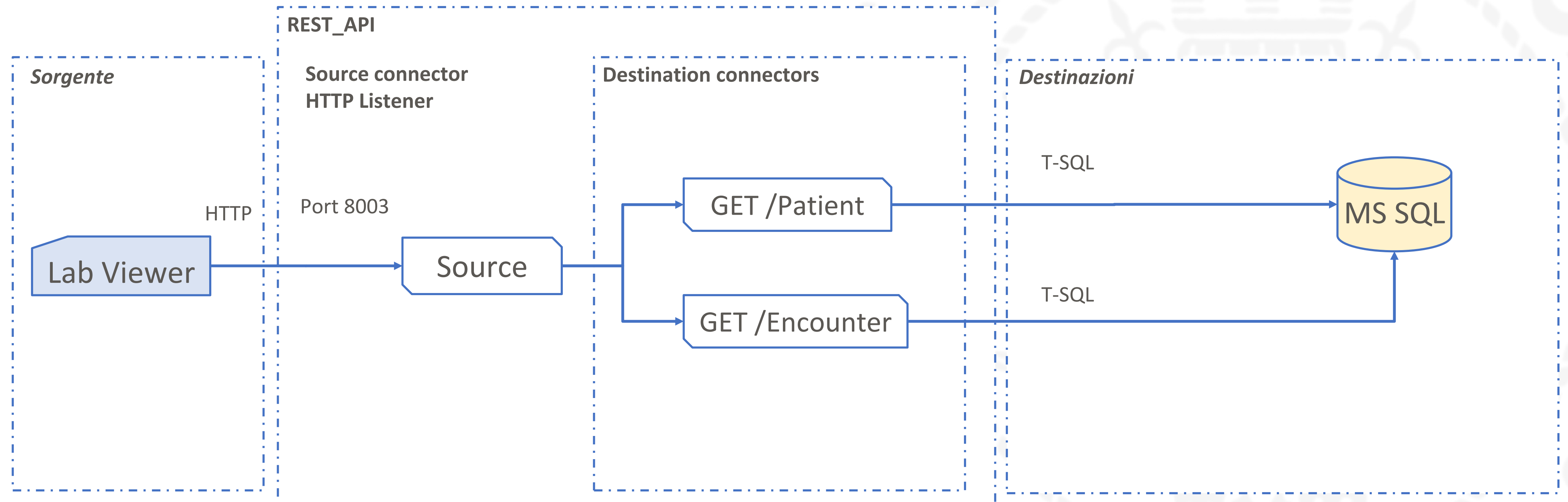
UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

UN ESEMPIO PRATICO DI INTEGRAZIONE FHIR

IMPLEMENTAZIONE FHIR

Schema REST API

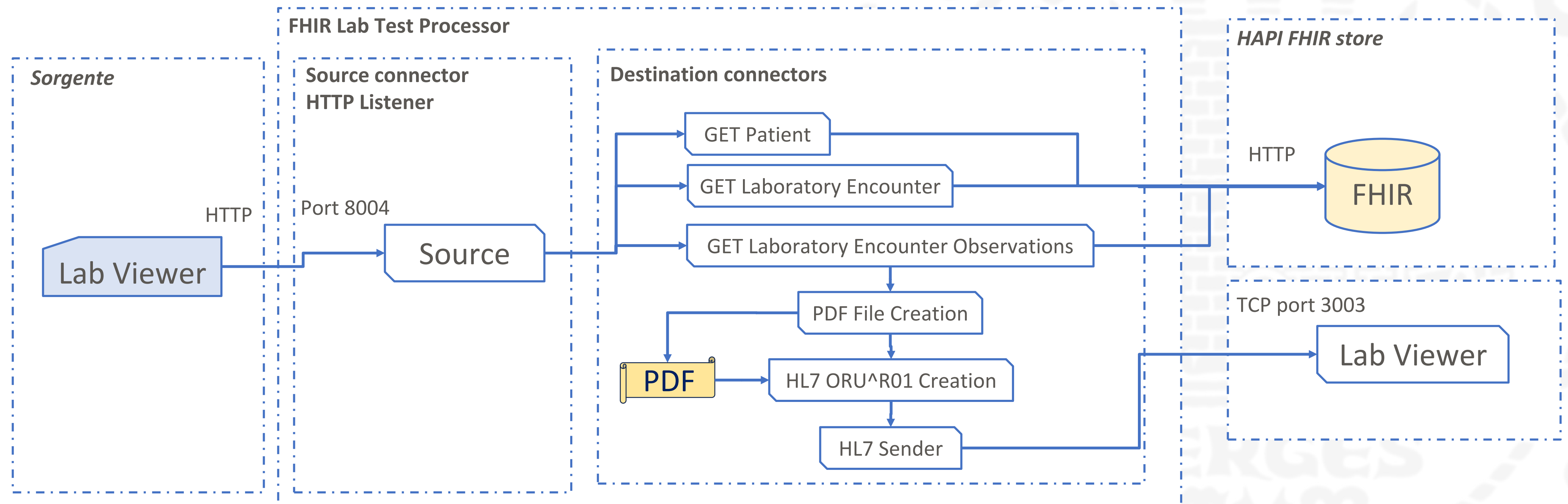
Un applicativo web chiamato Lab Viewer (implementato in *node.js*) permette la possibilità di ricerca paziente e di ricerca di encounters di laboratorio tramite REST API implementata in Mirth e interfacciata con un database MS SQL che si può assumere essere alimentato da un feed HL7 di tipo ADT^A28/A31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

The screenshot shows the HAPI FHIR Test Server interface. The top navigation bar includes 'Home', 'Server: HAPI Test Server (R4 FHIR)', 'Source Code', and 'About This Server'. The left sidebar contains 'Options' (Encoding: default, XML, JSON; Pretty: default, On, Off; Summary: none, true, text, data, count) and 'Server' (Server Home/Actions, HFQL/SQL, Resources: Observation 4210440, Specimen 1875902, Composition 938649, Patient 839641, Bundle 402816, Encounter 186662, Claim 134763, Condition 108640, AuditEvent 95336, QuestionnaireResponse 90182, ExplanationOfBenefit 89371, MedicationStatement 88034).

The main content area features the HAPI FHIR logo and a warning: "You are accessing the public FHIR server HAPI Test Server (R4 FHIR). This server is hosted elsewhere on the internet but is being accessed using the HAPI client implementation. **This is not a production server!** Do not store any information here that contains personal health information or any other confidential information. This server will be regularly purged and reloaded with fixed test data."

Server	HAPI FHIR Test/Demo Server R4 Endpoint
Software	HAPI FHIR Server - 7.7.0-SNAPSHOT/6fca981c51/2024-10-31
FHIR Base	http://hapi.fhir.org/baseR4

Server Actions:

- Retrieve the server's **conformance** statement.
- Retrieve the update **history** across all resource types on the server. Since Limit # (opt)
- Post a bundle containing multiple resources to the server and store all resources within a single atomic transaction. Bundle* (place transaction bundle body here)

IMPLEMENTAZIONE FHIR

Source connector FHIR processor

irth Connect Administrator - (4.4.1)

Edit Channel - FHIR Lab Test Processor

Summary | Source | Destinations | Scripts

Connector Type: HTTP Listener

Listener Settings

Local Address: All interfaces Specific interface: 0.0.0.0

Local Port:

Source Settings

Source Queue: OFF (Respond after processing)

Queue Buffer Size:

Response:

Process Batch: Yes No

Batch Response: First Last

Max Processing Threads:

HTTP Authentication

Authentication Type:

HTTP Listener Settings

Base Context Path:

Receive Timeout (ms):

Message Content: Plain Body XML Body

Parse Multipart: Yes No

Include Metadata: Yes No

Binary MIME Types: Regular Expression

HTTP URL:

Response Content Type:

Response Data Type: Binary Text

Charset Encoding:

Response Status Code:

Response Headers: Use Table Use Map:

Name

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.

Edit Channel - FHIR Lab Test Processor - Source Transformer

Enabled	#	Name
<input checked="" type="checkbox"/>	0	Read Patient Id and Encounter Id

Step | Generated Script

```
1 if (sourceMap.get('method') == 'GET') {
2
3   var query = sourceMap.get('query');
4   var querySplit = query.split("&");
5
6
7
8   var patientId = querySplit[0].replace('PatientId=', '');
9   var encounterId = querySplit[1].replace('EncounterId=', '');
10
11 channelMap.put('patientId', patientId);
12 channelMap.put('encounterId', encounterId);
```

IMPLEMENTAZIONE FHIR

Destination connector FHIR processor

1 Connect Administrator - (4.4.1)

Edit Channel - FHIR Lab Test Processor

Summary \ Source \ Destinations \ Scripts \

Status	Destination
<input checked="" type="checkbox"/> Enabled	Get Patient
<input checked="" type="checkbox"/> Enabled	Get Laboratory Encounter
<input checked="" type="checkbox"/> Enabled	Get Laboratory Encounter Observations
<input checked="" type="checkbox"/> Enabled	PDF File Creation
<input checked="" type="checkbox"/> Enabled	PDF encoding
<input checked="" type="checkbox"/> Enabled	HL7 Sender

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

IMPLEMENTAZIONE FHIR

Get Patient (1)

Mirth Connect Administrator - (4.4.1)

Edit Channel - FHIR Lab Test Processor

Summary | Source | Destinations | Scripts

Status	Destination
<input checked="" type="radio"/> Enabled	Get Patient
<input checked="" type="radio"/> Enabled	Get Laboratory Encounter
<input checked="" type="radio"/> Enabled	Get Laboratory Encounter Observations
<input checked="" type="radio"/> Enabled	PDF File Creation
<input checked="" type="radio"/> Enabled	PDF encoding
<input checked="" type="radio"/> Enabled	HL7 Sender Lab Viewer

Connector Type: Wait for previous destination

Destination Settings

Queue Messages: Never On Failure Always

Advanced Queue Settings: 0 Retries

Validate Response: Yes No

Reattach Attachments: Yes No

HTTP Sender Settings

URL:

Use Proxy Server: Yes No

Proxy Address:

Proxy Port:

Method: POST GET PUT DELETE PATCH

Multipart: Yes No

Send Timeout (ms):

Response Content: Plain Body XML Body

Parse Multipart: Yes No

Include Metadata: Yes No

Binary MIME Types: Regular Expression

Authentication: Yes No

Authentication Type: Basic Digest Preemptive

Username:

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)

rtm Connect Administrator - (4.4.1)

Edit Channel - FHIR Lab Test Processor - Get Patient Response Transformer

Enabled	#	Name	Type
<input checked="" type="checkbox"/>	0	FirstName	Mapper
<input checked="" type="checkbox"/>	1	LastName	Mapper
<input checked="" type="checkbox"/>	2	DateOfBirth	Mapper
<input checked="" type="checkbox"/>	3	Gender	JavaScript

Step | Generated Script

Variable: Add to:

Mapping:

Default Value:

String Replacement	Regular Expression	Replace With

Inbound Message Template

Data Type: Properties

```
{
  "use" : "official",
  "family" : "Villegas15",
  "given" : [
    "Ernesto186"
  ],
  "prefix" : [
    "Mr."
  ]
},
"telecom" : [
  {
    "system" : "phone",
    "value" : "555-166-3296",
    "use" : "home"
  }
],
"gender" : "male",
"birthDate" : "1963-07-01",
"address" : [
  {
```

Outbound Message Template

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/sex).

IMPLEMENTAZIONE FHIR

Get Laboratory Encounter

Mirth Connect Administrator - (4.4.1)

Edit Channel - FHIR Lab Test Processor

Summary | Source | Destinations | Scripts

Status	Destination
Enabled	Get Patient
Enabled	Get Laboratory Encounter
Enabled	Get Laboratory Encounter Observations
Enabled	PDF File Creation
Enabled	PDF encoding
Enabled	HL7 Sender Lab Viewer

Connector Type: HTTP Sender Wait for previous destination

Destination Settings

Queue Messages: Never On Failure Always

Advanced Queue Settings: 0 Retries

Validate Response: Yes No

Reattach Attachments: Yes No

HTTP Sender Settings

URL:

Use Proxy Server: Yes No

Proxy Address:

Mirth Connect Administrator - (4.4.1)

Edit Channel - FHIR Lab Test Processor - Get Laboratory Encounter Response Transformer

Reference | Message Trees | Message Templates

Inbound Message Template

Data Type: JSON

```
{
  "individual" : {
    "reference" : "Practitioner/624185",
    "display" : "Dr. Necole468 Bashirian201"
  },
  "period" : {
    "start" : "2013-07-01T18:22:17+02:00",
    "end" : "2013-07-01T18:37:17+02:00"
  },
  "serviceProvider" : {
    "reference" : "Organization/624184",
    "display" : "PCP32115"
  }
}
```

Step | Generated Script

Variable: Add to:

Mapping:

Default Value:

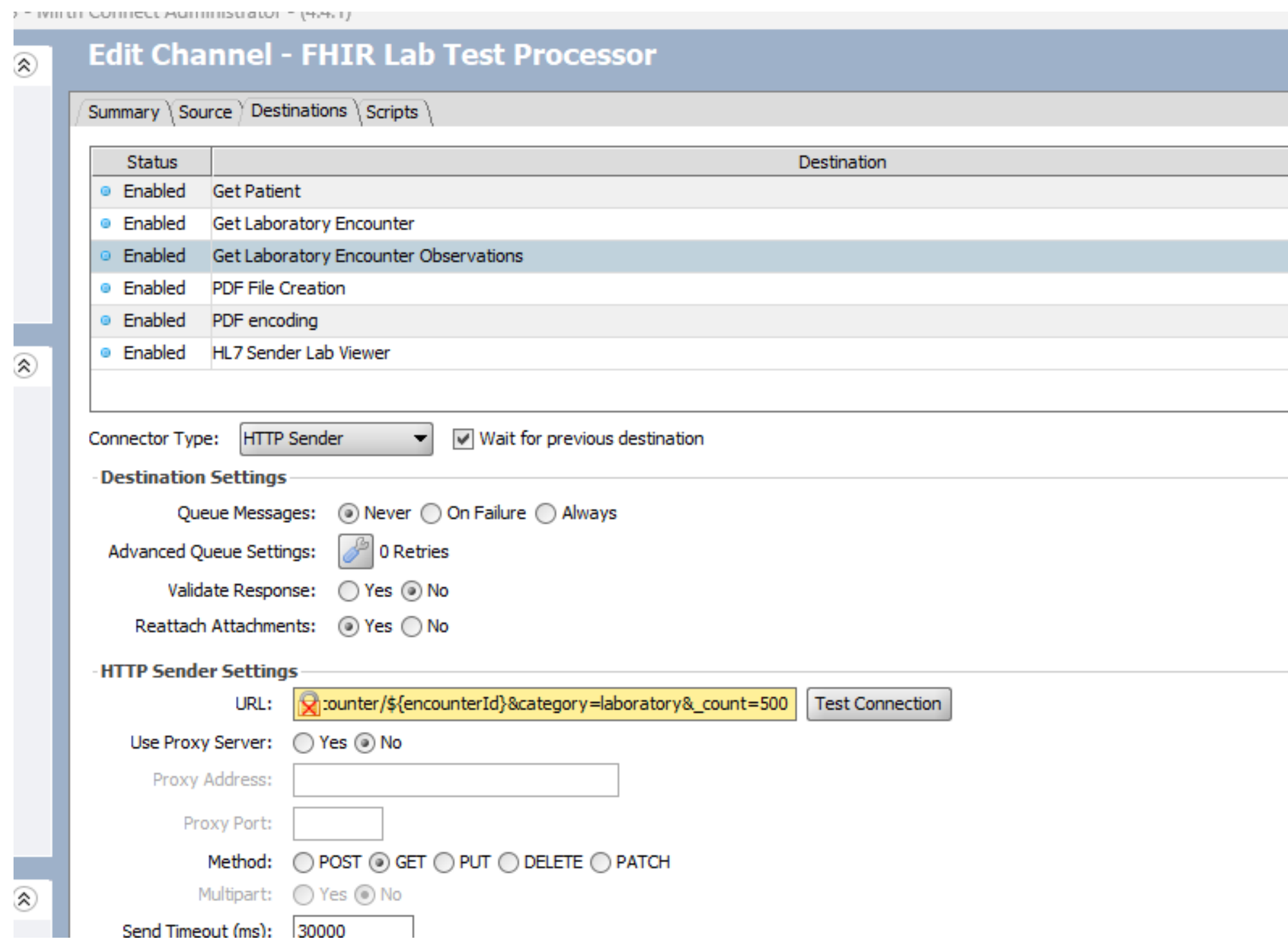
String Replacement:

Regular Expression	Replace With
<input type="text"/>	<input type="text"/>

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)



Status	Destination
<input checked="" type="radio"/> Enabled	Get Patient
<input checked="" type="radio"/> Enabled	Get Laboratory Encounter
<input checked="" type="radio"/> Enabled	Get Laboratory Encounter Observations
<input checked="" type="radio"/> Enabled	PDF File Creation
<input checked="" type="radio"/> Enabled	PDF encoding
<input checked="" type="radio"/> Enabled	HL7 Sender Lab Viewer

Connector Type: Wait for previous destination

Destination Settings

Queue Messages: Never On Failure Always

Advanced Queue Settings: Retries

Validate Response: Yes No

Reattach Attachments: Yes No

HTTP Sender Settings

URL:

Use Proxy Server: Yes No

Proxy Address:

Proxy Port:

Method: POST GET PUT DELETE PATCH

Multipart: Yes No

Send Timeout (ms):

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

IMPLEMENTAZIONE FHIR

Get Laboratory Encounter Observations (2)

The screenshot displays the 'Edit Channel' interface for a FHIR Lab Test Processor. The main window shows a JavaScript script for processing laboratory encounter observations. The script includes steps for collecting patient-level data, converting test dates, and initializing an array of entries. It then iterates through the 'entry' array, safely accessing description, value, and unit information from each resource.

```
1 // Collect patient-level data
2 tmp['PatientID'] = $('patientId');
3 tmp['LastName'] = $('LastName');
4 tmp['FirstName'] = $('FirstName');
5 tmp['DateOfBirth'] = $('DateOfBirth');
6 tmp['EncounterID'] = $('encounterId');
7
8 // Convert TestDate
9 var TestDateConverted = DateUtil.convertDate("yyyy-MM-dd'T'HH:mm:ssXXX", "MM/dd/yyyy", $('TestDate'));
10 tmp['TestDate'] = TestDateConverted;
11
12 // Initialize Entries as an empty array
13 tmp.Entries = [];
14
15 // Check if msg['entry'] exists and is an array
16 if (msg['entry'] && Array.isArray(msg['entry'])) {
17   for (var i = 0; i < msg['entry'].length; i++) {
18     var resource = msg['entry'][i]['resource'];
19
20     // Safe access for Description
21     var description = "";
22     if (resource.code && resource.code.coding && resource.code.coding.length > 0) {
23       description = resource.code.coding[0].display || "";
24     }
25
26     // Safe access for Value and Unit
27     var value = "";
28     var unit = "";
29     if (resource.valueQuantity) {
30       value = resource.valueQuantity.value != null ? resource.valueQuantity.value : "";
31       unit = resource.valueQuantity.unit || "";
32     } else if (resource.valueString) {
33       value = resource.valueString;
34     } else if (resource.valueCodeableConcept) {
35       value = resource.valueCodeableConcept.text || "";
36     } else if (resource.valueBoolean != null) {
37       value = resource.valueBoolean;
38     } else if (resource.valueInteger != null) {
39       value = resource.valueInteger;
40     }
41   }
42 }
```

The right-hand pane shows the 'Inbound Message Template' and 'Outbound Message Template' in JSON format. The inbound template is a searchset for observations, and the outbound template is a structured HTML string containing patient information and the processed entries.

```
{
  "resourceType": "Bundle",
  "id": "3050cb63-37a5-411b-8e63-76bc6670cbc9",
  "meta": {
    "lastUpdated": "2024-10-27T11:42:39.989+00:00"
  },
  "type": "searchset",
  "total": 14,
  "link": [
    {
      "relation": "self",
      "url": "https://hapi.fhir.org/baseR4/Observation"
    }
  ],
  "entry": [
    {
      "fullUrl": "https://hapi.fhir.org/baseR4/Observation/624251",
      "resource": {
        "resourceType": "Observation",
        "id": "624251",
        "meta": {
          "versionId": "1",
        }
      }
    }
  ]
}
```

```
{
  "PatientID": "",
  "LastName": "",
  "FirstName": "",
  "DateOfBirth": "",
  "EncounterID": "",
  "TestDate": "",
  "Entries": [
    {
      "Description": "",
      "Value": "",
      "Unit": ""
    }
  ]
}
```

La risposta è un array (FHIR *bundle*) contenente 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

irth Connect Administrator - (4.4.1)

Edit Channel - FHIR Lab Test Processor

Summary | Source | Destinations | Scripts

Status	Destination
<input checked="" type="radio"/> Enabled	Get Patient
<input checked="" type="radio"/> Enabled	Get Laboratory Encounter
<input checked="" type="radio"/> Enabled	Get Laboratory Encounter Observations
<input checked="" type="radio"/> Enabled	PDF File Creation
<input checked="" type="radio"/> Enabled	PDF encoding
<input checked="" type="radio"/> Enabled	HL7 Sender Lab Viewer

Connector Type: Wait for previous destination

Destination Settings

Queue Messages: Never On Failure Always

Advanced Queue Settings: Retries

Validate Response: Yes No

Reattach Attachments: Yes No

Document Writer Settings

Output: File Attachment Both

Directory:

File Name:

Document Type: PDF RTF

Encrypted: Yes No

Password:

Page Size: x

HTML Template:

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**).

irth Connect Administrator - (4.4.1)

Edit Channel - FHIR Lab Test Processor - PDF File Creation Transformer

Enabled	#	Name
<input checked="" type="checkbox"/>	0 <input type="text" value="filename"/>

Generated Script

```
1 var filename = $('encounterId') + "_" + $('patientId') + "_" + $('LastName');
2
3 channelMap.put("filename", filename);
```

IMPLEMENTAZIONE FHIR

PDF Encoding

Mirth Connect Administrator - (4.4.1)

Edit Channel - FHIR Lab Test Processor

Summary | Source | Destinations | Scripts

Status	Destination	Id	
Enabled	Get Patient	4	HTTP Sen
Enabled	Get Laboratory Encounter	6	HTTP Sen
Enabled	Get Laboratory Encounter Observations	2	HTTP Sen
Enabled	PDF File Creation	7	Document Wri
Enabled	PDF encoding	8	JavaScript Wr
Enabled	HL7 Sender Lab Viewer	10	TCP Sender

Connector Type: JavaScript Writer Wait for previous destination

Destination Settings

Queue Messages: Never On Failure Always

Advanced Queue Settings: Retries

Validate Response: Yes No

Reattach Attachments: Yes No

JavaScript Writer Settings

JavaScript:

```
1 var contents = FileUtil.readBytes('C:/lab-viewer/TestResult/' + $('filename') + '.pdf');
2
3 var encData = FileUtil.encode(contents);
4
5 channelMap.put("encodedPDF", encData);
```

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

Mirth Connect Administrator - (4.4.1)

Edit Channel - FHIR Lab Test Processor

Summary | Source | Destinations | Scripts

Status	Destination
<input checked="" type="radio"/> Enabled	Get Patient
<input checked="" type="radio"/> Enabled	Get Laboratory Encounter
<input checked="" type="radio"/> Enabled	Get Laboratory Encounter Observations
<input checked="" type="radio"/> Enabled	PDF File Creation
<input checked="" type="radio"/> Enabled	PDF encoding
<input checked="" type="radio"/> Enabled	HL7 Sender Lab Viewer

Connector Type: Wait for previous destination

Destination Settings

Queue Messages: Never On Failure Always

Advanced Queue Settings: Retries

Validate Response: Yes No

Reattach Attachments: Yes No

TCP Sender Settings

Transmission Mode:

MLLP Sample Frame:

Mode: Client Server

Remote Address:

Remote Port:

Override Local Binding: Yes No

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

The screenshot displays the 'Edit Channel - FHIR Lab Test Processor - HL7 Sender Transformer' interface. It is divided into two main sections: a code editor on the left and a message template editor on the right.

Code Editor (Generated Script):

```
1 var timestamp = DateUtil.getCurrentDate("yyyyMMddHHmmss");
2
3 // MSH segment transformations
4
5 tmp['MSH']['MSH.7']['MSH.7.1'] = timestamp;
6 tmp['MSH']['MSH.10']['MSH.10.1'] = UUIDGenerator.getUUID();
7
8 // PID segment transformations
9
10 tmp['PID']['PID.3']['PID.3.1'] = $('patientId');
11
12 tmp['PID']['PID.5']['PID.5.1'] = $('LastName');
13
14 tmp['PID']['PID.5']['PID.5.2'] = $('FirstName');
15
16 tmp['PID']['PID.7']['PID.7.1'] = DateUtil.convertDate("yyyy-dd-mm", "yyyymmdd", $('DateOfBirth'));
17
18 tmp['PID']['PID.8']['PID.8.1'] = $('gender');
19
20
21 // OBR segment transformations
22 tmp['OBR']['OBR.4']['OBR.4.5'] = $('filename') + ".pdf";
23 tmp['OBR']['OBR.7']['OBR.7.1'] = DateUtil.convertDate("yyyy-MM-dd'T'HH:mm:ssXXX", "yyyyMMddHHmmss", $('TestDate'));
24
25 // OBX segment transformations
26
27
28 tmp['OBX']['OBX.5']['OBX.5.5'] = $('encodedPDF'); ← Base64 PDF
```

Message Template Editor:

The 'Outbound Message Template' section shows the following HL7 message structure:

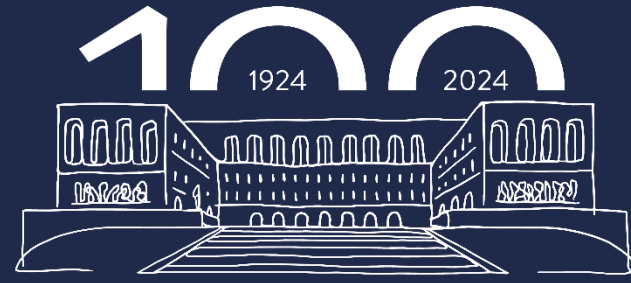
```
MSH|^~\&||SENDER|DEST_LOC|DEST_SYS|||ORU^R01||P|2.4|
PID||||^^^MRN||^
OBR||||^Laboratory Test Result^^^
OBX|1|ED|Blood Test^|^APPLICATION^PDF^BASE64^^|
```

A red arrow points from the text 'Scheletro di un ORU^R01' to the message template. Another red arrow points from the text 'Base64 PDF' to the line in the code editor: `tmp['OBX']['OBX.5']['OBX.5.5'] = $('encodedPDF');`

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

Patient Search

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

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

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

IMPLEMENTAZIONE FHIR

Dimostrazione end-to-end

Patient Search

Demographics

MRN: 624175 NHS: 111 111 1111
Name: Ernesto Villegas15 DOB: 1963-06-30
Gender: 1 Phone: 07777777778
Email: test1@test.com
Address: 35 Great Road, London, Greater London, SW1A 1AA

Encounters

EncounterId	EncounterType	EncounterDate	HospitalNumber	Action
624245	Lab Result	2013-07-01 00:00:00.0	624175	<input type="button" value="Search Encounter"/>
624314	Lab Result	2014-07-07 00:00:00.0	624175	<input type="button" value="Search Encounter"/>

Channel Messages - REST_API

Start Time: 07:30 pm All Day RECEIVED
End Time: 07:30 pm TRANSFORMED
Text Search: Regex FILTERED
Page Size: QUEUED
 SENT
 ERROR
 PENDING

Id	Connector	Status	Received
205	Source	TRANSFORMED	2025-12-15 11:5
	GET request	FILTERED	2025-12-15 11:5
	POST request	FILTERED	2025-12-15 11:5
	GET encounter	SENT	2025-12-15 11:5
204	Source	TRANSFORMED	2025-12-15 11:5

Messages | Mappings

Raw Encoded Response

Status:
SENT

Response:

```
[  
  {  
    "EncounterInternalId" : "961C527B-9A64-4AB6-99DF-0173758C8CE4",  
    "EncounterId" : "624245",  
    "EncounterType" : "Lab Result",  
    "EncounterDate" : "2013-07-01 00:00:00.0",  
    "PatientInternalId" : "F030C57A-6A28-4216-8100-6CB5E7289BB0",  
    "HospitalNumber" : "624175"  
  },  
  {  
    "EncounterInternalId" : "B3639356-2210-4264-BB07-7CF793356018",  
    "EncounterId" : "624314",  
    "EncounterType" : "Lab Result",  
    "EncounterDate" : "2014-07-07 00:00:00.0",  
    "PatientInternalId" : "F030C57A-6A28-4216-8100-6CB5E7289BB0",  
    "HospitalNumber" : "624175"  
  }  
]
```

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

Channel Messages - FHIR Lab Test Processor

Start Time: 07:30 pm All Day RECEIVED
End Time: 07:30 pm TRANSFORMED
Text Search: Regex FILTERED
Page Size: 20 QUEUED
 SENT
 ERROR
 PENDING

Current Search
Max Message Id: 136
Date Range: (any) to (any)
Statuses: (any)
Connectors: (any)

Id	Connector	Status	Received Date	Response Date
136	Source	TRANSFORMED	2025-12-15 12:01:48.083	2025-12-15 12:01:50.450
	Get Patient	SENT	2025-12-15 12:01:48.087	2025-12-15 12:01:48.213
	Get Laboratory Encounter	SENT	2025-12-15 12:01:48.247	2025-12-15 12:01:49.337
	Get Laboratory Encounter Observations	SENT	2025-12-15 12:01:49.387	2025-12-15 12:01:50.230
	PDF File Creation	SENT	2025-12-15 12:01:50.407	2025-12-15 12:01:50.420
	PDF encoding	SENT	2025-12-15 12:01:50.420	2025-12-15 12:01:50.430
	HL7 Sender Lab Viewer	SENT	2025-12-15 12:01:50.430	2025-12-15 12:01:50.443

Messages | Mappings

Raw Encoded Response

Select a message to view the raw message.

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

Channel Messages - FHIR Lab Test Processor

Start Time: 12:23 pm All Day
End Time: 12:23 pm
Text Search: Regex
Page Size: 20 Advanced... Reset Search

Current Search
Max Message Id: 80
Date Range: (any) to (any)
Statuses: (any)
Connectors: (any)

Id	Connector	Status	Received Date	Response Date
80	Source	TRANSFORMED	2024-11-16 16:55:49.277	--
	Get Patient	SENT	2024-11-16 16:55:49.280	2024-11-16 16:55:49.280
	Get Laboratory Encounter	SENT	2024-11-16 16:55:49.753	2024-11-16 16:55:50.123
	Get Laboratory Encounter Observations	SENT	2024-11-16 16:55:50.733	2024-11-16 16:55:51.123

Messages Mappings
 Raw Encoded Sent Response

Status: SENT

Response:

```
{
  "resourceType": "Bundle",
  "id": "2185b510-730b-4a02-88c0-f50d901826d3",
  "meta": {
    "lastUpdated": "2024-11-16T16:54:58.579+00:00"
  },
  "type": "searchset",
  "total": 1,
  "link": [
    {
      "relation": "self",
      "url": "https://hapi.fhir.org/baseR4/Patient?_id=Patient%2F624175"
    }
  ],
  "entry": [
    {
      "fullUrl": "https://hapi.fhir.org/baseR4/Patient/624175",
      "resource": {
        "resourceType": "Patient",
        "id": "624175",
        "meta": {
          "versionId": "1",
          "lastUpdated": "2020-02-18T10:20:57.687+00:00",
          "source": "H0evMc30fwqshBcpx",
          "profile": [
            "https://www.fhir.philips.com/4.0/StructureDefinition/common/resource/general/patient-v1/ILSPatient"
          ]
        }
      }
    }
  ]
}
```

Id	Connector	Status
80	Source	TRANSFORMED
	Get Patient	SENT
	Get Laboratory Encounter	SENT
	Get Laboratory Encounter Observations	SENT

Messages Mappings
 Raw Encoded Sent Response

URL: https://hapi.fhir.org/baseR4/Patient?_id=Patient/624175
METHOD: GET

[HEADERS]

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

rth Connect Administrator - (4.4.1)

Channel Messages - FHIR Lab Test Processor

Start Time: 12:23 pm All Day RECEIVED
End Time: 12:23 pm TRANSFORMED
Text Search: Regex FILTERED
Page Size: 20 QUEUED
 SENT
 ERROR
 PENDING

Current Search
Max Message Id: 80
Date Range: (any) to (any)
Statuses: (any)
Connectors: (any)

Id	Connector	Status	Received Date	Response
80	Source	TRANSFORMED	2024-11-16 16:55:49.277	--
	Get Patient	SENT	2024-11-16 16:55:49.280	2024-11-16 16:55:49.280
	Get Laboratory Encounter	SENT	2024-11-16 16:55:49.753	2024-11-16 16:55:49.753
	Get Laboratory Encounter Observations	SENT	2024-11-16 16:55:50.733	2024-11-16 16:55:50.733

Messages | Mappings

Raw Encoded Sent Response

Status: SENT

Response:

```
}
],
"entry" : [
  {
    "fullUrl" : "https://hapi.fhir.org/baseR4/Encounter/624245",
    "resource" : {
      "resourceType" : "Encounter",
      "id" : "624245",
      "meta" : {
        "versionId" : "1",
        "lastUpdated" : "2020-02-18T10:20:57.687+00:00",
        "source" : "#QevMc30HwqsbBcpx"
      },
      "status" : "finished",
      "class" : {
        "system" : "http://terminology.hl7.org/CodeSystem/v3-ActCode",
        "code" : "AMB"
      },
      "type" : [
        {
          "coding" : [
            {
              "system" : "http://snomed.info/sct",
              "code" : "162673000",
              "display" : "General examination of patient (procedure)"
            }
          ],
          "text" : "General examination of patient (procedure)"
        }
      ]
    }
  }
],
]
```

Id	Connector	Status	Response
80	Source	TRANSFORMED	2
	Get Patient	SENT	2
	Get Laboratory Encounter	SENT	2
	Get Laboratory Encounter Observations	SENT	2

Messages | Mappings

Raw Encoded Sent Response

URL: https://hapi.fhir.org/baseR4/Encounter?_id=Encounter/624245
METHOD: GET

[HEADERS]

[PARAMETERS]

[CONTENT]

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

IMPLEMENTAZIONE FHIR

Dimostrazione end-to-end

Channel Messages - FHIR Lab Test Processor

Start Time: 12:23 pm All Day RECEIVED
End Time: 12:23 pm TRANSFORMED
Text Search: Regex FILTERED
Page Size: 20 QUEUED
 SENT
 ERROR
 PENDING

Current S
Max Messa
Date Rang
Statuses: (
Connectors:

Id	Connector	Status	Received Date	Response Date	Errors
80	Source	TRANSFORMED	2024-11-16 16:55:49.277	--	--
	Get Patient	SENT	2024-11-16 16:55:49.280	2024-11-16 16:55:49.737	--
	Get Laboratory Encounter	SENT	2024-11-16 16:55:49.753	2024-11-16 16:55:50.717	--
	Get Laboratory Encounter Observations	SENT	2024-11-16 16:55:50.733	2024-11-16 16:55:51.737	--
	PDF File Creation	SENT	2024-11-16 16:55:51.753	2024-11-16 16:55:51.767	--

Messages | Mappings

Raw Encoded Sent Response

Status:
SENT

Response:

```
"profile" : [
  "https://www.fhir.philips.com/4.0/StructureDefinition/common/resource/general/ok
],
"status" : "final",
"category" : [
  {
    "coding" : [
      {
        "system" : "http://terminology.hl7.org/CodeSystem/observation-category",
        "code" : "laboratory",
        "display" : "laboratory"
      }
    ]
  }
],
"code" : {
  "coding" : [
    {
      "system" : "http://loinc.org",
      "code" : "49765-1",
      "display" : "Calcium"
    }
  ],
  "text" : "Calcium"
},
"subject" : {
```

Id	Connector	Status	Received Date	Response Date	Errors
80	Source	TRANSFORMED	2024-11-16 16:55:49.277	--	--
	Get Patient	SENT	2024-11-16 16:55:49.280	2024-11-16 16:55:49.737	--
	Get Laboratory Encounter	SENT	2024-11-16 16:55:49.753	2024-11-16 16:55:50.717	--
	Get Laboratory Encounter Observations	SENT	2024-11-16 16:55:50.733	2024-11-16 16:55:51.737	--
	PDF File Creation	SENT	2024-11-16 16:55:51.753	2024-11-16 16:55:51.767	--

Messages | Mappings

Raw Encoded Sent Response

URL: https://hapi.fhir.org/baseR4/Observation?patient=Patient/624175&encounter=Encounter/624245&category=laboratory&_count=500
METHOD: GET

[HEADERS]
[PARAMETERS]
[CONTENT]

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

IMPLEMENTAZIONE FHIR

Dimostrazione end-to-end

The screenshot shows the Mirth Connect Administrator interface for a channel named 'Channel Messages - FHIR Lab Test Processor'. The interface includes filters for Start Time, End Time, Text Search, and Page Size. A table displays the message processing details for message ID 136, showing a successful 'TRANSFORMED' status with various steps like 'Get Patient', 'Get Laboratory...', 'PDF File Creation', and 'PDF encoding'. Below the table, the 'Messages' tab is selected, showing the output file path and the content of the generated PDF document.

Id	Connector	Status	Received Date	Response Date	Errors	SOURCE	TYP
136	Source	TRANSFORMED	2025-12-15 12:01:48.083	2025-12-15 12:01:50.450	--	--	--
	Get Patient	SENT	2025-12-15 12:01:48.087	2025-12-15 12:01:48.213	--	--	--
	Get Laboratory...	SENT	2025-12-15 12:01:48.247	2025-12-15 12:01:49.337	--	--	--
	Get Laboratory...	SENT	2025-12-15 12:01:49.387	2025-12-15 12:01:50.230	--	--	--
	PDF File Creation	SENT	2025-12-15 12:01:50.407	2025-12-15 12:01:50.420	--	--	--
	PDF encoding	SENT	2025-12-15 12:01:50.420	2025-12-15 12:01:50.430	--	--	--

Messages | Mappings

Raw Encoded Sent Response

OUTPUT: File
URI: C:/lab-viewer/TestResult/624245_624175_Villegas15.pdf
DOCUMENT TYPE: pdf

[CONTENT]
<html><head><title>Patient Report</title></head><body><h1>Patient Report</h1><p>Patient ID: 624175</p></body></html>

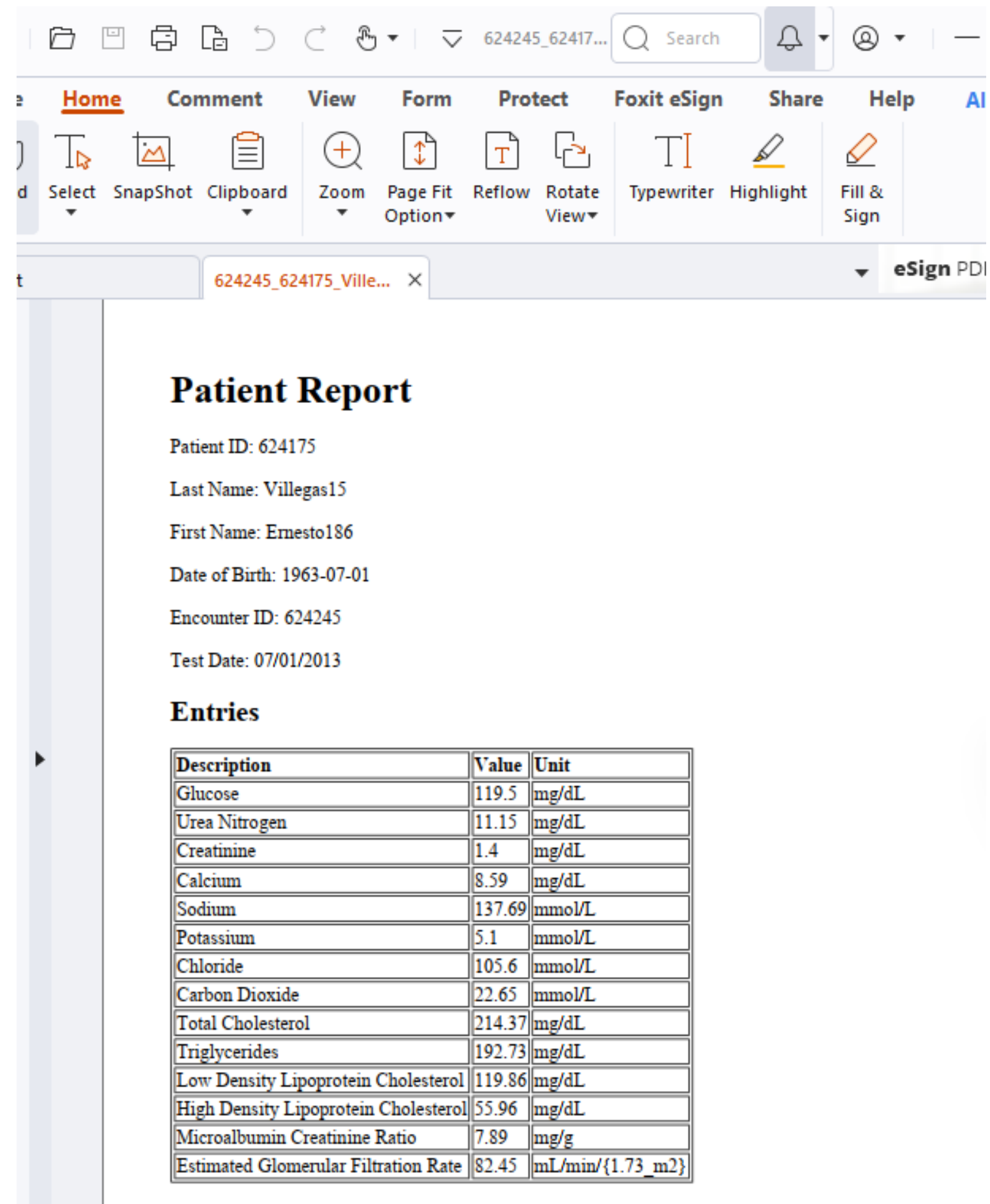
The screenshot shows a Windows File Explorer window for the path 'Local Disk (C:) > lab-viewer > TestResult'. It displays two PDF files: '624245_624175_Villegas15.pdf' (5 KB) and '624314_624175_Villegas15.pdf' (10 KB), both dated 15/12/2025 12:01. The files are of type 'Foxit PDF Reader ...'.

Name	File ownership	Date modified	Type	Size
624245_624175_Villegas15.pdf		15/12/2025 12:01	Foxit PDF Reader ...	5 KB
624314_624175_Villegas15.pdf		15/12/2025 12:01	Foxit PDF Reader ...	10 KB

Il file PDF viene creato correttamente nella cartella di destinazione.

IMPLEMENTAZIONE FHIR

Dimostrazione end-to-end



The screenshot shows a PDF viewer interface with a menu bar (Home, Comment, View, Form, Protect, Foxit eSign, Share, Help) and a toolbar (Select, Snapshot, Clipboard, Zoom, Page Fit Option, Reflow, Rotate View, Typewriter, Highlight, Fill & Sign). The document content is a Patient Report for Patient ID 624175, Last Name: Villegas15, First Name: Ernesto186, Date of Birth: 1963-07-01, Encounter ID: 624245, and Test Date: 07/01/2013. The report includes a table of lab test results.

Description	Value	Unit
Glucose	119.5	mg/dL
Urea Nitrogen	11.15	mg/dL
Creatinine	1.4	mg/dL
Calcium	8.59	mg/dL
Sodium	137.69	mmol/L
Potassium	5.1	mmol/L
Chloride	105.6	mmol/L
Carbon Dioxide	22.65	mmol/L
Total Cholesterol	214.37	mg/dL
Triglycerides	192.73	mg/dL
Low Density Lipoprotein Cholesterol	119.86	mg/dL
High Density Lipoprotein Cholesterol	55.96	mg/dL
Microalbumin Creatinine Ratio	7.89	mg/g
Estimated Glomerular Filtration Rate	82.45	mL/min/{1.73_m2}


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

IMPLEMENTAZIONE FHIR

Dimostrazione end-to-end

Patient Report

1 / 1 | - 100% + | [Print] [Share] [Refresh] [Back] [Forward]



1

Patient Report

Patient ID: 624175

Last Name: Villegas15

First Name: Ernesto186

Date of Birth: 1963-07-01

Encounter ID: 624245

Test Date: 07/01/2013

Entries

Description	Value	Unit
Glucose	119.5	mg/dL
Urea Nitrogen	11.15	mg/dL
Creatinine	1.4	mg/dL
Calcium	8.59	mg/dL
Sodium	137.69	mmol/L
Potassium	5.1	mmol/L
Chloride	105.6	mmol/L
Carbon Dioxide	22.65	mmol/L
Total Cholesterol	214.37	mg/dL
Triglycerides	192.73	mg/dL
Low Density Lipoprotein Cholesterol	119.86	mg/dL

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*), trasformarle 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.