

Artículo

[Muhammad Waseem](#) · Dic 22, 2021 Lectura de 3 min

Cómo crear y validar cualquier recurso FHIR usando el esquema FHIR con la ayuda de IntelliSense y la función de autocompletado en VS Code

La interoperabilidad en la asistencia sanitaria es esencial para mejorar la atención a los pacientes, reducir los costes de los proveedores de atención médica y ofrecer una imagen más precisa a los proveedores. Pero con tantos sistemas diferentes, los datos se presentan en diferentes formatos. Se han creado muchos estándares para tratar de resolver este problema, incluyendo HL7v2, HL7v3 y CDA, pero cada una tiene sus limitaciones.

FHIR, o Fast Healthcare Interoperability Resources, es un estándar para el intercambio de datos de salud, que tiene como objetivo resolver estos problemas. Fue desarrollado por Health Level Seven International (HL7), una organización que también desarrolló HL7v2, HL7v3 y CDA.

En este artículo descubriremos cómo crear y validar recursos FHIR usando el esquema FHIR con la ayuda de IntelliSense y la función de autocompletado en VS Code.

Paso 1: Descargar el archivo JSON Schema para Validación de Recursos desde la web oficial de FHIR: <https://www.hl7.org/fhir/>

The screenshot shows the HL7 FHIR Release 4 documentation website. The navigation menu includes Home, Getting Started, Documentation, Resources, Profiles, Extensions, Operations, and Terminologies. A green arrow points to the 'Documentation' link in the menu. Below the menu, there is a 'Table of Contents > Documentation Index' link. A yellow banner at the top of the content area states: 'This page is part of the FHIR Specification (v4.0.1: R4 - Mixed Normative and STU). This is the current published version. For a full list of available versions, see the Directory of published versions'. The main heading is '1.1 Documentation Index'. Below this, there are three columns of links. The first column is 'Framework', the second is 'Exchanging Resources', and the third is 'Adopting & Using FHIR'. A green arrow points to the 'Downloads - Schemas, Code, Tools' link in the 'Adopting & Using FHIR' column. Below this, there are sections for 'Safety & Security' and 'Implementation Advice'.

FHIR Infrastructure Work Group	Maturity Level: N/A	Standards Status: Informative
--	---------------------	-------------------------------

This page provides an index to the key commonly used documentation pages for FHIR.

- Framework**
 - Conformance Rules **N**
 - Resource Life Cycles
 - References between Resources **N**
 - Compartments
 - Narrative **N**
 - Extensibility **N**
 - Formats: **N** XML **N**, **N** JSON **N**, & **N** RDF
 - Terminologies **N** (Code Systems, Value Sets)
 - FHIRPath **N**
 - Mappings to other standards
- Version Management**
 - Change Management & Versioning **N**
 - Managing Multiple FHIR Versions
 - Version History
 - Differences to Release 3
- Exchanging Resources**
 - RESTful API (HTTP)** **N**
 - Search **N** (Search Param Registry)
 - Operations **N**
 - Asynchronous Use
 - Using GraphQL
 - Documents
 - Messaging
 - Services
 - Persistence/Data bases
- Base Types**
 - Data Types (Base) **N**
 - Metadata Types **N**
 - Resource **N**
 - DomainResource **N**
 - Element **N**
- Adopting & Using FHIR**
 - Profiling FHIR **N**
 - FHIR Workflow
 - Downloads - Schemas, Code, Tools**
 - Managing Multiple FHIR Versions
 - Validating Resources
 - Best Practices for Implementers
 - Mapping Language (tutorial)
 - Testing Implementations
- Safety & Security**
 - Security, Security Labels & Signatures
 - Clinical Safety
- Implementation Advice**
 - Managing Resource Identity
 - Guide to Resources

hl7.org/fhir/downloads.html

This page is part of the FHIR Specification (v4.0.1: R4 - Mixed Normative and STU). This is the current published version. For a full list of available versions, see the [Directory of published versions](#)

7.1 Downloads

FHIR Infrastructure [Work Group](#) Maturity Level: N/A Standards Status: Informative

Specification Downloads

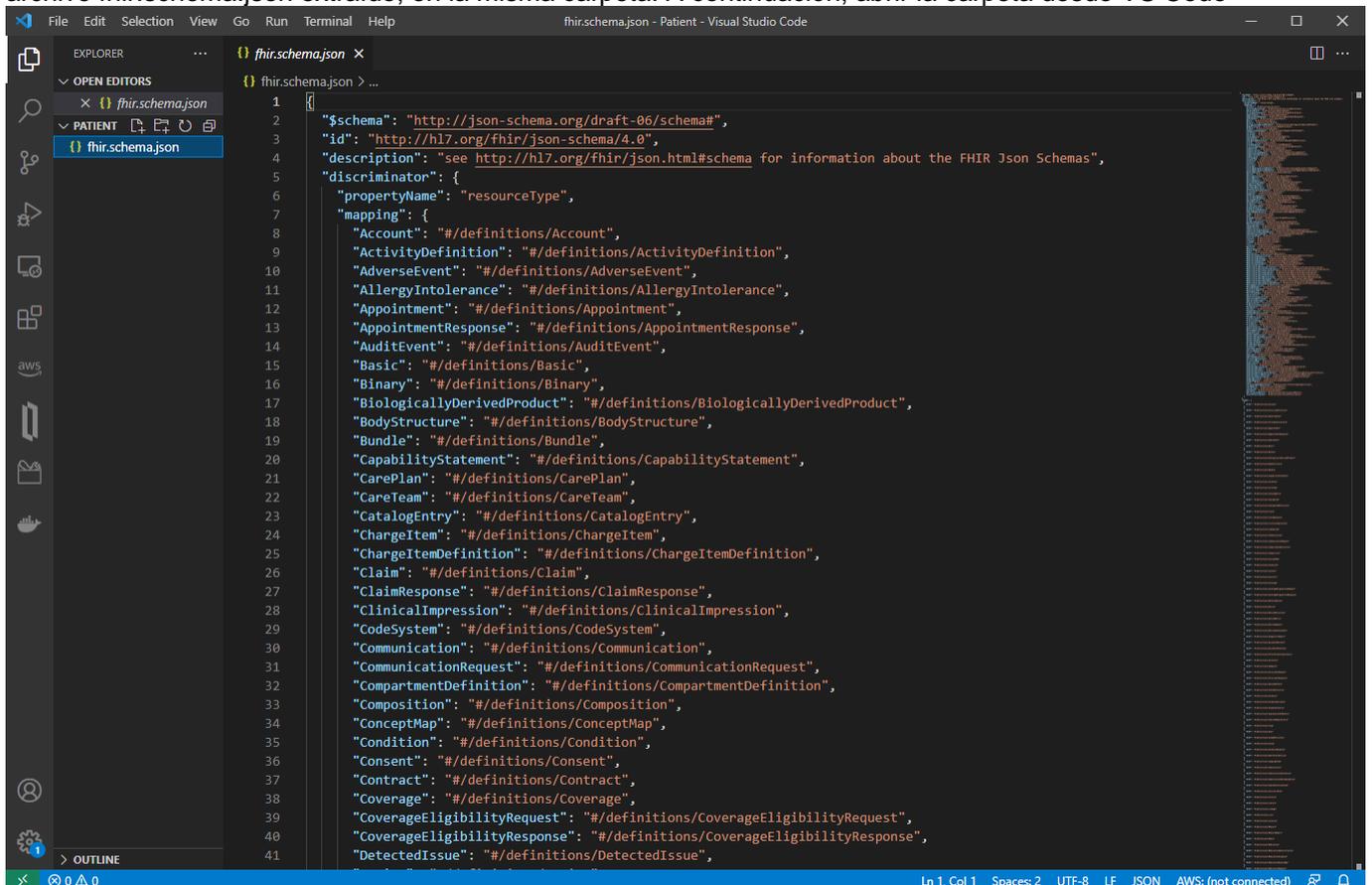
FHIR Definitions All the value sets, profiles, etc. defined as part of the FHIR specification, and the included implementation guides:

- XML
- JSON
- Forge: Special version of definitions for [Forge](#) (temporary)

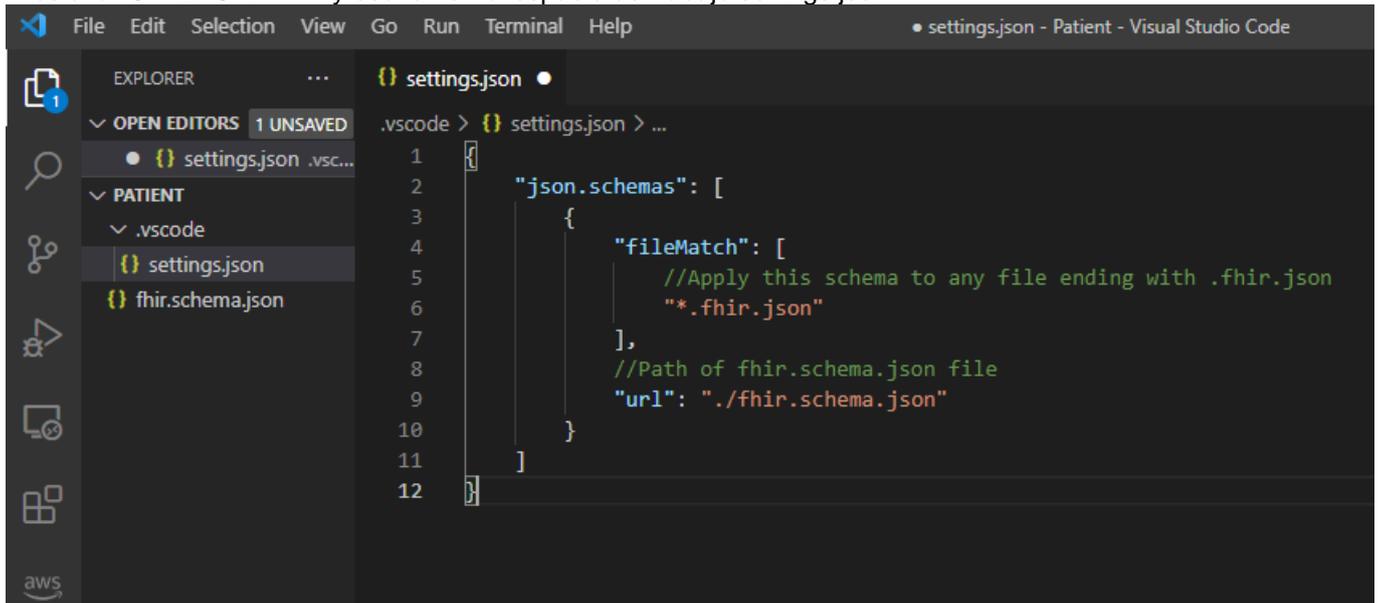
This is the master set of definitions that should be the first choice whenever generating any implementation artifacts. All the other forms below include only subsets of the information available in these definition files, and do not contain all of the rules about what makes resources valid. Implementers will still need to be familiar with the content of the specification and with any profiles that apply to the resources in order to make a conformant implementation.

- XML
- [Examples](#) - all the example resources in XML format
 - [Validation Schemas](#) (includes support schemas, resource schemas, modular & combined schemas, and Schematrons)
 - [Code Generation Schemas](#) (see [notes about code-generation schemas](#))
Note that names relevant for code generation, including resource names, element & slice names, codes, etc. may collide with reserved words in the relevant target language, and code generators will need to handle this
- JSON
- [Examples](#) - all the example resources in JSON format
 - [JSON Schema \(Needs JSON Schema draft-06 or more recent\)](#)
 - [Examples](#) - all the example resources in ND-JSON format (bulk data format)
- RDF
- [Turtle Examples](#) - all the example resources in Turtle format
 - [ShEx Schemas - ShEx](#) definitions for validating RDF resources
 - [Definitions](#) - the formal definitions that define the predicates and classes used in the RDF format (not up to date)

Paso 2: Crear una carpeta (en este ejemplo estoy usando la carpeta Patient y el recurso Patient) y copiar el archivo fhir.schema.json extraído, en la misma carpeta. A continuación, abrir la carpeta desde VS Code

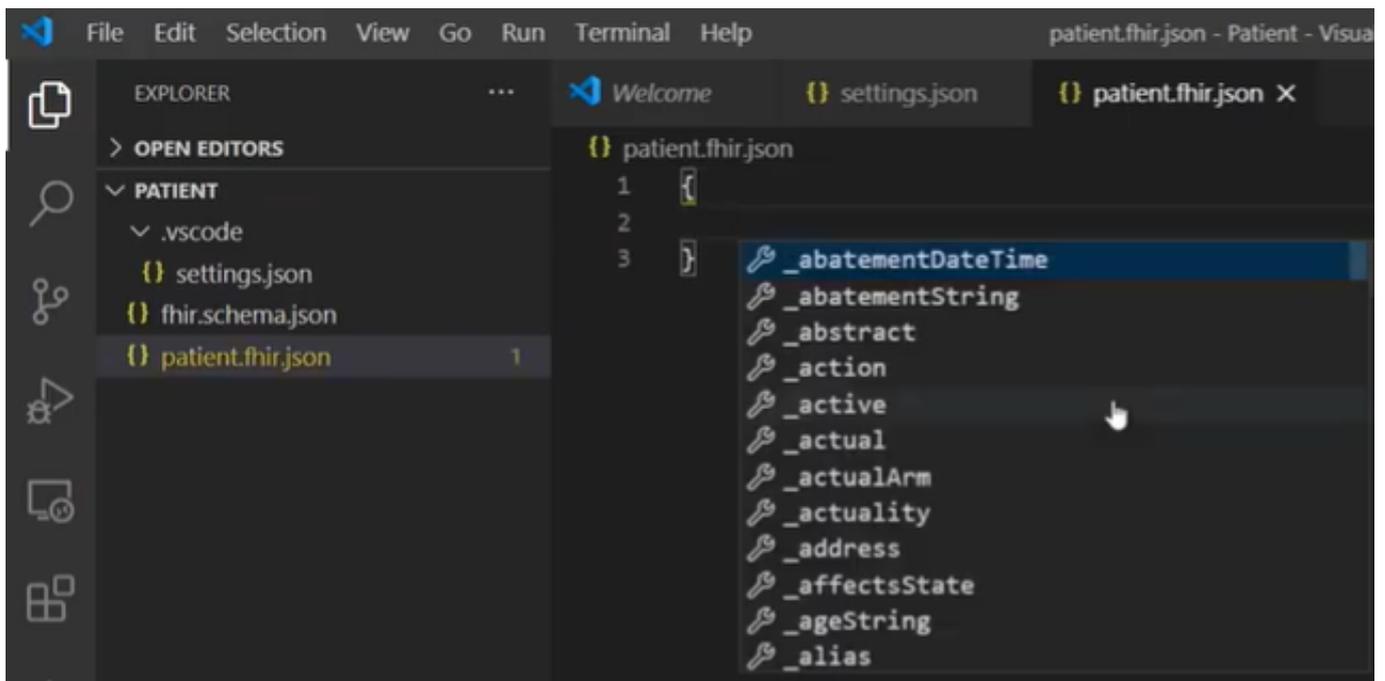


Paso 3: Configurar el código VS para reconocer el esquema FHIR modificando el archivo settings.json. Presionar CTRL+SHIFT+P y escribir en el espacio de trabajo settings.json



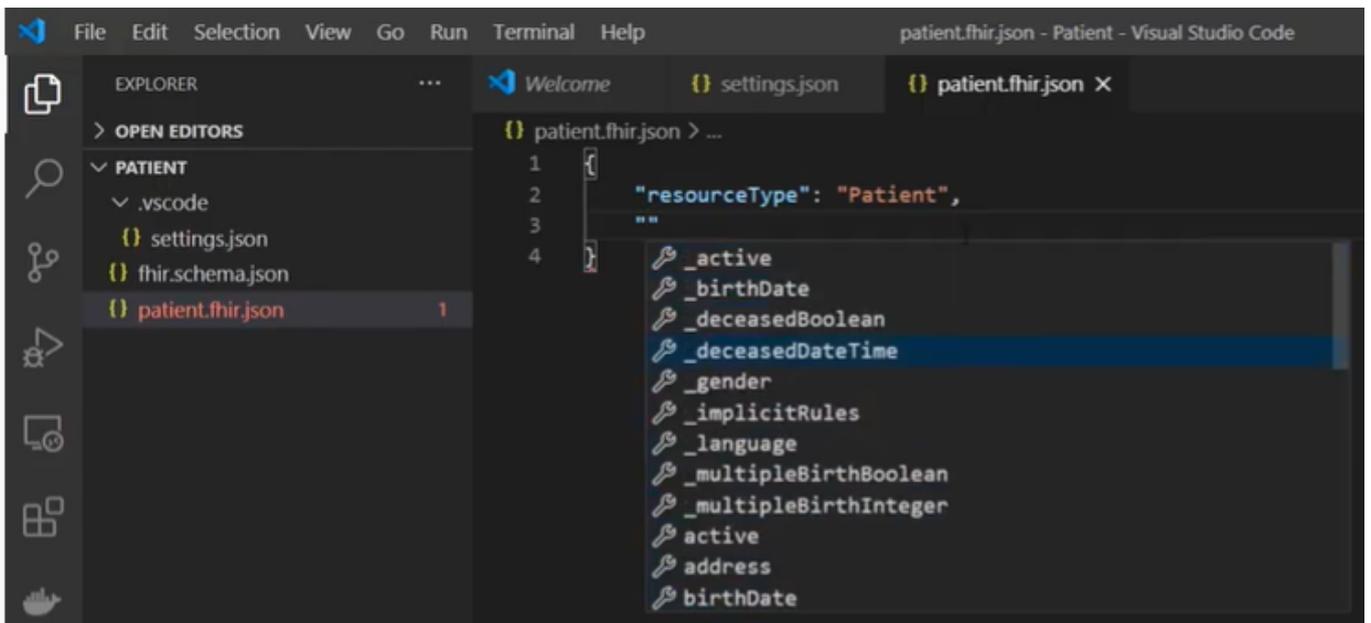
```
1 {
2   "json.schemas": [
3     {
4       "fileMatch": [
5         //Apply this schema to any file ending with .fhir.json
6         "**.fhir.json"
7       ],
8       //Path of fhir.schema.json file
9       "url": "./fhir.schema.json"
10    }
11  ]
12 }
```

Paso 4: Crear un nuevo archivo patient.fhir.json, en la misma carpeta. Pulsar Ctrl+ESPACIO y se obtendrán todos los atributos de los recursos de FHIR a través de IntelliSense

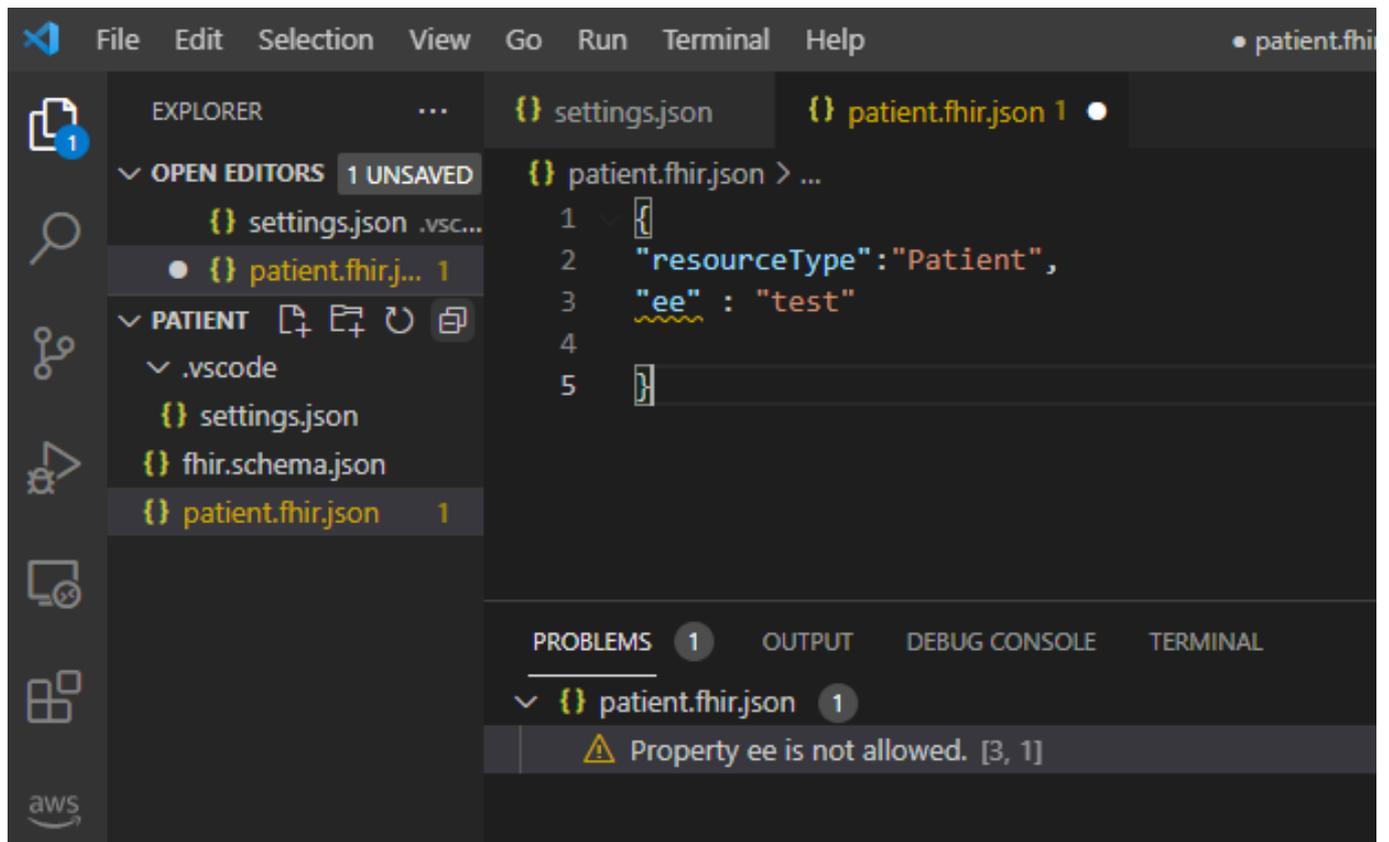


```
1 {
2
3   _abatementDateTime
4   _abatementString
5   _abstract
6   _action
7   _active
8   _actual
9   _actualArm
10  _actuality
11  _address
12  _affectsState
13  _ageString
14  _alias
```

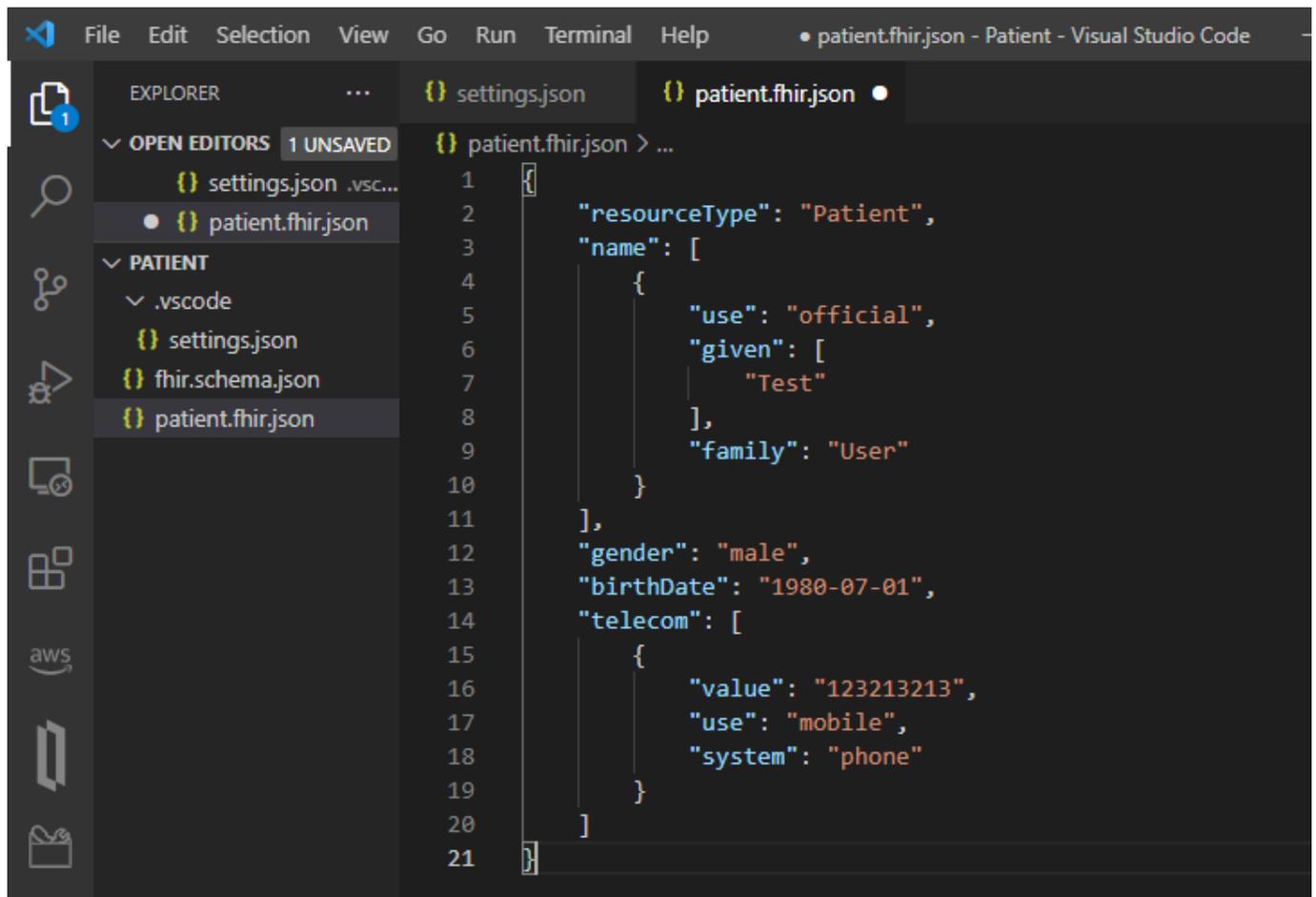
Escribir "resourceType": "Patient" y todos los atributos relacionados con el recurso Patient comenzarán a aparecer en el IntelliSense.



VS Code validará automáticamente la estructura y la sintaxis del recurso.



Con la ayuda de IntelliSense y la función de autocompletado hemos creado y validado nuestro recurso para los pacientes.



The screenshot shows the Visual Studio Code interface with the 'patient.fhir.json' file open. The Explorer sidebar on the left shows the project structure under 'PATIENT', including files like 'settings.json', 'fhir.schema.json', and 'patient.fhir.json'. The main editor area displays the following JSON content:

```
1  {}
2  "resourceType": "Patient",
3  "name": [
4    {
5      "use": "official",
6      "given": [
7        "Test"
8      ],
9      "family": "User"
10   }
11 ],
12 "gender": "male",
13 "birthDate": "1980-07-01",
14 "telecom": [
15   {
16     "value": "123213213",
17     "use": "mobile",
18     "system": "phone"
19   }
20 ]
21 }
```

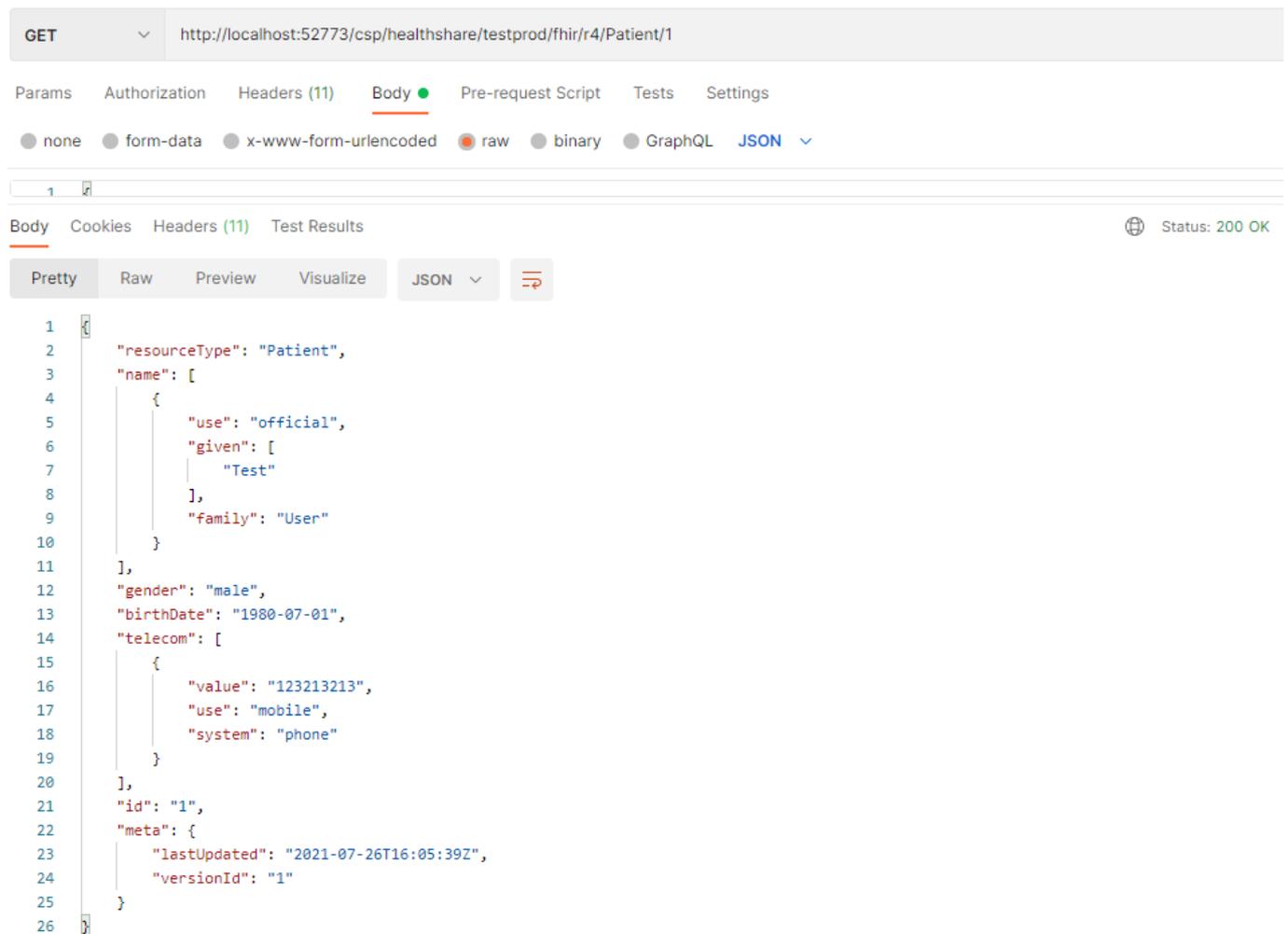
Paso 5: Publicar el recurso creado en el servidor FHIR de InterSystems usando la API Rest de Postman

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** `http://localhost:52773/csp/healthshare/testprod/fhir/r4/Patient` (indicated by a green arrow pointing to the URL field)
- Body:** A JSON object representing a patient resource:

```
1 {
2   "resourceType": "Patient",
3   "name": [
4     {
5       "use": "official",
6       "given": [
7         "Test"
8       ],
9       "family": "User"
10    }
11  ],
12  "gender": "male",
13  "birthDate": "1980-07-01",
14  "telecom": [
15    {
16      "value": "123213213",
17      "use": "mobile",
18      "system": "phone"
19    }
20  ]
21 }
```
- Status:** 201 Created (indicated by a green arrow pointing to the status bar)
- Response Format:** JSON
- Response View:** Pretty

Recuperar el recurso de Patient Creado utilizando el método GET



```
1  {
2    "resourceType": "Patient",
3    "name": [
4      {
5        "use": "official",
6        "given": [
7          "Test"
8        ],
9        "family": "User"
10     }
11  ],
12  "gender": "male",
13  "birthDate": "1980-07-01",
14  "telecom": [
15    {
16      "value": "123213213",
17      "use": "mobile",
18      "system": "phone"
19    }
20  ],
21  "id": "1",
22  "meta": {
23    "lastUpdated": "2021-07-26T16:05:39Z",
24    "versionId": "1"
25  }
26  }
```

¡ Enhorabuena! Hemos creado y validado nuestro recurso Patient, y lo hemos publicado y recuperado correctamente en el Servidor FHIR de InterSystems usando Postman.

De esta manera podemos crear y validar fácilmente cualquier Recurso FHIR.

[#API REST](#) [#FHIR](#) [#Caché](#) [#Ensemble](#) [#InterSystems IRIS for Health](#) [#VSCode](#)

URL de fuente: <https://es.community.intersystems.com/post/c%C3%B3mo-crear-y-validar-cualquier-recurso-fhir-usando-el-esquema-fhir-con-la-ayuda-de-intellisense>