Article
Lucas Enard · Aug 17 9m read

Open Exchange

The simplest template with REST CRUD for InterSystems IRIS with ONLY Python

In this GitHub based on this InterSystems community rest api template Guillaume and I have created this example of all the import CRUD operations usable using ONLY Python on IRIS and using Flask.

Using the IRIS ORM or by simply doing SQL requests as both methods are seen in the GitHub.

1. intersystems-iris-docker-rest-template

This is a template of a REST API application built in python in InterSystems IRIS.
It also has OPEN API spec, can be developed with Docker and VSCode.

2. Prerequisites

Make sure you have git and Docker desktop installed.

It is to be noted that the table Sample.Person was already created in advance for the demo using in the management portal in the sql tab:

```
CREATE TABLE Sample.Person (

        Company                                 VARCHAR(50),
        DOB                                     DATE,
        Name                                    VARCHAR(4096),
        Phone                                   VARCHAR(4096),
        Title                                   VARCHAR(50)
)
```

3. Installation

3.1. Installation for development

Clone/git pull the repo into any local directory e.g. like it is shown below:

```
$ git clone https://github.com/grongierisc/iris-python-flask-api-template.git
```

Open the terminal in this directory and run:

```
$ DOCKER_BUILDKIT=1 docker-compose up -d --build
```

3.2. Management Portal and VSCode

This repository is ready for VS Code.

Open the locally-cloned formation-template-python folder in VS Code.

If prompted (bottom right corner), install the recommended extensions.

3.3. Having the folder open inside the container

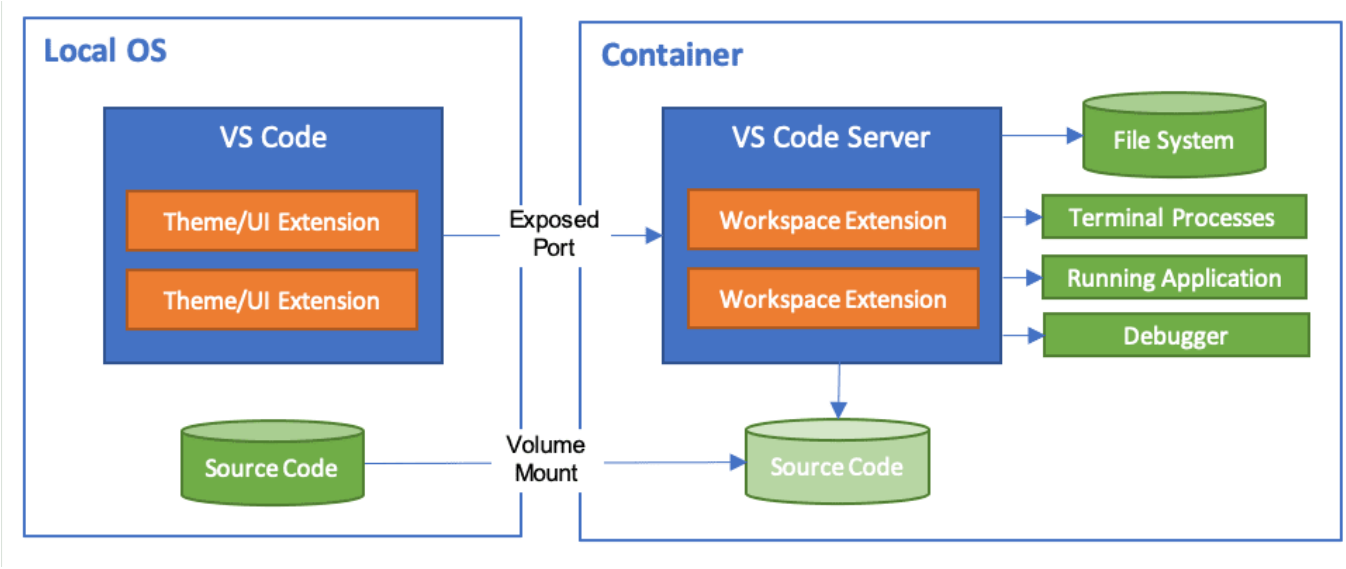
It is really important to be *inside* the container before coding.

For this, docker must be on before opening VSCode.

Then, inside VSCode, when prompted (in the right bottom corner), reopen the folder inside the container so you will be able to use the python components within it.

The first time you do this it may take several minutes while the container is readied.

More information here



By opening the folder remote you enable VS Code and any terminals you open within it to use the python components within the container. Configure these to use /usr/irissys/bin/irispython

4. How it works

The app.py, once launched (inside the container) will gather CRUD request.

Depending on the type of the request, the right message will be created to send to the FlaskService, this service will call the CrudPerson operation that will, depending on the type of the message send from the service to it, dispatch the information needed to do the action requested.

For more details you can check the How it works part of this fully documented demo.

5. How to Work With it

This template creates /crud REST web-application on IRIS which implements 4 types of communication: GET, POST, PUT and DELETE aka CRUD operations.
These interface works with a sample persistent class Person found in src/python/person/obj.py.

First of all, it is needed to start the 'app.py' situated in src/python/person/app.py using flask.

To do this, go in the app.py file, then to the run and debug window in VSCode and select Python: Flask then run. This will run the app.

5.1. POST request

5.1.1. Testing POST request

Create a POST request, for example in Postman or in RESTer for mozilla, with raw data in JSON like:

```
{"name":"Elon Musk","title":"CEO","company":"Tesla","phone":"123-123-1233","dob":"198
```

```
2-01-19"}
```

Using Content-Type as application/json

Adjust the authorisation if needed - it is basic for container with default login and password for IRIS Community edition container.

Send the POST request to localhost:5000/persons/

This will create a record in the table Sample.Person of IRIS and return the id of the newly added Person

of the POST request to add Elon Musk to the table.

5.1.2. How POST request works

```
    def create_person(self,request:CreatePersonRequest):
        """
        > Create a new person in the database and return the new person's ID

        :param request: The request object that was passed in from the client
        :type request: CreatePersonRequest
        :return: The ID of the newly created person.
        """

        # sqlInsert = 'insert into Sample.Person values (?,?,?,?,?)'
        # iris.sql.exec(sqlInsert,request.person.company,dob,request.person.name,requ
est.person.phone,request.person.title)

        # IRIS ORM
        person = iris.cls('Sample.Person')._New()
        if (v:=request.person.company) is not None: person.Company = v
        if (v:=request.person.name) is not None: person.Name = v
        if (v:=request.person.phone) is not None: person.Phone = v
        if (v:=request.person.title) is not None: person.Title = v
        if (v:=request.person.dob) is not None: person.DOB = v

        Utils.raise_on_error(person._Save())

        return CreatePersonResponse(person._Id())
```

Using IRIS ORM we can create a new Person and save into our database.

5.2. GET requests

5.2.1. Testing GET request

To test GET you need to have some data. You can create it with a POST request.

This REST API exposes two GET requests: all the data and one record.
To get all the data in JSON call:

```
localhost:5000/persons/all
```

To request the data for a particular record provide the id in GET request like 'localhost:5000/persons/id', here is an example:

```
localhost:5000/persons/1
```

This will return JSON data for the person with ID=1, something like that:

```
{"name":"Elon Musk","title":"CEO","company":"Tesla","phone":"123-123-1233","dob":"198
2-01-19"}
```

5.2.2. How GET request works

```
    def get_person(self,request:GetPersonRequest):
        """
        > The function takes a `GetPersonRequest` object, executes a SQL query, and r
eturns a
        `GetPersonResponse` object

        :param request: The request object that is passed in
        :type request: GetPersonRequest
        :return: A GetPersonResponse object
        """
        sql_select = """
            SELECT
                Company, DOB, Name, Phone, Title
            FROM Sample.Person
            where ID = ?
            """
        rs = iris.sql.exec(sql_select,request.id)
        response = GetPersonResponse()
        for person in rs:
            response.person= Person(company=person[0],dob=person[1],name=person[2],ph
one=person[3],title=person[4])
        return response

    def get_all_person(self,request:GetAllPersonRequest):
        """
        > This function returns a list of all the people in the Person table

        :param request: The request object that is passed to the service
        :type request: GetAllPersonRequest
        :return: A list of Person objects
        """

        sql_select = """
            SELECT
                Company, DOB, Name, Phone, Title
            FROM Sample.Person
            """
        rs = iris.sql.exec(sql_select)
        response = GetAllPersonResponse()
        response.persons = list()
        for person in rs:
            response.persons.append(Person(company=person[0],dob=person[1],name=perso
n[2],phone=person[3],title=person[4]))
```

```
        return response
```

This time, using the iris python sql.exec function, we can directly run SQL code inside the IRIS database, gather the information needed and send it back to the API and to the user.

5.3. PUT request

5.3.1. Testing PUT request

PUT request could be used to update the records. This needs to send the similar JSON as in POST request above supplying the id of the updated record in URL.
For example we want to change the record with id=5. Prepare the JSON in raw like following:

```
{"name":"Jeff Besos","title":"CEO","company":"Amazon","phone":"123-123-1233","dob":"1
982-01-19"}
```

and send the put request to:

```
localhost:5000/persons/5
```

5.3.2. How PUT request works

```
    def update_person(self,request:UpdatePersonRequest):
        """
        > Update a person in the database

        :param request: The request object that will be passed to the service
        :type request: UpdatePersonRequest
        :return: UpdatePersonResponse()
        """

        # IRIS ORM
        if iris.cls('Sample.Person')._ExistsId(request.id):
            person = iris.cls('Sample.Person')._OpenId(request.id)
            if (v:=request.person.company) is not None: person.Company = v
            if (v:=request.person.name) is not None: person.Name = v
            if (v:=request.person.phone) is not None: person.Phone = v
            if (v:=request.person.title) is not None: person.Title = v
            if (v:=request.person.dob) is not None: person.DOB = v
            Utils.raise_on_error(person._Save())

        return UpdatePersonResponse()
```

Using IRIS ORM we can check if the id leads to a Person, if it does, we can update it using our new information and save it into our database.

5.4. DELETE request

5.4.1. Testing DELETE request

For delete request this REST API expects only the id of the record to delete. E.g. if the id=5 the following DELETE

call will delete the record:

```
localhost:5000/persons/5
```

5.4.2. How DELETE request works

```
    def delete_person(self,request:DeletePersonRequest):
        """
        > Delete a person from the database

        :param request: The request object that is passed to the service
        :type request: DeletePersonRequest
        :return: The response is being returned.
        """

        sql_select = """
            DELETE FROM Sample.Person as Pers
            WHERE Pers.id = ?
            """
        rs = iris.sql.exec(sql_select,request.id)
        response = DeletePersonResponse()
        return response
```

This time, using the iris python sql.exec function, we can directly run SQL code inside the IRIS database and delete the person.

6. How to start coding

This repository is ready to code in VSCode with InterSystems plugins.
Open /src/python/person/app.py to change anything on the api.
Open /src/python/person/bo.py to be able to change things related to the internal requests, this is where you can use SQL - it will be compiled in running IRIS docker container.

7. What's inside the repo

7.1. Dockerfile

The simplest dockerfile to start IRIS.
Use the related docker-compose.yml to easily setup additional parametes like port number and where you map keys and host folders.

7.2. .vscode/settings.json

Settings file to let you immedietly code in VSCode with VSCode ObjectScript plugin)

7.3. .vscode/launch.json

Config file if you want to debug with VSCode ObjectScript

#API #Embedded Python #Python #InterSystems IRIS

Check the related application on InterSystems Open Exchange