Article
Yuri Marx · Aug 2, 2021 30m read

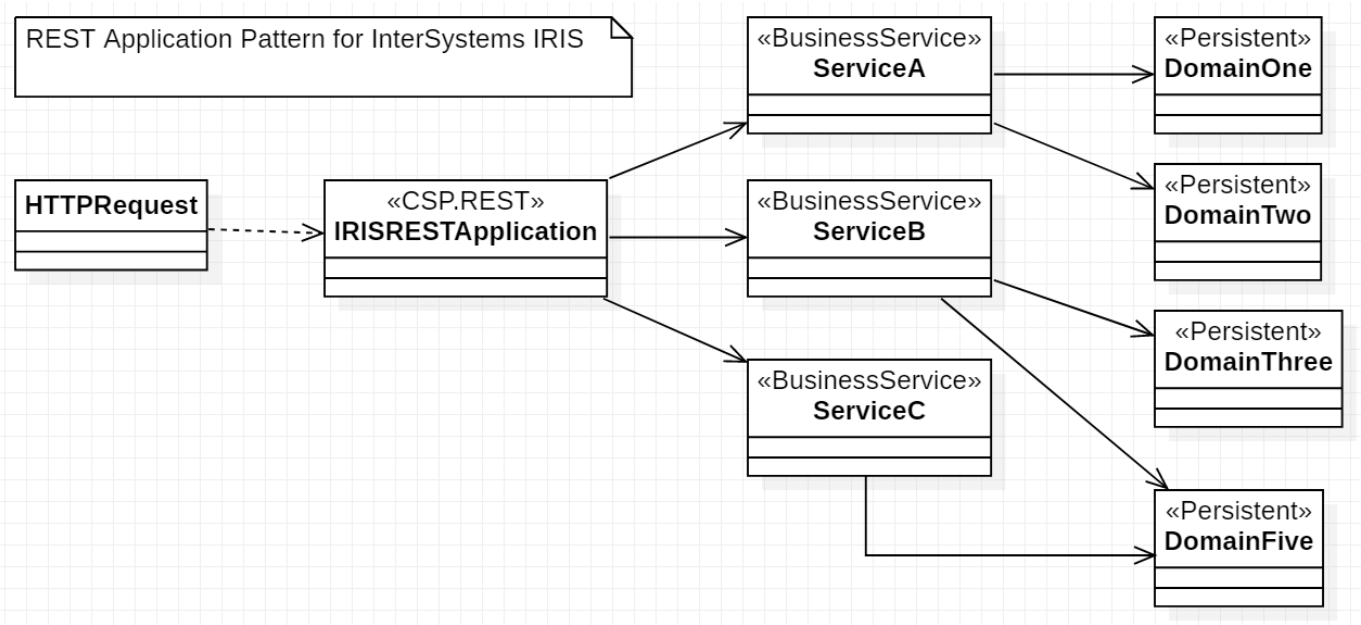
InterSystems IRIS REST Application Patterns

This article suggests to you some patterns to create REST API applications using IRIS.

Note: source code in https://github.com/yurimarx/movie

Class Pattern to the REST Application

To begin, see my suggestion for classes needed to create IRIS API applications:



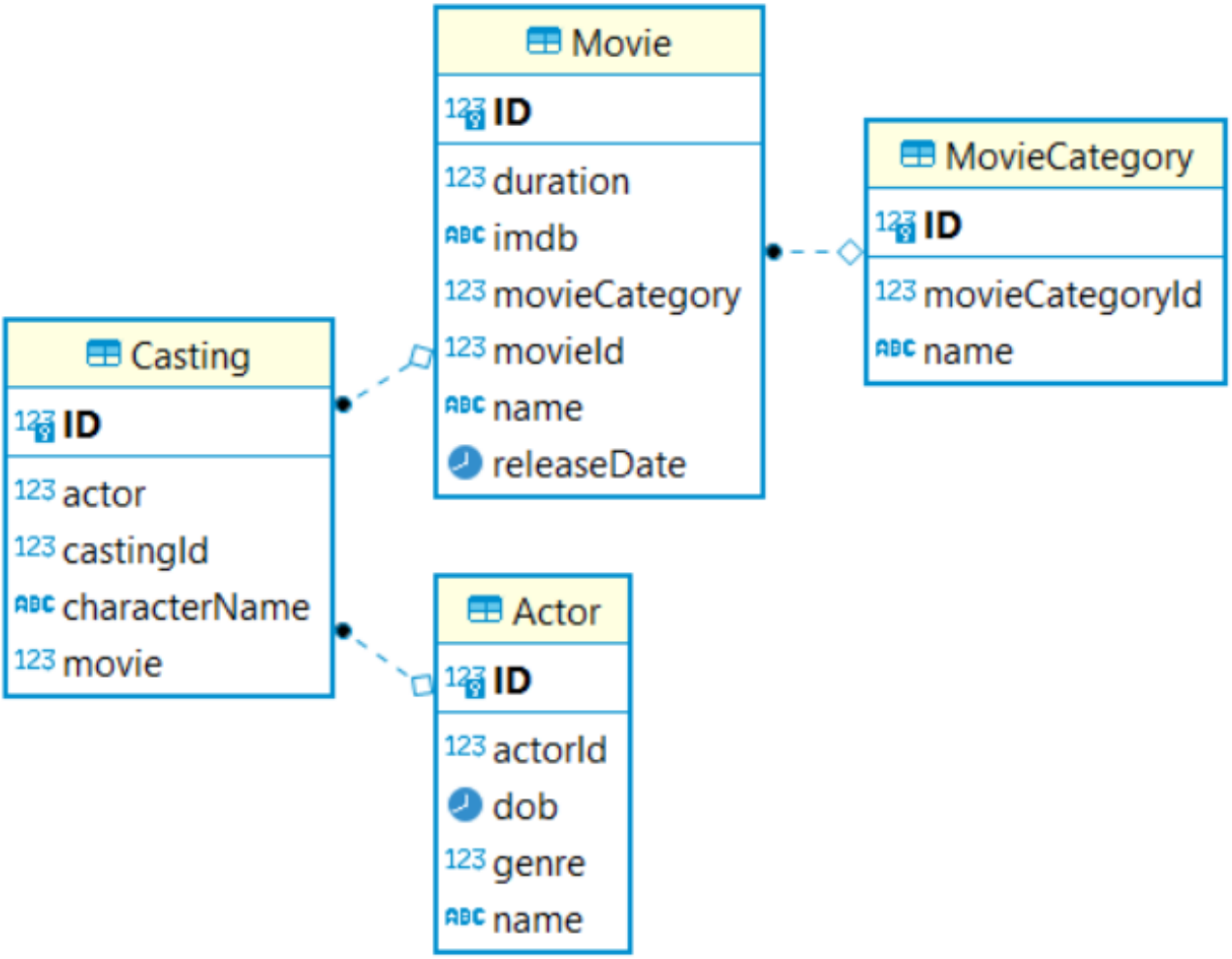
- **IRISRESTApplication**: CSP.REST class that will be the central controller for all REST requests and responses processed by the business services.
- **BusinessService**: class with a business topic implementation. It can use one or more Persistent Domain Classes to persist and query data required by the business topic requirements.
- **Persistent Domain**: persistent class to manage a SQL table.

Prereqs

- VSCode;
- Docker Desktop;
- InterSystems ObjectScript Extension Pack.

Class Diagram to the Sample Application

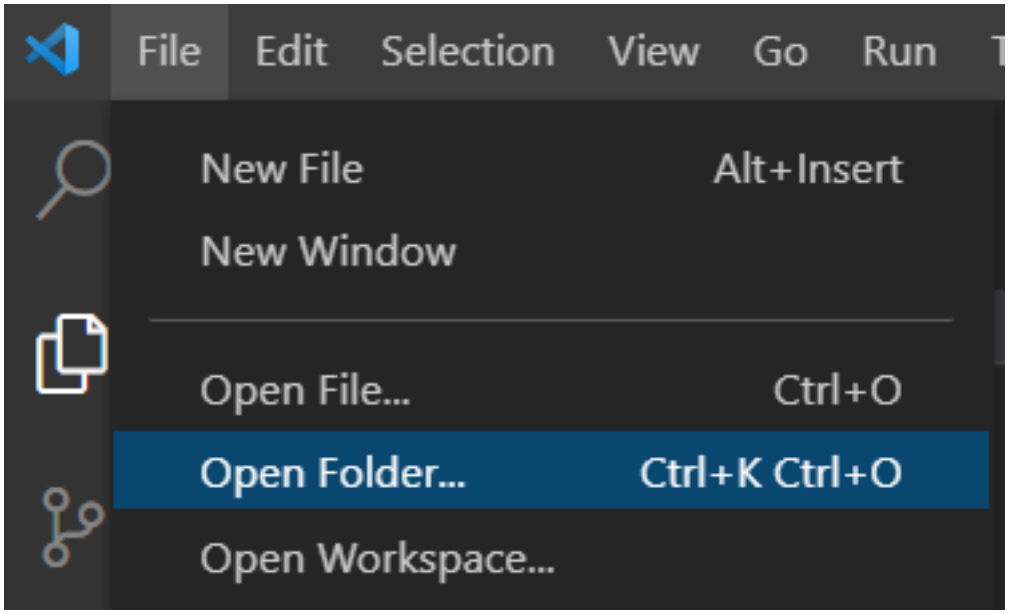
I will create a Movie Catalog application to demonstrate the patterns suggested in the article:



Note: thanks to the https://openexchange.intersystems.com/package/iris-rest-api-template application. It was the base to this tutorial.

Setup the Sample Application

1. Create a folder movie in your file system. Open this folder in a new VSCode window.



2. Create the Dockerfile file inside movie folder to run IRIS Community edition into a Docker container instance. Content:

Docker file content

3. Create the docker-compose.yml file inside movie folder to allows you run your docker instance and other instances together (not in this sample, but it is a good practice run from docker-compose instead dockerfile. Content:

Docker composer content

4. Create the iris.script file inside movie folder to do some actions before run IRIS. This file is important to do custom terminal actions necessary for the application, like disable password expiration. Content:

iris.script content

5. Create the module.xml file inside movie folder to install and run your application using ZPM. This file is important to do the application endpoint configuration and install swagger-ui (web app used to run and test your API using swagger file). Content:

Module.xml content

You can see CSPApplication tag, used to run the application API in the /movie-api URI and enable or disable password to consume the API.

6. Create the LICENSE file inside movie folder to setting the license of your application. Content:

LICENSE content

7. Create the README.md file inside movie folder to document your application to the users using markdown language. Content:

movie-rest-application

This is a sample of a REST API application built with ObjectScript in InterSystems IRIS.

8. Create .vscode folder inside movie folder. Create settings.json file inside .vscode folder to configure server connection between VSCode and your IRIS instance. Content:

Settings content

9. Create the folder src inside movie folder to put your source code folders and files.

10. Create dc folder inside src folder. This is a convention when your build projects to the InterSystems Developer Community, otherwise is not necessary.

11. Create movie folder inside dc folder. This folder will be the folder to your objectscript classes.

12. Create our first class, MovieRESTApp.cls file, inside src\dc\movie folder. This file will be the IRISRESTApplication class. Content:

MovieRESTApp content

Note 1: The class extends CSP.REST to be used as the REST Endpoint.

Note 2: the parameter chaset is used to encode requests and responses with UTF-8.

Note 3: the CONVERTINPUTSTREAM is used to force the request content in the UTF-8, without this you can have problems with special latin chars.

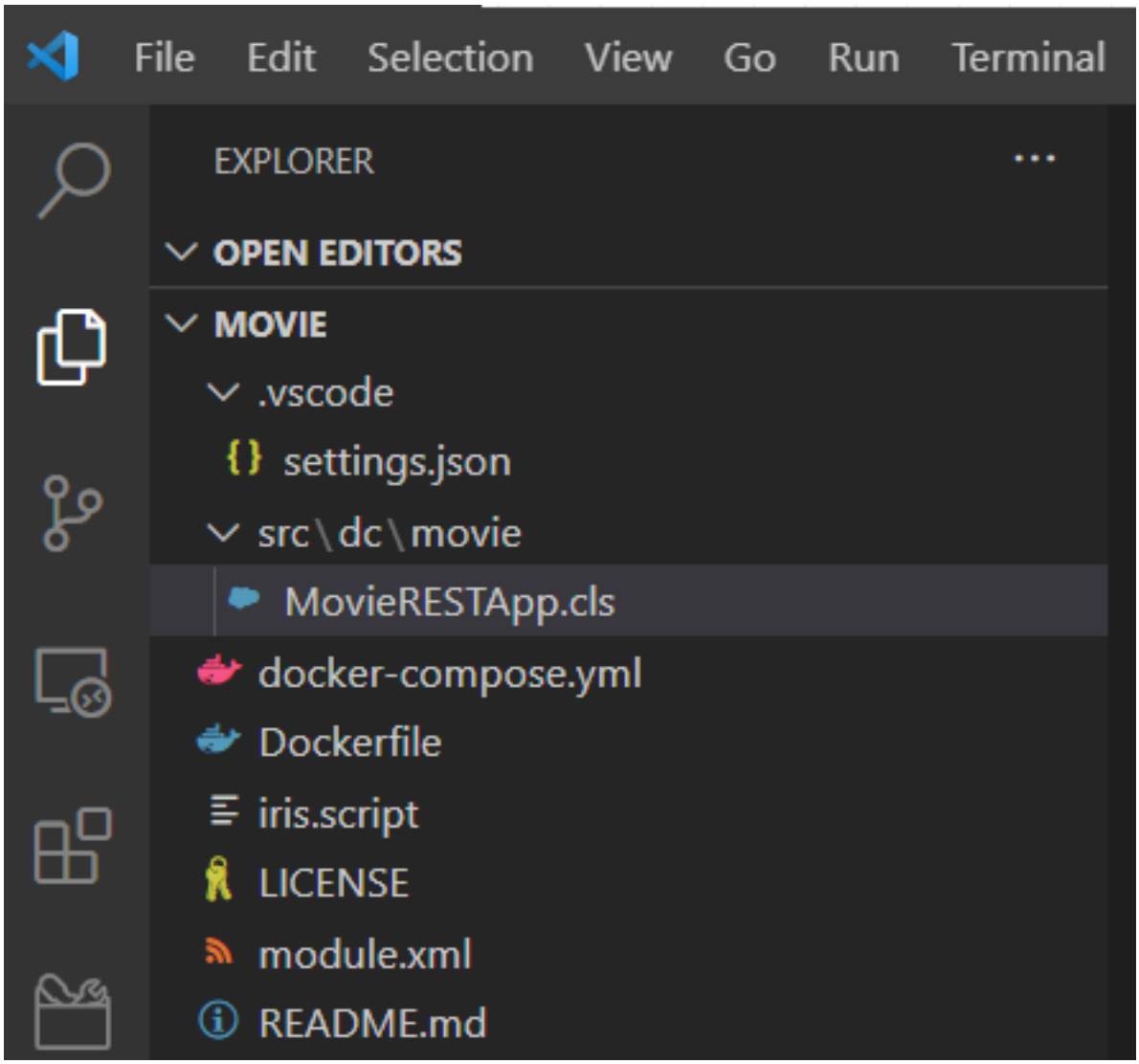
Note 4: CONTENTTYPE is used to declare the content using JSON, not XML.

Note 5: HandleCorsRequest = 1 is necessary to allows you consume the API from other servers different from the IRIS server.

Note 6: Routes are used to declare API URI to each class method.

Note 7: SwaggerSpec from CSP.REST class allows you generate the API swagger (API web documentation) content.

Now you have the following folders and files:



13. Open VSCode Terminal (menu Terminal > New Terminal) and type:

```
docker-compose up -d --build
```

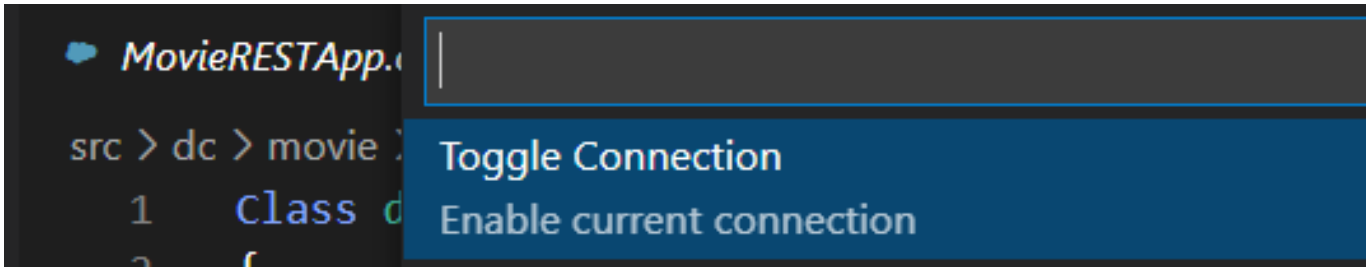
This will build the docker instance and run it.

14. Test your API with Swagger-UI. In the browser and type: http://localhost:52773/swagger-ui/index.html. Pay attention to the address bar (fix the url, if necessary to correct address)

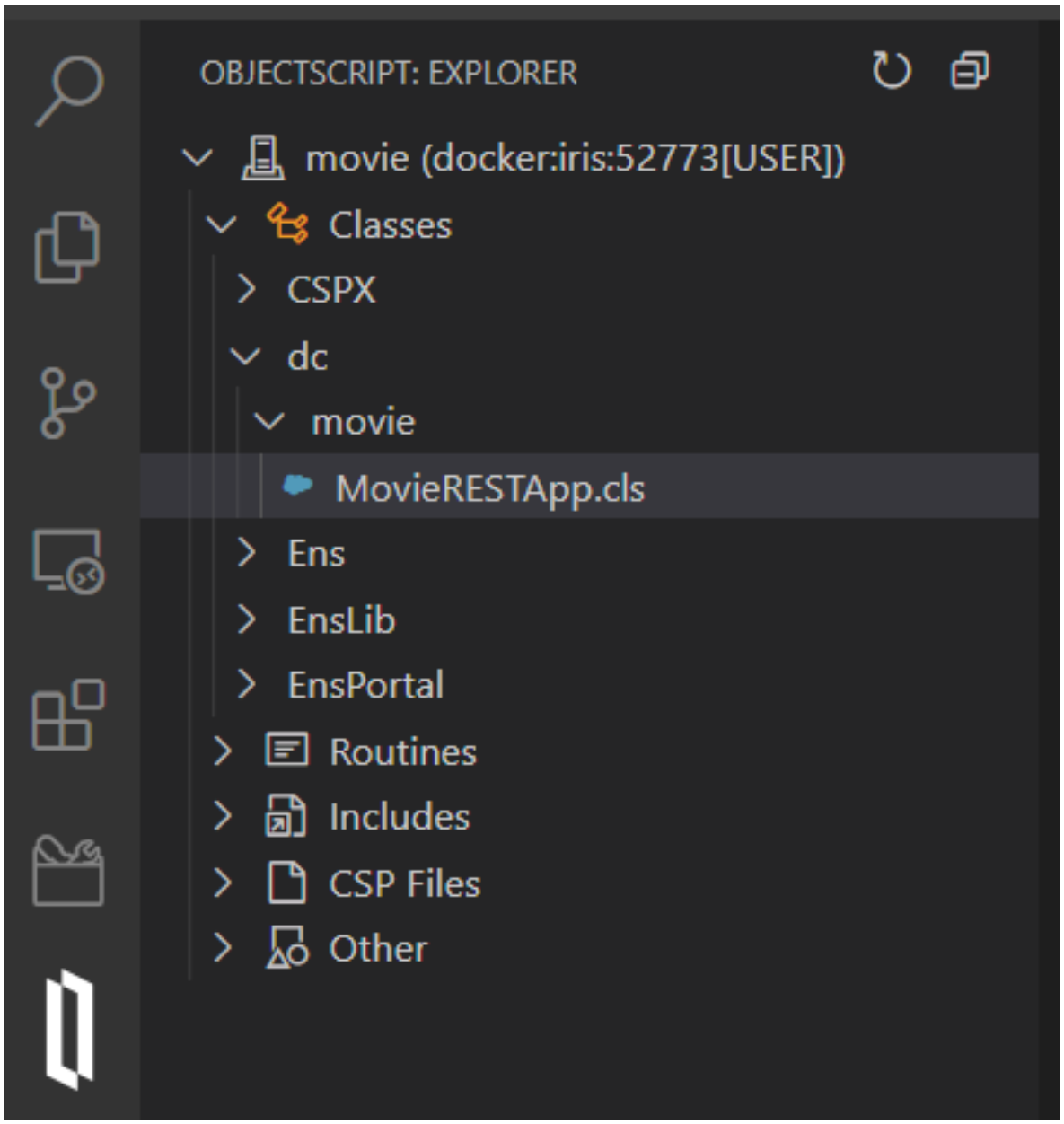
Connection beetween VSCode and IRIS

1. Click in the ObjectScript bar (in the VSCode footer)

2. Select Toogle Connection in the Top:

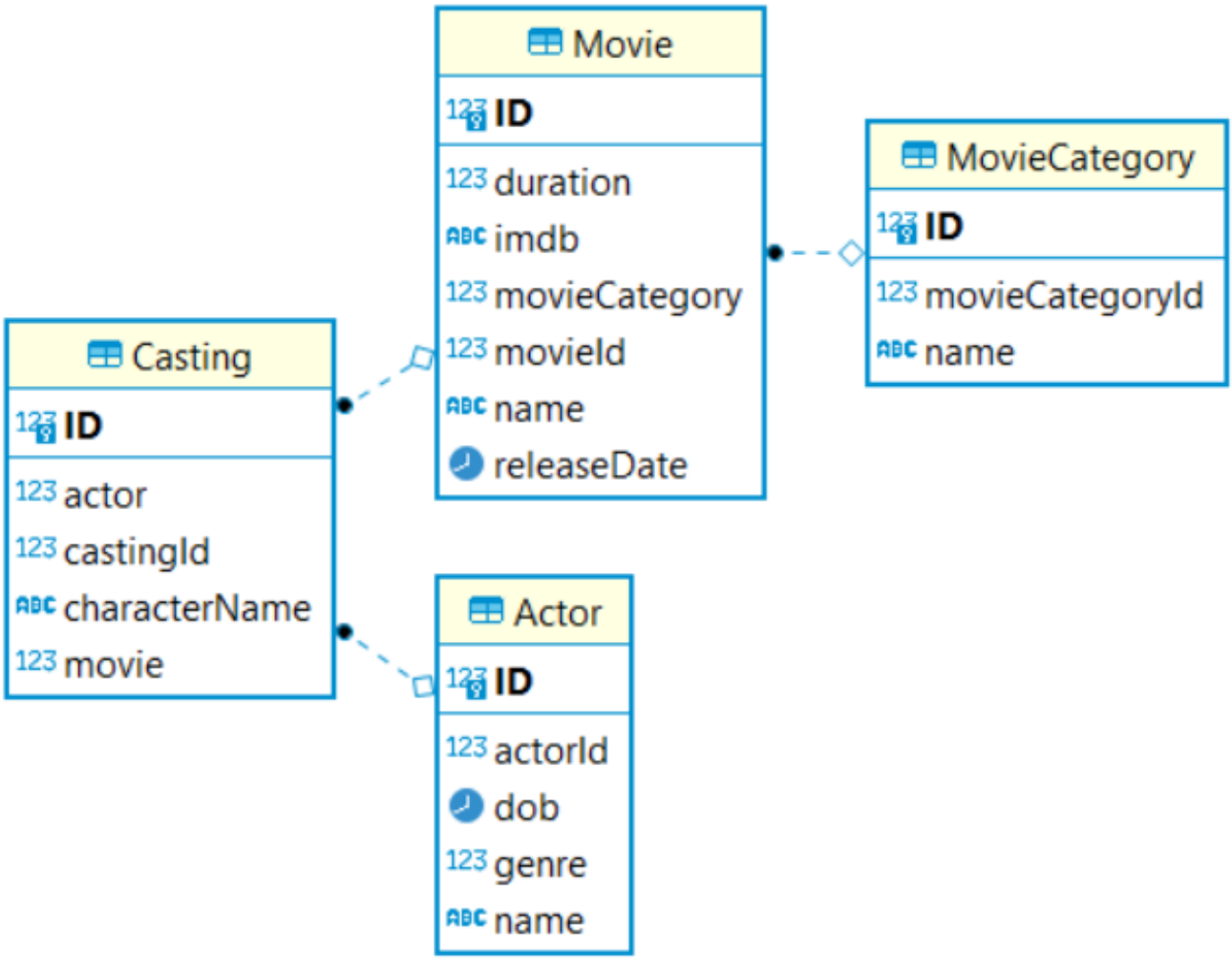


3. Check the connection status into ObjectScript Explorer (you will be able to see folders and classes created):



Persistent Classes to the Movie Catalog Application

In this section we will create the persistent domain classes to store and query the business data. See the DBeaver Diagram:



1. Create the folder model inside src\dc\movie folder.

2. Create the Actor.cls file inside model folder. Write the content:

```
Class dc.movie.model.Actor Extends (%Persistent, %JSON.Adaptor)
{
 Parameter %JSONREFERENCE = "ID";
 Property actorId As %Integer [ Calculated, SqlComputeCode = { set {*}={
%%ID}}, SqlComputed ];
Property name As %VarString(MAXLEN = 120); Property dob As %Date;
Property genre As %Integer(VALUELIST = ",1,2");
 }
```

3. Create the Movie.cls file inside model folder. Write the content:

```
Class dc.movie.model.Movie Extends (%Persistent, %JSON.Adaptor)
{
 Parameter %JSONREFERENCE = "ID";
 Property movieId As %Integer [ Calculated, SqlComputeCode = { set {*}={
%%ID}}, SqlComputed ];
Property name As %VarString(MAXLEN = 120);
```

```
Property releaseDate As %Date;
Property duration As %Integer;
Property imdb As %String(MAXLEN = 300);
Property movieCategory As dc.movie.model.MovieCategory;
ForeignKey MovieCategoryFK(movieCategory) References dc.movie.model.
MovieCategory();

}
```

4. Create the MovieCategory.cls file inside model folder. Write the content:

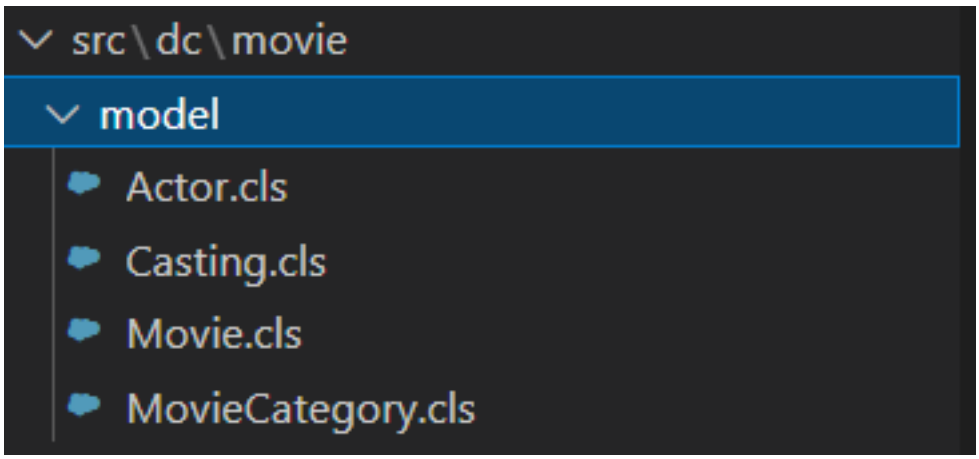
```
 Class dc.movie.model.MovieCategory Extends (%Persistent, %JSON.Adaptor)
{
  Parameter %JSONREFERENCE = "ID";
Property movieCategoryId As %Integer [ Calculated,
SqlComputeCode = { set {*}={%%ID}}, SqlComputed ];
Property name As %VarString(MAXLEN = 120);

}
```

5. Create the Casting.cls file inside model folder. Write the content:

```
 Class dc.movie.model.Casting Extends (%Persistent, %JSON.Adaptor)
{
  Parameter %JSONREFERENCE = "ID";
  Property castingId As %Integer [ Calculated, SqlComputeCode = { set {*}={
%%ID}}, SqlComputed ];
Property movie As dc.movie.model.Movie;
ForeignKey MovieFK(movie) References dc.movie.model.Movie();
Property actor As dc.movie.model.Actor;
ForeignKey ActorFK(actor) References dc.movie.model.Actor();
Property characterName As %String(MAXLEN = 100);
Index CastingIndex On (movie, actor) [ Unique ];
  }
```

See the created files:



Note 1: Parameter %JSONREFERENCE = "ID" allows return ID value inside JSON response.

Note 2: Property actorId As %Integer [Calculated, SqlComputeCode = { set {*}={%%ID}}, SqlComputed] and the other similar properties are used to return class+id into JSON response.

Note 3: (VALUELIST = "1,2") set possible values to 1 or 2 only.

Note 4: ForeignKey MovieFK(movie) References dc.movie.model.Movie() and similar are used to create a SQL foreign key reference.

Note 5: Index CastingIndex On (movie, actor) [Unique] and similar are used to not allows duplicate values combining properties in the On (movie and actor).

Note 6: I'm using Camel Case to property names because a best practice for JSON attribute names.

Business Service Classes to the Movie Catalog Application

In this section we will create the classes with business logic (methods to do persistence, query and calculations).

1. Create the service folder inside src\dc\movie.

2. Create CrudUtilService.cls file inside service folder. Write the content:

CrudUtilService content

3. Create MovieService.cls file inside service folder. Write the content:

MovieService content

4. Create MovieCategoryService.cls file inside service folder. Write the content:

MovieCategoryService content

5. Create ActorService.cls file inside service folder. Write the content:

ActorService content

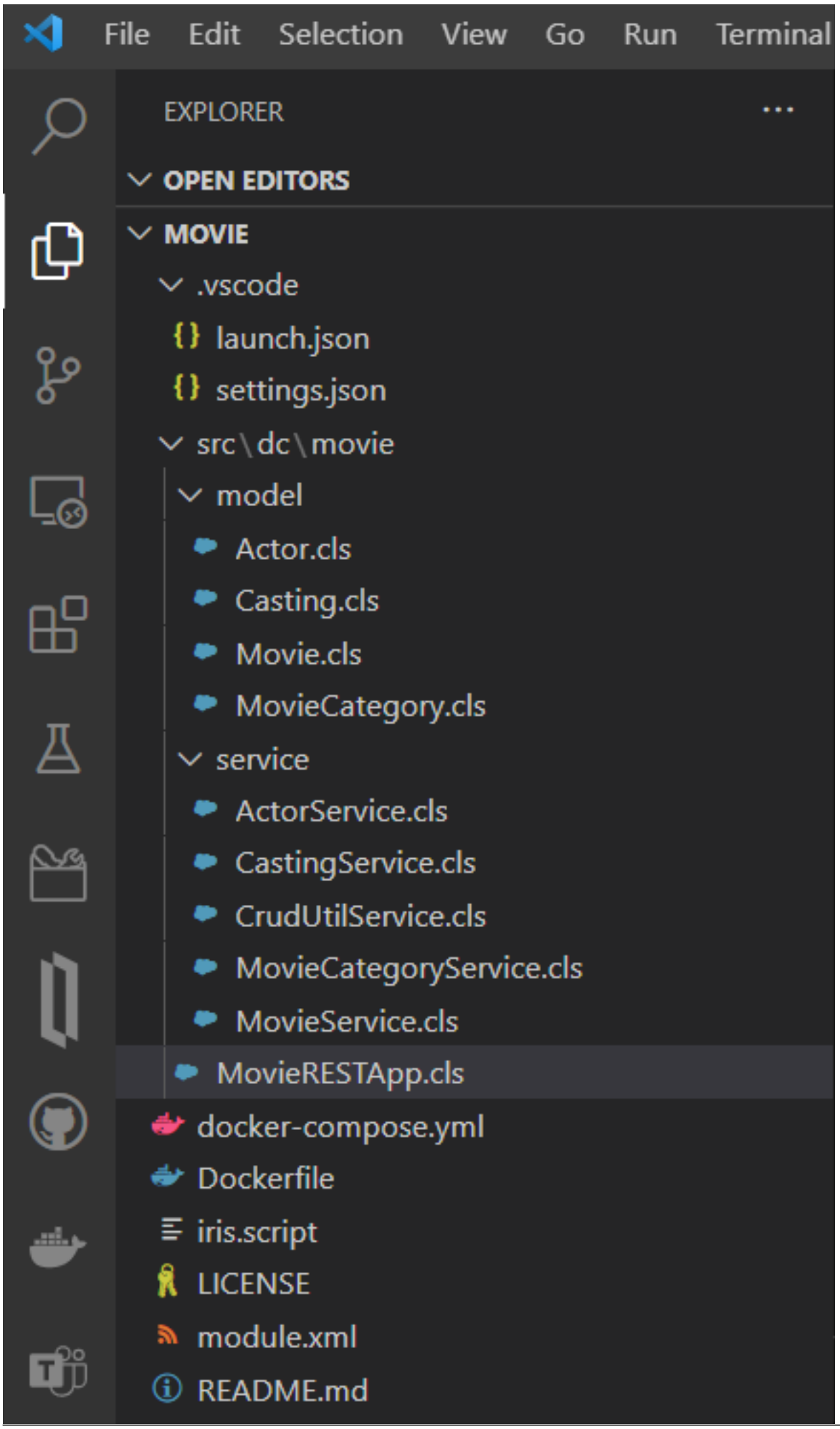
6. Create CastingService.cls file inside service folder. Write the content:

CastingService content

7. Update the file MovieRESTApp.cls to create paths to all new service class methods. Write the content:

MovieRESTApp updated content

8. The files and folders to the final project are:


File Edit Selection View Go Run Terminal
EXPLORER
OPEN EDITORS
MOVIE
.vscode
launch.json
settings.json
src\dc\movie
model
Actor.cls
Casting.cls
Movie.cls
MovieCategory.cls
service
ActorService.cls
CastingService.cls
CrudUtilService.cls
MovieCategoryService.cls
MovieService.cls
MovieRESTApp.cls
docker-compose.yml
Dockerfile
iris.script
LICENSE
module.xml
README.md

9. Test your new methods acessing http://localhost:52773/swagger-ui/index.html.

Note 1: REST paths are following business topic in plural with /id when we need pass id to the entity and camel case to paths to.

Note 2: We use verb GET to queries, POST to new records, PUT to update record and DELETE to delete a record.

Note 3: In <Route Url="/movies/casting/:id" Method="GET" Call="GetMovieCasting" /> I used /casting indicating a second purpose (get the movie and it casting). This method runs ToJSON(), because is a DynamicArray ([]) with Dynamic items ({}).

Note 4: I created the CrudUtilService class utility to do generic CRUD methods, following the Dont Repeat Yourself principle.

Enjoy this tutorial!

#REST API #Tutorial #InterSystems IRIS