Article
Dmitry Maslennikov · Oct 6, 2020 6m read

Open Exchange

RealWorld Application with InterSystems IRIS

Let's imagine if you would like to write some real web application, for instance, some simple clone of medium.com. Such sort of application can be written using any different language on the backend side, or with any framework on the frontend side. So many ways to do the same application, and you can look at this project. Which offers a bunch of frontends and backends realizations for exactly the same application. And you can easily mix them, any chosen frontend should work with any backend.

Let me introduce the same application realization for InterSystems IRIS on a backend side.

The RealWorld project uses REST and offers already prepared swagger specification, and Postman/Newman collection with tests. So, it helps to implement exactly the same REST API. And fortunately, InterSystems already implemented the way to generate REST API implementation by swagger specification. How to do it is best described here.

So my steps to implement this application were

- Generate API from swagger specification
- Add a few persistent classes, for every type of objects used in the application, and it's
  - Users
  - Articles
  - Comments
- Implement API, test it with Postman
- Finally, start with any frontend to see it in life.

Start with docker

You can try what came out of it.

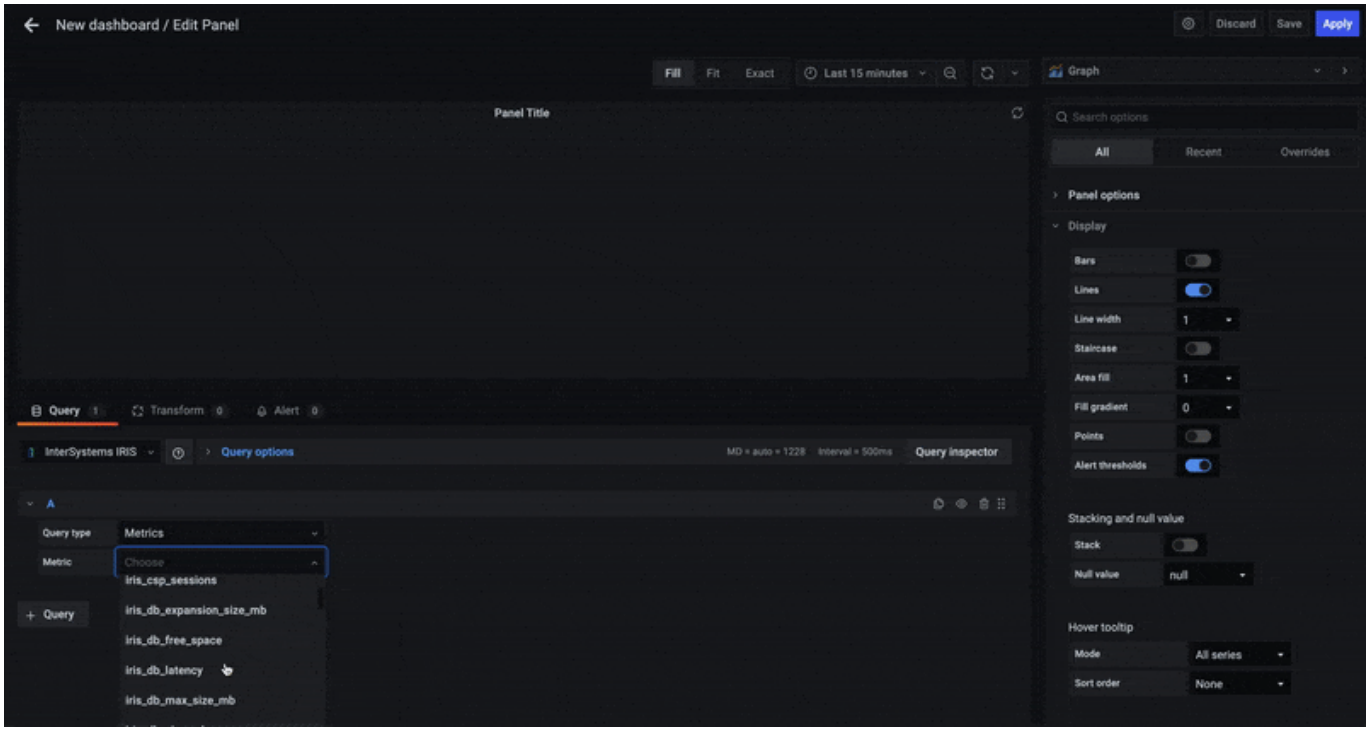
```
// clone github repository
git clone https://github.com/daimor/realworld-intersystems-iris.git
cd realworld-intersystems-iris

// build and run it with docker-compose
docker-compose up -d --build
```

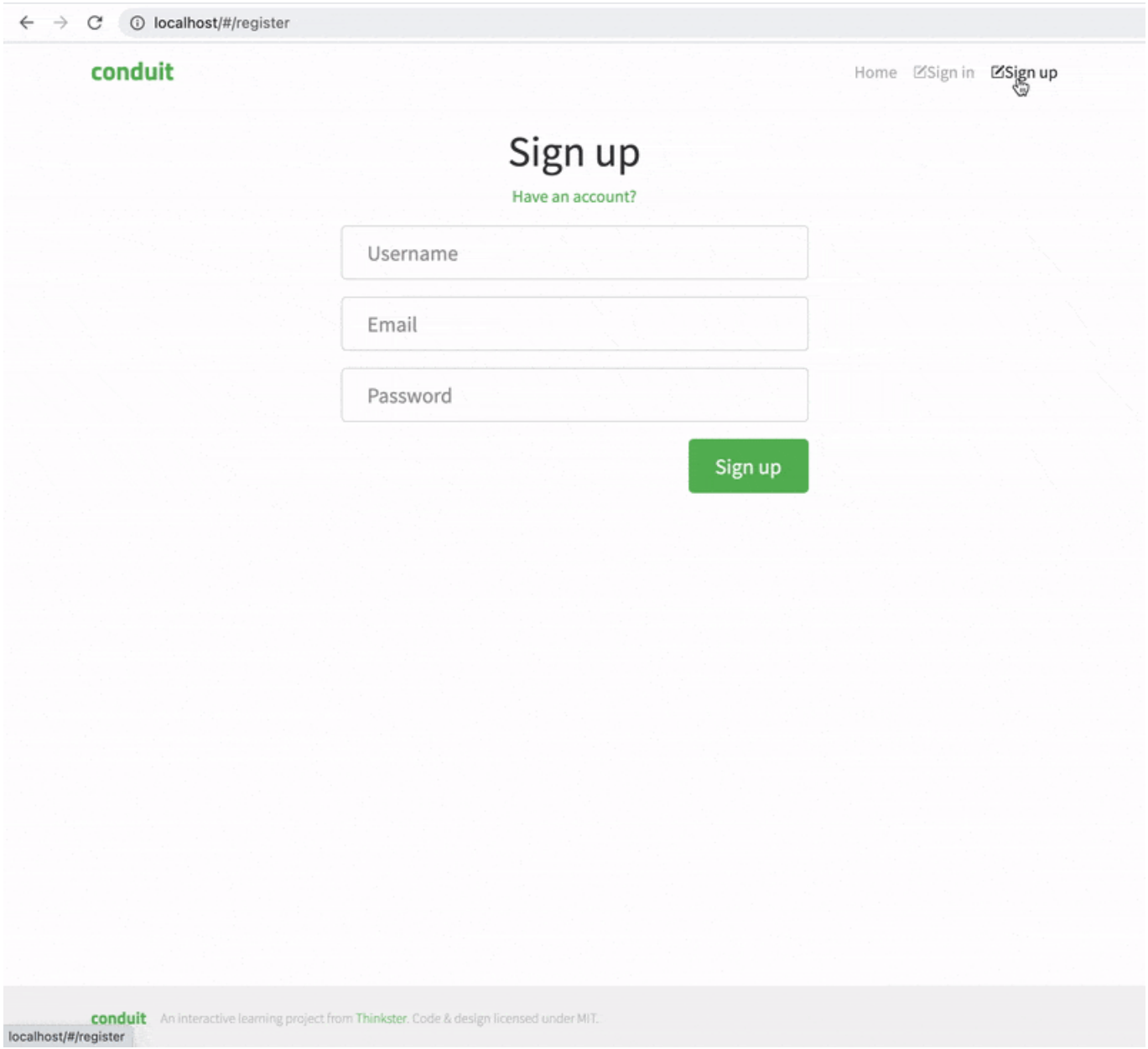
After start REST API in IRIS will be available by URL http://localhost:12000/conduit, and it can be tested with newman, you will need npm and npx package installed

```
APIURL=http://localhost:12000/conduit ./run-api-tests.sh
```

Same tests running Postman



Frontend is available by URL http://localhost/



UnitTests is available to run with zpm, just enter to iris session

```
$ docker-compose exec server iris session iris


Node: 0790684cf488, Instance: IRIS

CONDUIT>zpm
zpm: CONDUIT>test realworld [realworld]    Reload START
[realworld]    Reload SUCCESS
[realworld]    Module object refreshed.
[realworld]    Validate START
[realworld]    Validate SUCCESS
[realworld]    Compile START
[realworld]    Compile SUCCESS
[realworld]    Activate START
[realworld]    Configure START
[realworld]    Configure SUCCESS
[realworld]    Activate SUCCESS
```

```
[realworld]    Test START
Use the following URL to view the result:
http://172.22.0.3:52773/csp/sys/%25UnitTest.Portal.Indices.cls?Index=48&$NAMESPACE=CO
NDUIT
All PASSED
[realworld]    Test SUCCESS
zpm: CONDUIT>
```

By default, it goes with Vue frontend, but it's possible to run Angular and React

```
web=angular docker-compose up -d --build web

web=react docker-compose up -d --build web

web=vue docker-compose up -d --build web
```

Install with ZPM

InterSystems IRIS part (backend) can be installed with ZPM

```
USER>zpm
zpm: USER>install realworld [realworld]    Reload START
[realworld]    Reload SUCCESS
[realworld]    Module object refreshed.
[realworld]    Validate START
[realworld]    Validate SUCCESS
[realworld]    Compile START
[realworld]    Compile SUCCESS
[realworld]    Activate START
[realworld]    Configure START
[realworld]    Configure SUCCESS
[realworld]    Activate SUCCESS
zpm: USER>
```

And it will create /conduit`web application, so it should be able to be tested with newman as well, just set correct port

```
APIURL=http://localhost:52773/conduit ./run-api-tests.sh
```

And UnitTests can be run with ZPM

```
zpm: USER>test realworld [realworld]    Reload START
[realworld]    Reload SUCCESS
[realworld]    Module object refreshed.
[realworld]    Validate START
[realworld]    Validate SUCCESS
[realworld]    Compile START
[realworld]    Compile SUCCESS
[realworld]    Activate START
[realworld]    Configure START
[realworld]    Configure SUCCESS
[realworld]    Activate SUCCESS
[realworld]    Test START
```

```
Use the following URL to view the result:
http://172.17.0.2:52773/csp/sys/%25UnitTest.Portal.Indices.cls?Index=4&$NAMESPACE=USE
R
All PASSED
[realworld]    Test SUCCESS
```

Notes

There are a few issues I'm faced with during the development of this project.

- %JSON.Adaptor
  - It works quite well for importing entirely new objects. But, if you would need to partially update the existing object, %JSONImport will not work, for required fields, which it expects to see in a coming JSON.
    - So, instead of using %JSONImport for updating objects, I've used a simple set from incoming JSON to object if a value is defined.
  - Export available only to string, stream, and to the output device. Export to Native JSON not available.
    - API required to return any object wrapped by another object with a property named as a type of returned object. And solved it with using %JSONExportToString, and for arrays, converted to Native JSON
  - Ignores export for empty collection properties, such as array and list. While an application may expect to get even empty array for the field, but it does not get any field at all.
    - Did not solve it. Too tricky to solve it without solving it on %JSON.Adapter side.
- %REST - Generator for REST implementation, and REST implementation itself
  - Any compilation of ѝpec`class will update ѝmpl`class, even if no changes were there. So, it's important to keep generated parts, such as Method names, parameters list, and variable names, the same, or it will be overwritten with the next compile of ѝpec,` which may happen when the application will be built for production.
  - It's ok for REST to have endpoint like /ѝsers`and get requests as /ѝsers/`and in this case, it should act the same. But %CSP.REST does not recognize second if defined only the first way.
    - To solve this issue, had to change swagger specification, with just duplication of /ѝsers` with new endpoint /ѝsers/`
  - Swagger specification defines default values for parametrized requests, and generator just ignores it
    - Manually set default values, right in code of methods/ Generator may just overwrite methods definition, and setting default values to parameters will be removed. So, it may break implementation after deployment.
  - Methods from %REST.REST not available for overriding, it used by ðisp`class only, and will be completely overwritten by compilation of ѝpec`class.
    - No access to OnPreDispatch method for instance, so, no way to have some more control on, like to check access
  - Swagger specification defines which endpoint is public and which requires authorization. %REST generator just does not use it.
    - API requires to use JWT to authorize requests and had to manually check which endpoint needs to check access. Using JWT also quite tricky in IRIS, outside of %OAuth2 implementation.
  - Generated Methods in ѝmpl`class supposed to return Native JSON objects, streams, or string. But I think it would be good if it would accept %JSON.Adaptor objects as well.

Anyway, it was very interesting to implement such kind of application. And see how it is possible to do it with IRIS.

This application uses this list of features in IRIS.

- Native JSON + %JSON.Adaptor
- REST, and it's implementation generator by swagger specification

- JWT from OAuth2
- Containerization

Contest

This project is participating in InterSystems Full Stack Contest, please vote there if you like it.

#Angular #CaretDev #Contest #JSON #OAuth2 #React #REST API #Vue.js #InterSystems IRIS
Check the related application on InterSystems Open Exchange