Article
Rob Tweed · Oct 6, 2020 4m read

Open Exchange

qewd-conduit: Showcasing QEWD's Unique integration of Node.js/JavaScript and IRIS

Those of you who are following the FullStack competition here in the Developer Community will know that I submitted an entry named qewd-conduit. I wanted to summarise why I think it's something worth you taking a bit of time to check out.

qewd-conduit uses the Node.js-based QEWD framework alongside IRIS to implement the back-end REST APIs for something known as the RealWorld Conduit application:

https://github.com/gothinkster/realworld

This is a very cool initiative as it provides a platform for lots of different people to implement technically different solutions for both the back- and front-ends of a specified application. So, qewd-conduit is just one of a pretty large number of solutions that conform to the same REST API back-end specification. Similarly, you can try out any one of a large number of different front-end clients that implement the exact same UI/UX, and integrate via REST with any of the Conduit back-ends including qewd-conduit.

So it's a great way of being able to compare and contrast different frameworks and technologies, all of which, at the end of the day, do the exact same thing.

The RealWorld application is a nice balance between being relevant and non-trivial (ie it's not just another ToDo app!) and not too overwhelmingly complex, either in terms of UI/UX or back-end APIs. As such it's a great way of showcasing and illustrating how a particular technology can be used to implement the RealWorld's specified functionality.

So in the case of qewd-conduit, I'm showcasing how you can use QEWD as a Node.js framework for implementing, entirely in JavaScript, a set of REST APIs that maintain the RealWorld Conduit application's data in IRIS.

The really interesting and unusual things about QEWD are that not only is your access to IRIS from within Node.js/JavaScript via an extremely high-performance in-process connection, but also, as far as you are concerned, the IRIS database is abstracted as persistent JavaScript Objects. What does that mean? Well, I describe it as the JavaScript equivalent of the ObjectScript up-arrow which seamlessly and transparently differentiates in-memory and on-disk arrays. In the case of QEWD's abstraction (known as QEWD-JSdb), you effectively not only have the normal in-memory JavaScript objects, but also objects that happen to reside physically on disk within the IRIS database. So when you manipulate these QEWD-JSdb JavaScript objects, you're actually accessing and manipulating their properties directly within the IRIS database!

You can watch a video of a presentation I gave back in January on QEWD-JSdb to the London Node.js Users Group:

https://www.youtube.com/watch?v=1TlAKTw167s&list=PLam80-FY3vSPW9apMaczTN4dtke9GYM

One of my main incentives for putting together qewd-conduit was as a way of showcasing this QEWD-JSdb abstraction for a well-known and understood application and use-case - ie the RealWorld Conduit REST API and its associated data set.

With QEWD, then, you maintain and access data in an IRIS database entirely in JavaScript: qewd-conduit contains no ObjectScript code (and neither does any part of the QEWD stack), and all you need to know about IRIS is how to model and manipulate data using the QEWD-JSdb abstraction from within JavaScript. The high-speed, in-

process connection between Node.js and IRIS is provided by the Open Source mg-dbx interface (created and maintained by my colleague Chris Munt): this creates the required intimate relationship between JavaScript and the IRIS database, with a level of performance close to that of native ObjectScript.

If you're interested in finding out more about how I used QEWD-JSdb to model the data needed to support the REST APIs of the RealWorld Conduit application's back-end, I've included a document in the qewd-conduit repository:

https://github.com/robtweed/qewd-conduit/blob/master/QEWD-JSdb.md

If you then want to dig deeper, background information on QEWD-JSdb (including a detailed tutorial) can be found here:

https://github.com/robtweed/qewd-jsdb

For more information on the mg-dbx interface, see:

https://github.com/chrisemunt/mg-dbx

One final thing: qewd-conduit goes one step further than the RealWorld Conduit specification by also implementing those same RealWorld Conduit REST APIs as corresponding WebSocket messages - WebSocket support is seamlessly integrated into QEWD, and it made sense to expose the same back-end logic used for handling the REST APIs as WebSocket messages. That way you have a unique opportunity to compare and contrast the use and relative performance of REST versus WebSockets doing the exact same job! I'd like to thank Ward De Backer for putting together a WebSocket-enabled version of the Vue.js-based RealWorld client - see the main qewd-conduit documentation for more information. (Spoiler alert: you should find the WebSocket messages quite a bit faster than their REST API equivalents!)

So that's a quick bit of extra background on qewd-conduit. As you'll have hopefully begun to realise, it's just the tip of a very interesting technical iceberg!

Anyway. if you've found this interesting and useful, do please vote for qewd-conduit in the Fullstack competition!

Rob Tweed

#JavaScript #Node.js #REST API #InterSystems IRIS
Check the related application on InterSystems Open Exchange

Source URL:https://community.intersystems.com/post/qewd-conduit-showcasing-qewds-unique-integration-nodejsjavascript-and-iris