The app is a node app calling Watson Question and Answer REST service. its on github. Make sure you have git installed

git clone <https://github.com/iwinoto/qa-sample-node.git>

When run locally, its hard-wired to my instance of the Watson QA service so you can do:

npm install

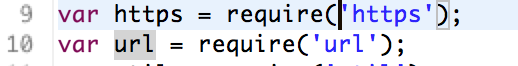
npm start

Then go to <http://localhost:3000/>

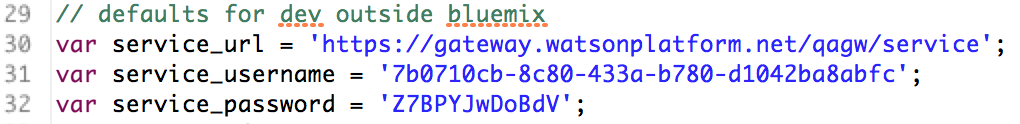
All the logic is in app.js. The view stuff is in views/ and public/. The app uses express for the HTTP serving & request routing and Jade templating engine to generate the HTML. Jade makes human readable files into HTML. The .jade template files are in views/.

Walking through the code:

To do the REST requests, I need some modules

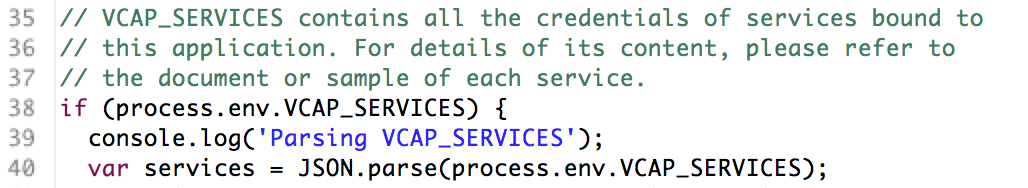


Set up default service credentials for running locally

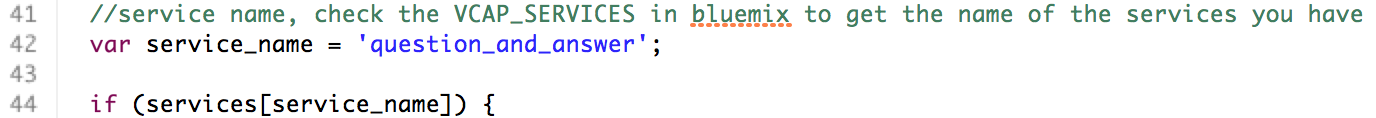


If we are running on Bluemix, then the service credentials will be in VCAP\_SERVICES environment. This block tests for the variable and extracts the service credentials.

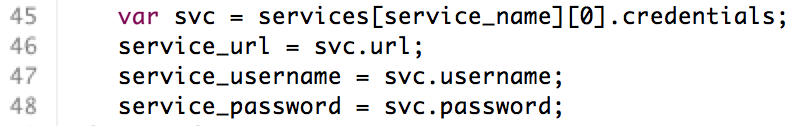
If it exists, get VCAP\_SERVICES json object into a variable.



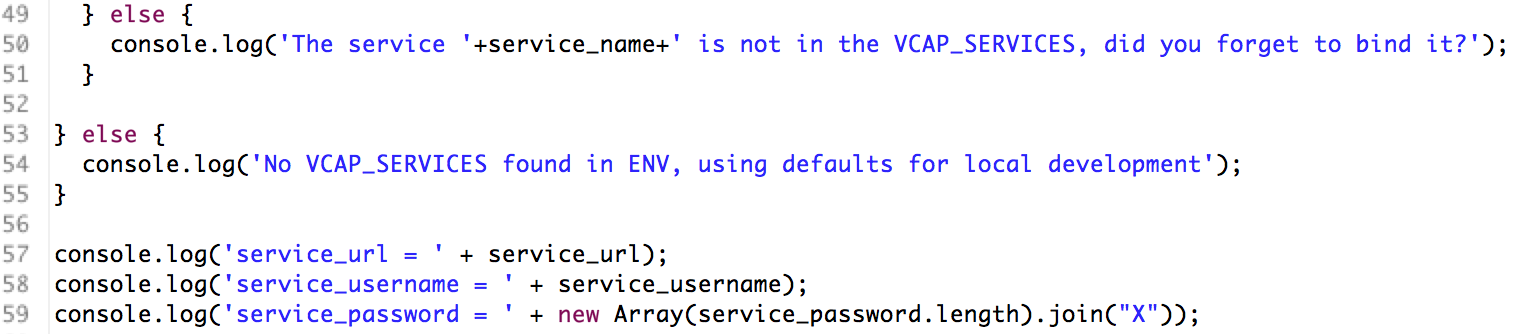
The service type is known as “question\_and\_answer”. We can find this from Bluemix. Look for a service with this name



If its found, then get the credentials and extract the relevant information. For the Watson Q&A, we want the URL, username and password. This is common, but for some services, like APIs from APImanagement, we want the client ID and key as well as the URL.



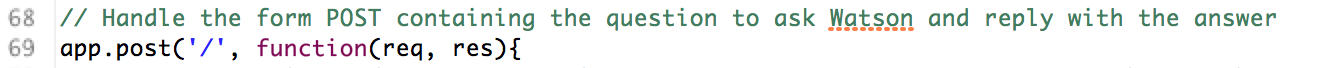
Log any problems



The Watson REST service wants to base64 encode the credentials. Store the encoded string for later use.

Macintosh HD:Users:iwinoto:Desktop:Screen Shot 2015-10-01 at 22.43.50.png

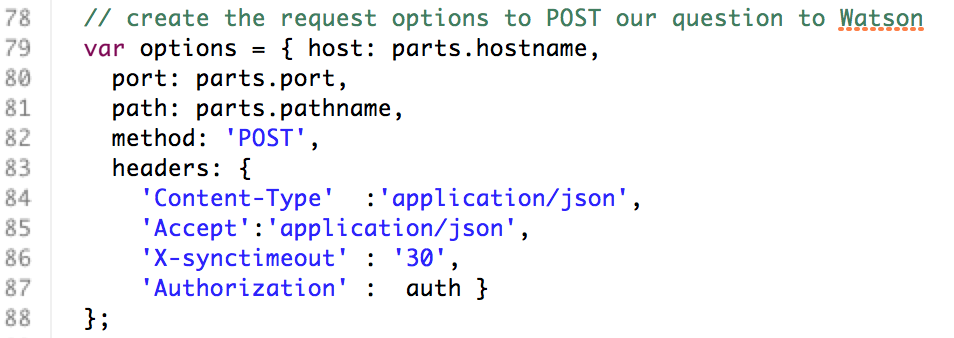
This next bit is an actual REST call to POST a question. It happens in the function to handle a POST request from the application web page.



First we get the URL (string) and parse it into its parts. This one is specific to Watson Q&A

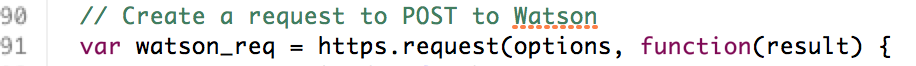
Macintosh HD:Users:iwinoto:Desktop:Screen Shot 2015-10-01 at 22.46.25.png

Create the json object to hold the HTTP request options



Now we get into call-back heaven. Node.js is non-blocking, so you have to code as asynchronous. This means, almost all methods will send a call back that you need to handle. In this case we first create the HTTPS request object that will handle the response from sending Watson a question and store this request object in a variable called “watson\_req”.

The call back handler for this request object start with the function declaration **function(result)** where **result** is the function parameter that will hold the object to deal with HTTPS results.



We expect the result to be UTF-8 encoded, so we make sure the result object knows this.

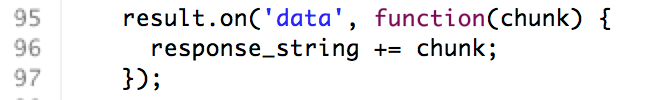
Macintosh HD:Users:iwinoto:Desktop:Screen Shot 2015-10-01 at 23.06.14.png

We also create and empty variable to hold the results when it arrives

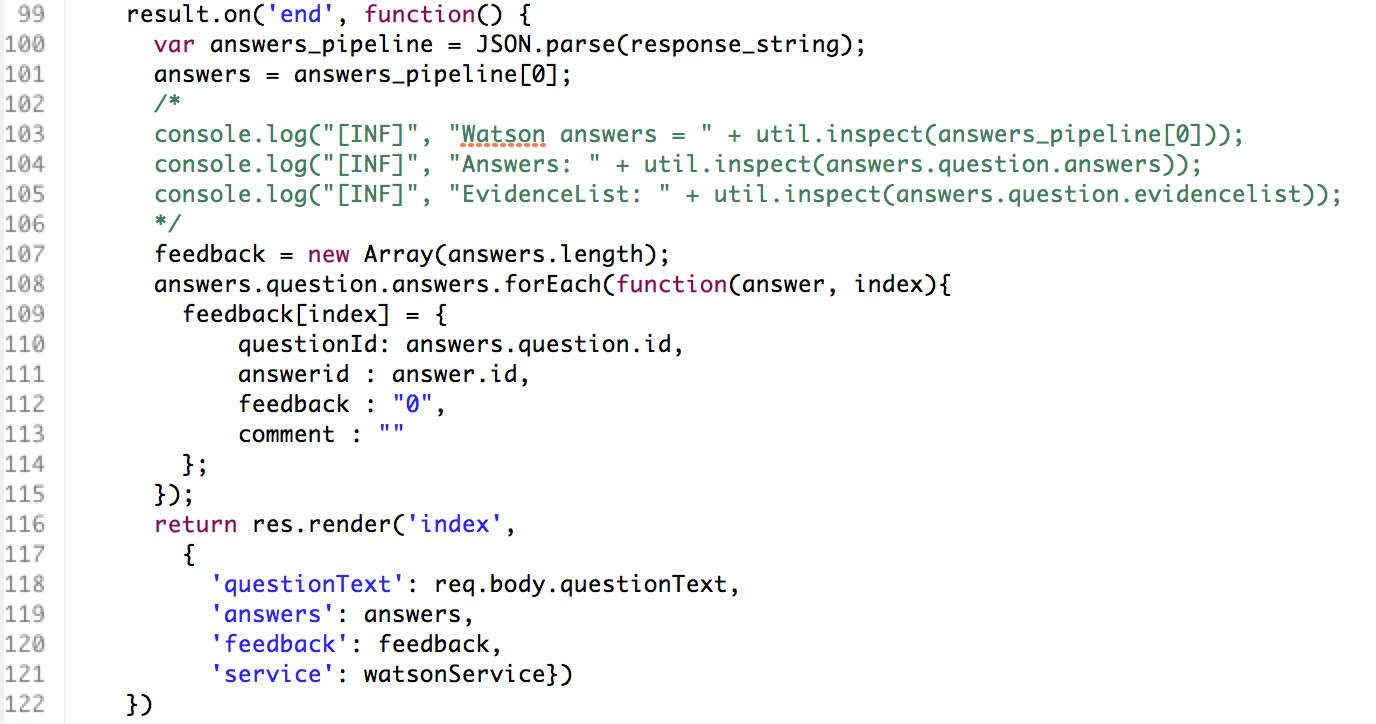
Macintosh HD:Users:iwinoto:Desktop:Screen Shot 2015-10-01 at 23.07.07.png

The **result** object will emit various events some of which we are interested in. Using the **on** method, we add some event handler methods to the **result** object for the events we are interested in

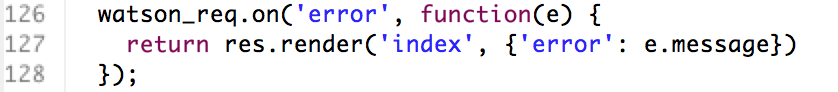
‘**data’**: Data comes in chunks. Each time a chunk of data is received, the **data** event is emitted. We need to get the data and add it to our result string:



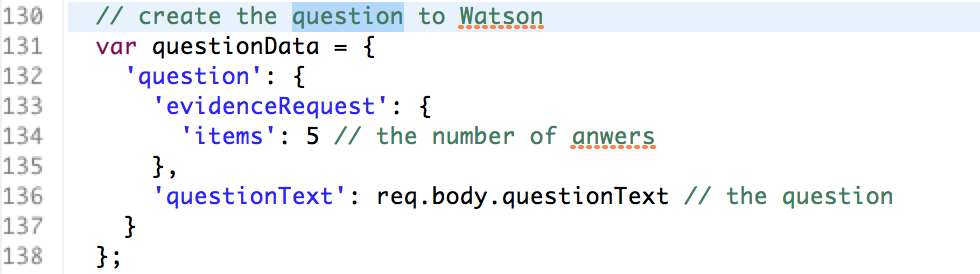
‘**end’**: At the end of the data stream, the **end** event is emitted. Now we know we’ve got the complete result, so we can parse it and display it. How you handle this depends on how your API responds. In most cases its JSON, so you can parse the string into a JSON object as we do here. This one deals with a JSON array of answers to the question that we’re going to send.



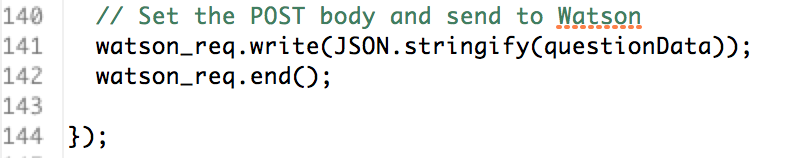
‘**error**’: we always need to handle the error



Now that we’ve told **result** how to handle events, we can send the actual request with the payload. In this case, the payload is a JSON object containing the question and some meta information.



Now we can write the payload to the prepared HTTPS request object and end the request. The call to **end()** is important, otherwise the request object will just wait and not close the request.



The rest of the code is more of the same.