Personality Insights

Follow instructions in this section to augment the application by exposing a REST API `/api/pi`. This API only accepts HTTP POST method with a payload. It uses [Node.js Watson Cloud SDK (https://github.com/watson-developer-cloud/node-sdk)](https://github.com/watson-developer-cloud/node-sdk) to invoke IBM Bluemix Personality Insights service. [API documentation](#) provided at the end.

In order to test the API, [Advance REST Client (https://advancedrestclient.com/)](https://advancedrestclient.com/) is used. It is a Chrome extension installed in student VM.

```
NOTE: If you are using your own computer to work through this tutorial, use API test client
of your choice. The tutorial assumes that you have Advance REST Client installed in your
computer.
```

Update Watson Helper Module

- Launch VS Code text editor and open the directory where you extracted the code in [Setup (02-text-to-speech/text-to-speech.md)](#) section.
- In [previous (02-text-to-speech/text-to-speech.md)](#) section, a new helper module `watson.js` was created. Here, lets modify this helper module and add new methods to invoke IBM Watson Personality Insights service.
- Open `watson.js` in VS Code text editor.
- Locate `Module Exports` comment.

  ```
  //
  // Module Exports
  //
  ```

- Paste the following code right above the `Module Exports` comment. Make sure to replace `YOUR_PI_SVC_NAME` reference with the name of Personality Insights Service instance in Bluemix (it is the full name with random characters suffixed). You can also locate the correct service name inside `manifest.yml` file located in the root of your app. Finally, save the file. We will go over the code in next step.

```
// Personality Insights setup
var piService = appEnv.getService("YOUR_PI_SVC_NAME")
var piClient = watson.personality_insights({
username: piService.credentials.username,
password: piService.credentials.password,
version: 'v2'
});

// Private method for extracting Big5 personality attributes from
// IBM Watson Personality Insights API response.
var extractBig5 = function(data) {
    var characteristicsArr = data.tree.children;

    var big5Arr = [];
    for (var i = 0; i<characteristicsArr.length; i++) {
        if (characteristicsArr[i].id === "personality") {
        big5Arr = characteristicsArr[i].children[0].children;
        }
    }

    // remove the children attribute from big 5 array
    for (var i = 0; i<big5Arr.length; i++) {
        delete big5Arr[i].children;
    }

    return big5Arr;
}
```

- Lets go over the code.
    - Using the watson (https://github.com/watson-developer-cloud/node-sdk) variable, Personality Insights client `piClient` is created.
    - Next, a private method `extractBig5` is implemented. As the name suggests, this method extracts Big 5 Personality attributes from IBM Watson Personality Insights service response. The Big 5 traits are `Openness`, `Conscientiousness`, `Extraversion`, `Agreeableness`, and `Neuroticism`. The Personality Insights service returns a lot of additional personality attributes but to keep this tutorial simple, we manipulate the service response and extract only certain traits.
- Now lets implement a public method `getPersonalityInsights` in this module that will use the `piClient` created above and invoke IBM Watson Personality Insights service.
- Locate `module.exports = {` line in `watson.js`
- Paste the following code right after the `module.exports = {` line. Do not forget the trailing `,` at the end of this code. A comma is needed because there are now two public methods in `watson.js` helper module, i.e. `getPersonalityInsights` and `getSynthesizeSpeech`. Finally, save the file. We will go over the code in next step.

```
// invoke Personality Insight to retrieve Big 5 personality traits
getPersonalityInsights: function (textToAnalyze) {
  var deferred = q.defer();

  console.log("invoke personality insight API profile");

  piClient.profile({
      "text": textToAnalyze,
      "language": "en"
    },
    function (err, response) {
      if (err) {
        console.log("personality insight error:", err);
        deferred.reject(err);
      }
      else {
        console.log("personality insight response: word_count,
word_count_message", response.word_count, response.word_count_message);
        // modify the response from watson by removing redundant fields
and
        // simplify the response sent to caller
        var result = {
          "word_count": response.word_count,
          "word_count_message": response.word_count_message,
          "big_5": extractBig5(response)
        }
        deferred.resolve(result);
      }
    }
  );
  return deferred.promise;
},
```

- Lets review the code.
    - [Watson Personality Insights Node API (https://www.ibm.com/watson/developercloud/personality-insights/api/v2/)](https://www.ibm.com/watson/developercloud/personality-insights/api/v2/) exposes `profile` method. This method accepts a set of parameters; specifically parameters include text to analyse, and the language text represents.
    `profile` method returns a JSON response containing personality attributes including the big5 attributes.
    - When a successful response is received from `profile` method, a `result` JSON structure is created. The big5 attributes are extracted using `extractBig5` private method discussed earlier.
    - Finally the response, containing the `result` JSON object, is returned to the caller.

Implement API

- Using VS Code, create a new file in routes folder and call it `insights.js`
- Paste the following code into `insights.js` file and save the file. We will discuss the code in next step.

```
'use strict';

//
// Module dependencies
//
var express = require('express'),
router = express.Router(),
bodyParser = require("body-parser"),
watson = require('../helpers/watson');

//
// Private Variables & Functions
//
console.log("setting up insights module");

router.route('/')
.post(
    // handle JSON body
    bodyParser.json(),
    // handler
    function (req, res) {
    console.log(">> personality insight api invoked");

    // get a reference to the json in request body
    var reqBodyJSON = req.body;
    console.log("text:", reqBodyJSON.text);

    // perform payload validation
    if (!reqBodyJSON.text || typeof reqBodyJSON.text == "undefined") {
        res.status(500).send('Invalid JSON request.  Missing required
fields');
        return;
    }

    watson.getPersonalityInsights(reqBodyJSON.text).then(
        // success handler
        function(result) {
        res.json(result);
        },
        // error handler
        function(er) {
        res.status(404).send(er);
        }
    );

    }
);
```

```
//
// Module Exports
//
module.exports = router;
```

- Lets review the code.
    - First, the code imports required node modules, `express`, `body-parser` and `watson`
      helper created in previous section.
    - Next, using `express.Router()`, the code creates a route `/` that accepts HTTP `POST`
      method.
    - In order to parse the POST body, `bodyParser.json()` is used such that `req.body` contains a JSON object.
    - Next, request validation is done to ensure that the payload contains a required JSON string property, `text`. If this property is missing, `500` error code is returned and `watson` helper module is not invoked.
    - When request validation is successful, the code invokes `watson` helper module to retrieve the big 5 attributes and send it back to the caller.
    - If an error occurs during invocation, `404` error code is returned with
      a message explaining the cause. Note that the error from `watson` helper module is sent as is to the caller. Error handling can be further refined as needed.

Create API Route

- Using VS Code, open *app.js*.
- Locate the following code:

```
app.use("/api/t2s", require("./routes/text2speech"));
```

- Add a new route right after the code above:

```
app.use("/api/pi", require("./routes/insights.js"));
```

- The code above creates a new route `/api/pi`. Node `route/insights`
  module is setup to handle the implementation of this API route.
- Save the file.

Publish app to Bluemix

- Open Terminal window.
- Change directory to the folder where you extracted the source code. Replace `<email id>` with
  your email id.

```
cd <email id>-cognitive-api1
```

- Type the following command to publish application updates to Bluemix.

```
cf push
```

- You should see the following output when `cf push` command is successful.

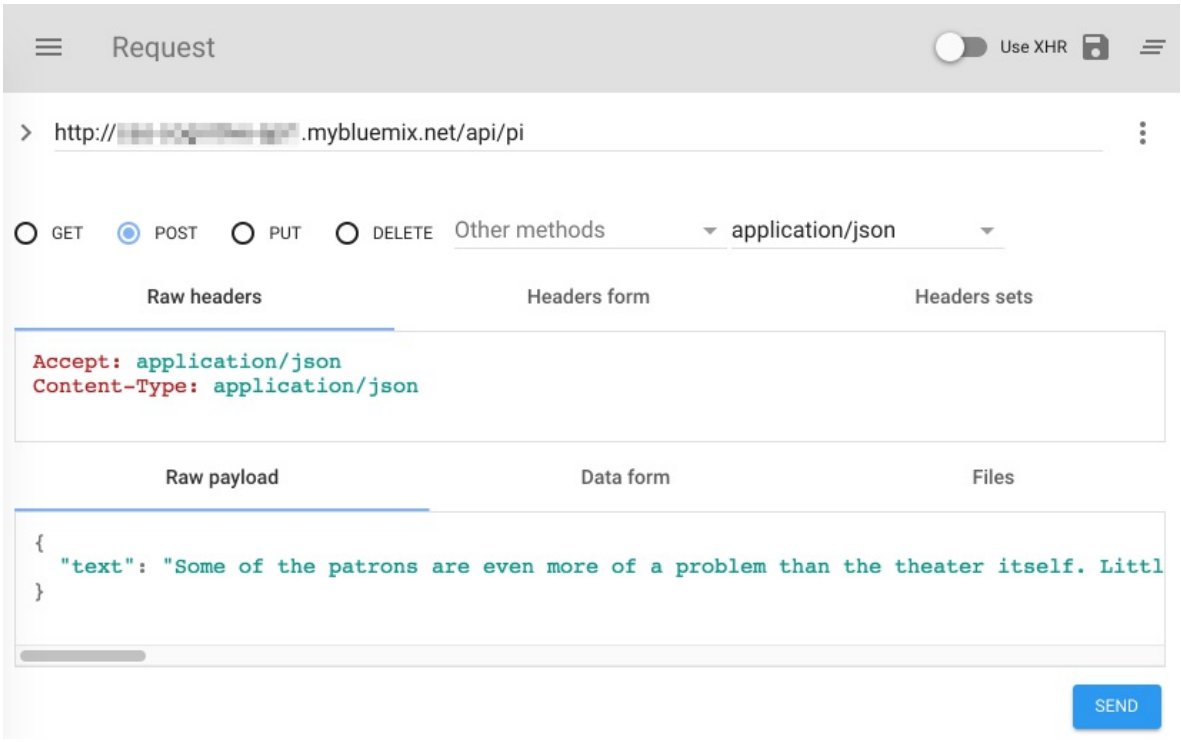
```
App YOUR_APP_NAME was started using this command
`./vendor/initial_startup.rb`

Showing health and status for app YOUR_APP_NAME in org YOUR_ORG / space
cascon as YOUR_ORG...
OK

requested state: started
instances: 1/1
usage: 256M x 1 instances
urls: YOUR_APP_NAME.mybluemix.net
last uploaded: Wed Oct 19 18:15:52 UTC 2016
stack: unknown
buildpack: SDK for Node.js(TM) (ibm-node.js-4.6.0, buildpack-v3.8-
20161006-1211)

    state     since                    cpu    memory           disk
details
#0   running   2016-10-19 02:16:51 PM   0.0%   100.4M of 256M   96.3M of
1G
```

Test API

We will use both Advance REST Client and cURL command line to test the API.

- Launch Advance REST Client.
- Complete the Advance REST Client screen as shown below. Note that the URL should `http://<email id>-cognitive-api1.mybluemix.net/api/pi`. Replace `<email id>` with
  your email id. A sample short story is provided below. Feel free to use your
  own text instead for analysis.

```
Once when a lion, the king of the jungle, was asleep, a little mouse
began
running up and down on him. This soon awakened the lion, who placed his
huge paw on the mouse, and opened his big jaws to swallow him.  Pardon,
O King! cried the little mouse. Forgive me this time. I shall never
repeat
it and I shall never forget your kindness. And who knows, I may be able
to
do you a good turn one of these days!  The lion was so tickled by the
idea
of the mouse being able to help him that he lifted his paw and let him
go.
Sometime later, a few hunters captured the lion, and tied him to a tree.
After that they went in search of a wagon, to take him to the zoo.  Just
then
the little mouse happened to pass by. On seeing the lion's plight, he ran
up to him and gnawed away the ropes that bound him, the king of the
jungle.
Was I not right? said the little mouse, very happy to help the lion.
```



- Verify that the response big_5 attributes.

```
{
    "big_5": [ {....} ]
}
```

Next, lets use cURL to invoke the Personality Insights API:

- Open Terminal window.

- Change directory to the folder where you extracted the source code. Replace <email id> with
your email id.

```
cd <email id>-cognitive-api1
```

- Type the following cURL command to invoke the API. As before, use
the correct application URL
http://<email id>-cognitive-api1.mybluemix.net/api/pi. Replace <email id>
with your email id. Also ensure to replace TYPE_100_WORDS with
at least 100 words.

```
curl -H "Accept: application/json" -H "Content-Type: application/json" \
     -X POST "http://<email id>-cognitive-api2.mybluemix.net/api/pi" \
    -d '{"text": "TYPE_100_WORDS"}'
```

Note that a sample JSON request is provided in solutions folder,
03-personality-insights/sample-request.json. Instead of typing hundred words on
command
line, which can get tedious, you may use the following cURL command that reads the
sample-request.json.

```
curl -H "Accept: application/json" -H "Content-Type: application/json" \
     -X POST "http://<email id>-cognitive-api2.mybluemix.net/api/pi" \
    -d @~/workshop/solutions/03-personality-insights/sample-request.json
```

- Verify that the response contains big_5 attributes.

Troubleshooting

- In order to view the application logs in Bluemix, type the following cf command.
Replace <email id> with your email id.

```
cf logs <email id>-cognitive-api1
```

- This starts a running tail on the application log. All API activity is logged.

API Documentation

Path

```
http://<email id>-cognitive-api1.mybluemix.net/api/pi
```

HTTP Method

```
POST
```

Headers

```
Content-Type: application/json
Accept: application/json
```

Body

Payload:

```
{
  "text": "Some free form text.  There should be at least 100+ words"
}
```

Following fields are required in the payload.

- `text`

Response

```
{
    "word_count": 139
    "word_count_message": "There were 139 words in the input. We need a minimum
of 600, preferably 1,200 or more, to compute statistically significant
estimates"
    "big_5": [
        {
            "id": "Openness"
            "name": "Openness"
            "category": "personality"
            "percentage": 0.9707150616109989
            "sampling_error": 0.0653507898
        },
        {
            "id": "Conscientiousness"
            "name": "Conscientiousness"
            "category": "personality"
            "percentage": 0.05379315899528442
            "sampling_error": 0.0816390731
        },
        {
            "id": "Extraversion"
            "name": "Extraversion"
            "category": "personality"
            "percentage": 0.2895550725189394
            "sampling_error": 0.061139683300000004
        },
        {
            "id": "Agreeableness"
            "name": "Agreeableness"
            "category": "personality"
            "percentage": 0.0003545380928280939
            "sampling_error": 0.10198415
        },
        {
            "id": "Neuroticism"
            "name": "Emotional range"
            "category": "personality"
            "percentage": 0.15906236917449357
            "sampling_error": 0.0965721185
        }
    ]
}
```


Error Response

Following HTTP response codes are returned when an error occurs:

- **404** – Error while invoking IBM Bluemix Watson API
- **500** – Missing required payload parameters