Text to Speech

Follow instructions in this section to augment the application by exposing a REST API `/api/t2s`. 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)
to invoke IBM Bluemix Text to Speech service. API documentation provided
at the end.

In order to test the API, Advance REST Client (https://advancedrestclient.com/) is used. It is a Chrome extension
installed in student VM. In addition, cURL command utility is also used to capture the response from API and process it further.

```
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 and cURL command line
installed in your computer.
```

Create 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.
- Create a new file in `helpers` folder and call it `watson.js`

  ```
  NOTE: In VS Code, in order to create a file or folder, first highlight the directory
  where you want to create the file or folder, then click on the new file or folder
  icon in VS Code file explorer.
  ```

- Paste the following code into `watson.js` file. Make sure to replace `YOUR_TEXT_TO_SPEECH_SVC_NAME` reference with the name of Text to Speech 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.

  ```
  'use strict';

  ////
  // Text to Speech Watson API:
  https://www.ibm.com/watson/developercloud/text-to-speech/api/v1/?node#synthesize audio
  // Personality Insights Watson API:
  https://www.ibm.com/watson/developercloud/personality-insights/api/v2/

  //
  ```

```
// Module dependencies
//
var watson = require('watson-developer-cloud'),
  q = require("q"),
  cfenv = require('cfenv');

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

var appEnv = cfenv.getAppEnv();

// Text to speech setup
var t2sService = appEnv.getService("YOUR_TEXT_TO_SPEECH_SVC_NAME");
var t2sClient = watson.text_to_speech({
  username: t2sService.credentials.username,
  password: t2sService.credentials.password,
  version: 'v1'
});

//
// Module Exports
//
module.exports = {

  getSynthesizeSpeech: function (text, voice) {
    var deferred = q.defer();

      // Sample voices:
      // en-US_AllisonVoice
      // en-US_MichaelVoice
      // en-US_LisaVoice
      // en-GB_KateVoice
      // es-ES_EnriqueVoice
      // fr-FR_ReneeVoice
      var params = {
        "text": text,
        "voice": voice,
        "accept": "audio/ogg;codecs=opus"
      };

      var transcript = t2sClient.synthesize(params);

      // stream returned 'response'
      transcript.on('response', function (response) {
        console.log("text to speech response status:",
response.statusCode);
      });
      // stream returned 'error'
      transcript.on('error', function (error) {
        console.log("text to speech error:", JSON.stringify(error));
```

```
        deferred.reject(error);
      });

      // stream returned 'data'

      // convert binary voice data into base64
      var buffers = [];
      transcript.on('data', function(buffer) {
        buffers.push(buffer);
      });

      // stream returned ended
      transcript.on('end', function() {
        var buffer = Buffer.concat(buffers);
        var base64Audio = buffer.toString('base64');
        console.log("stream completed.  returning base64Audio.  length:",
base64Audio.length);

        var result = {
          "base64_audio": base64Audio
        };
        deferred.resolve(result);
      });

    return deferred.promise;
  }

};
```

- Lets go over the code.
    - First the code imports node modules `watson-developer-cloud`, `q` and `cfenv` and assigns to appropriate variables for later use.
    - `cfenv` is a handy node module to access various features of a cloud foundary application. In this case, `cfenv` is used to look up Text to Speech service bound to the application during Setup. In specific, the code accesses Text to Speech credentials that were automatically generated during service instance creation.
    - Using the [watson (https://github.com/watson-developer-cloud/node-sdk)](https://github.com/watson-developer-cloud/node-sdk) variable, Text To Speech client `t2sClient` is created.
    - Finally, a new public method `getSynthesizeSpeech` is created. This method will be used in the next section.
    - [Watson Text to Speech Node API (https://www.ibm.com/watson/developercloud/text-to-speech/api/v1/)](https://www.ibm.com/watson/developercloud/text-to-speech/api/v1/) exposes `synthesize` method. This method accepts a set of parameters; specifically parameters include text to translate into audio, the type of human voice to use and the audio format of the returned voice. `synthesize` method returns [Node readable stream (https://nodejs.org/api/stream.html#stream_readable_streams)](https://nodejs.org/api/stream.html#stream_readable_streams), which in turns

return binary audio data. In order to read the binary data, the code listens for four stream events, i.e. `response`, `error`, `data` and `end`. For each event, the code performs various steps to properly capture the binary data. The `data` event is raised when the stream has a chunk (or byte array buffer) of audio data available.
Each chunk (or byte array) of data is captured into an array of buffers during `data` event. Finally, when the stream raises `end` event, code concatenates all buffer using native JavaScript Buffer object and converts it into Base64 string.

- The final Base64 string is passed back to the caller using following JSON structure:

```
{
  "base64_audio" : "......"
}
```

Implement API

- Using VS Code, create a new file in `routes` folder and call it `text2speech.js`
- Paste the following code into `text2speech.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 text2speech module");

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

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

      // perform payload validation
```

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

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

    }
  );

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

- Lets go over 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 two required JSON string properties, `text` and `voice`. If these properties are missing, `500` error code is returned and `watson` helper module is not invoked.
    - When request validation is successful, the code invokes `watson` helper module, created in earlier section, to get the synthesized Base64 audio 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*.
- Overwrite *app.js* code with the contents below and save the file. We will discuss the code in next step.

```
/*eslint-env node*/

//
================================================================================
====
// BASE SETUP
//
================================================================================
====

// import modules
var express = require("express");
var app = express();
var cors = require("cors");

// cfenv provides access to your Cloud Foundry environment
// for more info, see: https://www.npmjs.com/package/cfenv
var cfenv = require('cfenv');

// serve the files out of ./public as our main files
app.use(express.static(__dirname + '/public'));

// enable cross-origin resource sharing (CORS)
// this will allow a web application hosted on a different host/domain
name
// to invoke our APIs
app.use(cors({
  "origin": "*",
  "methods": ["GET", "POST", "DELETE", "OPTIONS", "PUT"],
  "allowHeaders": ["Content-Type", "Authorization"]
}));

//
================================================================================
====
// ROUTES FOR OUR API
//
================================================================================
====

// get an instance of the express Router
var router = express.Router();

// REGISTER OUR ROUTES - all of our routes will be prefixed with /api
app.use("/api/t2s", require("./routes/text2speech"));

// REGISTER OUR ROUTES - all of our routes will be prefixed with /api
```

```
app.use("/api", router);

//
================================================================================
====
// START THE SERVER
//
================================================================================
====

// get the app environment from Cloud Foundry
var appEnv = cfenv.getAppEnv();

// start server on the specified port and binding host
var server = app.listen(appEnv.port, '0.0.0.0', function() {
  // print a message when the server starts listening
  console.log("server starting on " + appEnv.url);
});

//
================================================================================
====
// EXPORT THE SERVER
//
================================================================================
====

// export 'server' so that we can test it and invoke 'server.close()'
// 'app' object does not expose .close() method.  Only app.listen()
returns
// a handle for .close().  It is crutial to invoke server.close() after
// gulp test execution is done to properly terminate gulp run.
exports = module.exports = server;
```

- Lets go over the code.
    - First, the code imports required node modules, `express`, `cfenv` and invokes `express()` to create application instance.
    - Next, a static route is created to serve HTML pages from `/public` folder. If this line is commented, `public/index.html` will no longer be accessible using web browser.
    - CORS support is added next. This will inject a `Access-Control-Allow-Origin: *` header in the response from APIs. It allows application clients, especially web applications running in a browser, to successfully invoke the API. If this header is not present in API response, the browser will block the call.
    - API routes are created next. In specific, a new route `/api/t2s` is created and `route/text2speech` is set to handle the implementation of this API route.
    - Finally, `app.listen()` starts the express server.

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
```

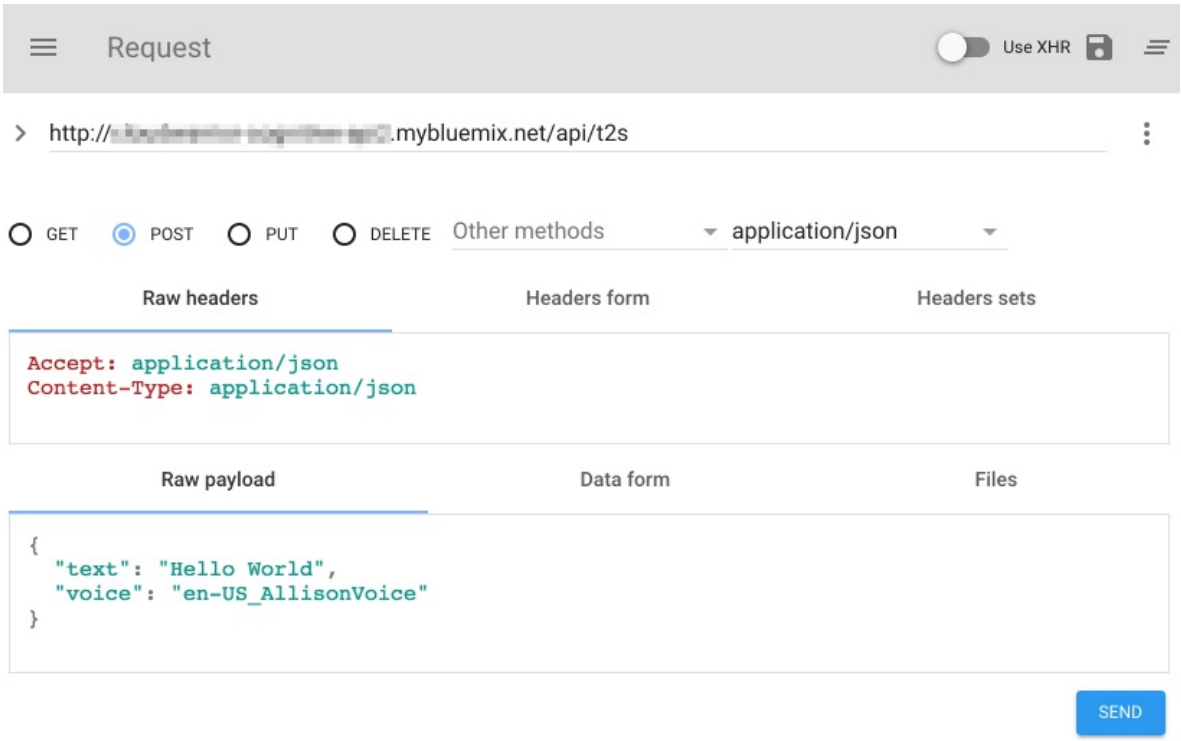
- You should see the following output when cf push command is successful.

```
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/t2s. Replace <email id> with
your email id.



- Verify that the response contains Base64 audio.

```
{
    "base64_audio": "....."
}
```

Next, lets try to decode this audio and listen to the synthesize voice.

- 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. Note that we will redirect
  the output from the command into a text file. As before, use the correct application
  URL http://<email id>-cognitive-api1.mybluemix.net/api/t2s. Replace <email id>
  with your email id.

```
curl -H "Accept: application/json" -H "Content-Type: application/json" \
     -X POST "http://<email id>-cognitive-api1.mybluemix.net/api/t2s" \
    -d '{"text": "Hello World", "voice":"en-US_AllisonVoice"}' >
audio.ogg.json
```

- Next, lets unwrap the JSON contents from the text file such that it only contains
  base64 string that we can process to binary audio

```
cat audio.ogg.json | \
    sed 's/{//g' | sed 's/}//g' | sed 's/"//g' | \
    awk -F ':' '{print $2}' > audio.ogg.txt
```

- Finally, lets decode the Base64 to binary audio

```
base64 --decode audio.ogg.txt > audio.ogg
```

- Double click audio.ogg file and it will open in Chrome browser. Listen to the synthesized audio.
- Clean up the audio files from project folder as we don't want to publish these to Bluemix.

```
rm audio.*
```

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/t2s
```

HTTP Method

```
POST
```

Headers

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

Body

Payload:

```
{
  "message": "Hello World",
  "voice": "en-US_AllisonVoice"
}
```

Following fields are required in the payload.

- message
- voice

Note that the voice must be one of the following as documented in IBM Watson [Text to Speech API Reference (https://www.ibm.com/watson/developercloud/text-to-speech/api/v1/)]

- de-DE_BirgitVoice

- de-DE_DieterVoice
- en-GB_KateVoice
- en-US_AllisonVoice
- en-US_LisaVoice
- en-US_MichaelVoice (the default)
- es-ES_LauraVoice
- es-ES_EnriqueVoice
- es-US_SofiaVoice
- fr-FR_ReneeVoice
- it-IT_FrancescaVoice
- ja-JP_EmiVoice
- pt-BR_IsabelaVoice

Response

```
{
    "base64_audio": "....."
}
```

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