**Ge Xia February 10, 2018**

**McKesson Deep Azure Final Project Report**

**Topic**: Image processing using Microsoft Cognitive Services Computer Vision API

**Problem**: Given an image URL, use Azure Computer Vision REST API to detect printed text, or recognize handwritten text in the images and extract the recognized characters into a machine-usable character stream.

**High Level Steps:**

1. Install and configure software tools for code development
2. Obtain Endpoint URL and subscription keys to Azure Cognitive Services
3. Create a web application using JAVA, JSP, Apache Tomcat
4. Implement code to access the REST API call to read text in an image
5. Run web application in a browser

**Data Set**: Image URLs are used in the demo.

**Hardware Used:** Windows 10, 64 bit processor laptop.

**Software, Tools and Technologies Used:**

* JDK 8, JSP 2.3
* Maven
* JSON
* HTML
* Eclipse Oxygen IDE
* Apache Tomcat 9
* Azure Microsoft Cognitive Services Computer Vision API

**Technology Overview:**

Azure cloud-based Computer Vision API provides developers with access to advanced algorithms for processing images and returning useful information. The smart Computer Vision algorithms can analyze visual content in different ways based on inputs and user choices. Users can get the processed image data by uploading an image or specifying an image URL.

With the Computer Vision API users can analyze images to:

* + - [Tag images based on content.](https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home)
    - [Categorize images.](https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home)
    - [Identify the type and quality of images.](https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home)
    - [Detect human faces and return their coordinates.](https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home)
    - [Recognize domain-specific content.](https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home)
    - [Generate descriptions of the content.](https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home)
    - [Use optical character recognition to identify printed text found in images.](https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home)
    - [Recognize handwritten text.](https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home)
    - [Distinguish color schemes.](https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home)
    - [Flag adult content.](https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home)
    - [Crop photos to be used as thumbnails.](https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home)

**Project Overview:**

In this project, I created two JSP pages using Computer Vision API to accomplish the following tasks:

* + - Detect and extract printed text from an image (OCR)

Reference:

[https://docs.microsoft.com/en-us/azure/cognitive-services/Computer-vision/quickstarts/java#optical-character-recognition-ocr-with-computer-vision-api-using-javaa-nameocr-a](https://docs.microsoft.com/en-us/azure/cognitive-services/Computer-vision/quickstarts/java)

* + - Detect and extract handwritten text from an image

Reference:

[https://docs.microsoft.com/en-us/azure/cognitive-services/Computer-vision/quickstarts/java#text-recognition-with-computer-vision-api-using-javaa-namerecognizetext-a](https://docs.microsoft.com/en-us/azure/cognitive-services/Computer-vision/quickstarts/java)

**Input data set requirements:**

Input method: raw image binary or image URL. I used image URL in this project.

Supported image formats: JPEG, PNG, GIF, BMP.

Image file size: Less than 4 MB.

Image dimension: Greater than 50 x 50 pixels.

Image URLs with printed text that I used:

<https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSJj9lLB9vU3HoGEGEIc8BWx-Mx_I-cJOTEDFAsL2_YqsMwxtGm>

<http://cdn.newsapi.com.au/image/v1/e10aad3abe8040f34c54ee73add530cf>

Image URLs with handwritten text that I used:

<https://d1qhuz9ahqnrhh.cloudfront.net/wp-content/uploads/2014/05/handwritten-banner2.png>

<http://www.productivity501.com/wp-content/uploads/2009/06/picture-9.png>

**Project Install/Config/Set up:**

* **Install JDK 8 by downloading from**

[**http://www.oracle.com/technetwork/pt/java/javase/downloads/jdk8-downloads-2133151.html**](http://www.oracle.com/technetwork/pt/java/javase/downloads/jdk8-downloads-2133151.html)

* **Get Eclipse Oxygen IDE:**

[**http://www.eclipse.org/downloads/**](http://www.eclipse.org/downloads/)

* **Download Apache Tomcat 9:**

[**https://tomcat.apache.org/download-90.cgi**](https://tomcat.apache.org/download-90.cgi)

* **Obtain Endpoint URL and subscription keys to Azure Cognitive Services using my McKesson account.**

**Endpoint: https://westcentralus.api.cognitive.microsoft.com/vision/v1.0**

**Set up Eclipse project:**

First create a Dynamic Web Project in Eclipse IDE.

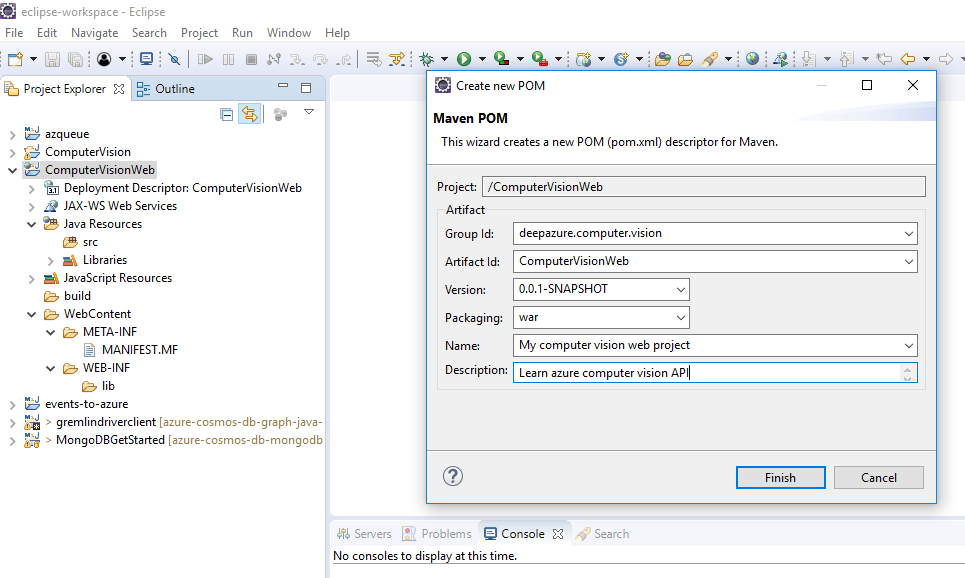
In Eclipse IDE, go to menu File > New > Dynamic Web Project to create a project for Java web application. Name the project as ComputerVisionWeb and keep the default settings for:

*Target Runtime: Apache Tomcat v9.0*

*Dynamic web module version (Servlet version): 3.1*

*Configuration: Default Configuration for Apache Tomcat v9.0*

Then add Maven as a build manager for the project, right click on the project and select Configure > Convert to Maven project. The *Create New POM* dialog appears, enter the following information:



Add dependencies in ComputerVisionWeb/pom.xml

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>deepazure.computer.vision</groupId>

<artifactId>ComputerVisionWeb</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<name>My computer vision web project</name>

<description>Learn azure computer vision API</description>

<dependencies>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet.jsp</groupId>

<artifactId>javax.servlet.jsp-api</artifactId>

<version>2.3.1</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.owasp.encoder</groupId>

<artifactId>encoder</artifactId>

<version>1.2</version>

</dependency>

<dependency>

<groupId>com.jayway.jsonpath</groupId>

<artifactId>json-path</artifactId>

<version>2.0.0</version>

</dependency>

</dependencies>

<build>

<sourceDirectory>src</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.7.0</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<plugin>

<artifactId>maven-war-plugin</artifactId>

<version>3.0.0</version>

<configuration>

<warSourceDirectory>WebContent</warSourceDirectory>

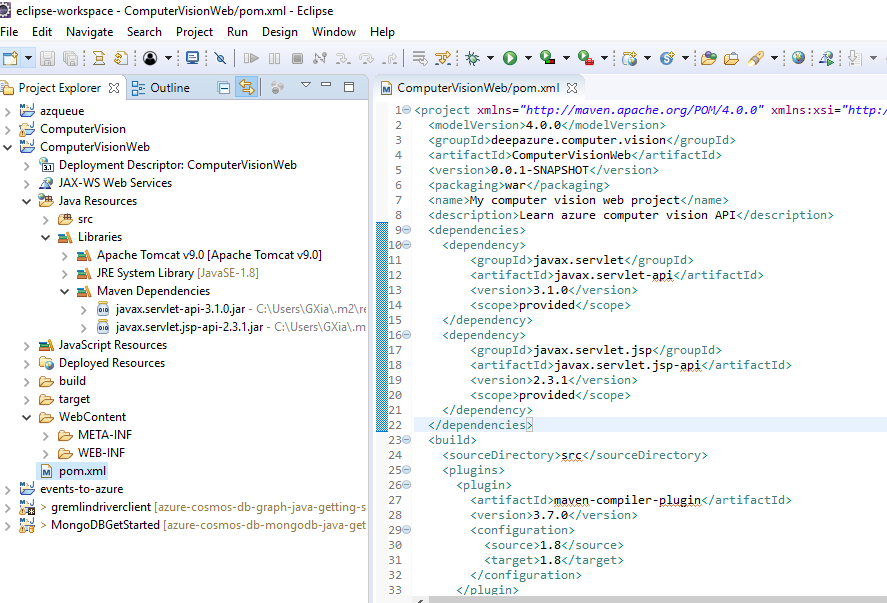
</configuration>

</plugin>

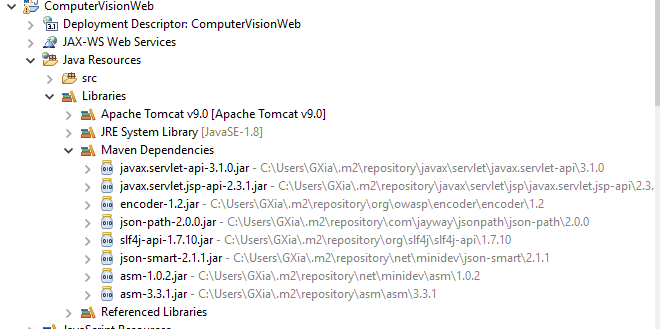
</plugins>

</build>

</project>



Maven will download the dependent JAR files in few seconds, as you can see under the Libraries section of the project:



**Creating JSP Pages:**

* Right click on the web project and select **New > JSP File**. Enter **ocrImage.jsp** as the file name in the *New JSP File* dialog.
* Click **Finish**. Eclipse created **ocrImage.jsp** file under the **WebContent** directory with some default code.
* Modify the jsp code using Azure Computer Vision Java API.
* Using the same way, create **handwrittenImage.jsp**
* API and sample code references online:

<https://docs.microsoft.com/en-us/azure/cognitive-services/Computer-vision/quickstarts/java>

-------------------------------------------------------------------------------------------------------------

Below is the complete code for these two jsp pages:

**ocrImage.jsp**

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"* pageEncoding=*"ISO-8859-1"*

import=*"java.net.URI,*

*org.apache.http.Header,*

*org.apache.http.HttpEntity,*

*org.apache.http.HttpResponse,*

*org.apache.http.client.HttpClient,*

*org.apache.http.client.methods.HttpGet,*

*org.apache.http.client.methods.HttpPost,*

*org.apache.http.entity.StringEntity,*

*org.apache.http.impl.client.CloseableHttpClient,*

*org.apache.http.impl.client.HttpClientBuilder,*

*org.apache.http.util.EntityUtils,*

*org.apache.http.client.utils.URIBuilder,*

*org.json.JSONObject,*

*org.json.JSONArray,*

*com.jayway.jsonpath.JsonPath,*

*org.owasp.encoder.Encode"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<%

String subscriptionKey = "30de815d24c24e94b82ff4462501c483";

String uriBase = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/ocr";

String myUrl = request.getParameter("myUrl");

String urlString = **null**;

**if**(myUrl != **null**){

urlString = "{\"url\":\"" + myUrl + "\"}";

}

CloseableHttpClient resultClient = HttpClientBuilder.create().build();

String outputText = "";

**if**(urlString != **null**){

**try** {

// Create the URI to access the REST API call to read text in an image.

URIBuilder uriBuilder = **new** URIBuilder(uriBase);

// Request parameters.

uriBuilder.setParameter("language", "unk"); //AutoDetect

uriBuilder.setParameter("detectOrientation ", "true");

// Prepare the URI for the REST API call.

URI uri = uriBuilder.build();

HttpPost textRequest = **new** HttpPost(uri);

// Request headers.

textRequest.setHeader("Content-Type", "application/json");

textRequest.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

// Request body.

StringEntity reqEntity = **new** StringEntity(urlString);

textRequest.setEntity(reqEntity);

HttpResponse resultResponse = resultClient.execute(textRequest);

HttpEntity responseEntity = resultResponse.getEntity();

**if** (responseEntity != **null**)

{

// Format and display the JSON response.

String jsonString = EntityUtils.toString(responseEntity);

JSONObject json = **new** JSONObject(jsonString);

outputText = json.toString(2);

System.out.println(outputText);

}

}

**catch** (Exception e)

{

// Display error message.

System.out.println(e.getMessage());

}

}

%>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>

<title>Read printed text in an image</title>

</head>

<body onload='document.getElementById("myUrl").focus();'>

<form id=*"myform"* name=*"myform"* method=*"post"* action=*"ocrImage.jsp"*>

<table width=*"100%"* height=*200* style="table-layout: *fixed*;" border=*1* bgcolor=*#FAFAD* cellpadding=*"2"*>

<col width=*"60%"*/> <col width=*"40%"*/>

<tr>

<td colspan=*"2"* align=*"center"*>

<b>Welcome to my Deep Azure final project</b><br />

This web application uses Computer Vision REST API to perform optical character recognition (OCR) to detect printed text in an image.<br />

Enter an image URL in the text field, and click the button on the right.<br />

The image will show in the left panel, and the JSON response will show in the text area on the right.

</td>

</tr>

<tr>

<td colspan=*"2"* style="margin-left: *15px*;">

Enter image URL: <input type=*"text"* id=*"myUrl"* name=*"myUrl"* size=*"100"*>&nbsp;&nbsp;&nbsp;&nbsp;

<input id=*"submit"* type=*"submit"* name=*"submit"* value=*"Analyze Image With Printed Text"* />

</td>

</tr>

<tr>

<td valign=*"top"*>

<% **if** (myUrl != **null**) { %>

<iframe width=*"100%"* height=*"100%"* src=*"*<%= myUrl %>*"* name=*"leftside"*></iframe>

<% } **else** { %>

<iframe width=*"100%"* height=*"100%"* src=*"about:blank"* name=*"leftside"*></iframe>

<% } %>

</td>

<td valign=*"top"*>

<textArea name=*"txtarea"* id=*"myText"* style="font-family:*Arial*;font-size:*8pt*;width:*100%*;height:*100vw*">

<%=Encode.forHtml(outputText)%>

</textArea>

</td>

</tr>

</table>

</form>

</body>

</html>

**handwrittenImage.jsp**

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"* pageEncoding=*"ISO-8859-1"*

import=*"java.net.URI,*

*org.apache.http.Header,*

*org.apache.http.HttpEntity,*

*org.apache.http.HttpResponse,*

*org.apache.http.client.HttpClient,*

*org.apache.http.client.methods.HttpGet,*

*org.apache.http.client.methods.HttpPost,*

*org.apache.http.entity.StringEntity,*

*org.apache.http.impl.client.CloseableHttpClient,*

*org.apache.http.impl.client.HttpClientBuilder,*

*org.apache.http.util.EntityUtils,*

*org.json.JSONObject,*

*org.json.JSONArray,*

*com.jayway.jsonpath.JsonPath,*

*org.owasp.encoder.Encode"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<%

String subscriptionKey = "30de815d24c24e94b82ff4462501c483";

String uriBase = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/recognizeText?handwriting=true";

String myUrl = request.getParameter("myUrl");

String urlString = **null**;

**if**(myUrl != **null**){

urlString = "{\"url\":\"" + myUrl + "\"}";

}

CloseableHttpClient textClient = HttpClientBuilder.create().build();

CloseableHttpClient resultClient = HttpClientBuilder.create().build();

String outputText = "";

**if**(urlString != **null**){

**try**

{

// This operation requires two REST API calls. One to submit the image for processing,

// the other to retrieve the text found in the image.

//

// Begin the REST API call to submit the image for processing.

URI uri = **new** URI(uriBase);

HttpPost textRequest = **new** HttpPost(uri);

// Request headers. Another valid content type is "application/octet-stream".

textRequest.setHeader("Content-Type", "application/json");

textRequest.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

// Request body.

StringEntity requestEntity = **new** StringEntity(urlString);

textRequest.setEntity(requestEntity);

// Execute the first REST API call to detect the text.

HttpResponse textResponse = textClient.execute(textRequest);

// Check for success.

**if** (textResponse.getStatusLine().getStatusCode() != 202)

{

// Format and display the JSON error message.

HttpEntity entity = textResponse.getEntity();

String jsonString = EntityUtils.toString(entity);

JSONObject json = **new** JSONObject(jsonString);

outputText = "Error: " + json.toString(2);

System.out.println(outputText);

}**else**{

String operationLocation = **null**;

// The 'Operation-Location' in the response contains the URI to retrieve the recognized text.

Header[] responseHeaders = textResponse.getAllHeaders();

**for**(Header header : responseHeaders) {

**if**(header.getName().equals("Operation-Location"))

{

// This string is the URI where you can get the text recognition operation result.

operationLocation = header.getValue();

**break**;

}

}

// NOTE: The response may not be immediately available. Handwriting recognition is an

// async operation that can take a variable amount of time depending on the length

// of the text you want to recognize. You may need to wait or retry this operation.

outputText = "Handwritten text submitted. Waiting 10 seconds to retrieve the recognized text.\n\n";

System.out.println(outputText);

Thread.sleep(10000);

// Execute the second REST API call and get the response.

HttpGet resultRequest = **new** HttpGet(operationLocation);

resultRequest.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

HttpResponse resultResponse = resultClient.execute(resultRequest);

HttpEntity responseEntity = resultResponse.getEntity();

**if** (responseEntity != **null**)

{

// Format and display the JSON response.

String jsonString = EntityUtils.toString(responseEntity);

JSONObject json = **new** JSONObject(jsonString);

outputText = json.toString(2);

System.out.println(outputText);

}

}

}

**catch** (Exception e)

{

System.out.println(e.getMessage());

}

}

%>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>

<title>Read hand written text in an image</title>

</head>

<body onload='document.getElementById("myUrl").focus();'>

<form id=*"myform"* name=*"myform"* method=*"post"* action=*"handwrittenImage.jsp"*>

<table width=*"100%"* height=*200* style="table-layout: *fixed*;" border=*1* bgcolor=*#FAFAD* cellpadding=*"2"*>

<col width=*"60%"*/> <col width=*"40%"*/>

<tr>

<td colspan=*"2"* align=*"center"*>

<b>Welcome to my Deep Azure final project</b><br />

This web application uses Computer Vision REST API to read hand written text in an image.<br />

Enter an image URL in the text field, and click the button on the right.<br />

The image will show in the left panel, and the JSON response will show in the text area on the right.

</td>

</tr>

<tr>

<td colspan=*"2"* style="margin-left: *15px*;">

Enter image URL: <input type=*"text"* id=*"myUrl"* name=*"myUrl"* size=*"100"*>&nbsp;&nbsp;&nbsp;&nbsp;

<input id=*"submit"* type=*"submit"* name=*"submit"* value=*"Analyze Image With Hand Written Text"* />

</td>

</tr>

<tr>

<td valign=*"top"*>

<% **if** (myUrl != **null**) { %>

<iframe width=*"100%"* height=*"100%"* src=*"*<%= myUrl %>*"* name=*"leftside"*></iframe>

<% } **else** { %>

<iframe width=*"100%"* height=*"100%"* src=*"about:blank"* name=*"leftside"*></iframe>

<% } %>

</td>

<td valign=*"top"*>

<textArea name=*"txtarea"* id=*"myText"* style="font-family:*Arial*;font-size:*8pt*;width:*100%*;height:*100vw*">

<%=Encode.forHtml(outputText)%>

</textArea>

</td>

</tr>

</table>

</form>

</body>

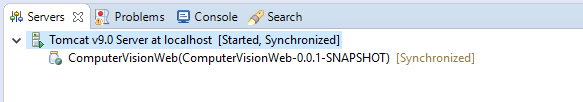
</html>

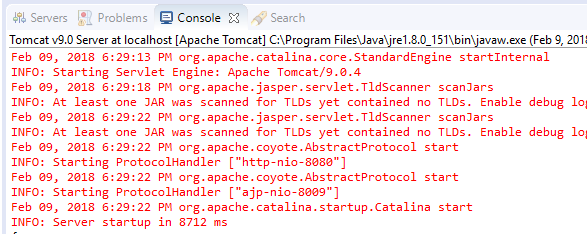
**Deploy web application:**

In eclipse, add Apache Tomcat server to the IDE.

To deploy the project on Tomcat, simply drag the project from the Project Explorer view to Servers view.

Right click on the server and select Start. Tomcat will be in Started state after few seconds.

****

****

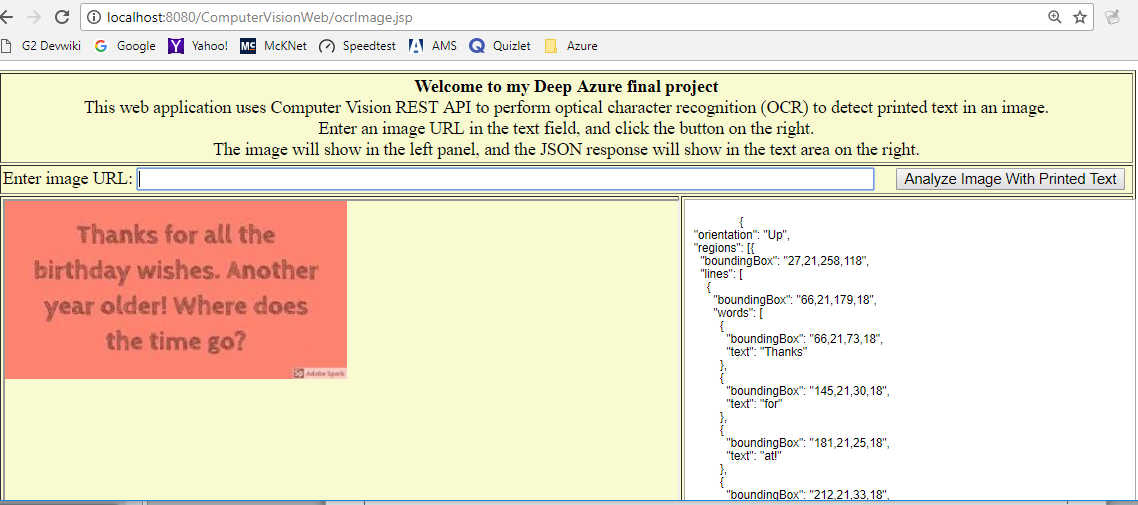
**After Tomcat server is started, run the jsp page in a browser.**

Open a browser e.g. Google Chrome and enter the following URL:

<http://localhost:8080/ComputerVisionWeb/ocrImage.jsp>

Enter below image url in the text field and click the button on the right.

<https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSJj9lLB9vU3HoGEGEIc8BWx-Mx_I-cJOTEDFAsL2_YqsMwxtGm>

****

**Full output in eclipse console:**

{

"orientation": "Up",

"regions": [{

"boundingBox": "27,21,258,118",

"lines": [

{

"boundingBox": "66,21,179,18",

"words": [

{

"boundingBox": "66,21,73,18",

"text": "**Thanks**"

},

{

"boundingBox": "145,21,30,18",

"text": "**for**"

},

{

"boundingBox": "181,21,25,18",

"text": "**at!**"

},

{

"boundingBox": "212,21,33,18",

"text": "**the**"

}

]

},

{

"boundingBox": "27,53,258,24",

"words": [

{

"boundingBox": "27,53,87,24",

"text": "**birthday**"

},

{

"boundingBox": "120,53,75,18",

"text": "**wishes.**"

},

{

"boundingBox": "201,53,84,19",

"text": "**Another**"

}

]

},

{

"boundingBox": "36,85,239,25",

"words": [

{

"boundingBox": "36,91,45,19",

"text": "**year**"

},

{

"boundingBox": "87,86,60,18",

"text": "**older!**"

},

{

"boundingBox": "153,85,67,19",

"text": "**Where**"

},

{

"boundingBox": "226,86,49,18",

"text": "**does**"

}

]

},

{

"boundingBox": "92,117,127,22",

"words": [

{

"boundingBox": "92,117,33,19",

"text": "**the**"

},

{

"boundingBox": "131,118,47,18",

"text": "**time**"

},

{

"boundingBox": "184,118,35,21",

"text": "**go?**"

}

]

}

]

}],

"textAngle": 0,

"language": "en"

}

**Enter below image url on the jsp page, and click the button:**

[**http://cdn.newsapi.com.au/image/v1/e10aad3abe8040f34c54ee73add530cf**](http://cdn.newsapi.com.au/image/v1/e10aad3abe8040f34c54ee73add530cf)



**Full eclipse console output:**

{

"orientation": "Up",

"regions": [

{

"boundingBox": "115,12,755,297",

"lines": [

{

"boundingBox": "209,12,661,52",

"words": [

{

"boundingBox": "209,12,397,51",

"text": "**EMERGENCY**"

},

{

"boundingBox": "627,12,243,52",

"text": "**ALERTS**"

}

]

},

{

"boundingBox": "115,231,627,78",

"words": [

{

"boundingBox": "115,232,426,77",

"text": "**Emergency**"

},

{

"boundingBox": "562,231,180,62",

"text": "**Alert**"

}

]

}

]

},

{

"boundingBox": "1135,13,49,49",

"lines": [{

"boundingBox": "1135,13,49,49",

"words": [{

"boundingBox": "1135,13,49,49",

"text": "x"

}]

}]

},

{

"boundingBox": "116,327,1042,364",

"lines": [

{

"boundingBox": "117,327,1041,63",

"words": [

{

"boundingBox": "117,327,387,63",

"text": "BALLISTIC"

},

{

"boundingBox": "532,327,299,63",

"text": "MISSILE"

},

{

"boundingBox": "855,327,303,62",

"text": "THREAT"

}

]

},

{

"boundingBox": "116,429,1028,62",

"words": [

{

"boundingBox": "116,429,364,62",

"text": "INBOUND"

},

{

"boundingBox": "505,429,103,62",

"text": "TO"

},

{

"boundingBox": "636,429,290,62",

"text": "HAWAII."

},

{

"boundingBox": "948,429,196,62",

"text": "SEEK"

}

]

},

{

"boundingBox": "116,529,1007,63",

"words": [

{

"boundingBox": "116,529,423,62",

"text": "IMMEDIATE"

},

{

"boundingBox": "564,529,360,63",

"text": "SHELTER."

},

{

"boundingBox": "947,529,176,63",

"text": "THIS"

}

]

},

{

"boundingBox": "116,629,584,62",

"words": [

{

"boundingBox": "116,629,63,62",

"text": "IS"

},

{

"boundingBox": "206,629,162,62",

"text": "NOT"

},

{

"boundingBox": "390,630,52,60",

"text": "A"

},

{

"boundingBox": "466,629,234,62",

"text": "DRILL."

}

]

}

]

}

],

"textAngle": 0,

"language": "en"

}

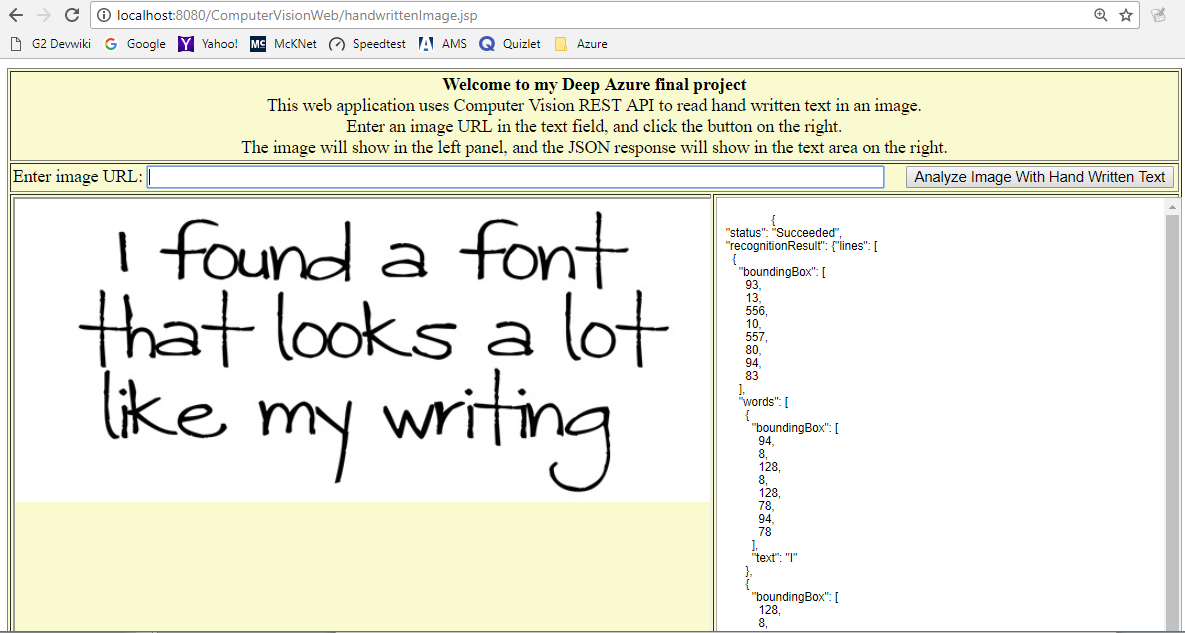
**Run *handwrittenImage.jsp:***

Open another tab in browser and enter the following URL:

<http://localhost:8080/ComputerVisionWeb/handwrittenImage.jsp>

Enter below image url in the text field and click the button on the right.

<https://d1qhuz9ahqnrhh.cloudfront.net/wp-content/uploads/2014/05/handwritten-banner2.png>



**Console output:**

Handwritten text submitted. Waiting 10 seconds to retrieve the recognized text.

{

"status": "Succeeded",

"recognitionResult": {"lines": [

{

"boundingBox": [

93,

13,

556,

10,

557,

80,

94,

83

],

"words": [

{

"boundingBox": [

94,

8,

128,

8,

128,

78,

94,

78

],

"text": "I"

},

{

"boundingBox": [

128,

8,

328,

10,

328,

80,

128,

78

],

"text": "found"

},

{

"boundingBox": [

333,

10,

380,

11,

379,

81,

332,

80

],

"text": "a"

},

{

"boundingBox": [

398,

11,

556,

12,

556,

82,

397,

81

],

"text": "font"

}

],

"text": "I found a font"

},

{

"boundingBox": [

59,

80,

593,

83,

592,

153,

58,

150

],

"words": [

{

"boundingBox": [

61,

76,

220,

78,

219,

154,

60,

152

],

"text": "that"

},

{

"boundingBox": [

220,

78,

409,

81,

408,

157,

219,

154

],

"text": "looks"

},

{

"boundingBox": [

429,

81,

480,

82,

479,

158,

428,

157

],

"text": "a"

},

{

"boundingBox": [

480,

82,

598,

83,

597,

159,

479,

158

],

"text": "lot"

}

],

"text": "that looks a lot"

},

{

"boundingBox": [

81,

140,

542,

157,

538,

266,

77,

248

],

"words": [

{

"boundingBox": [

80,

150,

183,

151,

186,

261,

82,

260

],

"text": "like"

},

{

"boundingBox": [

207,

151,

314,

153,

316,

263,

209,

262

],

"text": "my"

},

{

"boundingBox": [

318,

153,

546,

156,

548,

267,

320,

263

],

"text": "writing"

}

],

"text": "like my writing"

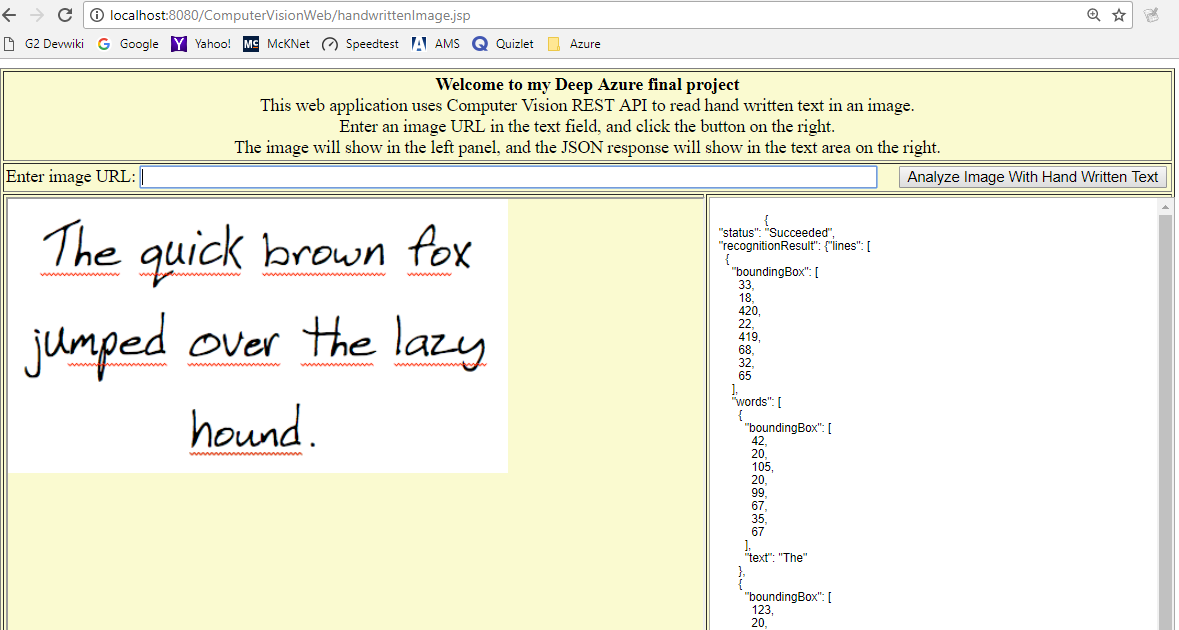
}

]}

}

**Enter this url on the page, then click the button.**

[**http://www.productivity501.com/wp-content/uploads/2009/06/picture-9.png**](http://www.productivity501.com/wp-content/uploads/2009/06/picture-9.png)



**Eclipse console output:**

Handwritten text submitted. Waiting 10 seconds to retrieve the recognized text.

{

"status": "Succeeded",

"recognitionResult": {"lines": [

{

"boundingBox": [

33,

18,

420,

22,

419,

68,

32,

65

],

"words": [

{

"boundingBox": [

42,

20,

105,

20,

99,

67,

35,

67

],

"text": "The"

},

{

"boundingBox": [

123,

20,

218,

20,

211,

67,

117,

67

],

"text": "quick"

},

{

"boundingBox": [

228,

20,

348,

20,

341,

67,

221,

67

],

"text": "brown"

},

{

"boundingBox": [

366,

20,

427,

20,

420,

67,

359,

67

],

"text": "fox"

}

],

"text": "The quick brown fox"

},

{

"boundingBox": [

127,

73,

146,

60,

150,

67,

132,

79

],

"words": [{

"boundingBox": [

132,

65,

146,

65,

146,

79,

132,

79

],

"text": "F"

}],

"text": "F"

},

{

"boundingBox": [

14,

105,

431,

97,

432,

158,

15,

166

],

"words": [

{

"boundingBox": [

20,

102,

155,

103,

150,

166,

15,

165

],

"text": "jumped"

},

{

"boundingBox": [

166,

103,

255,

104,

250,

167,

161,

166

],

"text": "over"

},

{

"boundingBox": [

269,

104,

337,

104,

332,

167,

264,

167

],

"text": "the"

},

{

"boundingBox": [

343,

104,

437,

105,

432,

168,

338,

167

],

"text": "lazy"

}

],

"text": "jumped over the lazy"

},

{

"boundingBox": [

168,

190,

266,

196,

264,

229,

166,

223

],

"words": [{

"boundingBox": [

168,

192,

277,

192,

278,

226,

169,

226

],

"text": "hound"

}],

"text": "hound"

}

]}

}

**Summary:**

* **Optical character recognition (OCR)**

Benefits: This technology can detect text in an image and extract the recognized words into a machine-readable character stream. It can analyze images to detect embedded text, generate character streams, and enable searching. It saves time and effort and can make you more productive by allowing you to take images of text, rather than having to transcribe it.

Limitations: On photos where text is dominant, false positives may come from partially recognized words. On some photos, especially photos without any text, precision can vary a lot depending on the type of image.

* **Recognize Handwritten Text**

Benefits: This technology makes it possible to digitize notes, which then allows you to implement quick and easy search. It also reduces paper clutter. It can be used for numerous other purposes like medical records, security, and banking.

Limitations: This technology is currently in preview and is only available for English text.

**Lessons learned:**

* Some code in the tutorial use deprecated java classes or methods. I was able to solve the issue in my application by doing some research on the required libraries.
* The response from the image processing API is in JSON format. To make the feature more usable, the next step I would like to do is to extract the recognized text from JSON string, concatenate all pieces together and display the complete text in the application.
* Depending on the image quality or the style of the text written in the image, inaccurate or partial recognized text sometimes can happen. We see this in one of the examples I used.

**YouTube Links:**

2 Min: <https://youtu.be/Eizzpb7XzMU>

15 Min: <https://youtu.be/4vOerTrCDmY>

**Public GitHub repository URL:**

<https://github.com/azurelearner/DeepAzureFinalProject.git>