Lab 4.2 Google Vision API

Instructions

1. Code on your computer and zip all your code before submission.
2. Please submit the assignment after you finish.

In this lab, we are going to explore Google Vision API. Google Cloud Vision API enables developers to understand the **content** of an image by utilizing powerful machine learning models in an easy-to-use and user friendly REST API. It can analyze images to extract text (OCR), labels, detect faces, identify objects, etc. The API can be integrated into applications to provide image analysis capabilities without requiring advanced machine learning expertise. In this lab exercise, we want to use Google Vision API to detect labels from a given image. Download the Google\_Vision\_API\_lab folder and work on the exercise.  
  
**Part 1**

1. Install all the require module with npm install
2. Check out the knexfile, create the database locally and configure the setting accordingly.
3. Create a new migration file with the following command:

| npx knex migrate:make create\_image\_table |
| --- |

1. Complete the migrate file with the following schema:

| exports.up = function (knex) {  return knex.schema.createTable("image", (table) => {  table.increments("id").primary();  table.text("name", 128).notNullable();  table.text("detected\_label").notNullable();  });  };  exports.down = function (knex) {  return knex.schema.dropTableIfExists("image");  }; |
| --- |

We want to create a new table named “image” with 3 fields, id of the image item, name of the image and the detected\_label that stores the label obtained from the Google Vision API.

1. Migrate the migration file with the following command after you finish:

| npx knex migrate:latest |
| --- |

1. checkout the app.js, routers/index.js and controllers/index.js, try to understand what each file is doing. In the routers/index.js and controllers/index.js, finish the APIs below:

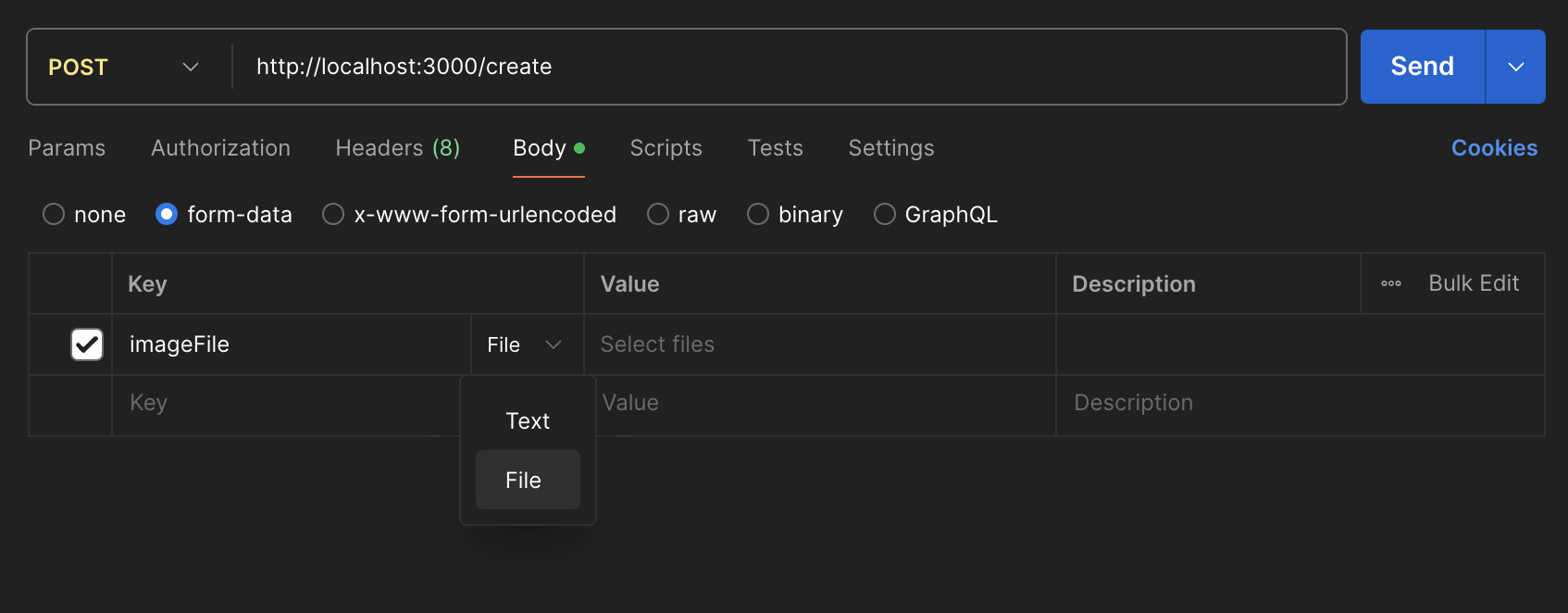
* GET /list
* GET /get:id
* POST /create

Complete the APIs accordingly that are marked with TODO.

**Few things to Note:**

When calling the POST /create api, we want to upload an image for calling Google Vision API to perform label detection and then retrieve all the description from the response object and store it in the database. Hence, the first step is to call the POST /create request with uploading an image. Secondly, inside the API logic, we will call the Google Vision API to do label detection. Thirdly, we obtain the result and store the result with knex.

However, how do we upload an image file with Postman? In the Postman console, navigate to the Body tab and select form-data. Under the form-data table, input a new key imageFile, under the dropdown menu, select File as the value type, then upload an image file on the value cell and click the Send button to call the POST /create api.



In routers/index.js, we have to use a new module called multer, you will find the configuration of multer we set in this project. This module allows us to handle file upload from POST requests in Express. Multer is a middleware for handling **multipart/form-data**, which is primarily used for uploading files. Multer will not process any form which is not multipart (multipart/form-data). Multer adds a body object and a file or files object to the request object. The body object contains the values of the text fields of the form, the file or files object contains the files uploaded via the form. Read more from the official documents:  
<https://github.com/expressjs/multer>  
<https://www.npmjs.com/package/multer>

The POST /create API will first run the upload.single("imageFile") middleware to add a file object into the request body, i.e. req.file, you will be able to find the details of the uploaded image in the req.file object.

In the routers/index.js, we create a multer.diskStorage, this allows us to store the uploaded image in a destination folder. Thus, if you upload the image file and call the /create api correctly, after the upload.single("imageFile") middleware is resolved, you should see the uploaded image stored under the public/images folder inside your project root folder. When the uploaded image is stored in the public images folder, we can read the image file content and convert it into Base64 format. Google vision API accepts images in Base64 format, this is the reason why we have to store the uploaded image locally and perform the format conversion. Read more from the official document to know how to use the Google Vision API:  
<https://cloud.google.com/vision/docs/request>

**Part 2**

If you are able to complete part 1, you should have the basic understanding of how to handle file upload in Express with Postman and know how to use Google Vision API to do label detection on images. Now, let’s explore further, this time we want to perform face detection on any image that includes human faces. Try to figure out how to do this with Google Vision API. Similarly, finish the APIs below:

* GET /list
* GET /get:id
* POST /create

For this time, we want to save the facial expression description of all the faces detected from the image when we call the POST /create api. Try to come up with the appropriate schema by yourself, how would you store the data? This is the response of the Face detection with Google Vision API:

{

"responses": [

{

"faceAnnotations": [

{

"boundingPoly": {

"vertices": [objects]

},

"fdBoundingPoly": {

"vertices": [objects]

},

"landmarks": [objects],

"rollAngle": -7.5747037,

"panAngle": -20.727795,

"tiltAngle": -6.7191052,

"detectionConfidence": 0.9609375,

"landmarkingConfidence": 0.5166721,

// we want to store all the items below

"joyLikelihood": "VERY\_UNLIKELY",

"sorrowLikelihood": "VERY\_UNLIKELY",

"angerLikelihood": "VERY\_UNLIKELY",

"surpriseLikelihood": "VERY\_UNLIKELY",

"underExposedLikelihood": "VERY\_UNLIKELY",

"blurredLikelihood": "VERY\_UNLIKELY",

"headwearLikelihood": "VERY\_UNLIKELY"

},

...more

]

}

]

}

You could add additional APIs if you want to make your project fully support CRUD.

However, this time you have to start from the beginning without any source project folder, from zero to everything. Time to recall how to create a new project with npm and NodeJS. You could take reference to your previous projects and notes.