GRT INSTITUTE OF ENGINEERING AND TECHNOLOGY, TIRUTTANI - 631209
Approved by AICTE, New Delhi Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROJECT TITLE

Image Recognition with IBM Cloud Visual Recognition

COLLEGE CODE:1103

PHASE : 4

SUNITHA.S

3rd yr, 5th sem

Reg no.:110321104050

Sunitha.sapanisss@gmail.com

CODE SUMMARY

Backend (server.js):

This component is a Node.js server that handles image uploads, storage on IBM Cloud Object Storage (COS), retrieval, and deletion.It uses the IBM COS SDK to connect to the cloud object storage service.It sets up an Express.js server, handles CORS, and listens on a specified port (default 3001).

It defines several routes:

Items: Retrieves the contents of the COS bucket with an optional prefix.
Images: Handles image uploads, storing them in the COS bucket.
results: Retrieves JSON data from the "results" folder in the COS bucket.
item: Deletes an item from the COS bucket based on the filename.
The code includes error handling middleware to manage unexpected situations and return proper error responses.It uses environment variables (loaded from a .env file) for configuration, such as COS endpoint, API key, and bucket name.

Frontend (app.js):

This component serves as the frontend for the image classification system. It's also a Node.js server using Express.It serves static files for an HTML interface, JavaScript, CSS, and images.
It includes routes displays the main page or redirects to a login page if the backend URL is not configured.
items: Passes GET requests to the backend to retrieve image information.
uploadimage: Handles image uploads and forwards them to the backend.
classifyimage: Requests image classification results from the backend.
image: Deletes an image item, forwarding the request to the backend.
It also includes proper error handling middleware.

Overall Review:

The backend appears to be properly structured with clear route handling and error management.The COS integration seems well-implemented, with functions for uploading, retrieving, and deleting objects.The frontend is designed to serve a web interface for interacting with the backend, but it might benefit from more detailed error handling and feedback to users.The use of environment variables for configuration is good practice.

However, it's crucial to ensure security in both applications, especially considering they use external services and handle file uploads. Ensure that proper access controls and authentication are in place to protect against unauthorized access.

Moreover, the code could be enhanced with more detailed comments and documentation to make it easier for other developers to understand and work with the system.
Lastly, consider implementing image classification and integration with an NLG service, as mentioned in your initial query, to fulfill the project's image recognition and caption generation requirements

Code:

DOCKERFILE:

```
FROM node:10-alpine
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
RUN npm ci --only=production
COPY . .
EXPOSE 3001
CMD [ "node", "server.js" ]
```

PACKAGE.JSON:

```
{
  "name": "codeengine-image-classification-backend",
  "version": "1.0.0",
  "description": "Backend server for Code Engine Image Classification",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Vidyasagar Machupalli",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^8.2.0",
    "express": "^4.17.1",
    "ibm-cos-sdk": "^1.6.0",
    "multer": "^1.4.2",
    "multer-s3": "^2.9.0"
  }
}
```

SERVER.JS:

```
"use strict";
const myCOS = require("ibm-cos-sdk");
const multer = require("multer");
const multerS3 = require("multer-s3");
const express = require("express");
const app = express();
const path = require("path");
require("dotenv").config({
  silent: true
});

const cors = require("cors");
app.use(cors());

const port = process.env.PORT || 3001;

function getCosClient() {
  var config = {
    endpoint:
      process.env.COS_ENDPOINT ||
      "s3.us-south.cloud-object-storage.appdomain.cloud",
    apiKeyId: process.env.COS_APIKEY,
    ibmAuthEndpoint: "https://iam.cloud.ibm.com/identity/token",
    serviceInstanceId: process.env.COS_RESOURCE_INSTANCE_ID,
  };


  var cosClient = new myCOS.S3(config);
  return cosClient;
}

function uploadFilesToCOS(req, res, next) {
  var upload = multer({
    storage: multerS3({
      s3: getCosClient(),
      bucket: process.env.COS_BUCKETNAME + "/images",
      metadata: function (req, file, cb) {
        cb(null, { fieldName: file.fieldname });
      },
      key: function (req, file, cb) {
        cb(null, file.originalname);
      } }),
  }).array("files", 10);

  upload(req, res, function (err) {
    if (err) {
      return next(err);
    }
    if (req.files.length === 0) {
      return res.send("Upload an image...");
    } else if (req.files.length > 1) {
      return res.send(
        "Successfully uploaded " + req.files.length + " images to Object Storage");
    } else {
      return res.send(
        "Successfully uploaded " + req.files.length + " image to Object Storage"};

    }  });

}
 async function getBucketContents(req, res, next, prefix) {
  try {
    let cos = getCosClient();
    let bucketName = process.env.COS_BUCKETNAME;
    var resultDict = {};
    var result;
    console.log(`Retrieving bucket contents from: ${bucketName}`);

    const data = await cos
      .listObjects({
        Bucket: bucketName,
        Prefix: prefix,
      })
      .promise();
    if (data != null && data.Contents != null) {
      for (var i = 0; i < data.Contents.length; i++) {
        var itemKey = data.Contents[i].Key;
        var itemSize = data.Contents[i].Size;
        console.log(`Item: ${itemKey} (${itemSize} bytes).`);
        result = await getItem(bucketName, itemKey, prefix);
        resultDict[itemKey] = result;
      }
      res.send(resultDict);
    }
  } catch (e) {
    console.error(`ERROR: ${e.code} - ${e.message}\n`);
    return next(e.message);
  }
}
 async function getItem(bucketName, itemName, prefix) {
  let cos = getCosClient();
  console.log(`Retrieving item from bucket: ${bucketName}, key: ${itemName}`);
  try {
    const data = await cos
      .getObject({
        Bucket: bucketName,
        Key: itemName,
      })
      .promise();
    if (data != null) {
      if (prefix === "results") {
        return JSON.parse(data.Body);
      } else {
        return Buffer.from(data.Body).toString("base64");
      }}
  } catch (e) {
    console.error(`ERROR: ${e.code} - ${e.message}\n`);
  }}
async function deleteItem(req, res, next, bucketName, itemName, prefix) {
  let cos = getCosClient();
  let bucketname = process.env.COS_BUCKETNAME;
  itemName = prefix + "/" + itemName;
  if (prefix === "results") {
    itemName = itemName + ".json";
  }
  console.log(`Deleting item: ${itemName}`);
  try {
    await cos
      .deleteObject({
        Bucket: bucketname,
        Key: itemName,
      })
      .promise();
    console.log(`Item: ${itemName} deleted!`);
    res.send(`Item: ${itemName} deleted!`);
  } catch (e) {
    console.error(`ERROR: ${e.code} - ${e.message}\n`);
  }}

 app.get("/", function (req, res, next) {
  res.send("Hello World! from backend");
});

app.get("/items", async (req, res, next) => {
  try {
    var prefix = req.query.prefix;

     await getBucketContents(req, res, next, prefix);
  } catch (error) {
```

```
    return next(error);
  }
});

app.post("/images", uploadFilesToCOS, function (req, res, next) {});

app.post("/results", async (req, res, next) => {
  try {
    await getBucketContents(req, res, next, "results");
  } catch (error)
    console.log(error);
    return next(error);
  }
});


app.delete("/item", async (req, res, next) => {
  var itemName = req.query.filename;
  console.log(itemName);
  await deleteItem(req, res, next, null, itemName, "images");
  await deleteItem(req, res, next, null, itemName, "results");
});

app.use((req, res, next) => {
  const error = new Error("Not found");
  error.status = 404;
  next(error);
});
app.use((error, req, res, next) => {
  console.log(error);
  if (res.headersSent) {
    return next(error);
  }
  res.status(error.status || 500).send({
    error: {
      status: error.status || 500,
      message: error.message || "Internal Server error",
    },
  });
});

app.listen(port, () => console.log(`App listening on port ${port}!`));
```

DOCKERFILE:

```
FROM node:10-alpine
WORKDIR /usr/src/app

COPY package*.json ./
RUN npm install
RUN npm ci --only=production
COPY . .
EXPOSE 3000
CMD [ "node", "app.js" ]
```

APP.JS:

```
"use strict";
const express = require("express");
const app = express();
const request = require("request");
const path = require("path");
require("dotenv").config({
  silent: true
});
const cors = require("cors");
app.use(cors());
app.use(express.static(__dirname + '/public/js'));
app.use(express.static(__dirname + '/public/images'))
app.use(express.static(__dirname + '/public/css'))
const port = process.env.PORT || 3000;

const backendURL = process.env.BACKEND_URL;
console.log("backend URL: " + backendURL);

app.get('/', function(req, res) {
    if (backendURL === undefined || backendURL === ""){
// if user is not logged-in redirect back to login page //
       res.sendFile(__dirname + "/public/501.html");
    }   else{
        res.sendFile(__dirname + "/public/index.html");
    }
});

app.get('/items', async(req, res) => {
  req.pipe(
   await request.get(
      {
        url: backendURL+"/items?prefix=images",
        agentOptions: {
          rejectUnauthorized: false
        }
      },
      function(error, resp, body) {
        if (error) {
          res.status(400).send(error.message);
        }
        else{
        //console.log(body);
        res.send({ data: body });
        }
      }
    )
  );
});
app.post("/uploadimage", async(req, res) => {
    req.pipe(
     await request.post(
        {
          url: backendURL+"/images",
          gzip: true,
          agentOptions: {
            rejectUnauthorized: false
          }
        },
        function(error, resp, body) {
          if (error) {
            console.log(error);
          res.status(400).send(error.message);
        }
        else{
        //console.log(body);
        res.send({ data: body });
        }})
    );

});

app.post("/classifyimage", async(req, res) => {
     req.pipe(
       await request.post(
       {
         url: backendURL+"/results",
         agentOptions: {
           rejectUnauthorized: false
         }
       },
       function(error, resp, body) {
         if (error) {
           console.log(error);
           res.status(400).send(error.message);
         }
         else{
         //console.log(body);
         res.send({ data: body });
         }})
    );
});

app.delete("/image", async (req, res) => {
  var itemName = req.query.filename;
  req.pipe(
    await request.delete(
      {
        url: backendURL+"/item?filename="+itemName,
        agentOptions: {
          rejectUnauthorized: false
        }
      },
      function(error, resp, body) {
        if (error) {
          console.log(error);
          res.status(400).send(error.message);
        }
        else{
        //console.log(body);
        res.send({ data: body });
        }}));

});
app.use(function(error, req, res, next) {
  res.status(500).send(error.message);
});

app.listen(port, () => console.log(`App listening on port ${port}!`));
```