CSCI 5410: Assignment 4

Part B: Build an event-driven serverless application using GCP ML.

GitLab Link: https://git.cs.dal.ca/drshah/dhrumilrakeshshah_csci5410.git

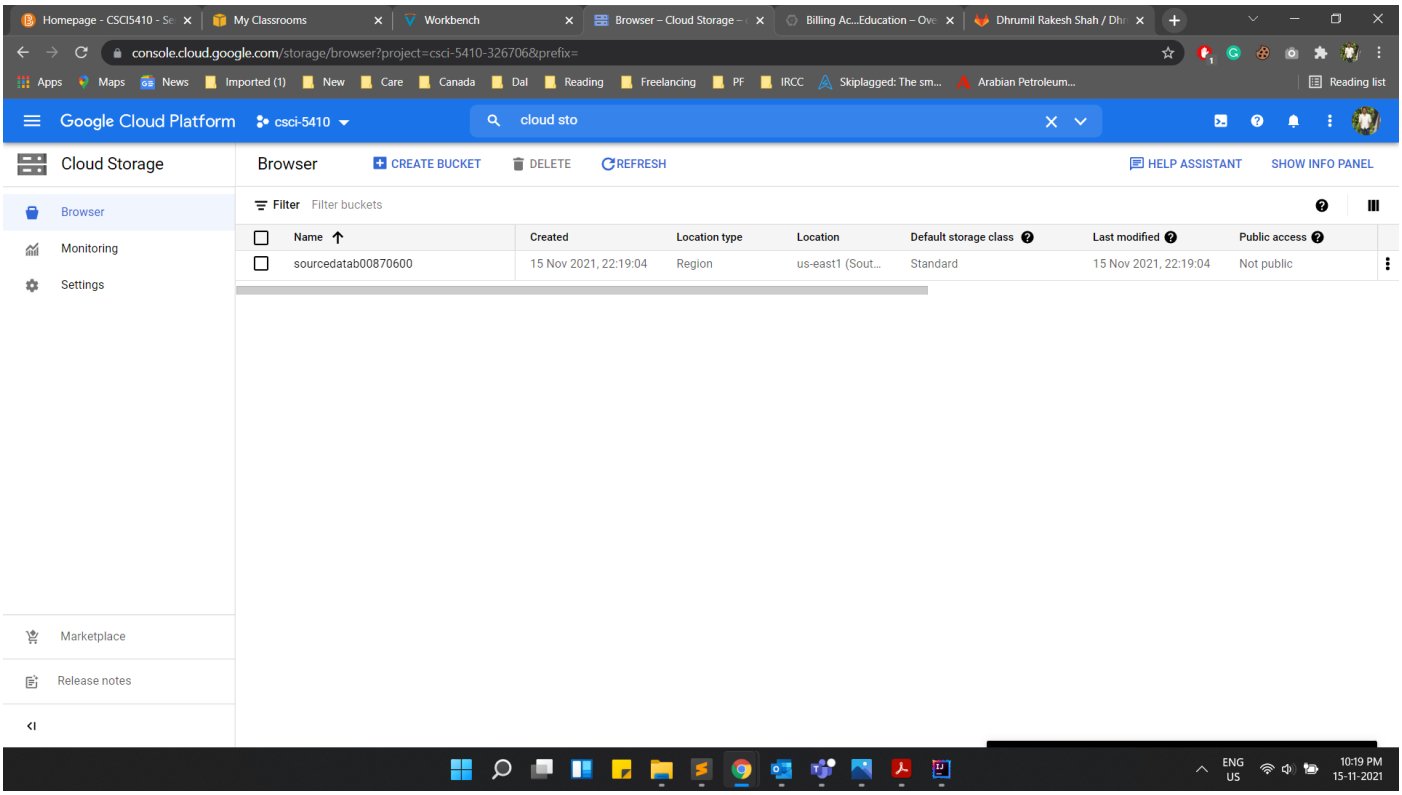

Figure 1: sourcedatab00870600 Bucket

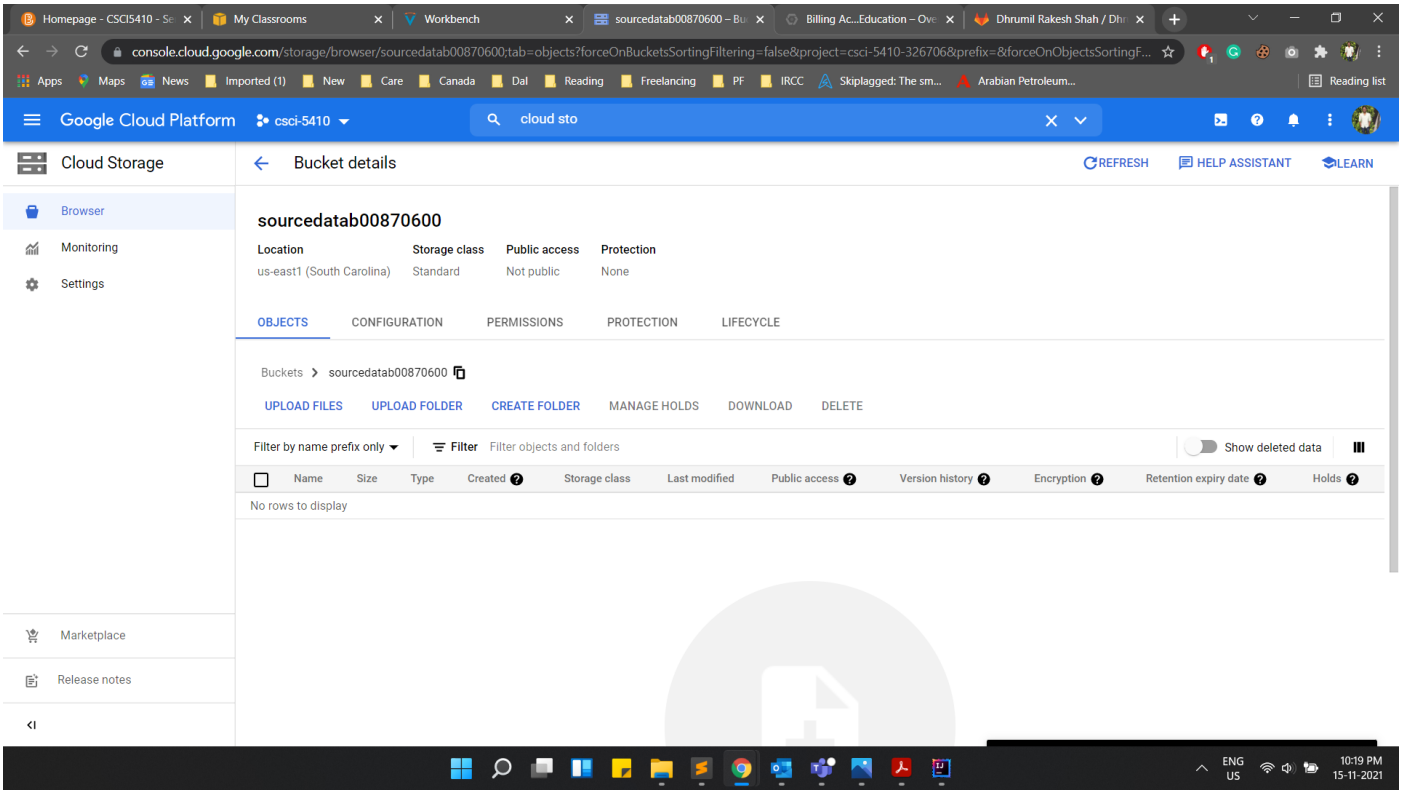

Figure 2: sourcedatab00870600 Empty Bucket

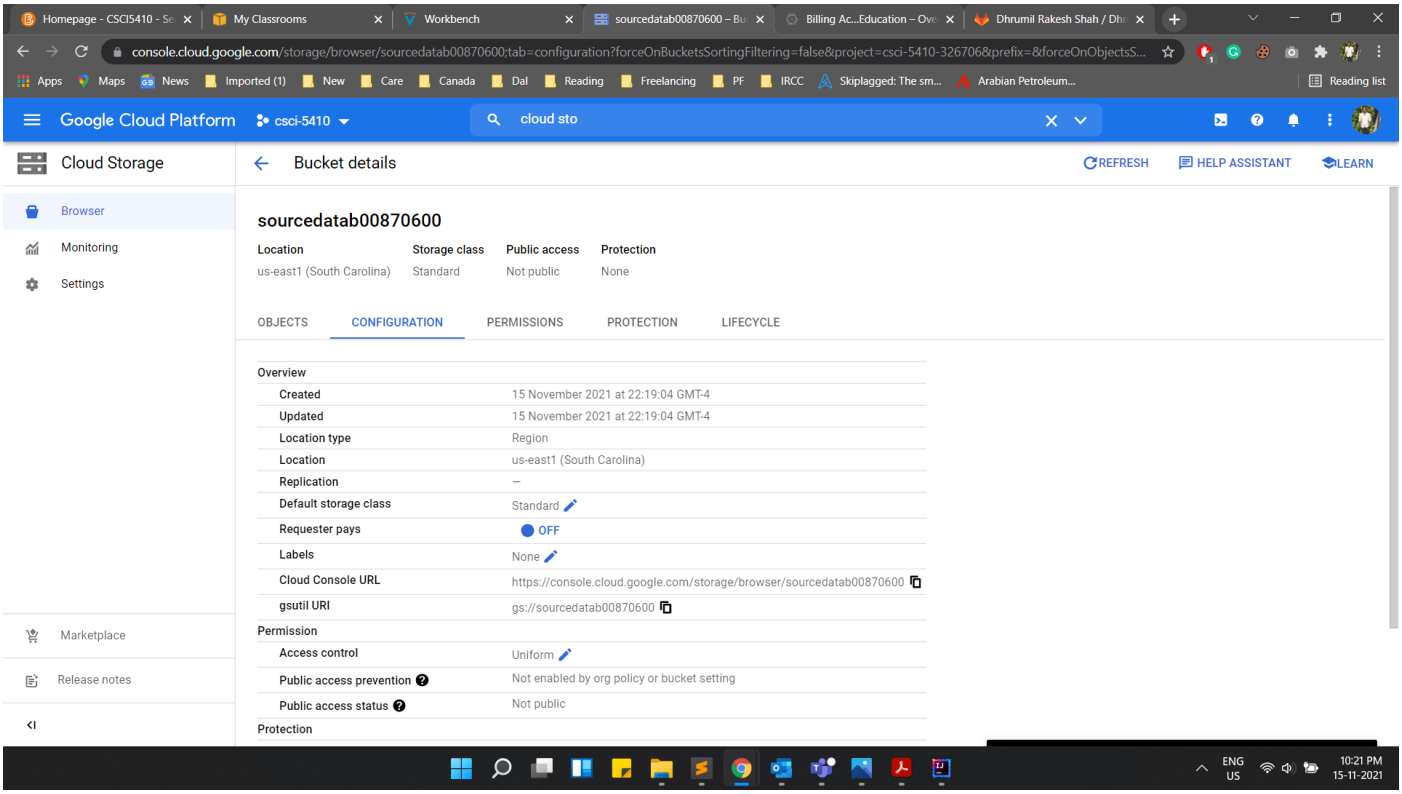

Figure 3: sourcedatab00870600 Bucket Config

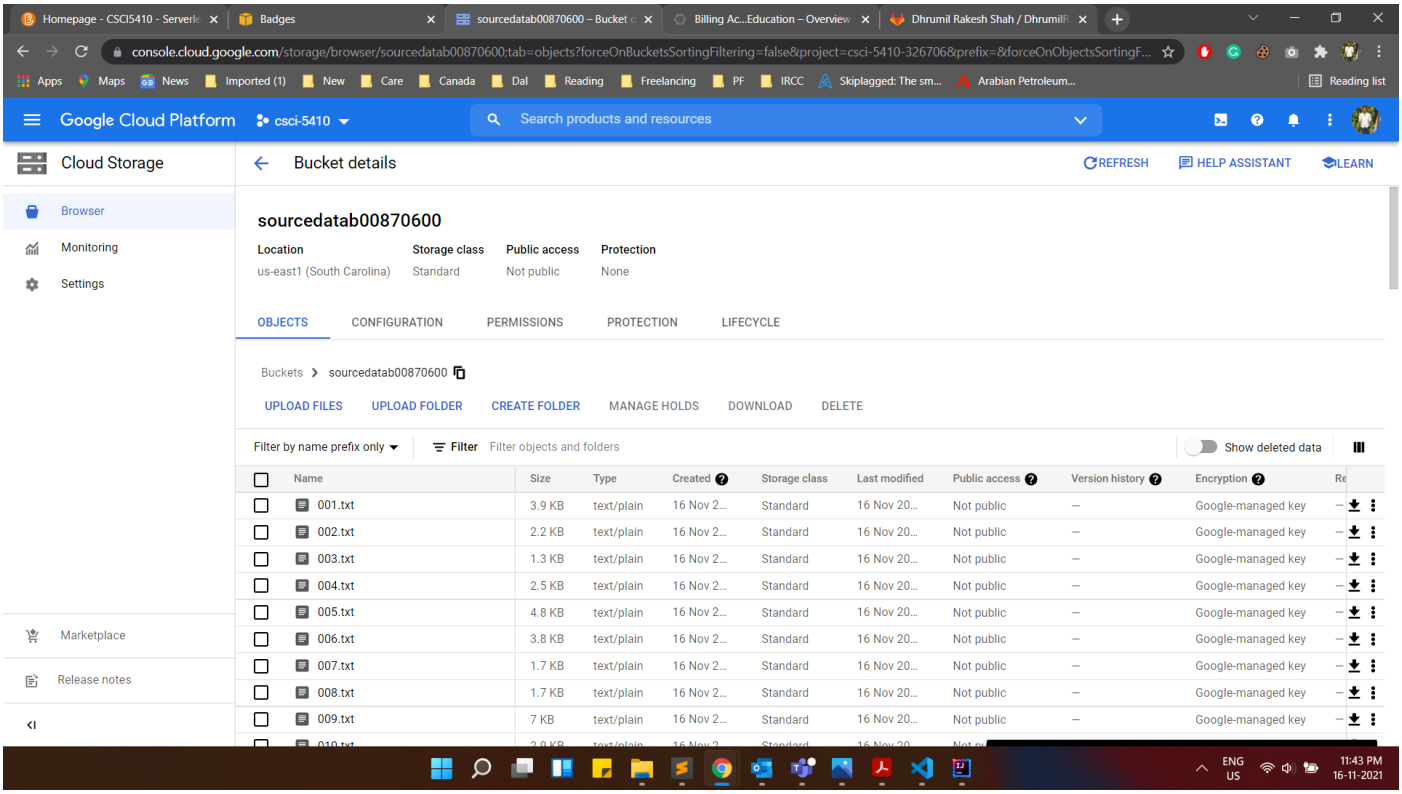

Figure 4: sourcedatab00870600 Bucket containing Train folder files

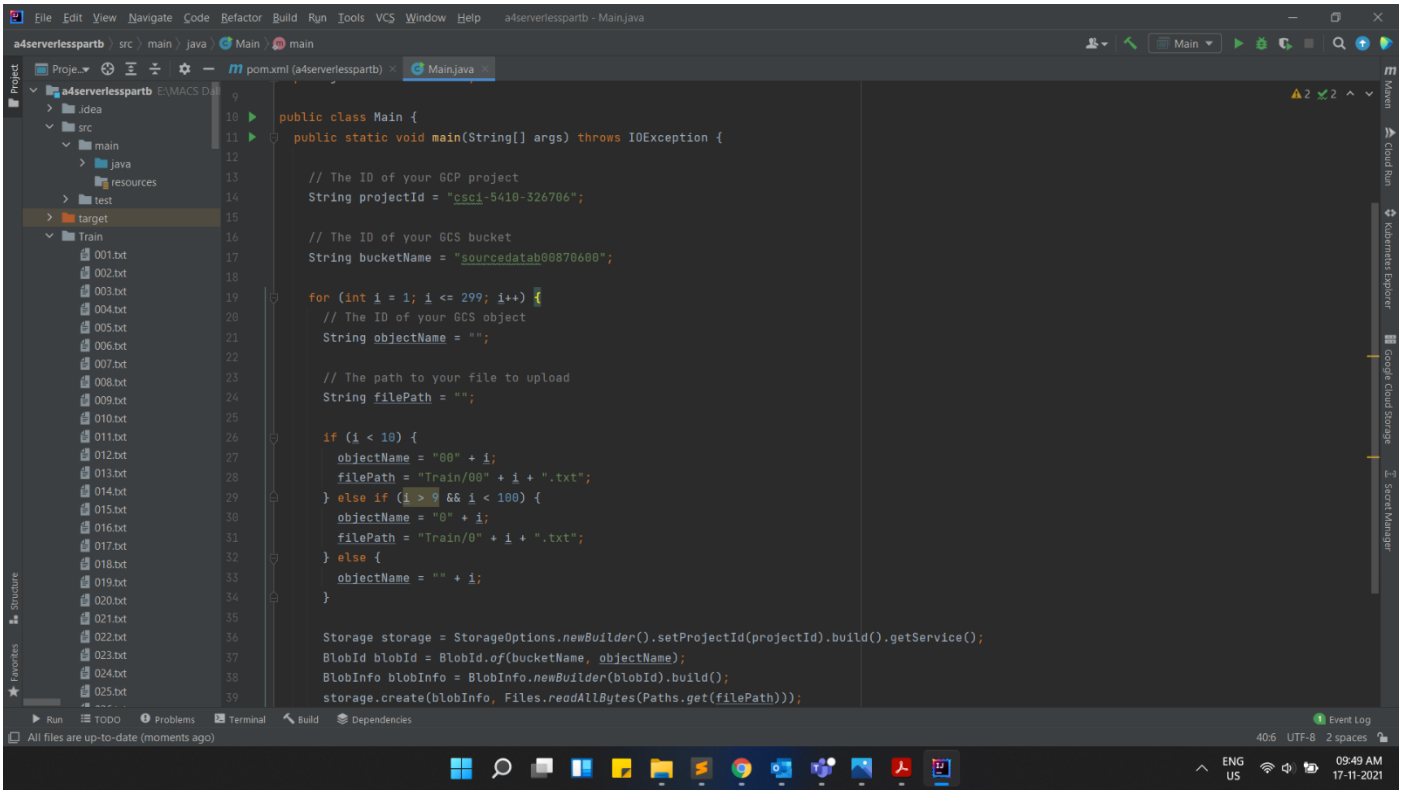

Figure 5: sourcedatab00870600 Bucket File Upload Code

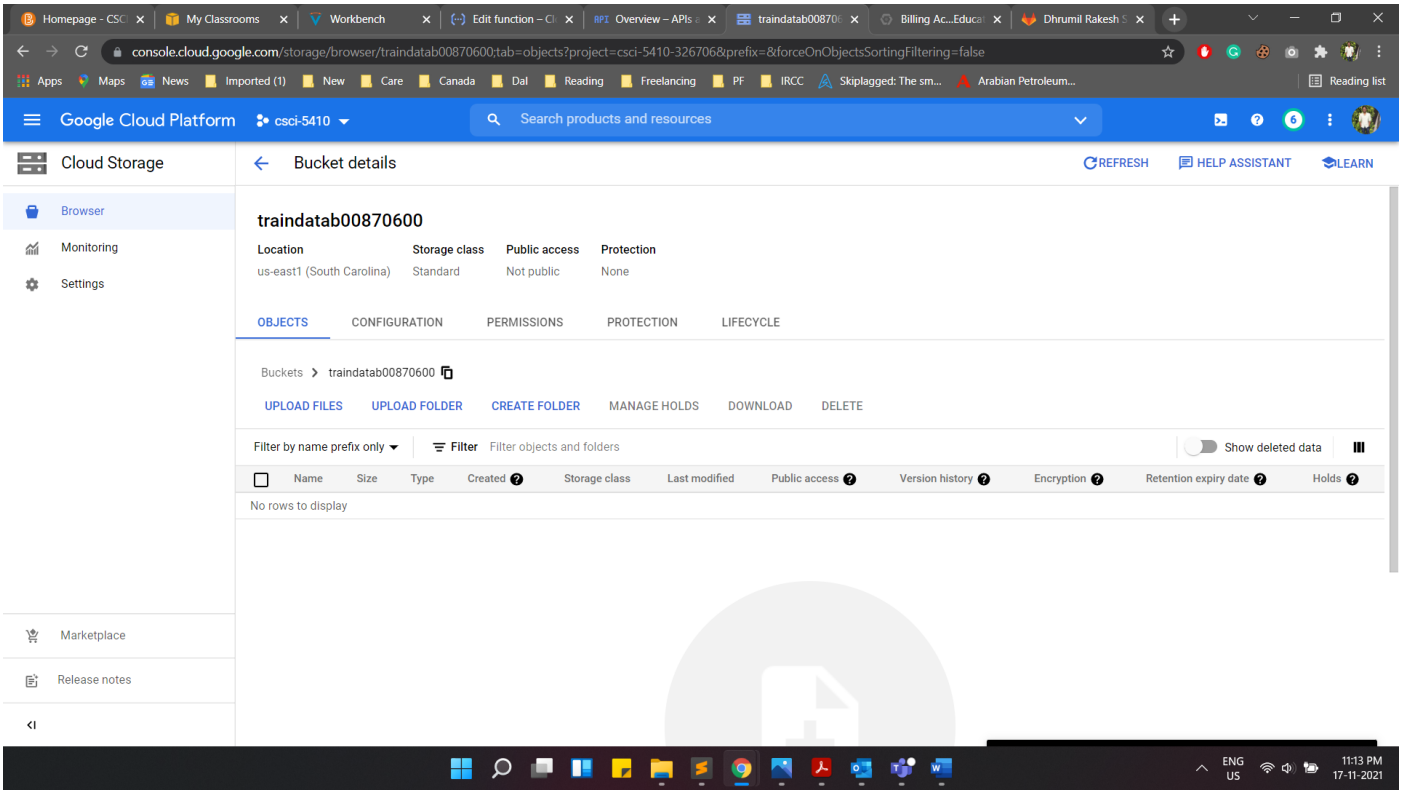

Figure 6: traindatab00870600 Empty Bucket

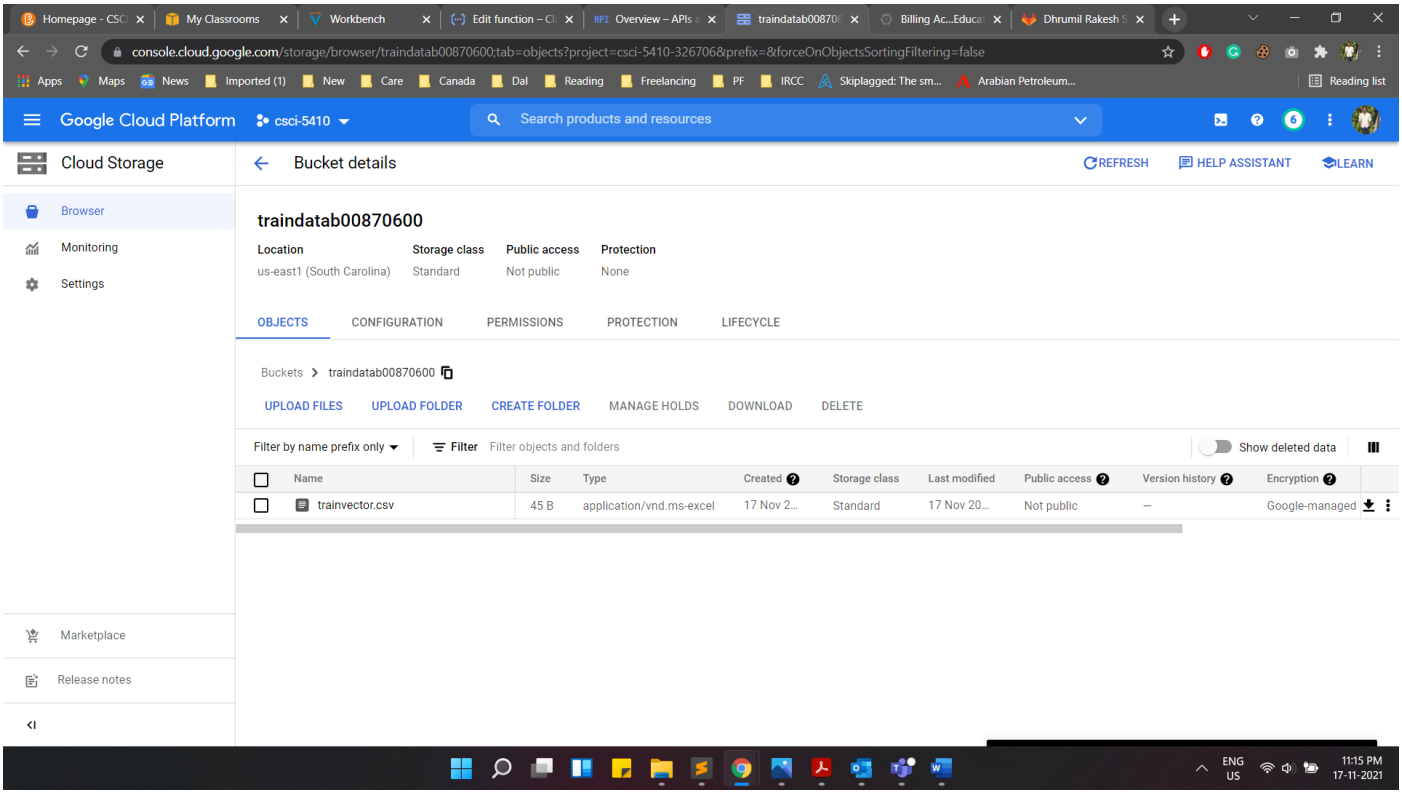

Figure 7: traindatab00870600 Bucket containing trainVector File

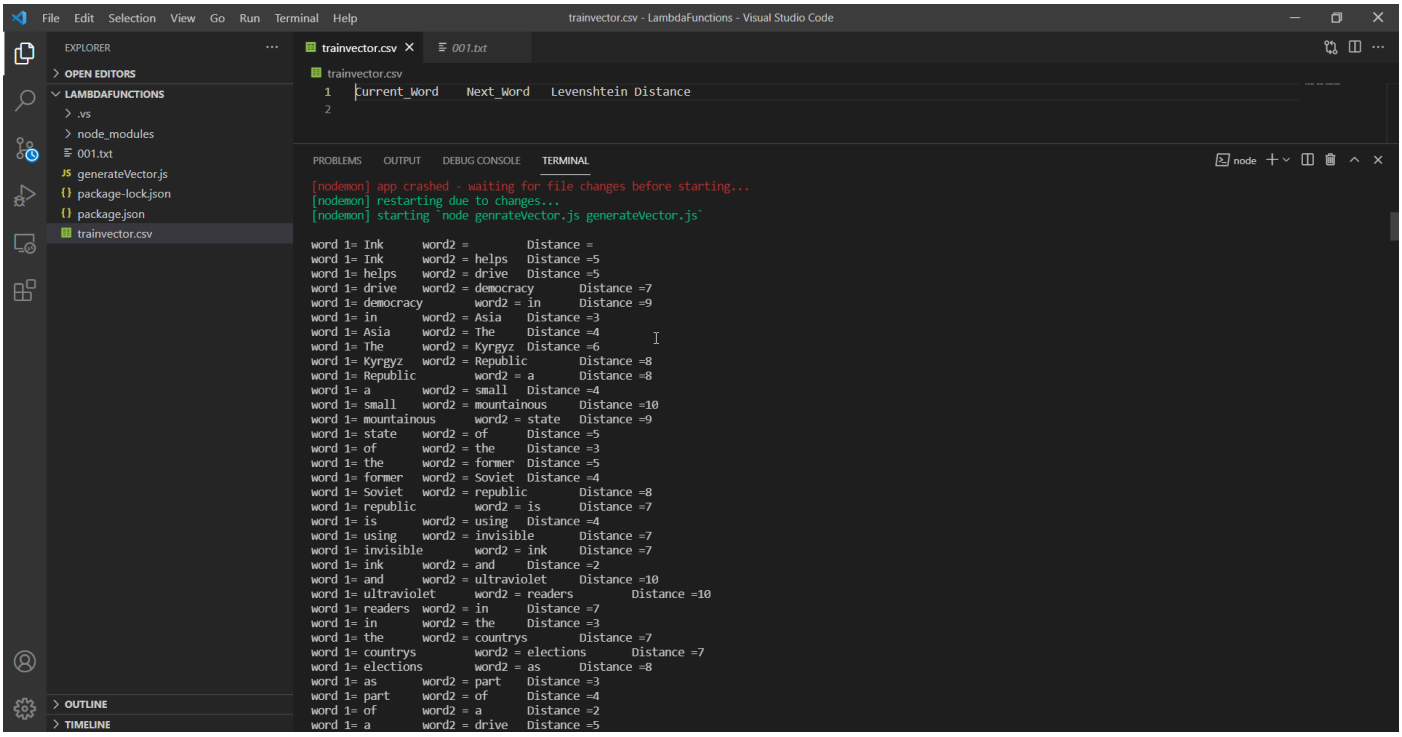

Figure 8: trainVector File Contents

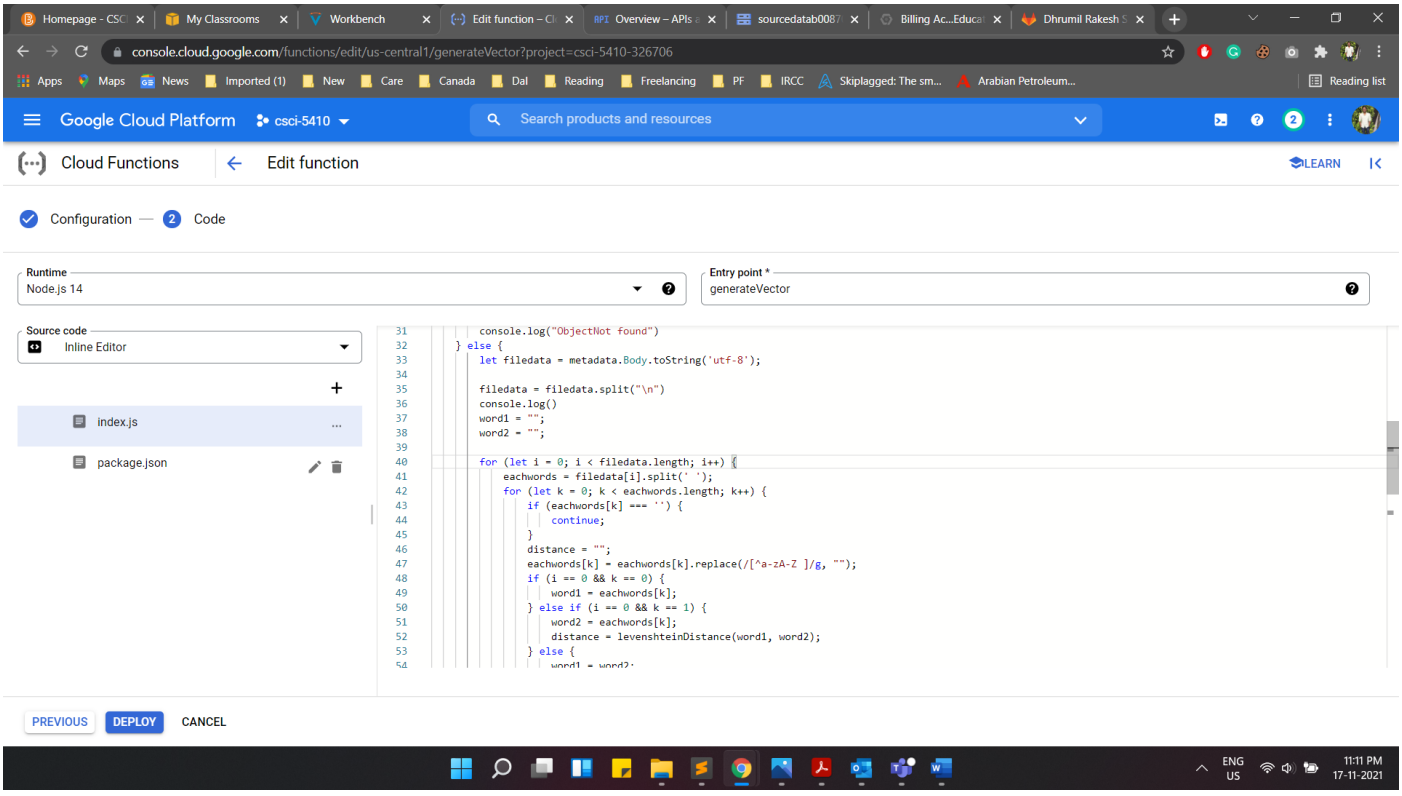

Figure 9: generateVector Cloud Function Code

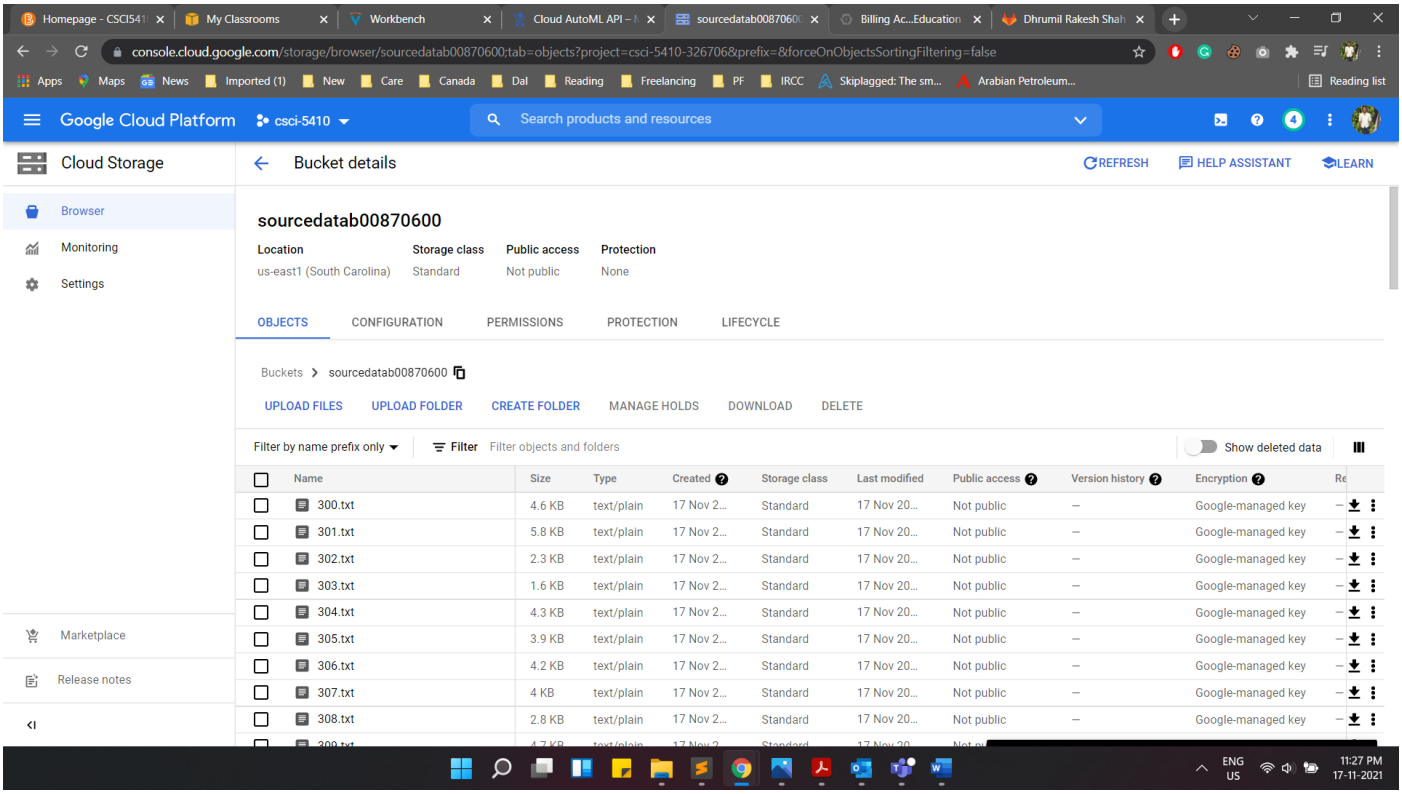

Figure 10: sourcedatab00870600 Bucket containing Test Files

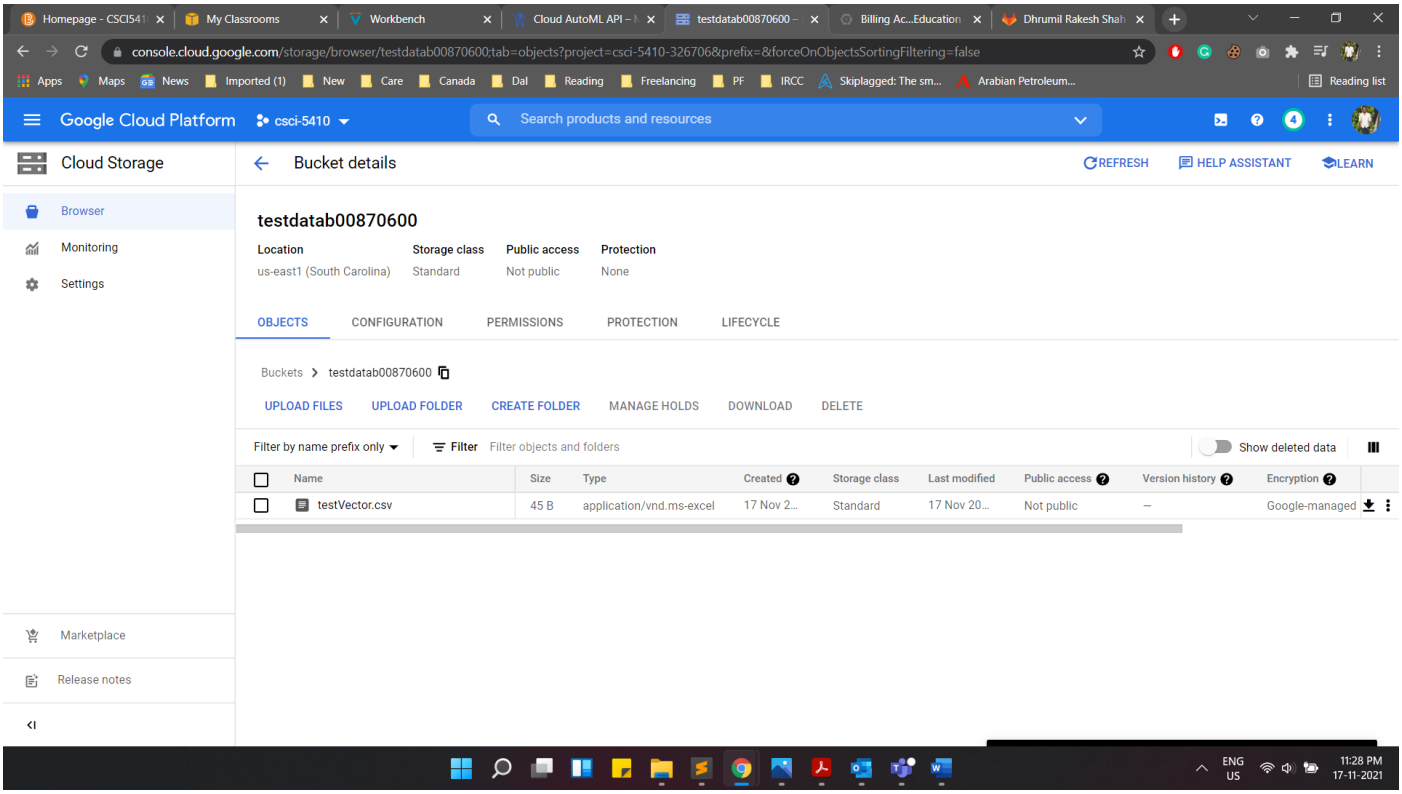

Figure 11: testdatab00870600 Bucket containing testVector File

generateVector Code:

```
/**
 * Responds to any HTTP request.
 *
 * @param {!express:Request} req HTTP request context.
 * @param {!express:Response} res HTTP response context.
 */
exports.helloWorld = (req, res) => {
  let message = req.query.message || req.body.message || 'Hello World!';
  res.status(200).send(message);
    if (err && err.code === 'NotFound') {
        console.log("ObjectNot found")
    } else {
        let filedata = metadata.Body.toString('utf-8');

        filedata = filedata.split("\n")
        console.log()
        word1 = "";
        word2 = "";

        for (let i = 0; i < filedata.length; i++) {
            eachwords = filedata[i].split(' ');
            for (let k = 0; k < eachwords.length; k++) {
                if (eachwords[k] === '') {
                    continue;
                }
                distance = "";
                eachwords[k] = eachwords[k].replace(/[^a-zA-Z ]/g, "");
                if (i == 0 && k == 0) {
                    word1 = eachwords[k];
                } else if (i == 0 && k == 1) {
                    word2 = eachwords[k];
                    distance = levenshteinDistance(word1, word2);
                } else {
                    word1 = word2;
                    word2 = eachwords[k];
                    distance = levenshteinDistance(word1, word2);
                }
                console.log("word 1= " + word1 + "\t word2 = " + word2 + "\t Distance =" +
distance);
            }
        }
    }

const levenshteinDistance = (str1 = '', str2 = '') => {
    const track = Array(str2.length + 1).fill(null).map(() =>
        Array(str1.length + 1).fill(null));
    for (let i = 0; i <= str1.length; i += 1) {
        track[0][i] = i;
```

```
    }
    for (let j = 0; j <= str2.length; j += 1) {
        track[j][0] = j;
    }
    for (let j = 1; j <= str2.length; j += 1) {
        for (let i = 1; i <= str1.length; i += 1) {
            const indicator = str1[i - 1] === str2[j - 1] ? 0 : 1;
            track[j][i] = Math.min(
                track[j][i - 1] + 1, // deletion
                track[j - 1][i] + 1, // insertion
                track[j - 1][i - 1] + indicator, // substitution
            );
        }
    }
    return track[str2.length][str1.length];
};
};
```

Main.java Code:

```
/**
 * Responds to any HTTP request.
 *
 * @param {!express:Request} req HTTP request context.
 * @param {!express:Response} res HTTP response context.
 */
exports.helloWorld = (req, res) => {
  let message = req.query.message || req.body.message || 'Hello World!';
  res.status(200).send(message);
    if (err && err.code === 'NotFound') {
        console.log("ObjectNot found")
    } else {
        let filedata = metadata.Body.toString('utf-8');

        filedata = filedata.split("\n")
        console.log()
        word1 = "";
        word2 = "";

        for (let i = 0; i < filedata.length; i++) {
            eachwords = filedata[i].split(' ');
            for (let k = 0; k < eachwords.length; k++) {
                if (eachwords[k] === '') {
                    continue;
                }
                distance = "";
                eachwords[k] = eachwords[k].replace(/[^a-zA-Z ]/g, "");
                if (i == 0 && k == 0) {
```

```
                    word1 = eachwords[k];
                } else if (i == 0 && k == 1) {
                    word2 = eachwords[k];
                    distance = levenshteinDistance(word1, word2);
                } else {
                    word1 = word2;
                    word2 = eachwords[k];
                    distance = levenshteinDistance(word1, word2);
                }
                console.log("word 1= " + word1 + "\t word2 = " + word2 + "\t Distance =" +
distance);
            }
        }
    }

const levenshteinDistance = (str1 = '', str2 = '') => {
    const track = Array(str2.length + 1).fill(null).map(() =>
        Array(str1.length + 1).fill(null));
    for (let i = 0; i <= str1.length; i += 1) {
        track[0][i] = i;
    }
    for (let j = 0; j <= str2.length; j += 1) {
        track[j][0] = j;
    }
    for (let j = 1; j <= str2.length; j += 1) {
        for (let i = 1; i <= str1.length; i += 1) {
            const indicator = str1[i - 1] === str2[j - 1] ? 0 : 1;
            track[j][i] = Math.min(
                track[j][i - 1] + 1, // deletion
                track[j - 1][i] + 1, // insertion
                track[j - 1][i - 1] + indicator, // substitution
            );
        }
    }
    return track[str2.length][str1.length];
};
};
```