```
const express = require('express');
const app = express();
const port = 3000;

var AWS = require('aws-sdk');
AWS.config.update({
    // Put in your access key ID
    accessKeyId: "",
    // Put in your secret access key
    secretAccessKey: "",
    region: 'eu-west-1'
});
const TABLE = 'Movies';
const BUCKET = "csu44000assignment220";
const OBJECT = "moviedata.json";
const INSERT_SET_SIZE = 25;
var s3 = new AWS.S3();
var ddb = new AWS.DynamoDB();

async function sendClientFile(_, res) {
    res.sendFile(__dirname + "/" + "client.html");
}

async function create(_, res) {
    var exists = await checkTableExists();
    if(exists) {
        console.log("Exists");
        returnMessage(res, "success", {});
        return;
    }

    console.log("Creating Table...");
    var json = await getS3Object(BUCKET, OBJECT);

    await createTable();
    await ddb.waitFor('tableExists', { TableName: TABLE}).promise();
    await insertIntoTable(json);
    console.log('Table Created');
    returnMessage(res, "success", {});
    return;
}

async function query(req, res){
    var movie = req.get('Movie');
    var year = parseInt(req.get('Year'));
    var rating = parseInt(req.get('Rating'));

    if(isNaN(year)) {
```

```
        returnMessage(res, "Failed, year not set", {});
    } else if(isNaN(rating)) {
        returnMessage(res, "Failed, rating not set", {})
    } else if(movie.length == 0) {
        returnMessage(res, "Failed, no starting text given", {})
    } else {
        console.log('Querying...');
        let data = await queryTable(movie.toLowerCase(), year.toString(),
rating.toString());
        console.log('Query Successful');
        returnMessage(res, "Success", data);
    }
}

async function destroy(_, res) {
    var exists = await checkTableExists();
    if(!exists) {
        console.log("Doesn't Exists");
        returnMessage(res, "success", {});
        return;
    }

    console.log("Deleting Table...");
    var params = { TableName: TABLE };
    await ddb.deleteTable(params).promise();
    console.log("Table Deleted");
    returnMessage(res, "success", {});
}

async function checkTableExists() {
    var allTables = await ddb.listTables({}).promise();
    return allTables.TableNames.includes(TABLE);
}

async function getS3Object() {
    var params = {
        Bucket: BUCKET,
        Key: OBJECT
    };
    var data = await s3.getObject(params).promise();
    return JSON.parse(data.Body.toString('utf-8'));
}

async function createTable() {
    var params = {
        TableName: TABLE,
        KeySchema: [
            { AttributeName: 'title', KeyType: 'HASH' },
            { AttributeName: 'release_date', KeyType: 'RANGE'}
```

```
        ],
        AttributeDefinitions: [
            { AttributeName: 'title', AttributeType: 'S' },
            { AttributeName: 'release_date', AttributeType: 'N' }
        ],
        ProvisionedThroughput: {
            ReadCapacityUnits: 1,
            WriteCapacityUnits: 1
        }
    }
    await ddb.createTable(params).promise();
}

async function insertIntoTable(json) {
    var sets = [], movies = [];
    for(var i = 0; i < json.length; i++) {
        if(movies.length == INSERT_SET_SIZE) {
            sets.push(movies);
            movies = [];
        }

        title = json[i].title;
        lowerTitle = json[i].title.toLowerCase();
        if(json[i].hasOwnProperty('year')) year = json[i].year.toString();
        else year = '-1';
        if(json[i].info.hasOwnProperty('rating')) rating =
json[i].info.rating.toString();
        else rating = '-1';
        if(json[i].info.hasOwnProperty('rank')) rank =
json[i].info.rank.toString();
        else rank = '-1';

        movies.push({
            PutRequest: {
                Item: {
                    title: { 'S': title },
                    release_date: { 'N': year },
                    rating: { 'N': rating },
                    lowerCaseTitle: {'S': lowerTitle},
                    rank: { 'N': rank }
                }
            }
        });
    }
    if(movies.length != 0) sets.push(movies);

    for(var i = 0; i < sets.length; i++) {
        var percentComplete = (((i+1) / (sets.length+1)) * 100).toFixed(2);
```

```
143         process.stdout.write(`\rData insertion progress:
${percentComplete}% complete\r`);
144         await ddb.batchWriteItem({ RequestItems: { [TABLE]: sets[i] }
}).promise();
145     }
146     process.stdout.write(`\rData insertion progress: 100% complete\r\n`)
147 }
148
149 async function queryTable(movie, year, rating) {
150     var params = {
151         TableName: TABLE,
152         ExpressionAttributeValues: {
153             ':y': {N: year},
154             ':t': {S: movie},
155             ':r': {N: rating}
156         },
157         FilterExpression: 'release_date = :y and begins_with
(lowerCaseTitle, :t) and rating >= :r'
158     }
159
160     var res = await ddb.scan(params).promise();
161     var data = [];
162
163     res.Items.forEach(function(item, _, _) {
164         data.push({
165             title: item.title.S,
166             year: item.release_date.N,
167             rating: item.rating.N,
168             rank: item.rank.N
169         });
170     });
171
172     return data;
173 }
174
175 function returnMessage(res, message, data) {
176     body = {
177         message: message,
178         data: data
179     };
180     res.send(body);
181     return;
182 }
183
184 app.use(express.json());
185 app.use(express.static('public'));
186 app.get('/', sendClientFile);
187 app.get('/create', create);
188 app.get('/query', query);
```

```
189 app.get('/destroy', destroy);
190 app.listen(port, () => console.log(`Example app listening on port
    ${port}!`));
```