```
const express = require('express');
const app = express();
const port = 3000;
const fetch = require('node-fetch');
const path = require('path');
let publicPath = path.resolve(__dirname, 'public');
let AWS = require('aws-sdk');
AWS.config.update({region: 'eu-west-1'});

app.use(express.static(publicPath));
app.get('/appCreate', appCreate);
app.get('/appDelete', appDelete);
app.get('/appQuery/:year/:prefix', appQuery);
app.listen(port, () => console.log(`App listening on port ${port}`));

const S3_BUCKET = 'cs4000-a2';
const S3_OBJECT = 'moviedata.json';
const DB_TABLE = 'movies';
const BATCH_SIZE = 25;

let s3 = new AWS.S3();
let dd = new AWS.DynamoDB();
let dc = new AWS.DynamoDB.DocumentClient();

async function appCreate(_, res) {
    console.log('=== CREATE ===');

    let tableExists = await checkDynamoTableExists(DB_TABLE);
    if (tableExists) {
        res.json({result: {
            success: false,
            message: 'Table already exists'
        }});
        return;
    }

    console.log('Fetching data...');
    let json = await getS3Object(S3_BUCKET, S3_OBJECT);
    console.log('Data fetched!');

    console.log('Creating table...');
    await createDynamoTable(DB_TABLE);
    await dd.waitFor('tableExists', { TableName: DB_TABLE }).promise(); // wait until
table has finished created
    console.log('Table created!');
    console.log('Inserting data...');
    await insertIntoDynamoTable(DB_TABLE, json);
    console.log('Data inserted!');

    res.json({result: {
        success: true,
        message: 'Creation successful!'
    }});
}
```

```javascript

async function appDelete(_, res) {
    console.log('=== DELETE ===');

    let tableExists = await checkDynamoTableExists(DB_TABLE);
    if (!tableExists) {
        res.json({result: {
            success: false,
            message: 'Table doesn\'t exist'
        }});
        return;
    }

    console.log('Deleting table...');
    await deleteDynamoTable(DB_TABLE);
    console.log('Table deleted!');

    res.json({result: {
        success: true,
        message: 'Deletion successful!'
    }});
}

async function appQuery(req, res) {
    console.log('=== QUERY ===');

    let year = parseInt(req.params.year, 10);
    let prefix = req.params.prefix;
    if (prefix == '_') prefix = '';

    if (isNaN(year)) {
        res.json({result: {
            success: false,
            message: 'Invalid year',
            movies: {}
        }})
    } else {
        console.log('Fetching results...');
        let data = await queryDynamoTable(DB_TABLE, year.toString(),
prefix.toLowerCase());
        console.log('Results fetched!');

        res.json({result: {
            success: true,
            message: 'OK',
            movies: data
        }});
    }
}

/* Helper functions which use the AWS SDK */

async function checkDynamoTableExists(tableName) {
    let data = await dd.listTables({}).promise();
    return data.TableNames.includes(tableName);
```

```
}

async function getS3Object(bucketName, objectName) {
    let params = {
        Bucket: S3_BUCKET,
        Key: S3_OBJECT
    };
    let data = await s3.getObject(params).promise();
    return JSON.parse(data.Body.toString('utf-8'));
}

async function createDynamoTable(tableName) {
    let params = {
        AttributeDefinitions: [
            { AttributeName: 'titleLower', AttributeType: 'S' },
            { AttributeName: 'releaseYear', AttributeType: 'N' },
        ],
        KeySchema: [
            { AttributeName: 'titleLower', KeyType: 'HASH' },
            { AttributeName: 'releaseYear', KeyType: 'RANGE' }
        ],
        ProvisionedThroughput: {
            ReadCapacityUnits: 5,
            WriteCapacityUnits: 5
        },
        TableName: tableName
    };
    await dd.createTable(params).promise();
}

async function insertIntoDynamoTable(tableName, json) {
    let batches = [], batch = [];
    for (var i = 0; i < json.length; i++) {
        if (batch.length == BATCH_SIZE) {
            batches.push(batch);
            batch = [];
        }
        batch.push({
            PutRequest: {
                Item: {
                    titleLower: {'S': json[i].title.toLowerCase() },
                    releaseYear: {'N': json[i].year?.toString() ?? '-1' },
                    title: {'S': json[i].title },
                    rating: {'N': json[i].info.rating?.toString() ?? '-1' }
                }
            }
        });
    }
    if (batch.length != 0) batches.push(batch);

    for (var i = 0; i < batches.length; i++) {
        console.log(`Inserting data batch ${i + 1}/${batches.length}`);
        await dd.batchWriteItem({ RequestItems: { [tableName]: batches[i] }
}).promise();
    }
```

```

}

async function deleteDynamoTable(tableName) {
    let params = { TableName: tableName };
    await dd.deleteTable(params).promise();
}

async function queryDynamoTable(tableName, year, prefix) {
    let params = {
        ExpressionAttributeValues: {
            ':y': {N: year},
            ':p': {S: prefix}
        },
        FilterExpression: 'releaseYear = :y and begins_with (titleLower, :p)',
        ProjectionExpression: 'title, releaseYear, rating',
        TableName: tableName
    }

    let raw = await dd.scan(params).promise();
    let data = [];

    raw.Items.forEach(function (item, _, _) {
        data.push({
            title: item.title.S,
            year: item.releaseYear.N,
            rating: item.rating.N
        });
    });

    return data;
}
```