```
const express = require('express');
const app = express();
const port = 3000;
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', appQuery); // in case no prefix is included
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();

async function appCreate(_, res) {
    let tableExists = await checkDynamoTableExists(DB_TABLE);
    if (tableExists) {
        res.json(generateResponse(false, 'Table already exists', {}));
        return;
    }

    console.log('Creating...');
    let json = await getS3Object(S3_BUCKET, S3_OBJECT);

    await createDynamoTable(DB_TABLE);
    await dd.waitFor('tableExists', { TableName: DB_TABLE }).promise(); // wait until
table has finished creating
    await insertIntoDynamoTable(DB_TABLE, json);
    console.log('Done!');

    res.json(generateResponse(true, 'Creation successful!', {}));
}

async function appDelete(_, res) {
    let tableExists = await checkDynamoTableExists(DB_TABLE);
    if (!tableExists) {
        res.json(generateResponse(false, 'Table doesn\'t exist', {}));
        return;
    }

    console.log('Deleting...')
    await deleteDynamoTable(DB_TABLE);
    console.log('Done!')

    res.json(generateResponse(true, 'Deletion successful!', {}));
```

```
54 }
55
56 async function appQuery(req, res) {
57     let year = parseInt(req.params.year, 10);
58     let prefix = req.params.prefix ?? '';
59
60     if (isNaN(year)) {
61         res.json(generateResponse(false, 'Invalid year', {}));
62     } else {
63         console.log('Querying...');
64         let data = await queryDynamoTable(DB_TABLE, year.toString(),
   prefix.toLowerCase());
65         console.log('Done!');
66         res.json(generateResponse(true, 'OK', data));
67     }
68 }
69
70 /* Helper functions which use the AWS SDK */
71
72 async function checkDynamoTableExists(tableName) {
73     let data = await dd.listTables({}).promise();
74     return data.TableNames.includes(tableName);
75 }
76
77 async function getS3Object(bucketName, objectName) {
78     let params = {
79         Bucket: S3_BUCKET,
80         Key: S3_OBJECT
81     };
82     let data = await s3.getObject(params).promise();
83     return JSON.parse(data.Body.toString('utf-8'));
84 }
85
86 async function createDynamoTable(tableName) {
87     let params = {
88         AttributeDefinitions: [
89             { AttributeName: 'titleLower', AttributeType: 'S' },
90             { AttributeName: 'releaseYear', AttributeType: 'N' },
91         ],
92         KeySchema: [
93             { AttributeName: 'titleLower', KeyType: 'HASH' },
94             { AttributeName: 'releaseYear', KeyType: 'RANGE' }
95         ],
96         ProvisionedThroughput: {
97             ReadCapacityUnits: 5,
98             WriteCapacityUnits: 5
99         },
100        TableName: tableName
101    };
102    await dd.createTable(params).promise();
103 }
104
105 async function insertIntoDynamoTable(tableName, json) {
106     // DynamoDB only allows batch inserts of 25 items
107     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;
}
```

```
162
163 /* Other helper functions */
164
165 function generateResponse(_success, _message, _movies) {
166     return { result: {
167         success: _success,
168         message: _message,
169         movies: _movies
170     }};
171 }
```