



ACIK Enterprises (OPC) Pvt Ltd


# Internship Screen Test

October 30, 2020

## Introduction

Thank you for showing interest in being part of our Internship program. At ACIK, we are always looking to engage with highly motivated and self-driven developers to grow our platform and team together. We are a product company with a strong focus on developing our flagship platform, Omnibus, on solid, modern, scalable technology using industry-standard Agile practices.

We are delighted with the overwhelming interest to be a part of our internship program, and we have decided to develop a screening test to help us make choices for our limited posts. We also believe this to be a great learning opportunity for you. In this document, we have described a practical problem statement - a significantly simplified semblance of the final problem. We are looking for developers who can code quickly and accurately.



## Table of Contents

[Introduction](#)

[Table of Contents](#)

[Problem Statement](#)

[Prerequisites:](#)

[Specifications](#)

[Figure 1.1](#)

[Create Product](#)

[Mark product unsellable](#)

[Delete product](#)

[List all sellable products](#)

[List all out of stock products](#)

[List top N sold products](#)

[Customer APIs](#)

[Place Order](#)

[Note:](#)

[Technology Stack](#)

[Submission](#)

## Problem Statement

As part of this exercise, you are required to build a simplified 'Online DVD Sales Platform' with the requirements mentioned below.

The platform will be used by people who plan to sell DVDs online. The sellers know which movie DVDs they have, and want to meet two objectives from this platform - Enrich the data by invoking a third-party service, Online Movie Database ([OMDb](#)), and add store-specific data like price and inventory.

For example, a seller, "Ravi DVD Centre" has 5 DVDs of "Harry Potter and the Sorcerer's Stone". On invoking the OMDb API, the platform will get information about the movie like Genre, Director, Release Year etc. The seller will also provide information like MRP (Rs 200), Selling Price (Rs 175), Inventory (5 pieces) etc. This combined information will be stored in the database for subsequent consumption.

The following section describes each API in detail with the expected responses. Developers are required to ensure that the inputs are thoroughly validated and appropriate error messages are returned if invalid data is passed. Developers should also handle third-party API failures and surface them appropriately.

## Prerequisites:

Create a free tier account with [OMDb RESTful web service](#) and make note of the API Key for future use.

## Specifications

Every product that is being sold on the platform consists of the following attributes:

Attribute name
Movie title
Release year
Length
Director
Plot
Cast
IMDB rating
Language
Genre
MRP
Inventory
Selling price
Shipping fee
Is sellable
Sold unit count

**Figure 1.1**

## Seller APIs

During all Seller API request calls, the request header `x-auth` should be set to `tst-seller`.

### Create Product

**POST** /api/v1/dvd-online/seller/products/create

```
{
  "movieTitle": "",
  "releaseYear": ""
}
```

Response:

```
{
  "err": false,
  "data": {
    "movieTitle": "",
    "releaseYear": "",
    "length": "",
    "director": "",
    "plot": "",
    "cast": "",
    "imdb": "",
    "language": "",
    "genre": "",
    "productId": ""
  },
  "message": "Product has been created successfully"
}
```

Using the movieTitle and releaseYear entered by the seller, fetch the corresponding movie information using the [OMDb RESTful web service](#) and save the product. The saved product should now have the attributes mentioned in **Grey** in the **Figure 1.1**. The response should contain the movie information and the id corresponding to that entry.

## Sell product

**POST** /api/v1/dvd-online/seller/products/sell

```
{
  "productId": "",
  "sellingPrice": "",
  "inventory": "",
  "mrp": "",
  "shippingPrice": ""
}
```

Response:

```
{
  "err": false,
  "message": "Product price & inventory saved successfully"
}
```

On executing this API, the price & inventory information of the product should be updated. The product should contain all the fields mentioned in **Figure 1.1**. *Is Sellable* should be set true. *Sold unit count* attribute should be set to a default value of 0 if this field is absent in the product.

## Mark product unsellable

**POST** /api/v1/dvd-online/seller/products/unsell

```
{
  "productId": ""
}
```

Response:

```
{
  "err": false,
  "message": "Product has been marked unsellable successfully"
}
```

## Delete product

**DELETE** /api/v1/dvd-online/seller/products/{:productId}

Response:

```
{
  "err": false,
  "message": "The product has been deleted successfully"
}
```

## List all sellable products

**GET** /api/v1/dvd-online/seller/products/list/sellable

Response:

```
{
  "err": false,
  "data": [{
    "movieTitle": "",
    "releaseYear": "",
    "length": "",
    "director": "",
    "plot": "",
    "cast": "",
    "imdb": "",
    "language": "",
    "genre": "",
    "productId": "",
    "sellingPrice": "",
    "inventory": "",
    "soldUnitCount": "",
    "mrp": "",
    "shippingPrice": ""
  }],
  "message": "Listed all sellable products successfully"
}
```

### List all out of stock products

**GET** /api/v1/dvd-online/seller/products/list/out-of-stock

#### Response:

Signature of response would be similar to that of previous API. On executing this API, all the products which are sellable and have inventory == 0 should be listed.

### List top N sold products

**GET** /api/v1/dvd-online/seller/products/list/top-n-sold/{:count}

#### Response:

Signature of response would be similar to that of previous API. On executing this API, the top *count* number of products should be listed in the descending order of sold unit counts.



## Customer APIs

During all Customer API request calls, the request header `x-auth` should be set to `tst-customer`.

### Place Order

**POST** /api/v1/dvd-online/customer/place-order

```
{
  "productId1": 3,
  "productId2": 5,
  "productId3": 1
  .
  .
  .
}
```

### Response:

```
{
  "err": false,
  "data": {
    "cost": ""
  },
  "message": "Order has been placed successfully"
}
```

Input consists of a map, where key consists of productId and value consists of the inventory count. Calculate the total cost for the order and return in response, also decrease the inventory and increase the *Sold unit count* of the corresponding products according to the input.

**Ref:** Cost per product unit = Selling price + Shipping price

**Note:**

Customers should not be able to execute the Seller APIs and vice versa. This should be ensured by intercepting the request using middleware and authenticating the user by validating the `x-auth` request header.

Essentially, requests with `x-auth` header equal to `tst-customer` should not be able to access Seller APIs and requests with `x-auth` header equal to `tst-seller` should not be able to access Customer APIs.

# Technology Stack

[Node.js](#) with [Express.js](#) framework

[Typescript](#)

[Postgres](#)

[Sequelize](#)

All business logic should be written with Typescript.

## Submission

Ensure that your README.md file contains any specific instructions required for us to run or evaluate your project.

Once your project is successfully completed, remove the `node_modules` folder and compress it to a .zip format.

We will run `npm install` to install dependencies from `package.json` prior to testing your project.

**Important** - Submissions with `node_modules` folder will be rejected

Completed submissions can be uploaded to our [Form here](#)  
[\[https://forms.gle/g19ZhjjRqy6bs5mY8\]](https://forms.gle/g19ZhjjRqy6bs5mY8).

We will proceed with the fastest submissions and highest quality

Good Luck