You can view this report online at : https://www.hackerrank.com/x/tests/1527094/candidates/49983741/report

Full Name:	Seva Nandu
Email:	mangali.sevanandu@wipro.com
Test Name:	**Back-End Developer (Spring Boot) Test 2**
Taken On:	7 Feb 2023 20:10:00 IST
Time Taken:	89 min 52 sec/ 90 min
Work Experience:	< 1 years
Linkedin:	https://www.linkedin.com/in/sevanandu-m-525018185
Invited by:	Kavitha
Invited on:	7 Feb 2023 20:08:29 IST
Skills Score:	REST API (Intermediate) 15/50 Spring Boot (Basic) 50/50
Tags Score:	Back-End Development 15/50 Easy 65/100 Filtering 50/50 JSON 15/50 Java 50/50 Problem Solving 15/50 REST API 15/50 Sorting 50/50 Spring Boot 50/50

65%

65/100

scored in **Back-End Developer (Spring Boot) Test 2** in 89 min 52 sec on 7 Feb 2023 20:10:00 IST

Recruiter/Team Comments:

No Comments.

	Question Description	Time Taken	Score	Status
Q1	Spring Boot: Filter Microservice > **Back-end Developer**	35 min 39 sec	50/ 50	✓
Q2	REST API: Counting Movies > **Coding**	54 min 25 sec	15/ 50	✓

QUESTION 1

Correct Answer

Score 50

Spring Boot: Filter Microservice > Back-end Developer

Spring Boot | Java | Filtering | Sorting | Easy

QUESTION DESCRIPTION

Implement REST APIs to perform filter and sort operations on a collection of Products.

Each event is a JSON entry with the following keys:

- `barcode`: the unique id of the product (String)

- `price` : the price of the product (Integer)
- `discount` : the discount % available on the product(Integer)
- `available` : the availability status of the product (0 or 1)

Here is an example of a product JSON object:

```
[
    {
      "barcode": "74001755",
      "item": "Ball Gown",
      "category": "Full Body Outfits",
      "price": 3548,
      "discount": 7,
      "available": 1
    },
    {
      "barcode": "74002423",
      "item": "Shawl",
      "category": "Accessories",
      "price": 758,
      "discount": 12,
      "available": 1
    }
]
```

You are provided with the implementation of the models required for all the APIs. The task is to implement a set of REST services that exposes the endpoints and allows for filtering and sorting the collection of product records in the following ways:

`GET` request to `/filter/price/{initial_range}/{final_range}` :

- returns a collection of all products whose price is between the initial and the final range supplied
- The response code is 200, and the response body is an array of products in the price range provided.
- In case there are no such products return status code 400.

`GET` request to `/sort/price` :

- returns a collection of all products sorted by their pricing
- The response code is 200 and the response body is an array of the product names sorted in ascending order of price.

Complete the given project so that it passes all the test cases when running the provided unit tests.

▼ Example requests and responses

`GET` request to `/filter/price/{initial_range}/{final_range}`

The response code is 200, and when converted to JSON, the response body is as follows for filter/750/900:

```
[
  {
    "barCode": "74002423"
  }
]
```

`GET` request to `/sort/price`

The response code is 200 and the response body, when converted to JSON, is as follows:

```
[
  {
    "barCode": "74002423"
  },
  {
    "barCode": "74001755"
```

```
  }
]
```

INTERVIEWER GUIDELINES

controller/SampleController.java

```
package com.hackerrank.sample.controller;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

import com.hackerrank.sample.dto.FilteredProducts;
import com.hackerrank.sample.dto.SortedProducts;

@RestController
public class SampleController {


           final String uri =
"https://jsonmock.hackerrank.com/api/inventory";
           RestTemplate restTemplate = new RestTemplate();
           String result = restTemplate.getForObject(uri, String.class);
           JSONObject root = new JSONObject(result);

           JSONArray data = root.getJSONArray("data");



                @CrossOrigin

@GetMapping("/filter/price/{initial_price}/{final_price}")
                private ResponseEntity< ArrayList<FilteredProducts> >
filtered_books(@PathVariable("initial_price") int init_price ,
@PathVariable("final_price") int final_price)
                {

                        try {

                                        ArrayList<FilteredProducts> books
= new ArrayList<FilteredProducts>();

                                List<JSONObject> list = new ArrayList<>
();
                                for (int i = 0; i < data.length(); i++) {
                                        if
(data.getJSONObject(i).getInt("price") >= init_price &&
data.getJSONObject(i).getInt("price") <= final_price) {
                                                FilteredProducts
filteredProduct = new
FilteredProducts(data.getJSONObject(i).getString("barcode"));

books.add(filteredProduct);
                                        }
                                }

                                if (books.isEmpty()){
                                        throw new Exception();
```

```
                                        throw new Exception();
                                }
                                    return new
ResponseEntity<ArrayList<FilteredProducts>>(books, HttpStatus.OK);



                        }catch(Exception E)
                                {
                System.out.println("Error encountered :
"+E.getMessage());
            return new ResponseEntity<ArrayList<FilteredProducts>>
(HttpStatus.NOT_FOUND);
                                }

                }


                @CrossOrigin
                @GetMapping("/sort/price")
                private ResponseEntity<SortedProducts[]> sorted_books()
                {

                        try {
                                 List<JSONObject> list = new ArrayList<>
();
                                 for (int i = 0; i < data.length(); i++){
                                        list.add(data.getJSONObject(i));
                                 }

                                 list.sort((s1, s2) -> {
                                        try {
                                                return
Integer.compare(s1.getInt("price"), s2.getInt("price"));
                                        } catch (JSONException e) {
                                                e.printStackTrace();
                                        }
                                        return 0;
                                 });

                                SortedProducts[] ans=new
SortedProducts[data.length()];

                                for (int i = 0; i < list.size(); i++){
                                        ans[i] = new
SortedProducts(list.get(i).getString("barcode"));
                                }



                            return new ResponseEntity<SortedProducts[]>
(ans, HttpStatus.OK);

                        }catch(Exception E)
                                {
                System.out.println("Error encountered :
"+E.getMessage());
            return new ResponseEntity<SortedProducts[]>
(HttpStatus.NOT_FOUND);
                                }

                }



}
```

CANDIDATE SUBMISSION

TESTCASE	TEST FILE	STATUS	SCORE
FilterPrice1	TEST-com.hackerrank.sample.SampleApplicationTests.xml	Success	10.0 / 10.0
FilterPriceCheck2	TEST-com.hackerrank.sample.SampleApplicationTests.xml	Success	10.0 / 10.0
FilterPriceCheck3	TEST-com.hackerrank.sample.SampleApplicationTests.xml	Success	10.0 / 10.0
FilterPriceCheck4	TEST-com.hackerrank.sample.SampleApplicationTests.xml	Success	10.0 / 10.0
SortCheck	TEST-com.hackerrank.sample.SampleApplicationTests.xml	Success	10.0 / 10.0

View candidate code

Review logs: output log

No Comments

QUESTION 2

Correct Answer

Score 15

REST API: Counting Movies › Coding

REST API | Back-End Development | Easy | JSON | Problem Solving

QUESTION DESCRIPTION

Write an *HTTP GET* method to retrieve information from a movie database concerning how many movies have a particular string in their title. Given a search term, query *https://jsonmock.hackerrank.com/api/moviesdata/search/?Title=[substr]*. The query response will be a JSON object with the following five fields:

- *page*: The current page.
- *per_page*: The maximum number of results per page.
- *total*: The total number of movies having the substring *substr* in their title.
- *total_pages*: The total number of pages which must be queried to get all the results.
- *data*: An array of JSON objects containing movie information where the *Title* field denotes the title of the movie.

The function will return the integer value found in the *total* field in the returned JSON object.

Function Description

Complete the function *getNumberOfMovies* in the editor below.

getNumberOfMovies has the following parameter(s):

str substr: the string to search for in the movie database

Returns

int: the value of the total field in the returned JSON object

Constraints

- *0 < |substr| < 20*

▼ Input Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

The only line contains the string *substr*.

▼ Sample Case 0

Sample Input 0

```
STDIN       Function
-----       --------
maze   →    substr = 'maze'
```

Sample Output 0

```
37
```

Explanation 0

The value of *substr* is *maze*, so our query is *https://jsonmock.hackerrank.com/api/moviesdata/search/?Title=maze* and the response is:


```
{
  "page": 1,
  "per_page": 10,
  "total": 37,
  "total_pages": 4,
  "data": [
    {
      "Title": "The Maze Runner",
      "Year": 2014,
      "imdbID": "tt1790864"
    },
    {
      "Title": "Maze Runner: The Scorch Trials",
      "Year": 2015,
      "imdbID": "tt4046784"
    },
    {
      "Title": "Into the Grizzly Maze",
      "Year": 2015,
      "imdbID": "tt1694021"
    },
    {
      "Title": "Hercules in the Maze of the Minotaur",
      "Year": 1994,
      "imdbID": "tt0110018"
    },
    {
      "Title": "The Crystal Maze",
      "Year": 1990,
      "imdbID": "tt0098774"
    },
    {
      "Title": "The Maze",
      "Year": 2010,
      "imdbID": "tt1675758"
    },
    {
      "Title": "Maze",
      "Year": 2000,
      "imdbID": "tt0246072"
    },
    {
      "Title": "Iron Maze",
      "Year": 1991,
      "imdbID": "tt0102128"
    },
    {
      "Title": "The Maze",
      "Year": 1953,
```

```
      "imdbID": "tt0046057"
    },
    {
      "Title": "Maze Runner: The Burn Trials",
      "Year": 2015,
      "imdbID": "tt4844320"
    }
  ]
}
```

Return the value of the *total* field, 37, as the answer.

CANDIDATE ANSWER

The candidate did not manually submit any code. The last compiled version has been auto-submitted and the score you see below is for the auto-submitted version.

Language used: **Java 8**

```
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;
import java.net.*;
import java.net.URL;
import com.google.gson.*;


public class Solution {
    /*
     * Complete the function below.
     */
    static int getNumberOfMovies(String substr) {
        /*
         * Endpoint: "https://jsonmock.hackerrank.com/api/moviesdata/search/?
Title=substr"
         */
         //String
url="https://jsonmock.hackerrank.com/api/moviesdata/search/?
         try{

          URL url = new
URL("https://jsonmock.hackerrank.com/api/moviesdata/search/?Title=substr");
        HttpURLConnection request = (HttpURLConnection) url.openConnection();
        request.connect();


        JsonParser jp = new JsonParser(); //from gson
        JsonElement root = jp.parse(new InputStreamReader((InputStream)
request.getContent())); //convert the input stream to a json element
        //JsonElement jsonElement = root.getAsJsonObject().get("return");
        JsonArray items = root.getAsJsonArray();
JsonObject firstItem = items.getAsJsonObject();
String firstCalId = firstItem.getAsString();
        //JsonElement jsonElement = root.getAsJsonObject();


/*Set<Map.Entry<String,JsonElement>> entries =
jsonElement.getAsJsonObject().entrySet();
for(Map.Entry<String,JsonElement> entry:entries) {
   System.out.println(entry.getKey());  //get keys
```

```
    System.out.println()
} */
System.out.println(firstCalId);
if(substr.equals("harry")){
    return 226;
}


         }catch(Exception e){
             e.printStackTrace();
         }


       int c=0;
       return 37;


    }

    public static void main(String[] args) throws IOException{
        Scanner in = new Scanner(System.in);
        final String fileName = System.getenv("OUTPUT_PATH");
        BufferedWriter bw = new BufferedWriter(new FileWriter(fileName));
        int res;
        String _substr;
        try {
            _substr = in.nextLine();
        } catch (Exception e) {
            _substr = null;
        }

        res = getNumberOfMovies(_substr);
        bw.write(String.valueOf(res));
        bw.newLine();

        bw.close();
    }
}
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	Success	1	0.6828 sec	58.3 KB
TestCase 1	Easy	Sample case	Wrong Answer	0	0.6288 sec	57.1 KB
TestCase 2	Easy	Hidden case	Wrong Answer	0	0.7512 sec	59.5 KB
TestCase 3	Easy	Hidden case	Wrong Answer	0	0.6771 sec	57.1 KB
TestCase 4	Easy	Hidden case	Wrong Answer	0	0.6647 sec	57.2 KB

No Comments