Fraud Prevention Challenge

Introduction:

As part of our hiring process, we ask candidates to develop some kind of minimal application in order to show off their skills & abilities. As such, this document serves as technical specification of a (simple) **HTTP/REST JSON API service**, describing requirements, use cases, and additional details which should be taken into account by candidates.

General Requirements:

- The service must be implemented in **C#, using .Net Core**. We provide a sample VS solution with an ASP.NET Core 3.0 WebAPI project. Feel free to use it directly or take it as reference and create your own.
- Valid requests may receive a 200 status code response, along with the content in JSON format. Requests not compliant with HTTP/1.1 specification, along with other special failure conditions may receive a 400 status error with a description of the failure. Any unexpected server error processing valid requests may receive a 500 status error.
- The service should produce diagnostics/execution logs as rotating (ie. daily) files, including enough information to be of use while diagnosing service issues.
- External dependencies (like databases, non-managed components, etc.) should be avoided as much as possible. This does not apply to .Net managed libraries and NuGets: Feel free to use whatever makes your task easier from within the .Net ecosystem.
- Implementing unit or integration tests, while not a must, will be a convenient addition.
- Code comments and notes which may help us understand your reasoning and decision taking while developing are appreciated.

All in all, this is an exercise designed in order to allow you to demonstrate your potential and skills. As such, we encourage you to try to provide us with clean, well-organized code, easy to review, showing off your talent. Also, even if this project is a small one, which may not directly benefit from it, do not hesitate on implementing coding patterns and/or using development techniques which may help illustrating your experience.

Service Description:

At Groupon we need to take steps to detect and prevent fraudulent purchases. One form of fraud is an attempt from the same user to purchase a deal more than once using different credit card information. Given a set of orders, your task is to identify the orders that fall under this type of fraud.

An order is considered fraudulent if any of the following conditions apply:

- Two orders have the same email address and deal id, but different credit card information, regardless of street address.
- Two orders have the same Address/City/State/Zip and deal id, but different credit card information, regardless of email address.

Remember, people are tricky and are actively trying to get past this fraud detector. Your code must be able to handle the following tricks:

- Email and addresses are case insensitive: bugs@bunny.com is the same as BuGS@BuNNy.COMand 123 Sesame St.is the same as 123 SeSAME st..
- The username portion of an email address can have ignored characters. A dot (.)in an email is flat out ignored, so bugs1@bunny.com and bugs.1@bunny.com are the same email address. A plus (+) in an email means the plus and everything after is ignored, so bugs@bunny.com and bugs+10@bunny.com are the same email address.
- Street addresses often have abbreviated words. 123 Sesame St. and 123 Sesame Streetare the same address. IL and Illinoisare the same state. For the purposes of not making this a typing problem, you can assume that the only abbreviated words you need to worry about are Street/St. and Road/Rd. and the only states you need to worry about are IL, CA, and NY.

We need this detection code to run quickly. The input request can contain thousands of purchases so that it will behoove you to make your code as fast as possible. That said, please remember that this fraud system is part of a larger system and one that might change over time, and we expect the structure of your code to reflect that fact.

REST/API description

External Application Programming Interface (in HTTP/REST format), which should be exposed by Fraud Prevention API.

Security & Authentication

No authentication, authorization or encryption requirements has been defined for this application.

Message formatting & serialization

Fraud Prevention API uses industry standard HTTP/1.1 as its underlying data communications transport, along with JSON as its message formatting and serialization.

POST /FraudPrevention/validate

Validate a list of purchases, returning the fraudulent ones.

Request details

A list of purchases in JSON format, each one containing the following information:

- Order id (numeric)
- Deal id (numeric)
- Email address
- Street address
- City
- State
- Zip Code
- Credit Card #

Sample request:

```
{"purchases": [{
        "orderId": 1,
        "dealId": 1,
        "emailAddress": "bugs@bunny.com",
        "streetAddress": "123 Sesame St.",
        "city": "New York",
        "state": "NY",
        "zipCode": "10011",
        "creditCardNumber": 12345689010
    },
       {
        "orderId": 2,
        "dealId": 1,
        "emailAddress": "elmer@fudd.com",
        "streetAddress": "123 Sesame St.",
        "city": "New York",
        "state": "NY",
        "zipCode": "10011",
        "creditCardNumber": 10987654321
   },
       {
        "orderId": 3,
        "dealId": 2,
        "emailAddress": "bugs@bunny.com",
        "streetAddress": "123 Sesame St.",
        "city": "New York",
        "state": "NY",
        "zipCode": "10011",
        "creditCardNumber": 12345689010
    }
]}
```

Response details

A list of fraudulent OrderId's in ascending order, in JSON format.

Sample response:

```
[1,2]
```

Sample Explanation

The first two orders are fraudulent, because they have the same address and deal, but different credit card information. The third order is not fraudulent because, although it shares personal information with the first order, it has the same credit card info and a different deal id.