Scenario

Our product manager has come back from a “security conference” where they saw an amazing demo of a “Superman Detector” from LexCorp that identifies logins by a user that occur from locations that are farther apart than a normal person can reasonably travel on an airplane. These locations were determined by looking up the source IP addresses of successful logins in a GeoIP database. This behavior is interesting because it can be a strong indication that a user has had their account credentials compromised.

We want our own Superman detector application. All of the sudden, a flurry of JIRA tickets are opened that are ill-defined but prioritize this project. After careful consultation with your technical compadres, the acceptance criteria for a proof-of-concept looks like the following:

Key Requirements:

- The Superman detector will have an API that accepts a JSON document containing:
    - UUID identifying the login attempt
    - Username
    - UNIX epoch timestamp in seconds
    - IPv4 address

  Note: The calls to this API will not necessarily be in event time order. In fact, these tuples often arrive to the system out-of-order. Therefore, you will have to take care to make proper comparisons correctly in proper time order.
- Your team debated whether a gRPC or JSON API should be used and the JSON advocates won. (See example section below for a JSON queries and responses)
- Use data structures or existing libraries for fast mapping from IP address to latitude and longitude geolocation lookups provided by a **local** copy of the MaxMind GeoIP database.
    - Download and use the MaxMind City database for IP geolocation: https://dev.maxmind.com/geoip/geoip2/geolite2/
- Our Superman speed threshold is anything above 500 miles per hour.
    - Assume the Earth is a perfect sphere where distances should be calculated via the Haversine formula (https://en.wikipedia.org/wiki/Haversine_formula).
- Vendor all of your external dependencies using `go mod vendor` (assumes go v1.11).
- Unit tests should be written for key components of the system. A README is also required.
- Use SQLite as your database. (Otherwise, you might be blocked on the devops team needing more time to set up a full RDS database with Terraform!)

- Package the build process and runtime image as a multi-stage Docker container.
- You are welcome to use any external libraries you need, but you must **document and cite any external libraries or resources you used.**

Key Considerations:

- Runtime performance for IP Geolocation lookups and the travel distance computations.
- Handling the inevitable out-of-order arrival time of login events.
- The IP Geolocation database includes a radius (in kilometers) around each latitude and longitude point to express uncertainty.

Expected Request

Your API should accept a POST query over HTTP that provides information about the time and ip that a user is connecting from. A sample curl request might look like:

```
$ curl -X POST -d
      '{"username": "bob",
        "unix_timestamp": 1514764800,
        "event_uuid": "85ad929a-db03-4bf4-9541-8f728fa12e42",
        "ip_address": "206.81.252.6"}' http://yourservice.local/v1/
```

Expected Response

In response to the above POST request, your API should return a JSON document informing the client about the geo information for the current IP access event as well as the nearest previous and subsequent events (if they exist). For the preceding/subsequent events, it should also include a field **`suspiciousTravel`** indicating whether travel to/from that geo is suspicious or not, as well as the **speed**.

```
{
    "currentGeo": {
        "lat": 39.1702,
        "lon": -76.8538,
        "radius": 20
     },
     “travelToCurrentGeoSuspicious”: true,
```

```
    “travelFromCurrentGeoSuspicious”: false,
    "precedingIpAccess": {
       "ip": "24.242.71.20",
       "speed": 55,
       "lat": 30.3764,
       "lon": -97.7078,
       "radius": 5,
       "timestamp": 1514764800
    },
    "subsequentIpAccess": {
       "ip": "91.207.175.104",
       "speed": 27600,
       "lat": 34.0494,
       "lon": -118.2641,
       "radius": 200,
       "timestamp": 1514851200
    }
}
```