Hi,
Thanks for taking the time to work on this assignment!

This engineering assignment is a close simulation of the work we do at Salt in our day to day, and is meant to assess your engineering and coding skills.
Once implemented, we will meet for a review session (about 45 mins) so you can present the solution and code and discuss it together.
Below you can find the assignment. Feel free to ask any questions you might have all through the process.

One of the existing modules in Salt is a service that identifies abnormal http requests being sent to customers' APIs. The service consists of 2 main parts:

1. Receiving API models for different endpoints (URLs) and HTTP methods being generated by Salt Security's algorithms. These models represent the "normal" structure of requests being sent to this endpoint with the defined method.

   Note: An API model is defined by its endpoint and method, such that for a given endpoint, multiple and different models should be supported (one for each HTTP method).
   Example structure of learned model data:

```
{
        "path": "/entity/create",
        "method": "POST",
        "query_params": [
                {
                        "name": "QueryParam1",
                        "types": ["type1", "type2",...],
                        "required": true / false
                },
                .
                .
                .
        ],
        "headers": [
                {
                        "name": "HeaderParam1",
                        "types": ["type1", "type2",...],
                        "required": true / false
                },
                .
                .
                .
        ],
        "body": [
                {
                        "name": "BodyParam1",
                                        "types": ["type1", "type2",...],
                                        "required": true / false
                              },
                              .
                              .
                              .
                    ]
          }
```

Each parameter can have multiple types and a flag specifying if it's required or not.
Available parameter types and their specifications:

Type	Subtype	Format	Example
Int	Primitive	- - -	8, 109, 722
String	Primitive	- - -	"Foo", "this is a string"
Boolean	Primitive	- - -	true, false
List	Primitive	List of any primitive or object	[1,2,3], [{"id": 123}, {"id": 456}]
Date	String	dd-mm-yyyy	"12-01-2022"
Email	String	RFC 5321	"foo@bar.com"
UUID	String	A global unique identifier composed of dashes (-), numbers and English letters	"46da6390-7c78-4a1c-9efa-7c0396067ce4"
Auth-Token	String	Authentication token that starts with "Bearer " and continues with a token composed of letters and numbers	"Bearer ebb3cbbe938c4776bd22a4ec2ea8b2ca"

2. Real time processing of requests data, that determines if the current request is "abnormal" - i.e. has some kind of anomaly in relation to the learned "normal" request, main cases for "abnormal" data are: type mismatch & missing required parameters.
Example structure of request data:

```
{
        "path": "/entity/test",
        "method": "GET",
        "query_params": [
                {
                        "name": "QueryParam1",
                        "value": "{test1}"
                },
                .
                .
                .
        ],
        "headers": [
                {
                        "name": "HeaderParam1",
                        "value": "{test2}"
                },
                .
                .
                .
        ],
        "body": [
                {
                        "name": "BodyParam1",
                        "value": "{test3}"
                },
                .
                .
                .
        ]
}
```

Your task is implement a micro-service that has 2 main routes:

1. Receives and stores the “learned” models (already generated by Salt Security’s algorithms, and out of scope for this assignment). a json file with a list of ready models for your tests is attached (in the format specified in #1 above).
2. Receives data for a single request and decides if the current request is “abnormal”. A json file with a list of requests data is attached (in the format specified in #2 above). Your service should return “200 OK” with json body stating whether the request is abnormal or valid, the abnormal fields if there are any and the reason for the abnormality in each field.

Your result should include an implementation of the micro-service specified above in a technology of your choice (preferred: Scala, Java, Node.js, Go, Python)

General guidelines:

- You can use any tool and online resource you want, and reach out to us for consultation - as you would if you were building this service as an engineer at Salt Security :-)
- Please implement the schema validation yourself - refrain from using JSON schema validation as it’s one of the main points of the assignment (we do our own validation for performance considerations)
- The implementation should be as clean, readable and efficient as possible - you should be able to explain the performance considerations.
- If you feel you need to use databases for your solution, feel free to implement all needs in-memory inside the code, but be sure to write down what solution you would use in real life production and the tradeoffs.
- Make sure to present the tradeoffs you made as part of presenting the solution during the design and implementation reviews.

Good luck!