How To Setup In Local:

1) Set up virtual environment.
   - sudo pip install virtualenv
   - virtualenv quartic_venv
   - virtualenv -p /usr/local/bin/python3.5 quartic_venv
   - source quartic_venv/bin/activate
2) Go to the directory where you want to start the project and install django.
   - cd ~/Documents/
   - pip install django
   - django-admin startproject RuleEngine
   - pythonmanage.py startapp rule_engine
3) Install all the dependencies specified in the project in requirements.txt
   - pip install -r requirements.txt
   - Path for requirements.txt(Repo/requirements.txt)
4) Export Environmental variables.
   - export DB_NAME='rule_engine'
   - export DB_HOST='localhost'
   - export DB_PORT=5432
   - export DB_USER='postgres'
   - export DB_PWD=' '
   - export DJANGO_SETTINGS_MODULE='RuleEngine.settings.development'

   Note: Please create database before exporting.
5) Migrate the models and run the server.
   - python3.5 manage.py makemigrations
   - python3.5 manage.py migrate
   - python3.5 manage.py runserver

API Contract:

1) **API**: http://ip:port/api/v1/rule/
   HTTP Method: GET
   Description: This API is used for getting all the rules created by user.
   Sample Response:
   Body:
```
[{
        "rule_id": 1,
        "rule_name": "Rule For ATL1",
        "signal": "ATL1",
        "value": "240","value_type": "Integer","criteria": "=="}]
```
   Status Code: 200 OK

2) **API**: http://ip:port/api/v1/rule/{ruleId}/

1) **HTTP Method**: GET
2) **Description**: This API is used for getting a rule based on Rule Id
3) **Sample Response**:
4) **Body**:
5) [{
6) "rule_id": 1,
7) "rule_name": "Rule For ATL1",
8) "signal": "ATL1",
9) "value": "240","value_type": "Integer","criteria": "=="}]
10) **Status Code**: 200 OK

3) **API**: http://ip:port/api/v1/rule/{ruleId}/

HTTP Method: PUT

Description: This API is used for updating rule based on rule id.

Sample Request:

Body:

11){

12) "rule_name": "Rule For ATL1",

13) "signal": "ATL2",

14) "value": "240","value_type": "Integer","criteria": "=="}

Sample Response:

Body:

15)[{

16) "rule_id": 1,

17) "rule_name": "Rule For ATL1",

18) "signal": "ATL2",

19) "value": "240","value_type": "Integer","criteria": "=="}]

Status Code: 200 OK

4) **API**: http://ip:port/api/v1/rule/{ruleId}/

HTTP Method: DELETE

Description: This API is used for deleting rule based on rule id.

Sample Response:

Body:

20) Rule successfully deleted

Status Code: 200 OK

5) **API**: http://ip:port/api/v1/rule/

HTTP Method: POST

Description: This API is used for creating rule.

Sample Request:

Body:

```
{
        "rule_name": "Rule For ATL1",
        "signal": "ATL2",
        "value": "240","value_type": "Integer","criteria": "=="}
```

Sample Response:

Body:

```
[{
        "rule_id": 1,
        "rule_name": "Rule For ATL1",
        "signal": "ATL2",
        "value": "240","value_type": "Integer","criteria": "=="}]
```

Status Code: 200 OK

6) **API**: http://ip:port/api/v1/filterdata/

HTTP Method: POST

Description: This API is used for filter data based on rules.

Sample Request:

Body:

```
{       "rules":[1,2,4],
        "data": [{"signal": "ATL1", "value_type": "Integer", "value": "240"},
                {"signal": "ATL2", "value_type": "String", "value": "LOW"}]
}
```

Sample Response:

Body:

```
[{
        "signal": "ATL1",
        "value_type": "Integer",
        "value": "63.679"
    },
    {
        "signal": "ATL2",
        "value_type": "String",
        "value": "LOW"
    }]
```

Status Code: 200 OK

7) **API** : http://ip:port/api/v1/health

HTTP Method : GET

Description : This API is used for checking the health of the application.

Sample Response :

Body : EMPTY

Status Code : 200 OK

Discussion Questions:

Q1. Briefly describe the conceptual approach you chose! What are the trade-offs?
A. I have created APIs for performing CRUD(Create, Retrieve, Update, Delete) operations on rules. There is a sepearte API provided for filtering data which takes Rule Ids and filter the data which violates the rule. Trade off is instead of having normalised database(Keeping seperate table for storing operations), I decided to keep opeartions in the same table as rules, so that extra database call will be saved.

Q2. What's the runtime performance? What is the complexity? Where are the bottlenecks?
A. For the sample data(raw_signal.json) provided in the question, the latency is 30-50ms with three rules applied. The complexity is O(N) where N is the number of rows of data.The major bottleneck I faced is defining the unified operations for all the value types.

Q3.If you had more time, what improvements would you make, and in what order of priority?
A. Firstly, I would have liked to introduce the cache service to it so that all the old rules will be stored in cache and DB calls will be made only for new rules. Next I would introduce a script for taking the latest code from Github and building a Docker image using Jenkins so that deployment will be fast.

Docker Deployment:

For deployment using docker, use the following steps:

- Install docker on the respective machine.
- Clone the repository and go to deploy folder(Path: Repo/deploy/)
- Run startDocker.sh file by giving executable permissions to it.

SQL Schema:

Schema is provided in django model. Path of file is
(Repo/orders/models/ruleModels.py)

Database:

I used postgres as database. All the tables are created by django using following steps:

- python3.5 manage.py makemigrations
- python3.5 manage.py migrate