

## **Real-time Data Classification API**

### **Introduction**

Welcome to the Real-time Data Classification System! This project aims to create a powerful and dynamic platform for processing and classifying live data streams based on user-defined rules. It leverages modern web technologies and efficient backend solutions to ensure that data is processed with high speed and accuracy, making it ideal for applications that require real-time decision-making and data management.

### **Purpose and Goals :**

The primary goal of this project is to provide a scalable, real-time data processing system where users can define their own classification rules. By integrating JWT for secure authentication, a robust database for rule storage, and real-time data streaming capabilities, this project ensures that user-defined rules can be applied promptly to incoming data, making the system suitable for various real-world applications.

### **Features**

- ❖ JWT Authentication
  - Secure access with JSON Web Tokens.
  - Middleware to validate incoming requests.
  - Sign-up and login endpoints for user management.
- ❖ Database Integration
  - Schema design for storing user and rule data.
  - CRUD operations for managing user-defined classification rules.
  - Integration with a database for persistent storage.
- ❖ User-Defined Classification Rules
  - Parser to interpret and validate user-defined rules.
  - API endpoints for rule management (create, read, update, delete).
- ❖ Real-time Data Processing
  - WebSocket or Server-Sent Events (SSE) for live data streaming.
  - Real-time classification engine to process data streams using defined rules.
- ❖ Comprehensive Testing
  - Unit tests for individual modules.
  - Integration tests for end-to-end system validation.
  - Load testing to ensure system stability under high data volumes.

### **Tools & Technologies**

- **Backend** : Fastify, NodeJS
- **Database** : MongoDB
- **Authentication** : Json Web Token (JWT)
- **Real-Time Processing** : WebSocket
- **Unit/Integration Testing** : Jest with Supertest

- **Load Testing** : Apache JMeter
- **API request Platform** : VS Code-Thunder Client

## Installation

- **Clone the repository**  
git clone https://github.com/biswojit65/LDP-Project-3.git  
cd LDP-Project-3
- **Install dependencies using npm**  
npm install
- **Set up the database**  
Setup mongodb database and connect it to the application using mongoose.

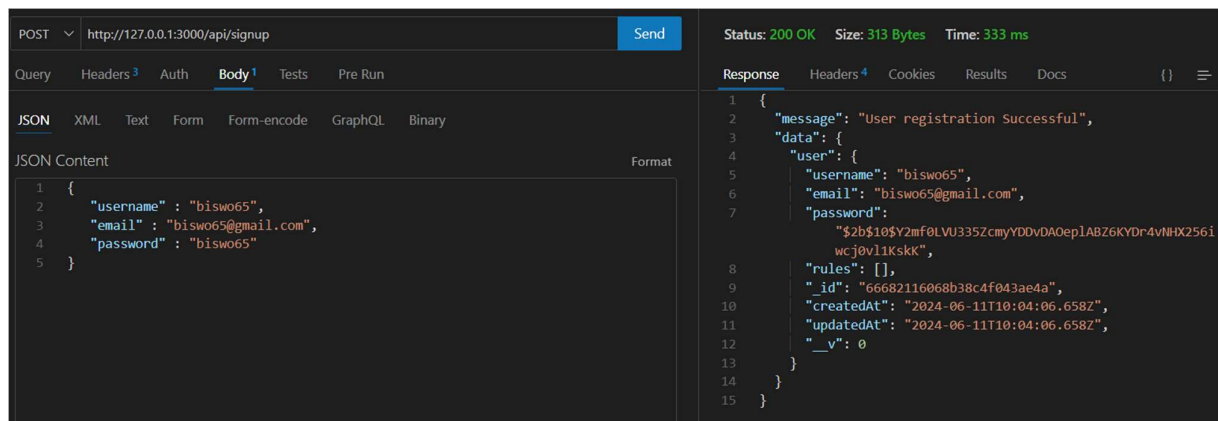
## Usage

- **Start the server**  
npm start
- **API Documentation**  
You can use tools like Postman/VS Code Thunder Client to interact with the API endpoints listed below.

## Endpoints

- **Create an User** : http://{{host}}:{{port}}/api/signup
- **Login a user** : http://{{host}}:{{port}}/api/login
- **Create a Rule** : http://{{host}}:{{port}}/api/newrules
- **Get All Rules** : http://{{host}}:{{port}}/api/getrules
- **Update a rule** : <http://{{host}}:{{port}}/api/updaterule>
- **Delete a Rule** : http://{{host}}:{{port}}/api/deleterule
- **Applying rules on inputstring** : http://{{host}}:{{port}}/api/check

## Images



Creating a new User

POST

http://127.0.0.1:3000/api/login

Send

Query

Headers 3

Auth

Body 1

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

Binary

JSON Content

Format

```
1 {
2   "username": "bismo65",
3   "password": "bismo65"
4 }
```

Status: 200 OK

Size: 204 Bytes

Time: 147 ms

Response

Headers 4

Cookies

Results

Docs

{}

≡

```
1 {
2   "message": "Successfully Logged in",
3   "data": {
4     "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
      .eyJpZCI6IjY2NjgyMTE2MDY4YjM4YzRmMDQzVWU0YSIsIm1hdCI6
      MTcxODEwMDI5N30uJhBssY3l_4n0TvOmXduv9WovDC80Sha
      -TBwvwlxvKUU"
5   }
6 }
```

Login a User

POST

http://127.0.0.1:3000/api/newrule

Send

Query

Headers 3

Auth

Body 1

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

Binary

JSON Content

Format

```
1 {
2   "rule" : "min(max(30,sumofallnumberfromstring),sumof2(count('a'),40)) <= 50"
3 }
```

Status: 200 OK

Size: 158 Bytes

Time: 550 ms

Response

Headers 4

Cookies

Results

Docs

{}

```
1 {
2   "message": "Your Rule Successfully Inserted",
3   "data": {
4     "pushRule": {
5       "acknowledged": true,
6       "modifiedCount": 1,
7       "upsertedId": null,
8       "upsertedCount": 0,
9       "matchedCount": 1
10    }
11  }
12 }
```

Creating a New Rule

POST

http://127.0.0.1:3000/api/newrule

Send

Query

Headers 3

Auth

Body 1

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

Binary

JSON Content

Format

```
1 {
2   "rule" : "max(count('ab'),sumof2 (sumofallnumberfromstring
      ,avgofallnumberfromstring)) >= min(max(32,count('cb')),avgof2
      (avgofallnumberfromstring,min(12,sumofallnumberfromstring)))"
3 }
```

Status: 200 OK

Size: 158 Bytes

Time: 409 ms

Response

Headers 4

Cookies

Results

Docs

```
1 {
2   "message": "Your Rule Successfully Inserted",
3   "data": {
4     "pushRule": {
5       "acknowledged": true,
6       "modifiedCount": 1,
7       "upsertedId": null,
8       "upsertedCount": 0,
9       "matchedCount": 1
10    }
11  }
12 }
```

Response

Chat

Creating anothor New Rule

PUT

http://127.0.0.1:3000/api/updaterule

Send

Query

Headers 3

Auth

Body 1

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

Binary

JSON Content

Format

```
1 {
2   "ruleID" : "66686627fa6a66f06e019882",
3   "rule" : "sumof2(count('a'),count('b')) <= count('c')"
4 }
```

Status: 200 OK

Size: 303 Bytes

Time: 414 ms

Response

Headers 4

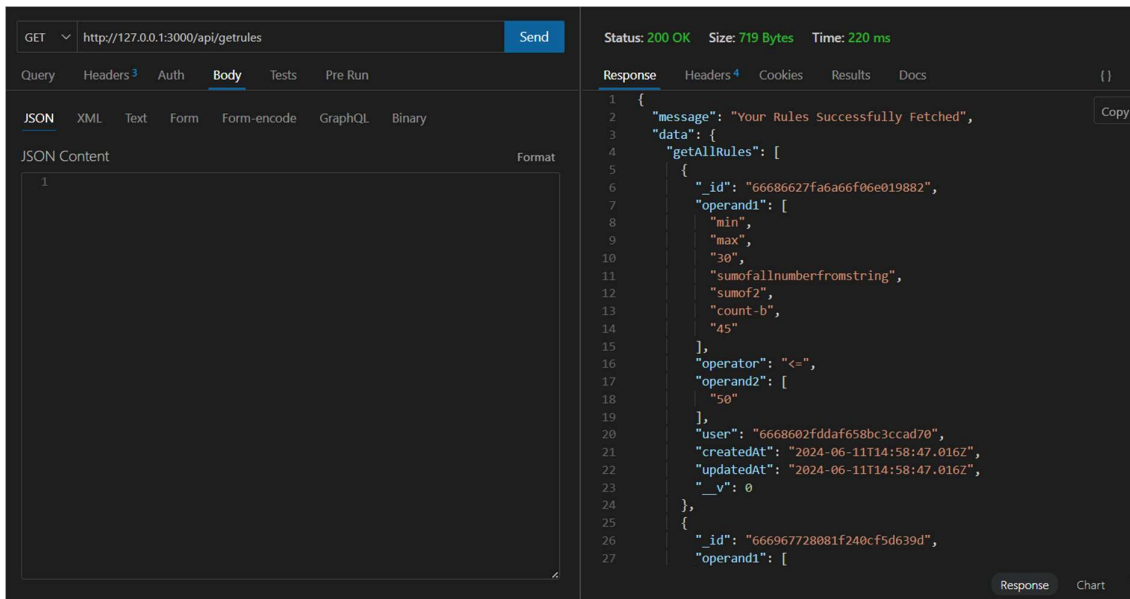
Cookies

Results

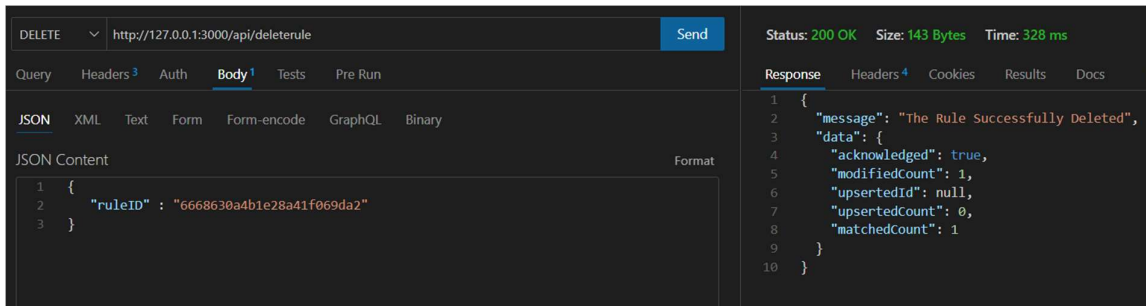
Docs

```
1 {
2   "message": "Your Rule Successfully Updated",
3   "data": {
4     "updatedrule": {
5       "_id": "66686627fa6a66f06e019882",
6       "operand1": [
7         "sumof2",
8         "count-a",
9         "count-b"
10      ],
11       "operator": "<=",
12       "operand2": [
13         "count-c"
14      ],
15       "user": "6668602fddaf658bc3ccad70",
16       "createdAt": "2024-06-11T14:58:47.016Z",
17       "updatedAt": "2024-06-19T05:13:27.623Z",
18       "__v": 0
19     }
20   }
21 }
```

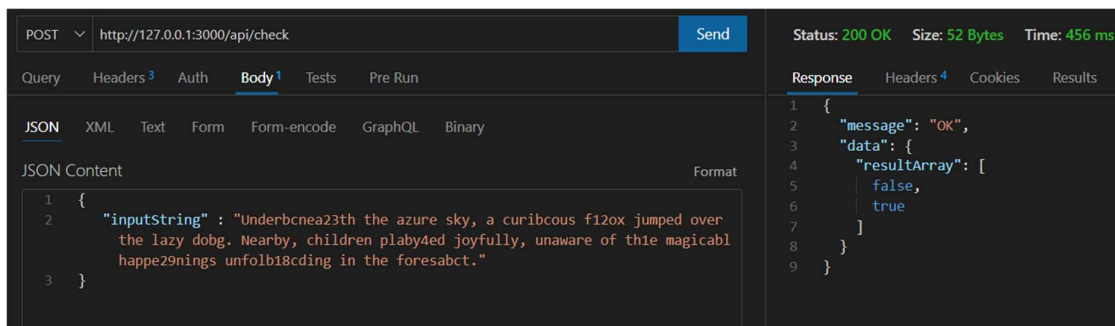
Update a Rule



## Get all the stored Rules for a user



## Delete a Rule



## Applying rules on inputstring and getting result

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
<b>All files</b>	<b>71.31</b>	<b>68.29</b>	<b>86.66</b>	<b>74.05</b>	
src	100	100	100	100	
createApp.js	100	100	100	100	
src/constants	100	100	100	100	
constant.js	100	100	100	100	
src/controllers	54.46	50.98	63.63	57.61	
controller.js	54.46	50.98	63.63	57.61	186-290,309-311,327-328,331-332,335-338
src/models	100	100	100	100	
rule.js	100	100	100	100	
user.js	100	100	100	100	
src/routes	100	100	100	100	
route.js	100	100	100	100	
src/services	98.26	96.66	100	100	
service.js	98.26	96.66	100	100	87,188
Test Suites: 3 passed, 3 total Tests: 75 passed, 75 total Snapshots: 0 total Time: 6.246 s					

## Unit Test/Integration Test/Code coverage

A	B	C	D	E	F	G	H	I	J	K
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	650	11075	381	14409	2736.04	0.00%	42.14485	9.18	20.5	223
TOTAL	650	11075	381	14409	2736.04	0.00%	42.14485	9.18	20.5	223

## Matrices obtained during Load Testing (Ramp up Period-1sec)

A	B	C	D	E	F	G	H	I	J	K
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	650	11075	381	14409	2736.04	0.00%	42.14485	9.18	20.5	223
TOTAL	650	11075	381	14409	2736.04	0.00%	42.14485	9.18	20.5	223

## Matrices obtained during Load Testing (Ramp up Period-5sec)

### WebSocket Data Receiver

[Get Data](#)

String 1: [Rule 1 : Failed , Rule 2 : Passed , ]
String 2: [Rule 1 : Failed , Rule 2 : Passed , ]
String 3: [Rule 1 : Failed , Rule 2 : Passed , ]
String 4: [Rule 1 : Failed , Rule 2 : Passed , ]
String 5: [Rule 1 : Passed , Rule 2 : Failed , ]
String 6: [Rule 1 : Failed , Rule 2 : Passed , ]
String 7: [Rule 1 : Passed , Rule 2 : Passed , ]
String 8: [Rule 1 : Failed , Rule 2 : Passed , ]
String 9: [Rule 1 : Passed , Rule 2 : Failed , ]
String 10: [Rule 1 : Failed , Rule 2 : Passed , ]
String 11: [Rule 1 : Passed , Rule 2 : Failed , ]
String 12: [Rule 1 : Failed , Rule 2 : Passed , ]
String 13: [Rule 1 : Failed , Rule 2 : Passed , ]
String 14: [Rule 1 : Failed , Rule 2 : Failed , ]

The screenshot shows a web browser window with a 'WebSocket Data Receiver' application. The application has a 'Get Data' button and a list of 14 strings, each representing a rule execution result (e.g., 'String 1: [Rule 1 : Failed , Rule 2 : Passed ,]'). To the right, the browser's developer console is open, displaying logs from 'index.html'. The logs show a connection to a WebSocket server and several 'Received' messages containing JSON data. The console also shows a 'new' logo and a message about moving tracks around with updated UI in the Performance panel.

## WebSocket Data streaming