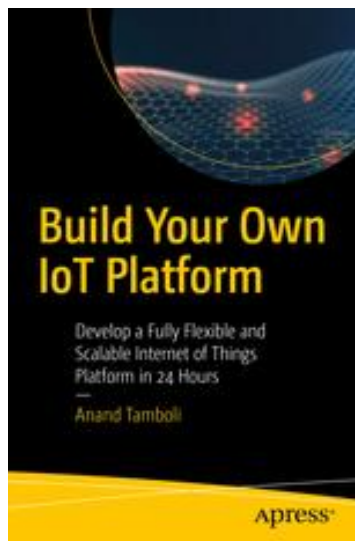


Proof of Concept (POC) - Prototype IoT Platform

Provisional Guidance

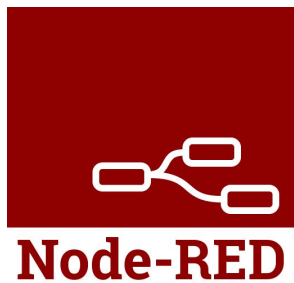
- Used 'Build Your Own IoT Platform' as an implementation guide
- It delineates the use of software utilities and applications which have been utilized to build this rudimentary prototype of an IoT platform



Technology Stack

- Mosquitto - MQTT Message Broker
- LAMP Stack - Linux + Apache + MySQL + PHP
- UFW - Uncomplicated Firewall
- phpMyAdmin
- Node.js, Node-RED
- WebSocket
- Eclipse Paho
- Twilio
- SendGrid

Technology Stack



Implementation Details of Individual Blocks

Message Broker

- MQTT used as a messaging and communication protocol
- WebSocket enables all MQTT features to a browser-based application
- Eclipse Paho utility used to check connectivity to MQTT broker from browser via WebSocket
- Mosquitto used as the broker
- Mosquitto configuration

```
112 # mosquitto start process alias
113 alias mosquitto_start_process='sudo mosquitto -c /etc/mosquitto/conf.d/broker.conf -v > mosquitto.log &'
```

```
niranay@niranay: ~/Documents/sem7/btech project/in24hrs master ? mosquitto_start_process
```

```
[1] 22099
```

```
niranay@niranay: ~/Documents/sem7/btech project/in24hrs master ? cat mosquitto.log
```

```
1573840589: mosquitto version 1.4.8 (build date Tue, 18 Jun 2019 11:59:34 -0300) starting
```

```
1573840589: Config loaded from /etc/mosquitto/conf.d/broker.conf.
```

```
1573840589: Opening ipv4 listen socket on port 1883.
```

```
1573840589: Opening websockets listen socket on port 8443.
```

```
niranay@niranay: ~/Documents/sem7/btech project/in24hrs master ?
```

```
✓ 10020 23:26:17
```

```
✓ 10021 23:26:29
```

```
✓ 10022 23:26:39
```

Mosquitto Log File

```
1 ./mosquitto.log  
2 1573729373: mosquitto version 1.4.8 (build date Tue, 18 Jun 2019 11:59:34 -0300) starting  
3 1573729373: Config loaded from /etc/mosquitto/conf.d/broker.conf.  
4 1573729373: Opening ipv4 listen socket on port 1883.  
5 1573729373: Opening websockets listen socket on port 8443.  
6 1573729583: New connection from 127.0.0.1 on port 1883.  
7 1573729583: New client connected from 127.0.0.1 as node-red (c1, k60, u'vaidnira').  
8 1573729583: Sending CONNACK to node-red (0, 0)  
9 1573729583: Received SUBSCRIBE from node-red  
10 1573729583: # (QoS 2)  
11 1573729583: node-red 2 #  
12 1573729583: Sending SUBACK to node-red  
13 1573729597: Received PUBLISH from node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
14 1573729597: Sending PUBLISH to node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
15 1573729612: Received PUBLISH from node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
16 1573729612: Sending PUBLISH to node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
17 1573729627: Received PUBLISH from node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
18 1573729627: Sending PUBLISH to node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
19 1573729642: Received PUBLISH from node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
20 1573729642: Sending PUBLISH to node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
21 1573729657: Received PUBLISH from node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
22 1573729657: Sending PUBLISH to node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
23 1573729672: Received PUBLISH from node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
24 1573729672: Sending PUBLISH to node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
25 1573729687: Received PUBLISH from node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
26 1573729687: Sending PUBLISH to node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
27 1573729702: Received PUBLISH from node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
28 1573729702: Sending PUBLISH to node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
29 1573729717: Received PUBLISH from node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
30 1573729717: Sending PUBLISH to node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
31 1573729732: Received PUBLISH from node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
32 1573729732: Sending PUBLISH to node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
33 1573729747: Received PUBLISH from node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
34 1573729747: Sending PUBLISH to node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
35 1573729762: Received PUBLISH from node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
36 1573729762: Sending PUBLISH to node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))  
37 1573729777: Received PUBLISH from node-red (d0, q0, r0, m0, 'timestamp', ... (13 bytes))
```

Time-Series Storage

- Data dump to single table
- Schema details
 - ID - Incremental unique number
 - Topic - Topic name
 - Payload - packet data
 - Timestamp - UNIX timestamp

Sr	Name	Type	Null	Extra
1	ID	int (11)	No	AUTO_INCREMENT
2	Topic	varchar (1024)	No	
3	Payload	varchar (2048)	No	
4	Timestamp	varchar (15)	No	

phpMyAdmin View

phpMyAdmin

Recent Favorites

information_schema
tSeriesDB

Server: localhost > Database: tSeriesDB > Table: thingData

Browse Structure SQL Search Insert Export Import Operations Tracking Triggers

Show query box

Showing rows 0 - 24 (5271 total, Query took 0.0007 seconds.)

```
SELECT * FROM `thingData` WHERE 1
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> Number of rows: 25 Filter rows: Search this table

Sort by key: None

+ Options

	id	topic	payload	timestamp	deleted
Edit Copy Delete	4	timestamp	1563480502551	1563480502.552	31
Edit Copy Delete	5	timestamp	1563480517566	1563480517.567	31
Edit Copy Delete	6	timestamp	1563480532576	1563480532.577	31
Edit Copy Delete	7	timestamp	1563481770945	1563481770.953	31
Edit Copy Delete	8	timestamp	1563481785951	1563481785.953	31
Edit Copy Delete	9	timestamp	1563481800952	1563481800.953	31
Edit Copy Delete	10	timestamp	1563481815953	1563481815.954	31
Edit Copy Delete	11	timestamp	1563481830956	1563481830.958	30

Console

Virtual Data Generation and Consumption

- Node-RED used to generate periodic timestamp data and publish on MQTT by establishing a flow
- HTTP POST request used to generate asynchronous user specific topic + payload data and publish on MQTT
- Node-RED used to establish a flow for a database listener expecting incoming data on MQTT
- Node-RED configuration

```
115 # node-red start process alias
116 alias node_red_start_process='forever start -l node-red.log --append /usr/local/bin/node-red'
```

```
niranjan@niranjan: ~/Documents/sem7/btech project node_red_start_process
warn: --minUptime not set. Defaulting to: 1000ms
warn: --spinSleepTime not set. Your script will exit if it does not stay up for at least 1000ms
info: Forever processing file: /usr/local/bin/node-red
niranjan@niranjan: ~/Documents/sem7/btech project
```

10034 00:09:19

10035 00:09:26

Node-RED Log File

```
1 ~/.forever/node-red.log
421 16 Nov 00:09:27 - [info]
422
423 Welcome to Node-RED
424 =====
425
426 16 Nov 00:09:27 - [info] Node-RED version: v0.20.7
427 16 Nov 00:09:27 - [info] Node.js version: v11.4.0
428 16 Nov 00:09:27 - [info] Linux 4.15.0-66-generic x64 LE
429 16 Nov 00:09:27 - [info] Loading palette nodes
430 16 Nov 00:09:28 - [warn] rpi-gpio : Raspberry Pi specific node set inactive
431 16 Nov 00:09:28 - [warn] rpi-gpio : Cannot find Pi RPi.GPIO python library
432 16 Nov 00:09:28 - [info] Settings file : /home/niramay/.node-red/settings.js
433 16 Nov 00:09:28 - [info] Context store : 'default' [module=memory]
434 16 Nov 00:09:28 - [info] User directory : /home/niramay/.node-red
435 16 Nov 00:09:28 - [warn] Projects disabled : editorTheme.projects.enabled=false
436 16 Nov 00:09:28 - [info] Flows file : /home/niramay/.node-red/flows_Niramay.json
437 16 Nov 00:09:28 - [info] Server now running at http://127.0.0.1:1880/admin/
438 16 Nov 00:09:28 - [warn]
439
440 -----
441 Your flow credentials file is encrypted using a system-generated key.
442
443 If the system-generated key is lost for any reason, your credentials
444 file will not be recoverable, you will have to delete it and re-enter
445 your credentials.
446
447 You should set your own key using the 'credentialSecret' option in
448 your settings file. Node-RED will then re-encrypt your credentials
449 file using your chosen key the next time you deploy a change.
450 -----
451
452 16 Nov 00:09:28 - [info] Starting flows
453 16 Nov 00:09:29 - [info] Started flows
454 16 Nov 00:09:29 - [info] [mqtt-broker:441d35c1.25fcc4] Connected to broker: node-red@mqtt://localhost:1883
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

Node-RED Timestamp Flow

The image displays the Node-RED web interface. On the left, the 'input' and 'output' node palettes are visible. The main workspace contains a flow with the following components:

- A blue 'timestamp' node (injector type) with a dropdown arrow.
- A green 'msg.payload' node.
- A purple 'timestamp' node (mqtt type) with a 'connected' status indicator.

The flow is connected as follows: the 'timestamp' injector node connects to the 'msg.payload' node, which then connects to the 'timestamp' mqtt node.

On the right, the 'info' sidebar shows the following details for the selected flow:

- Information**
 - Flow: `"c1116f8d.1c54"`
 - Name: generate timestamp flow
 - Status: Enabled
- Description**
 - Generates periodic timestamp Publishes it using MQTT

Node-RED User Topic + Payload Flow

The image displays the Node-RED web interface. The top bar includes the Node-RED logo, a search bar, and tabs for various flows: "generate timestamp", "publish user topic" (selected), "database listen flow", "retrieve data record", and "retrieve data record". The right sidebar shows the "info" panel for the selected flow, displaying its ID, name, status, and description.

Flow Diagram:

```
graph LR; A["[post] /pub/:topic/:payload"] --> B["f create message"]; B --> C["f create response"]; C --> D["mqtt"]; D --> E["http"];
```

The flow starts with an HTTP POST node labeled "[post] /pub/:topic/:payload". This node connects to a function node "f create message". The output of "f create message" connects to another function node "f create response". The output of "f create response" connects to an MQTT node labeled "mqtt" with a "connected" status. Finally, the output of the MQTT node connects to an HTTP node labeled "http".

Node-RED Interface Details:

- Left Panel (Nodes):** Includes input nodes (inject, catch, status, link, mqtt, http, websocket, tcp, udp) and output nodes (debug, link, mqtt).
- Right Panel (Info):**
 - Information:**
 - Flow: "ff8c85bb.ed5c7"
 - Name: publish user topic+payload flow
 - Status: Enabled
 - Description:**
 - Uses HTTP POST request to generate user specified topic+payload object Publishes it using HTTP response Publishes it using MQTT

localhost:1880/admin/#ff8c85bb.ed5c7

HTTP POST Request Validation using Postman

The screenshot displays the Postman application interface. On the left, a sidebar shows a collection named 'IoT-Platform-API-Testing' with 17 requests. The main panel shows a POST request to 'http://localhost:1880/pub/myTopic/myPayload'. The 'Body' tab is selected, showing a JSON response in 'Pretty' format. The response status is 200 OK, with a time of 103ms and a size of 333 B. The response body is:

```
{  "success": true,  "message": "published myTopic/myPayload"}
```

Request Details:

- Method: POST
- URL: http://localhost:1880/pub/myTopic/myPayload
- Environment: No Environment

Response Details:

- Status: 200 OK
- Time: 103ms
- Size: 333 B
- Save Response: [button]

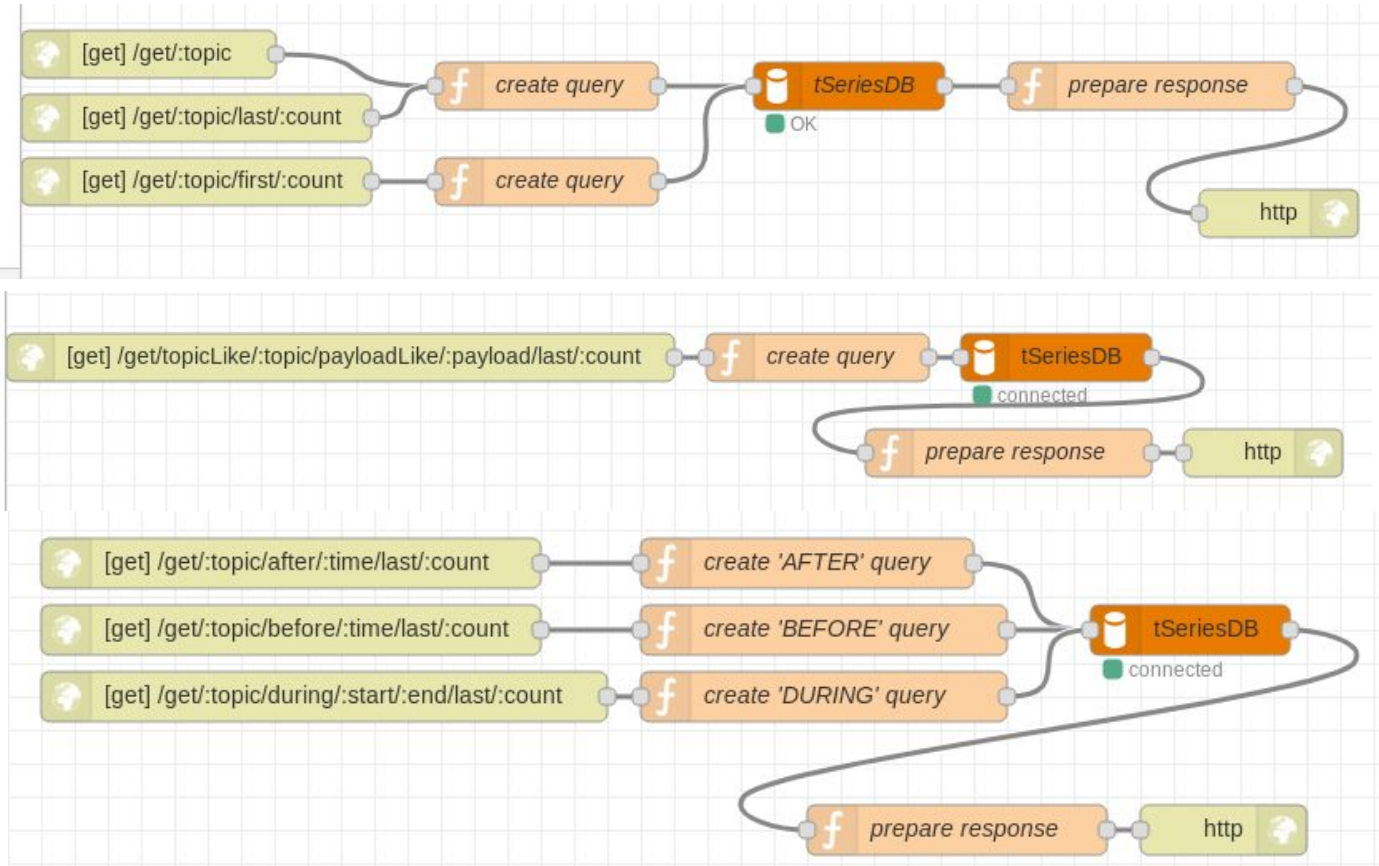
Response Body (JSON):

```
{  "success": true,  "message": "published myTopic/myPayload"}
```

REST API Interface

- Data Accessing APIs
 - Get a single record
 - Get several data records in series
 - Get one or several records based on certain condition(s)
 - Delete a single data record
 - Delete several data records in series
 - Delete one or several records based on certain condition(s)
- Tested using Postman, curl command line utility and Firefox web browser

Node-RED Date Retrieval Flows



Validation using Postman

The screenshot displays the Postman application interface. On the left, a sidebar shows a collection named "IoT-Platform-API-Testing" with 17 requests. The selected request is "retrieve data records flow (based on topic criteria)". The main panel shows the request details:

- Method:** GET
- URL:** `http://localhost:1880/get/myTopic/last/3`
- Params:** Query Params table with columns KEY, VALUE, and DESCRIPTION. The table contains one entry: Key, Value, Description.
- Body:** JSON (Pretty view) showing a successful response with status 200 OK, time 19ms, and size 522 B. The response body is a JSON array of three objects, each containing id, topic, payload, and timestamp.

```
1 {
2   {
3     "id": 5343,
4     "topic": "myTopic",
5     "payload": "myPayload",
6     "timestamp": "1573844048.900"
7   },
8   {
9     "id": 2506,
10    "topic": "myTopic",
11    "payload": "myPayload",
12    "timestamp": "1573668284.410"
13  },
14  {
15    "id": 1974,
16    "topic": "myTopic",
```

Validation using Postman

The screenshot displays the Postman application interface. On the left, a sidebar shows a collection named "IoT-Platform-API-Testing" with 17 requests. The main workspace shows a GET request to `http://localhost:1880/get/topicLike/time*/payloadLike*/last/3`. The "Query Params" section is empty. The "Body" tab is selected, showing a JSON response with three records. The status bar at the bottom indicates a 200 OK response with a time of 15ms and a size of 541 B.

Request Details:

- Method: GET
- URL: `http://localhost:1880/get/topicLike/time*/payloadLike*/last/3`
- Query Params: None
- Body: JSON

Response Body (JSON):

```
1  {
2    {
3      "id": 5437,
4      "topic": "timestamp",
5      "payload": "1573858698989",
6      "timestamp": "1573858698.991"
7    },
8    {
9      "id": 5436,
10     "topic": "timestamp",
11     "payload": "1573858683974",
12     "timestamp": "1573858683.976"
13   },
14   {
15     "id": 5435,
16     "topic": "timestamp",
```

Validation using Postman

The screenshot displays the Postman application interface. On the left, a sidebar shows a collection named "IoT-Platform-API-Testing" with 17 requests. The main area shows a GET request to the endpoint `http://localhost:1880/get/timestamp/during/1573656004.139/1573656109.209/last/5`. The request is configured with the following details:

- Method:** GET
- URL:** `http://localhost:1880/get/timestamp/during/1573656004.139/1573656109.209/last/5`
- Params:** Query Params table with columns KEY, VALUE, and DESCRIPTION.
- Body:** JSON body with the following structure:

```
1  [
2    {
3      "id": 1800,
4      "topic": "timestamp",
5      "payload": "1573656109207",
6      "timestamp": "1573656109.209"
7    },
8    {
9      "id": 1799,
10     "topic": "timestamp",
11     "payload": "1573656094192",
12     "timestamp": "1573656094.194"
13   },
14   {
15     "id": 1798,
16     "topic": "timestamp",
```

The response status is 200 OK, with a time of 25ms and a size of 715 B. The response body is displayed in JSON format.

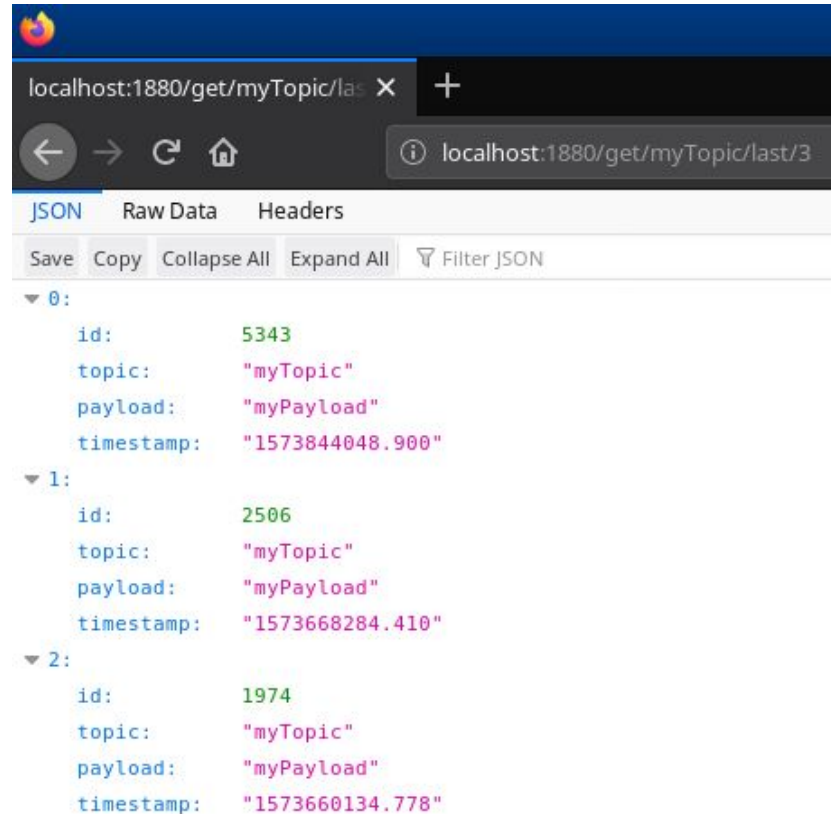
Validation using Curl

```
niranay@niranay: ~/Documents/sem7/btech_project curl -X GET "http://localhost:1880/get/myTopic/last/3" 127 10045 04:42:57
[{"id":5343,"topic":"myTopic","payload":"myPayload","timestamp":"1573844048.900"}, {"id":2506,"topic":"myTopic","payload":"myPayload","timestamp":"1573668284.410"}, {"id":1974,"topic":"myTopic","payload":"myPayload","timestamp":"1573660134.778"}]

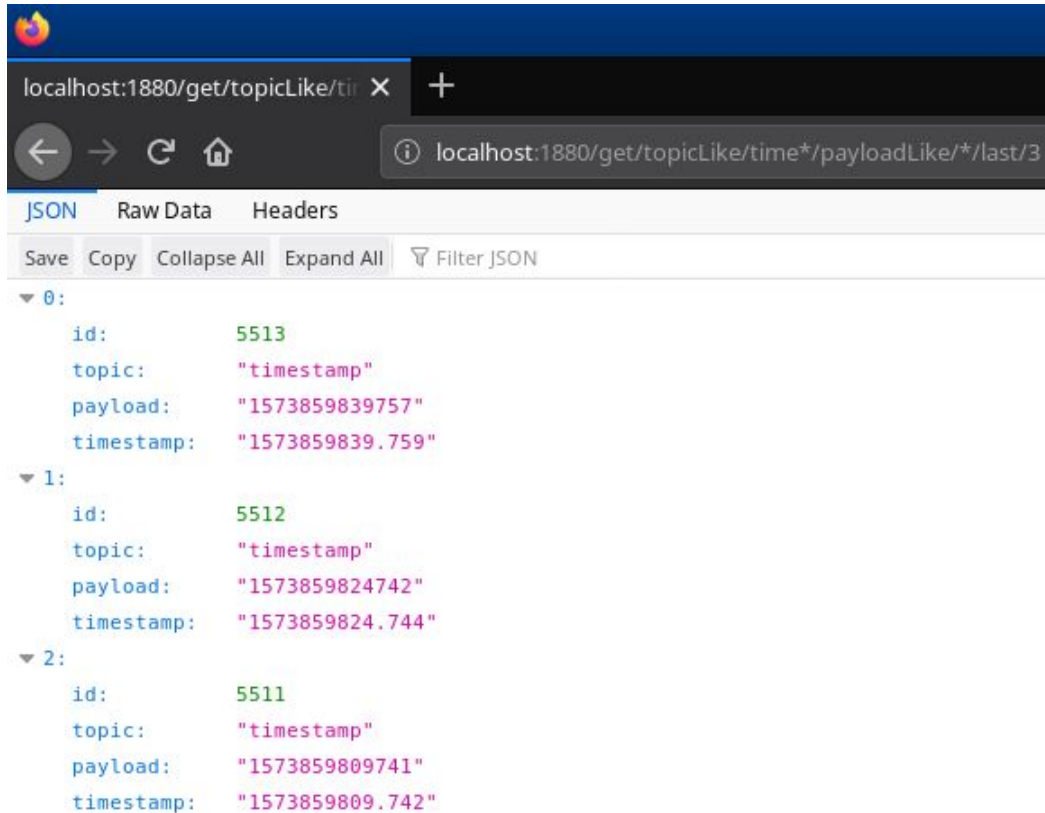
niranay@niranay: ~/Documents/sem7/btech_project curl -X GET "http://localhost:1880/get/topicLike/time*/payloadLike/*/last/3" 10046 04:43:16
[{"id":5497,"topic":"timestamp","payload":"1573859599610","timestamp":"1573859599.612"}, {"id":5496,"topic":"timestamp","payload":"1573859584608","timestamp":"1573859584.609"}, {"id":5495,"topic":"timestamp","payload":"1573859569595","timestamp":"1573859569.596"}]

niranay@niranay: ~/Documents/sem7/btech_project curl -X GET "http://localhost:1880/get/timestamp/during/1573656004.139/1573656109.209/last/5" 10048 04:43:47
[{"id":1800,"topic":"timestamp","payload":"1573656109207","timestamp":"1573656109.209"}, {"id":1799,"topic":"timestamp","payload":"1573656094192","timestamp":"1573656094.194"}, {"id":1798,"topic":"timestamp","payload":"1573656079177","timestamp":"1573656079.179"}, {"id":1797,"topic":"timestamp","payload":"1573656064162","timestamp":"1573656064.164"}, {"id":1796,"topic":"timestamp","payload":"1573656049159","timestamp":"1573656049.160"}]
```

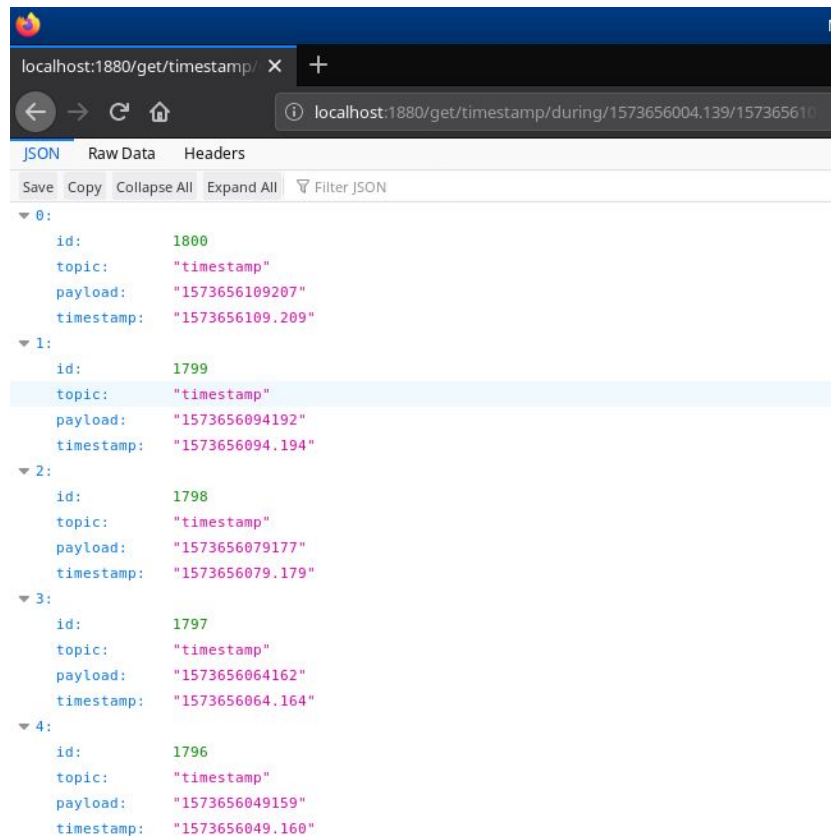
Validation using Firefox



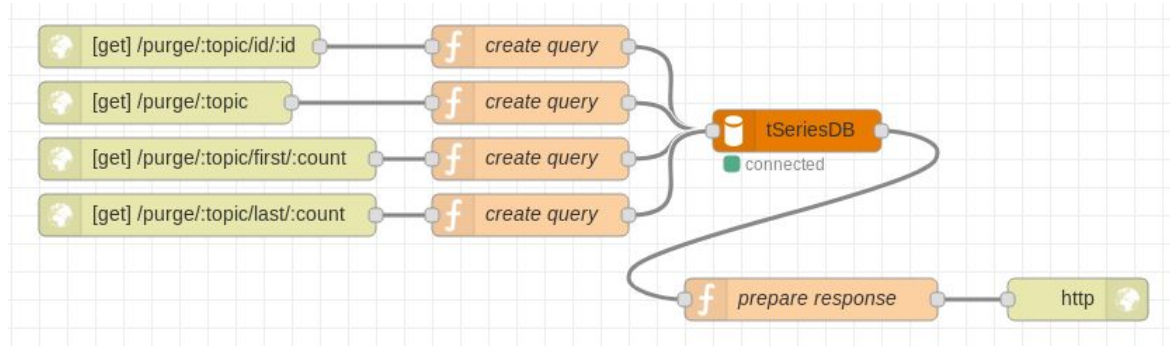
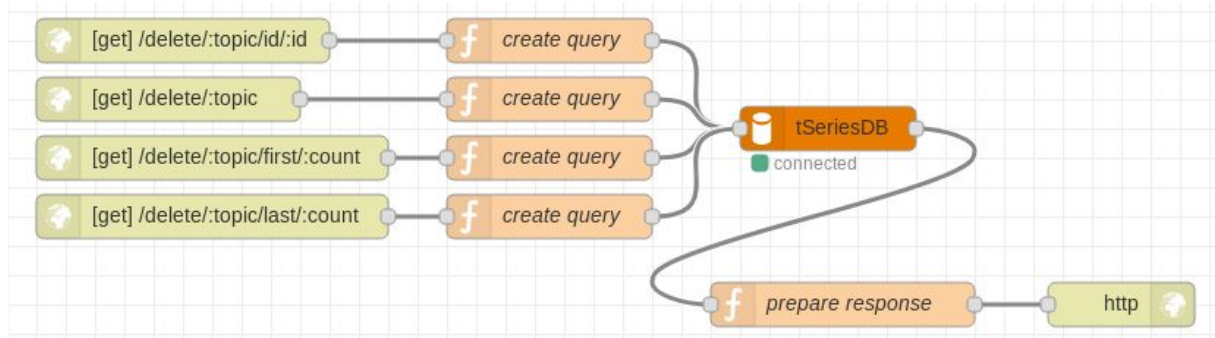
Validation using Firefox



Validation using Firefox



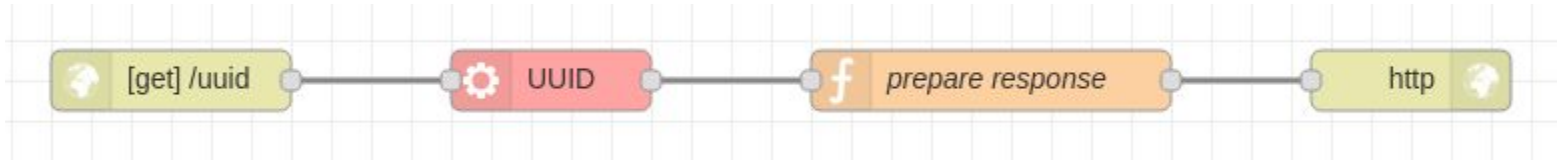
Node-RED Date Removal Flows



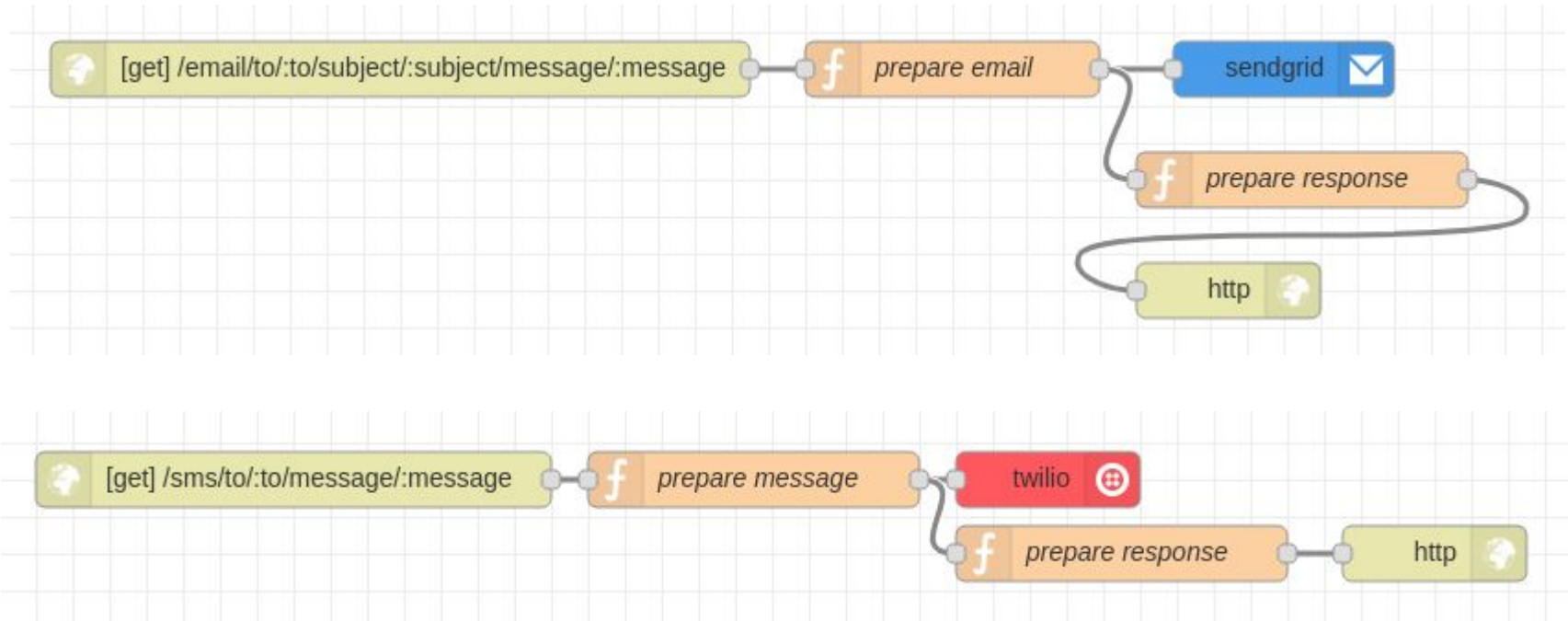
Microservices

- Twilio used for SMS notification
- SendGrid used for email notification
- Elementary microservices and utilities
 - Get current timestamp
 - Get unique or random number/string
 - Get UUID
 - Send an email
 - Send a text message
 - MQTT callback registration

Node-RED Microservices Flows



Node-RED Microservices Flows



Validation using Postman

The screenshot displays the Postman application interface. On the left, a sidebar shows a list of collections under the 'Collections' tab. The main workspace shows a GET request to `http://localhost:1880/timestamp` with a status of 200 OK. The response body is displayed in JSON format, showing a timestamp value.

Request Details:

- Method: GET
- URL: `http://localhost:1880/timestamp`
- Status: 200 OK
- Time: 42ms
- Size: 306 B

Response Body (JSON):

```
{  "timestamp": "1582211543158"}
```

Validation using Postman

The screenshot displays the Postman application interface. On the left, a sidebar shows a list of collections under the 'Collections' tab, with 'generate random code flow' selected. The main workspace shows a REST client request for the endpoint `http://localhost:1880/randomcode/32` using the GET method. The 'Params' tab is active, showing a single query parameter: 'Key' with the value 'Value'. The 'Body' tab is also visible, showing a JSON response in 'Pretty' format: `{ "code": "c6f23XzeyADnjIgtkjulnDkVWohwa1e" }`. The status bar at the bottom indicates a successful response with status 200 OK, time 30ms, and size 320 B.

Postman interface showing a REST client request for the endpoint `http://localhost:1880/randomcode/32` using the GET method. The request is configured with a single query parameter: 'Key' with the value 'Value'. The response is displayed in the 'Body' tab, showing a JSON object: `{ "code": "c6f23XzeyADnjIgtkjulnDkVWohwa1e" }`. The status bar indicates a successful response with status 200 OK, time 30ms, and size 320 B.

Validation using Postman

The screenshot displays the Postman application interface. On the left, a sidebar shows a list of collections under 'Collections', with 'generate uuid flow' selected. The main workspace shows a REST client request for the endpoint `http://localhost:1880/uuid` using the `GET` method. The 'Params' tab is active, showing a table for 'Query Params' with one parameter: 'Key' with value 'Value'. The 'Body' tab is also visible, showing a JSON response in 'Pretty' format: `{ "uuid": "0c02aa31-c19b-4164-95dd-e14c69c99e98" }`. The status bar at the bottom indicates a successful response with status `200 OK`, time `123ms`, and size `324 B`.

GET `http://localhost:1880/uuid` Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (8) Test Results Status: 200 OK Time: 123ms Size: 324 B Save Response

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   "uuid": "0c02aa31-c19b-4164-95dd-e14c69c99e98"
3 }
```

Validation using Postman

The screenshot displays the Postman application interface. On the left, a sidebar shows a list of collections under 'History' and 'Collections'. The 'Collections' tab is active, showing a list of API endpoints. The main workspace is titled 'My Workspace' and shows a collection named 'send sms flow'. The selected environment is 'No Environment'. The request is a GET method to the URL 'http://localhost:1880/sms/to/+917350911970/message/automated_message_generated_using_twilio'. The 'Send' button is highlighted. Below the URL bar, tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', and 'Settings' are visible. The 'Body' tab is active, showing a JSON response in 'Pretty' format. The response status is '200 OK', time is '45ms', and size is '373 B'. The response body is:

```
1 {
2   "smsTo": "+917350911970",
3   "message": "automated_message_generated_using_twilio",
4   "status": "queued"
5 }
```

At the bottom, there are buttons for 'Bootcamp', 'Build', and 'Browse'.

Validation using Postman

The screenshot displays the Postman application interface. On the left, a sidebar shows a list of collections under 'Collections' and 'APIs BETA'. The main workspace is titled 'My Workspace' and shows a collection named 'send email flow'. The selected request is a GET request to the URL 'http://localhost:1880/email/to/niramay.vaidya@gmail.com/subject/IoT_platform_email_microservice_testing/message/auto...'. The request is in the 'Params' tab, showing a 'Query Params' table with one entry: 'Key' with 'Value'. The response is shown in the 'Body' tab, displaying a JSON object: {'to': 'niramay.vaidya@gmail.com', 'status': 'email queued'}. The status is 200 OK, time is 472ms, and size is 334 B.

GET `http://localhost:1880/email/to/niramay.vaidya@gmail.com/subject/IoT_platform_email_microservice_testing/message/auto...` **Send** **Save**

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (8) Test Results Status: 200 OK Time: 472ms Size: 334 B Save Response

Pretty Raw Preview Visualize BETA JSON **Send**

```
1  
2  "to": "niramay.vaidya@gmail.com",  
3  "status": "email queued"  
4
```


Received Email

IoT_platform_email_microservice_testing  Inbox x



niramay.vaidya@gmail.com via sendgrid.net
to me ▼

Mon, Dec 16, 2019, 8:03 PM



automated_email_generated_using_sendgrid

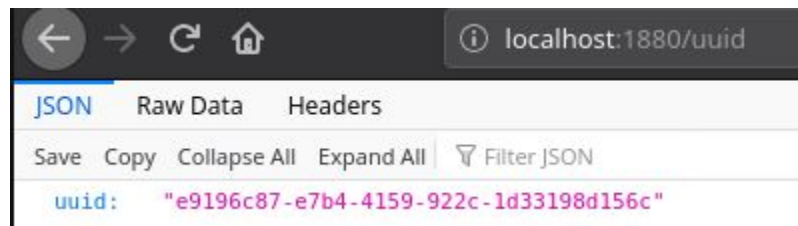
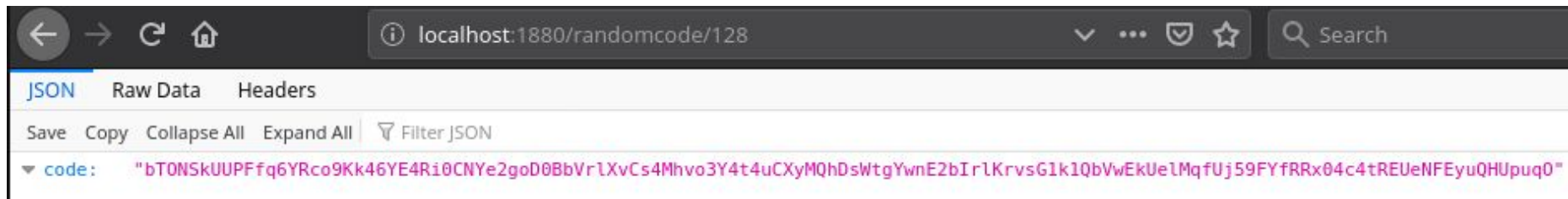
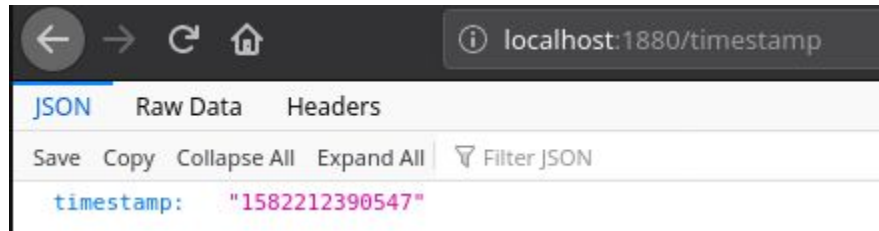
 Reply

 Forward

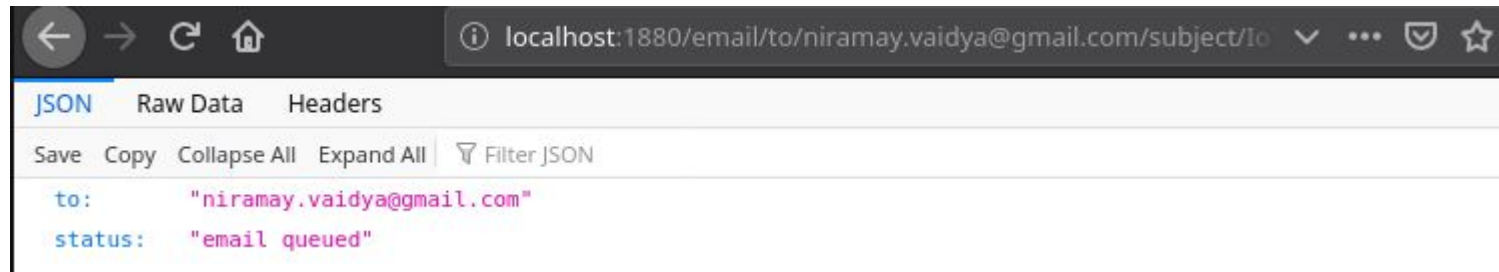
Validation using Curl

```
niranay@niranay: ~/Documents/sem7/btech project curl -X GET "http://localhost:1880/timestamp" 10081 20:29:07
{"timestamp":"1582212169520"}
niranay@niranay: ~/Documents/sem7/btech project curl -X GET "http://localhost:1880/randomcode/64" 10082 20:52:49
{"code":"evOvEApPTdPQhJFGWIQnub24yKhaxCxsp415xMEEpTu208PgPbOvERCWAu6Zyvie"}
niranay@niranay: ~/Documents/sem7/btech project curl -X GET "http://localhost:1880/uuid" 10083 20:53:13
{"uuid":"a4ccf5ad-1098-41ac-b55c-3fbfd8d8785f"}
niranay@niranay: ~/Documents/sem7/btech project curl -X GET "http://localhost:1880/sms/to/+917350911970/message/automated_message_generated_using_twilio"
{"smsTo":"+917350911970","message":"automated message generated using twilio","status":"queued"}
niranay@niranay: ~/Documents/sem7/btech project curl -X GET "http://localhost:1880/email/to/niranay.vaidya@gmail.com/subject/IoT_platform_email_microservice_testing
/message/automated_email_generated_using_sendgrid"
{"to":"niranay.vaidya@gmail.com","status":"email queued"}
niranay@niranay: ~/Documents/sem7/btech project 10086 20:53:57
```

Validation using Firefox




Validation using Firefox



Rule Engine

- Schema details
 - id - primary key
 - ruleName - name of the rule
 - active - rule active or not
 - topicPattern - basic on topic, decide rule
 - payloadPattern - based on payload, decide rule
 - method - based on topicPattern and payloadPattern decide which method to use
 - webHook - use this value to call using defined method
- Node-RED used to establish a flow for searching rules
- Node-RED used to establish flows for enabling and disabling specific or all rule and creating a new rule

Rule Engine Schema

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id 	int(11)			No	None		AUTO_INCREMENT
2	ruleName	varchar(255)	latin1_swedish_ci		No	None		
3	active	binary(1)			No			
4	topicPattern	varchar(1024)	latin1_swedish_ci		No	None		
5	payloadPattern	varchar(2048)	latin1_swedish_ci		No	None		
6	method	varchar(7)	latin1_swedish_ci		No	GET		
7	webHook	varchar(1024)	latin1_swedish_ci		No	None		

phpMyAdmin View

phpMyAdmin

Recent Favorites

information_schema
tSeriesDB

Server: localhost » Database: tSeriesDB » Table: ruleEngine

Browse Structure SQL Search Insert Export Import Operations Tracking Triggers

Showing rows 0 - 3 (4 total, Query took 0.0007 seconds.)

```
SELECT * FROM `ruleEngine`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 | Filter rows: Search this table

Sort by key: None

+ Options

	id	ruleName	active	topicPattern	payloadPattern	method	webHook
Edit Copy Delete	1	myTopic rule 1	1	myTopic%	%	POST	http://localhost:1880/pub/modifiedTopic/rule-engin...
Edit Copy Delete	2	myTopic rule 2	1	myTopic%	%	POST	http://localhost:1880/pub/modifiedTopic/rule-engin...
Edit Copy Delete	3	again rule	1	%	%again	POST	http://localhost:1880/pub/modifiedTopic/again-foun...
Edit Copy Delete	4	testRule	0	%Topic	%	GET	http://localhost:1880/email/to/niramay.vaidya@gmai...

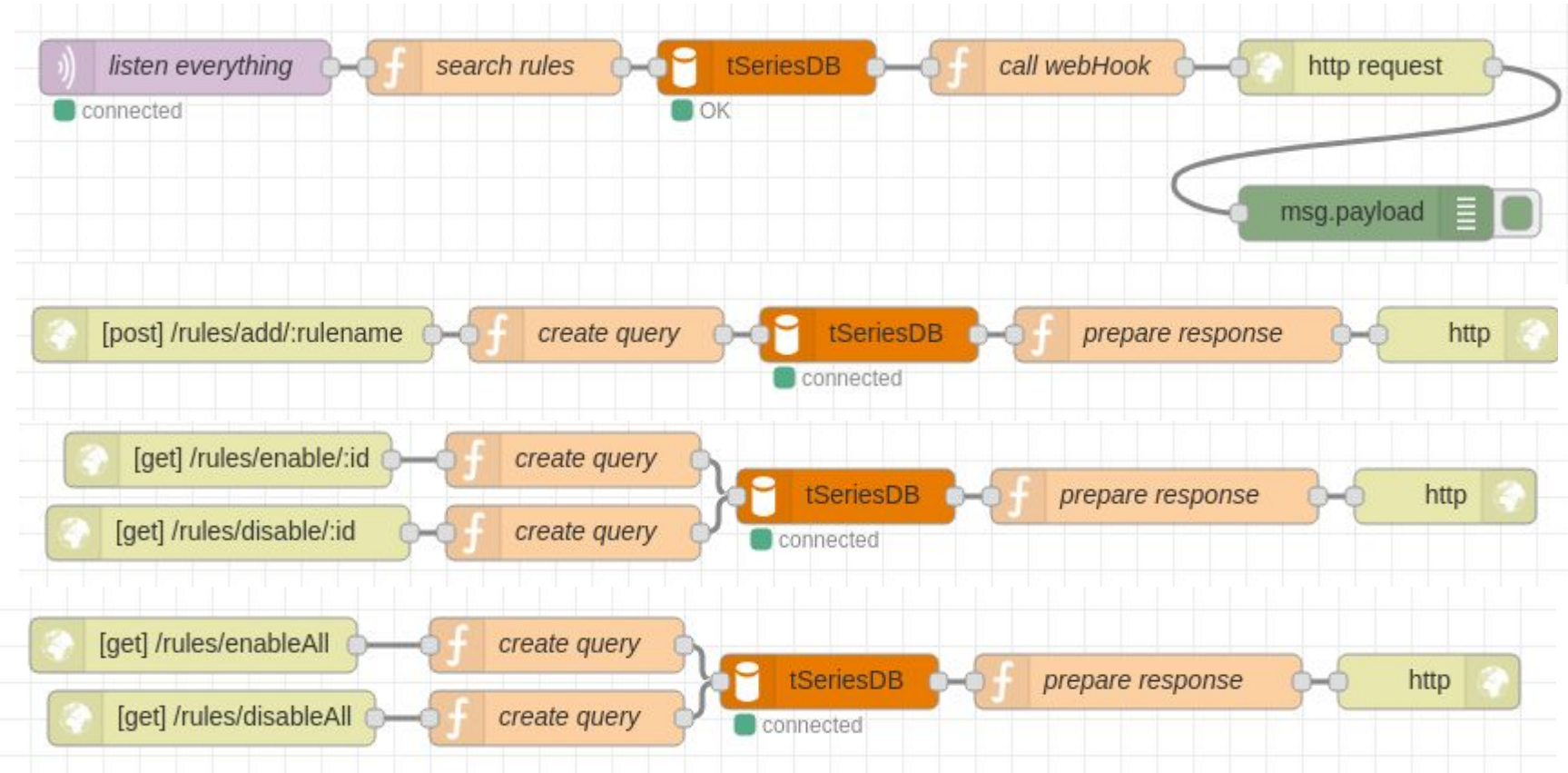
Check all With selected: Edit Copy Delete Export

Show all | Number of rows: 25 | Filter rows: Search this table

Query results operations

Console

Node-RED Microservices Flows



Validation using Postman

The screenshot displays the Postman application interface. On the left sidebar, the 'Collections' tab is active, showing a list of API collections. The 'enable rule flow' collection is selected. The main workspace shows a REST client request configured with the following details:

- Method:** GET
- URL:** http://localhost:1880/rules/enable/3
- Environment:** No Environment
- Query Params:** A table with one entry: Key (Value) with Description (Description).
- Body:** The 'Body' tab is selected, showing a JSON response: `{ "status": "enable success" }`.
- Status:** 200 OK, Time: 165ms, Size: 304 B.

The bottom status bar indicates the user is logged in as 'Bootcamp' and provides links to 'Build', 'Browse', and other utility icons.

Validation using Postman

The screenshot displays the Postman application interface. On the left sidebar, a list of collections is visible, including 'delete data records flow', 'purge data records flow', 'get current timestamp flow', 'generate random code flow', 'generate uuid flow', 'send sms flow', 'send email flow', 'enable rule flow', 'disable rule flow' (which is currently selected), 'enable all rules flow', 'disable all rules flow', and a 'postman-tutorial' folder containing 2 requests.

The main workspace shows a REST client request for the endpoint `http://localhost:1880/rules/disable/3`. The request method is `GET`. The 'Params' tab is active, showing a single query parameter with the key 'Key' and the value 'Value'. The 'Body' tab is also visible, showing a JSON response in 'Pretty' format: `{ "status": "disable success" }`. The status bar at the bottom indicates a successful response with status 200 OK, a time of 116ms, and a size of 305 B.

KEY	VALUE	DESCRIPTION
Key	Value	Description

```
1 {
2   "status": "disable success"
3 }
```

Validation using Postman

The screenshot displays the Postman application interface. On the left sidebar, the 'Collections' tab is active, showing a list of API collections. The 'enable all rules flow' collection is selected, and the 'enable all rules flow' request is highlighted. The main panel shows the details of this request, which is a GET request to the URL 'http://localhost:1880/rules/enableAll'. The 'Params' tab is active, showing a table of query parameters. The 'Body' tab is also visible, showing the response body in JSON format. The status bar at the bottom indicates the request was successful with a 200 OK status, a response time of 96ms, and a size of 308 B.

enable all rules flow

GET http://localhost:1880/rules/enableAll

Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (8) Test Results Status: 200 OK Time: 96ms Size: 308 B Save Response

Pretty Raw Preview Visualize BETA JSON

```
1
2  "status": "enable all success"
3
```

Bootcamp Build Browse

Validation using Postman

The screenshot displays the Postman application interface. On the left sidebar, the 'Collections' tab is active, showing a list of API collections. The 'disable all rules flow' collection is selected. The main workspace shows a REST client request for the endpoint `http://localhost:1880/rules/disableAll` using the GET method. The request is configured with 7 headers and 1 query parameter (Key: Value). The response is displayed in the 'Body' tab, showing a JSON object: `{ "status": "disable all success" }`. The status is 200 OK, with a response time of 139ms and a size of 309 B.

Request Details:

- Method: GET
- URL: `http://localhost:1880/rules/disableAll`
- Headers (7):
- Query Params (1):


KEY	VALUE	DESCRIPTION
Key	Value	Description

Response Details:

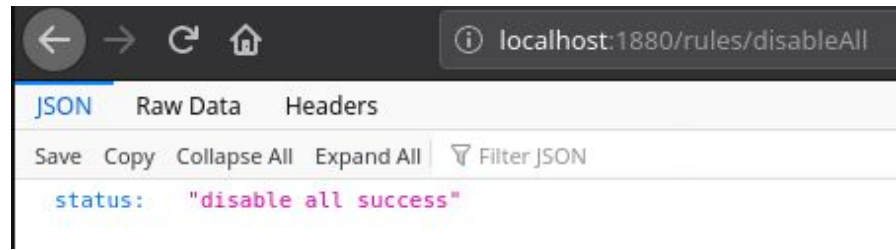
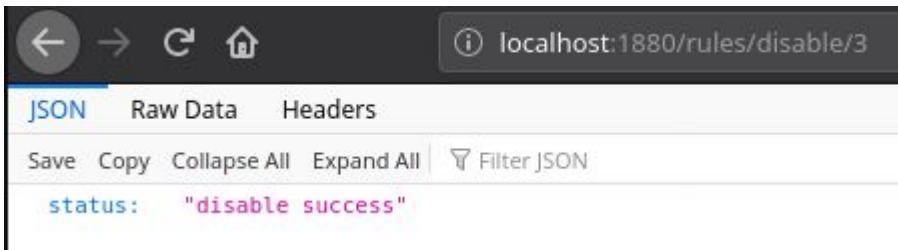
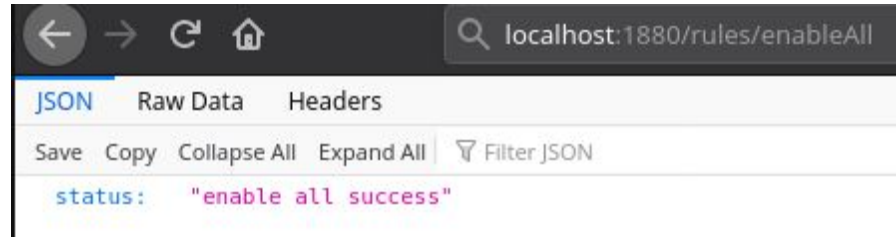
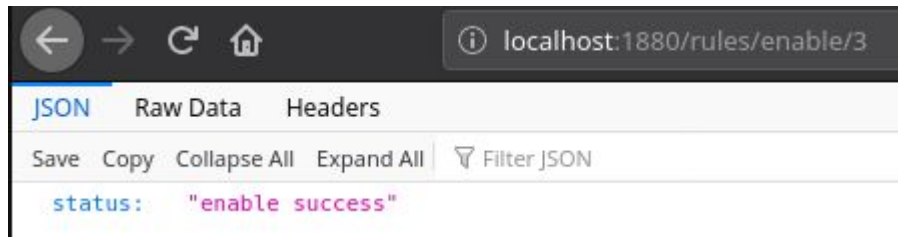
- Status: 200 OK
- Time: 139ms
- Size: 309 B
- Body (JSON): `{ "status": "disable all success" }`

Validation using Curl

```
niramay@niramay: ~/Documents/sem7/btech project$ curl -X POST "http://localhost:1880/rules/add/testRule" --data-urlencode "topicPattern=%myTopic" --data-urlencode "payloadPattern=%" --data-urlencode "method=GET" --data-urlencode "webHook=http://localhost:1880/email/to/niramay.vaidya@gmail.com/subject/IoT_platform_email_microservice_testing_rule_trigger/message/automated_email_generated_using_sendgrid" --data-urlencode '{"status":"rule added","ruleId":5}'
niramay@niramay: ~/Documents/sem7/btech project$
```



Validation using Firefox



The End Result

