DSS Backend Configuration Manual

Backend

Overview

DSS has two sides to it. One being the process in which the Data is pooled to ElasticSearch and the other being the way it is fetched, aggregated, computed, transformed and sent across.
As this revolves around a variety of Data Set, there is a need for making this configurable. So that, tomorrow, given a new scenario is introduced, then it is just a configuration away from getting the newly introduced scenario involved in this flow of process.

This document explains the steps on how to define the configurations for both sides of DSS. Analytics and Ingest Pipeline Services.

Abbreviations

Ingest: Micro Service which runs as a pipeline and manages to validate, transform and enrich the incoming data and pushes the same to ElasticSearch Index

Analytics: Micro Service which is responsible for building, fetching, aggregating and computing the Data on ElasticSearch to a consumable Data Response. This shall be later used for visualizations and graphical representations.

JOLT: JSON to JSON transformation library written in Java where the "specification" for the transform is itself a JSON document

Modules / Domain Level: These are the Services in this context. Each of the services, such as Property Tax, Trade License, Water and Sewerages are considered as Modules / Domains

Chart: Each individual graphical representation is considered as Chart in specific. For example, a Metric of Total Collection is considered as a Chart.

Visualization: Group of different Charts is considered as a Visualization. For example, the group of Total Collection, Target Collection and Target Achieved, is considered as a Metric Collection of Charts and thus it becomes Visualization.

Ingest Pipeline Configurations

Below are list of configurations

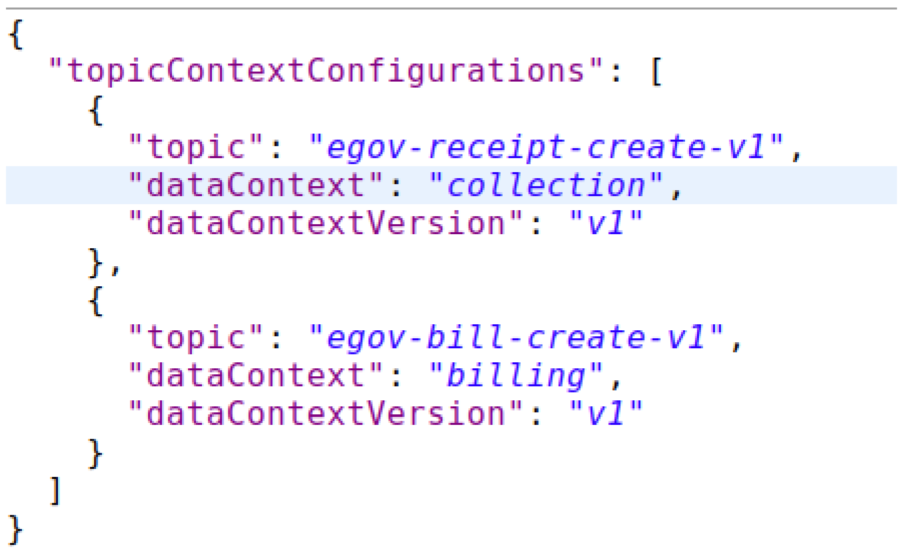
1. Topic Context Configurations
2. Validator Schema
3. JOLT Transformation Schema
4. Enrichment Domain Configuration
5. JOLT Domain Transformation Schema

Descriptions

1. Topic Context Configurations

Topic Context Configuration is an outline to define which data is received on which Kafka Topic.

Indexer Service and many other services are sending out data on different Kafka Topics. If the Ingest Service is asked to receive those data and pass it through the pipeline, the context and the version of the data being received has to be set. This configuration is used to identify as in which kafka topic consumed the data and what is the mapping for that.



```
{
  "topicContextConfigurations": [
    {
      "topic": "egov-receipt-create-v1",
      "dataContext": "collection",
      "dataContextVersion": "v1"
    },
    {
      "topic": "egov-bill-create-v1",
      "dataContext": "billing",
      "dataContextVersion": "v1"
    }
  ]
}
```

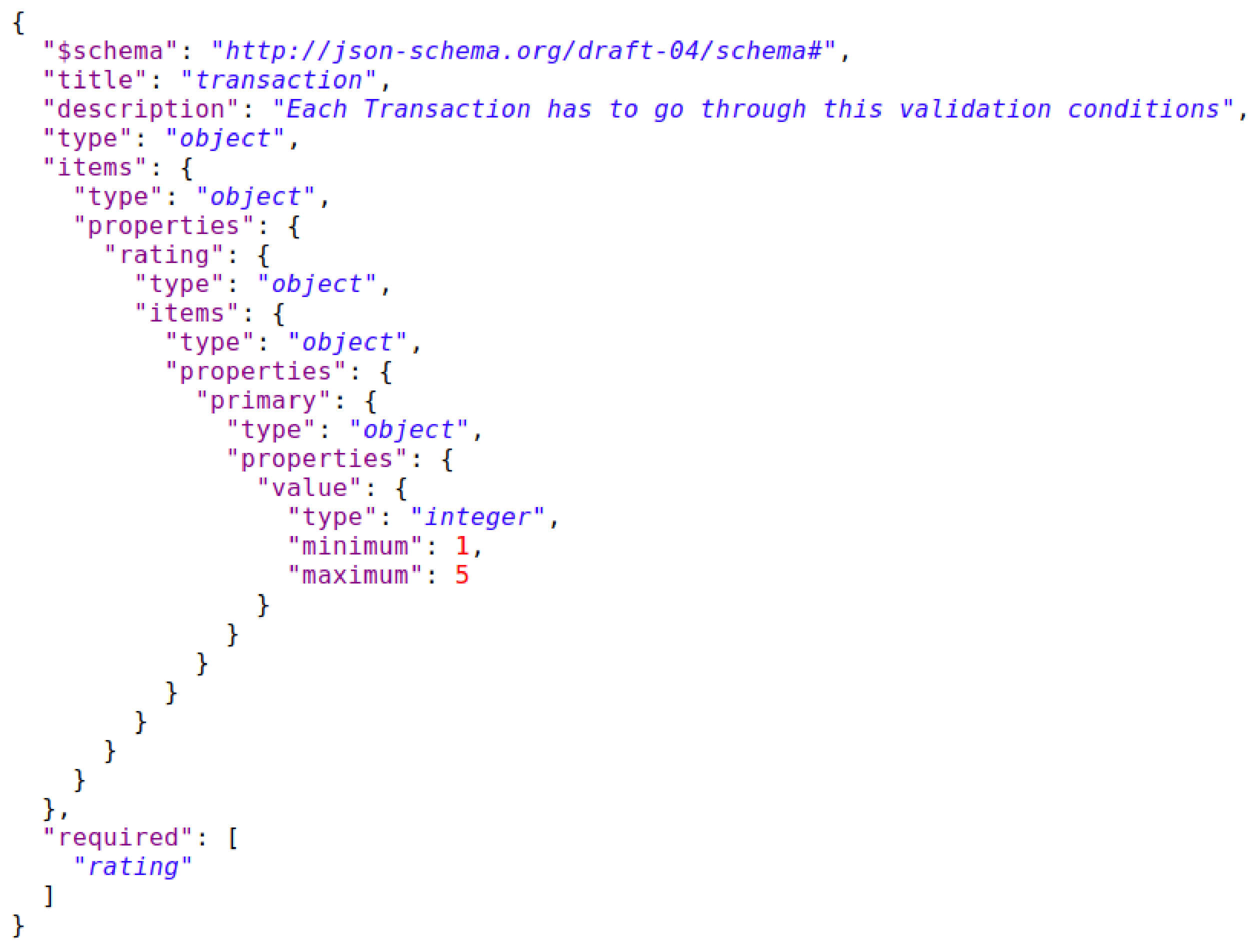
Figure 1

Click here for Full Configuration

Parameter Name	Description
topic	Holds the name of the Kafka Topic on which the data is being received
dataContext	Context Name which needs to be set for further actions in the pipeline
dataContextVersion	Version of the Data Structure is set here as there might be different structured data at different point of time

2. Validator Schema

Validator Schema is a configuration Schema Library from **Everit.** By passing the data against this schema, it ensures whether the data abides by the rules and requirements of the schema which has been defined.



```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "transaction",
  "description": "Each Transaction has to go through this validation conditions",
  "type": "object",
  "items": {
    "type": "object",
    "properties": {
      "rating": {
        "type": "object",
        "items": {
          "type": "object",
          "properties": {
            "primary": {
              "type": "object",
              "properties": {
                "value": {
                  "type": "integer",
                  "minimum": 1,
                  "maximum": 5
                }
              }
            }
          }
        }
      }
    }
  },
  "required": [
    "rating"
  ]
}
```

Figure 2

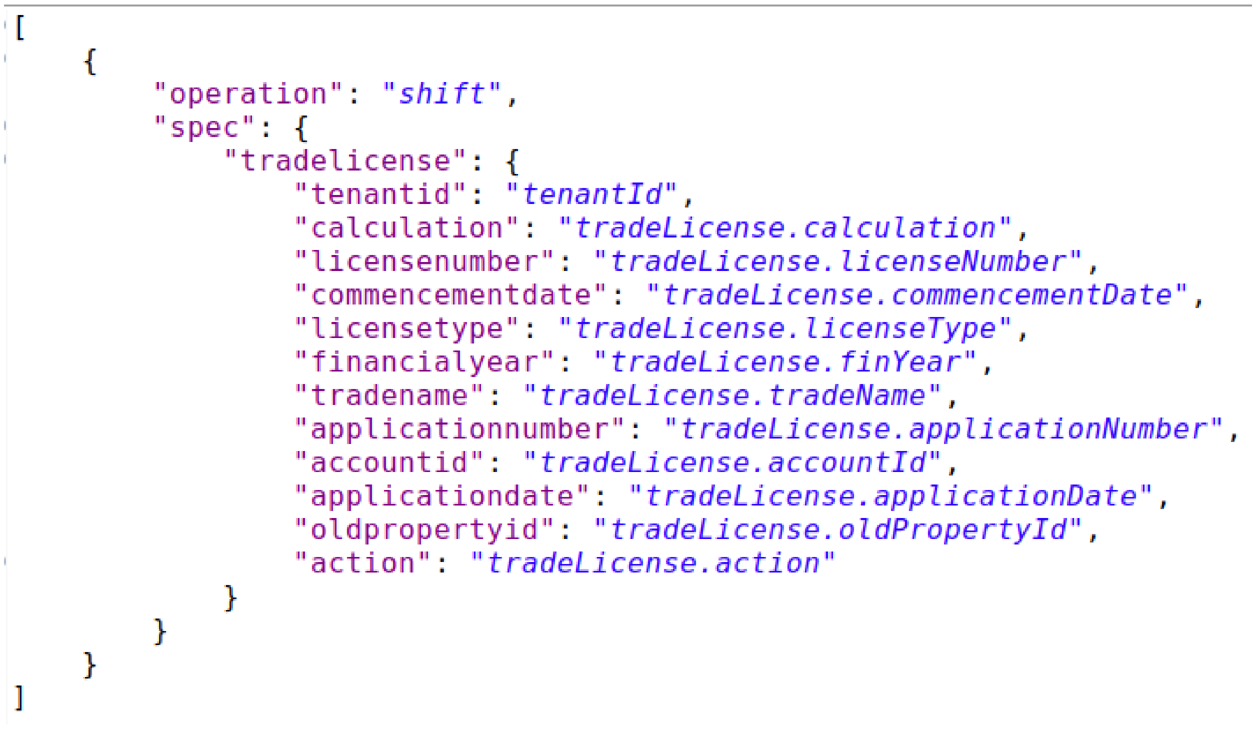
Click here for an example configuration

3. JOLT Transformation Schema

JOLT is a JSON to JSON Transformation Library. In order to change the structure of the data and transform it in a generic way, JOLT has been used.

While the transformation schemas are written for each Data Context, the data is transformed against the schema to obtain a transformed data.

Follow the slide deck for JOLT Transformations



```
[
    {
        "operation": "shift",
        "spec": {
            "tradelicense": {
                "tenantid": "tenantId",
                "calculation": "tradeLicense.calculation",
                "licensenumber": "tradeLicense.licenseNumber",
                "commencementdate": "tradeLicense.commencementDate",
                "licensetype": "tradeLicense.licenseType",
                "financialyear": "tradeLicense.finYear",
                "tradename": "tradeLicense.tradeName",
                "applicationnumber": "tradeLicense.applicationNumber",
                "accountid": "tradeLicense.accountId",
                "applicationdate": "tradeLicense.applicationDate",
                "oldpropertyid": "tradeLicense.oldPropertyId",
                "action": "tradeLicense.action"
            }
        }
    }
]
```

Figure 3

Click here for an example configuration

4. Enrichment Domain Configuration

This configuration defines and directs the Enrichment Process which the data goes through.

For example, if the Data which is incoming is belonging to a Collection Module data, then the Collection Domain Config is picked. And the based on the Business Type specified in the data, the right config is picked.

In order to enhance the data of Collection, the domain index specified in the configuration is queried with the right arguments and the response data is obtained, transformed and set.

Click here for an example configuration

Paremter Name	**Description**
id	Unique Identifier for the Configuration within the configuration document
businessType	This defines as in which kind of Domain / Service is the data related to. Based on this business type, query and enhancements are decided
indexName	Based on Business Type, Index Name is defined as to which index has to be queried to get the enhancements done from
query	Query to execute to get the Domain Level Object is defined here.
targetReferences sourceReference	Fields which are variables in order to get the domain level objects are defined here. The variables and where all the values has to be picked from are documented here

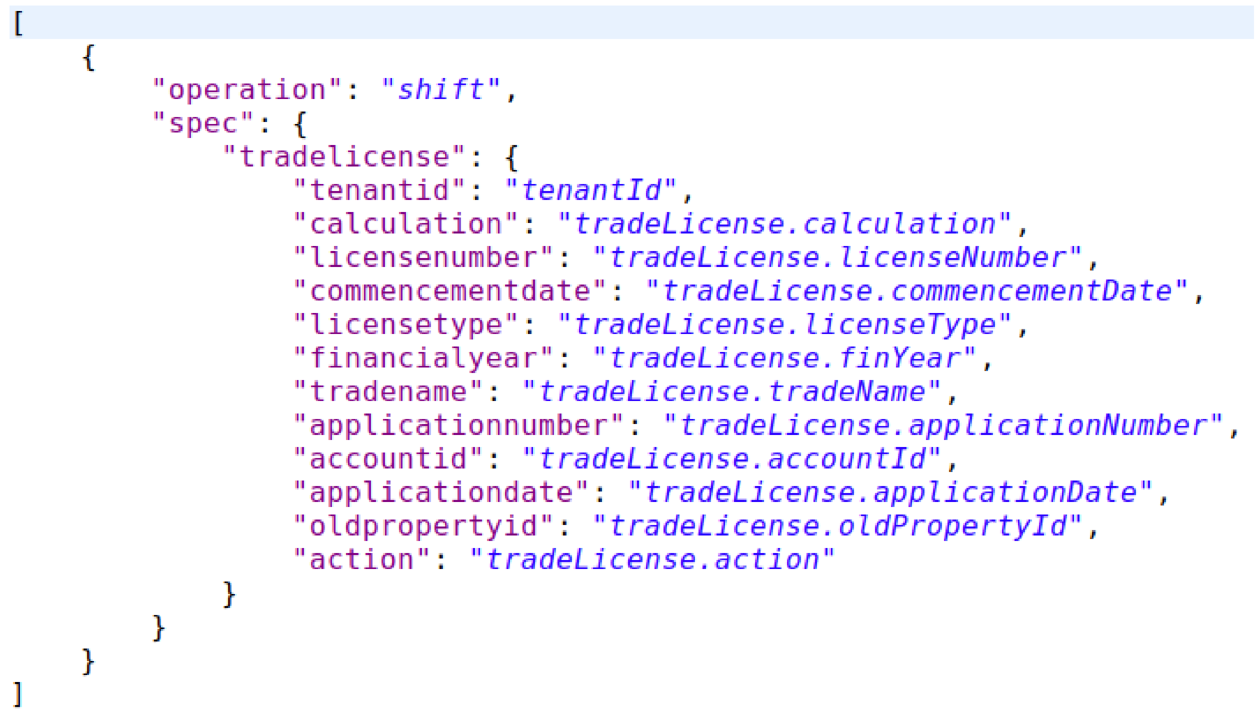
5. JOLT Domain Transformation Schema

As a part of Enhancement, once the domain level object is obtained, we might not need the complete document as is in the end data product.

Only those parameters which should be or can be used for an aggregation and representation are to be held and others are to be discarded.

In order to do that, we make use of JOLT again and write schemas to keep the required ones and discard the unwanted ones.

The above configuration is used to transform the data response in the enrichment layer.



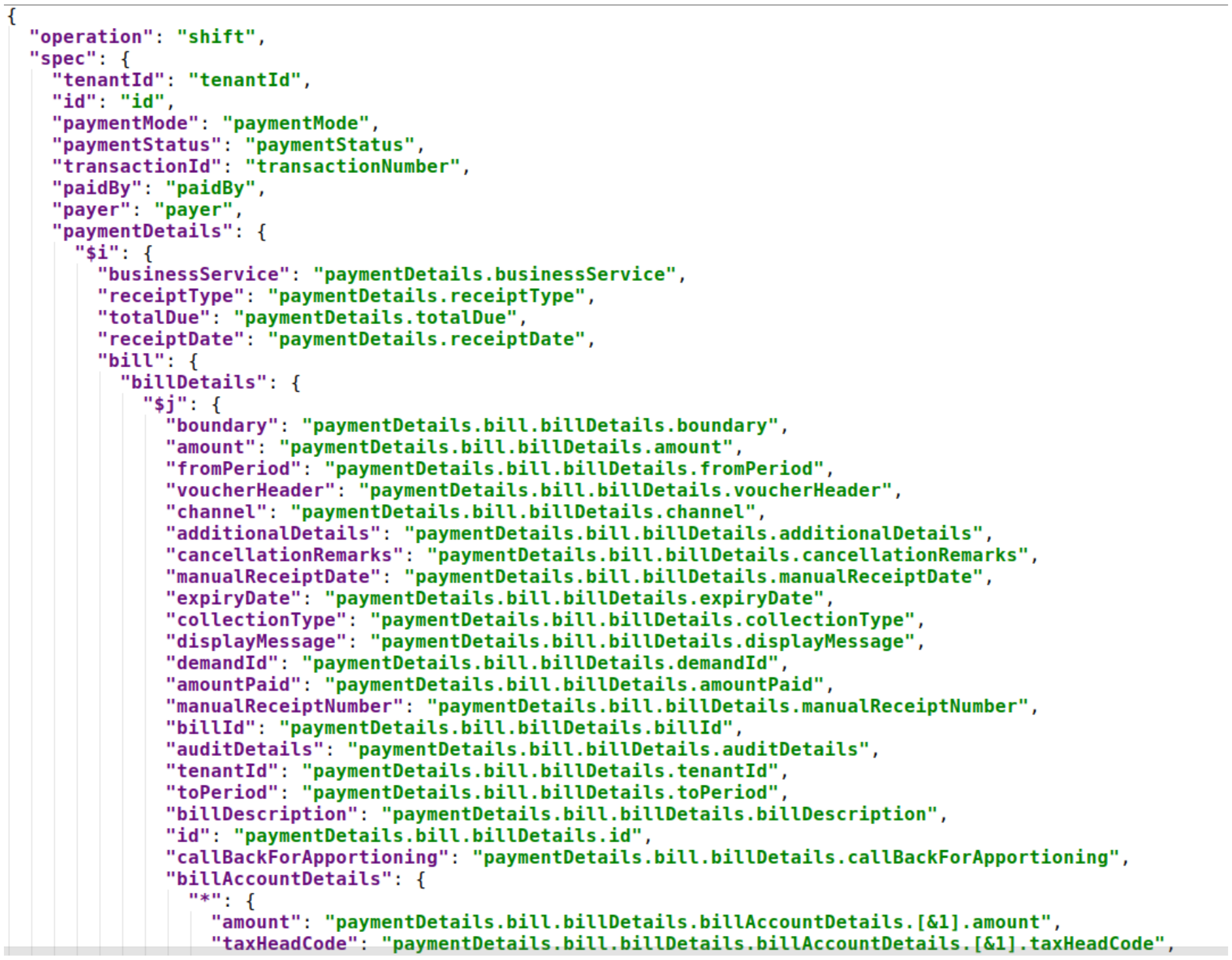
```
[
    {
        "operation": "shift",
        "spec": {
            "tradelicense": {
                "tenantid": "tenantId",
                "calculation": "tradeLicense.calculation",
                "licensenumber": "tradeLicense.licenseNumber",
                "commencementdate": "tradeLicense.commencementDate",
                "licensetype": "tradeLicense.licenseType",
                "financialyear": "tradeLicense.finYear",
                "tradename": "tradeLicense.tradeName",
                "applicationnumber": "tradeLicense.applicationNumber",
                "accountid": "tradeLicense.accountId",
                "applicationdate": "tradeLicense.applicationDate",
                "oldpropertyid": "tradeLicense.oldPropertyId",
                "action": "tradeLicense.action"
            }
        }
    }
]
```

Figure 4

[Click here for an example configuration]()

6. Use case: JOLT Transformation Schema for collection V2

JOLT transformation schema for payment-v1 has taken as a use case to explain the context collection and context version v2. The payment records are processed/transformed with the schema. The schema supports splitting the billing records in a independent new record. So if there are 2 bill items in the collection/payment incoming data then this results into 2 collection records in turn.



```
{
  "operation": "shift",
  "spec": {
    "tenantId": "tenantId",
    "id": "id",
    "paymentMode": "paymentMode",
    "paymentStatus": "paymentStatus",
    "transactionId": "transactionNumber",
    "paidBy": "paidBy",
    "payer": "payer",
    "paymentDetails": {
      "$i": {
        "businessService": "paymentDetails.businessService",
        "receiptType": "paymentDetails.receiptType",
        "totalDue": "paymentDetails.totalDue",
        "receiptDate": "paymentDetails.receiptDate",
        "bill": {
          "billDetails": {
            "$j": {
              "boundary": "paymentDetails.bill.billDetails.boundary",
              "amount": "paymentDetails.bill.billDetails.amount",
              "fromPeriod": "paymentDetails.bill.billDetails.fromPeriod",
              "voucherHeader": "paymentDetails.bill.billDetails.voucherHeader",
              "channel": "paymentDetails.bill.billDetails.channel",
              "additionalDetails": "paymentDetails.bill.billDetails.additionalDetails",
              "cancellationRemarks": "paymentDetails.bill.billDetails.cancellationRemarks",
              "manualReceiptDate": "paymentDetails.bill.billDetails.manualReceiptDate",
              "expiryDate": "paymentDetails.bill.billDetails.expiryDate",
              "collectionType": "paymentDetails.bill.billDetails.collectionType",
              "displayMessage": "paymentDetails.bill.billDetails.displayMessage",
              "demandId": "paymentDetails.bill.billDetails.demandId",
              "amountPaid": "paymentDetails.bill.billDetails.amountPaid",
              "manualReceiptNumber": "paymentDetails.bill.billDetails.manualReceiptNumber",
              "billId": "paymentDetails.bill.billDetails.billId",
              "auditDetails": "paymentDetails.bill.billDetails.auditDetails",
              "tenantId": "paymentDetails.bill.billDetails.tenantId",
              "toPeriod": "paymentDetails.bill.billDetails.toPeriod",
              "billDescription": "paymentDetails.bill.billDetails.billDescription",
              "id": "paymentDetails.bill.billDetails.id",
              "callBackForApportioning": "paymentDetails.bill.billDetails.callBackForApportioning",
              "billAccountDetails": {
                "*": {
                  "amount": "paymentDetails.bill.billDetails.billAccountDetails.[&1].amount",
                  "taxHeadCode": "paymentDetails.bill.billDetails.billAccountDetails.[&1].taxHeadCode",
```

Figure 5

Click here for an example configuration

Here: **$i**, the variable value that gets incremented for the number of records of paymentDetails

$j, the variable value that gets incremented for the number of records of bill Details.

Note: For kafka connect to work, Ingest pipeline application properties or in environments direct push must be disabled.

```
es.push.direct=false
```

Analytics Configurations

Below are list of configurations

1. Chart API Configuration
2. Master Dashboard Configuration
3. Role Dashboard Mappings Configuration

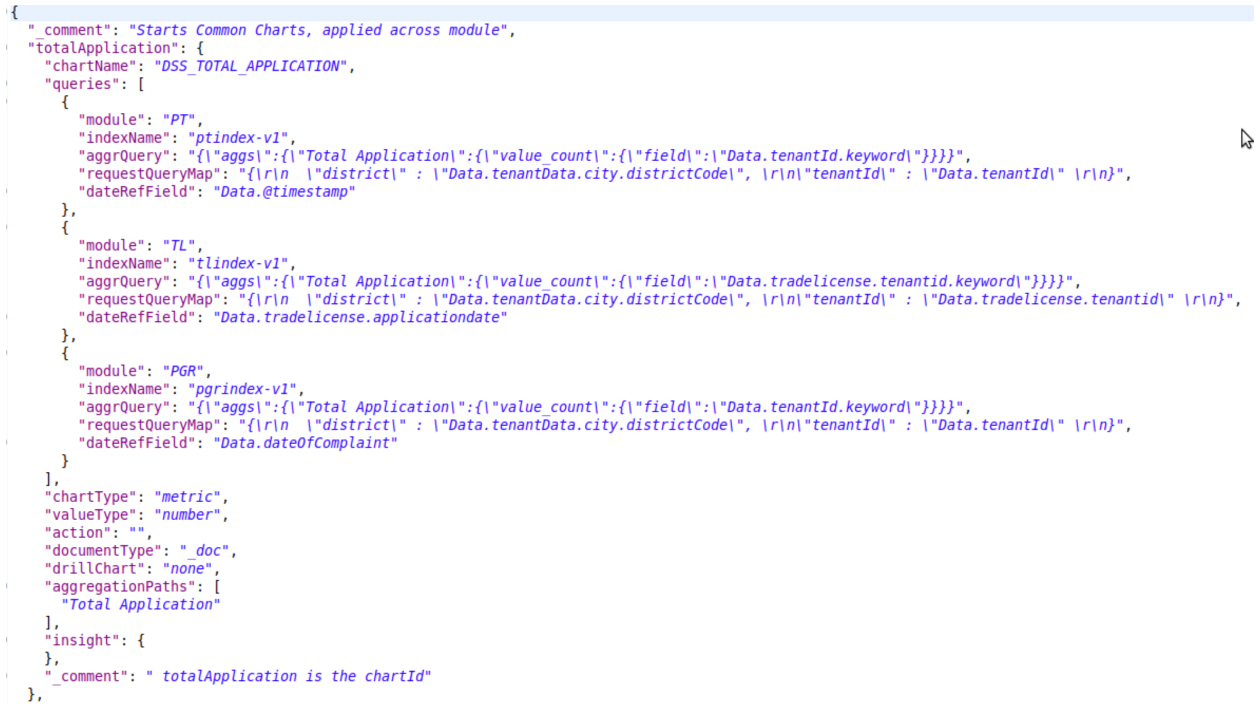
Description

Chart API Configuration

Each Visualization has its own properties. Each Visualization comes from different data sources (Sometimes it is a combination of different data sources)

In order to configure each visualization and its properties, we have Chart API Configuration Document.

In this, Visualization Code, which happens to be the key, will be having its properties configured as a part of configuration and are easily changeable.



```
{
  "_comment": "Starts Common Charts, applied across module",
  "totalApplication": {
    "chartName": "DSS_TOTAL_APPLICATION",
    "queries": [
      {
        "module": "PT",
        "indexName": "ptindex-v1",
        "aggrQuery": "{\"aggs\":{\"Total Application\":{\"value_count\":{\"field\":\"Data.tenantId.keyword\"}}}}",
        "requestQueryMap": "{\r\n  \"district\" : \"Data.tenantData.city.districtCode\", \r\n\"tenantId\" : \"Data.tenantId\" \r\n}",
        "dateRefField": "Data.@timestamp"
      },
      {
        "module": "TL",
        "indexName": "tlindex-v1",
        "aggrQuery": "{\"aggs\":{\"Total Application\":{\"value_count\":{\"field\":\"Data.tradelicense.tenantid.keyword\"}}}}",
        "requestQueryMap": "{\r\n  \"district\" : \"Data.tenantData.city.districtCode\", \r\n\"tenantId\" : \"Data.tradelicense.tenantid\" \r\n}",
        "dateRefField": "Data.tradelicense.applicationdate"
      },
      {
        "module": "PGR",
        "indexName": "pgrindex-v1",
        "aggrQuery": "{\"aggs\":{\"Total Application\":{\"value_count\":{\"field\":\"Data.tenantId.keyword\"}}}}",
        "requestQueryMap": "{\r\n  \"district\" : \"Data.tenantData.city.districtCode\", \r\n\"tenantId\" : \"Data.tenantId\" \r\n}",
        "dateRefField": "Data.dateOfComplaint"
      }
    ],
    "chartType": "metric",
    "valueType": "number",
    "action": "",
    "documentType": "_doc",
    "drillChart": "none",
    "aggregationPaths": [
      "Total Application"
    ],
    "insight": {
    },
    "_comment": " totalApplication is the chartId"
  },
```

Figure 6

Click here for an example configuration

Parameter Name	**Description**
Key (e.g: totalApplication)	This is the Visualization Code. This key will be referred in further visualization configurations. This is the key which will be used by client application to indicate which visualization is needed for display.
chartName	The name of the Chart which has to be used as a label on the Dashboard. The name of the Chart will be a detailed name. In this configuration, the Name of the Chart will be the code of Localization which will be used by Client Side
queries	Some visualizations are derived from a specific data source. While some other are derived from different data sources and are combined together to get a meaningful representation. The queries of aggregation which are to be used to fetch out the right data in the right aggregated format are configured here.
queries.module	The module / domain level, on which the query should be applied on. Property Tax is PT, Trade License is TL. If the query is applied across all modules, the module has to be defined as COMMON
queries.indexName	The name of the index upon which the query has to be executed is configured here.
queries.aggrQuery	The aggregation query in itself is added here. Based on the Module and the Index name specified, this query is attached to the filter part of the complete search request and then executed against that index
queries.requestQueryMap	Client Request would carry certain fields which are to be filtered. The parameters specified in the Client Request are different from the parameters in each of these indexed documents. In order to map the parameters of the request to the parameters of the ElasticSearch Document, this mapping is maintained
queries.dateRefField	Each of these modules have separate indexes. And all of them have their own date fields. When there is a date filter applied against these visualizations, each of them has to apply it against their own date reference fields.

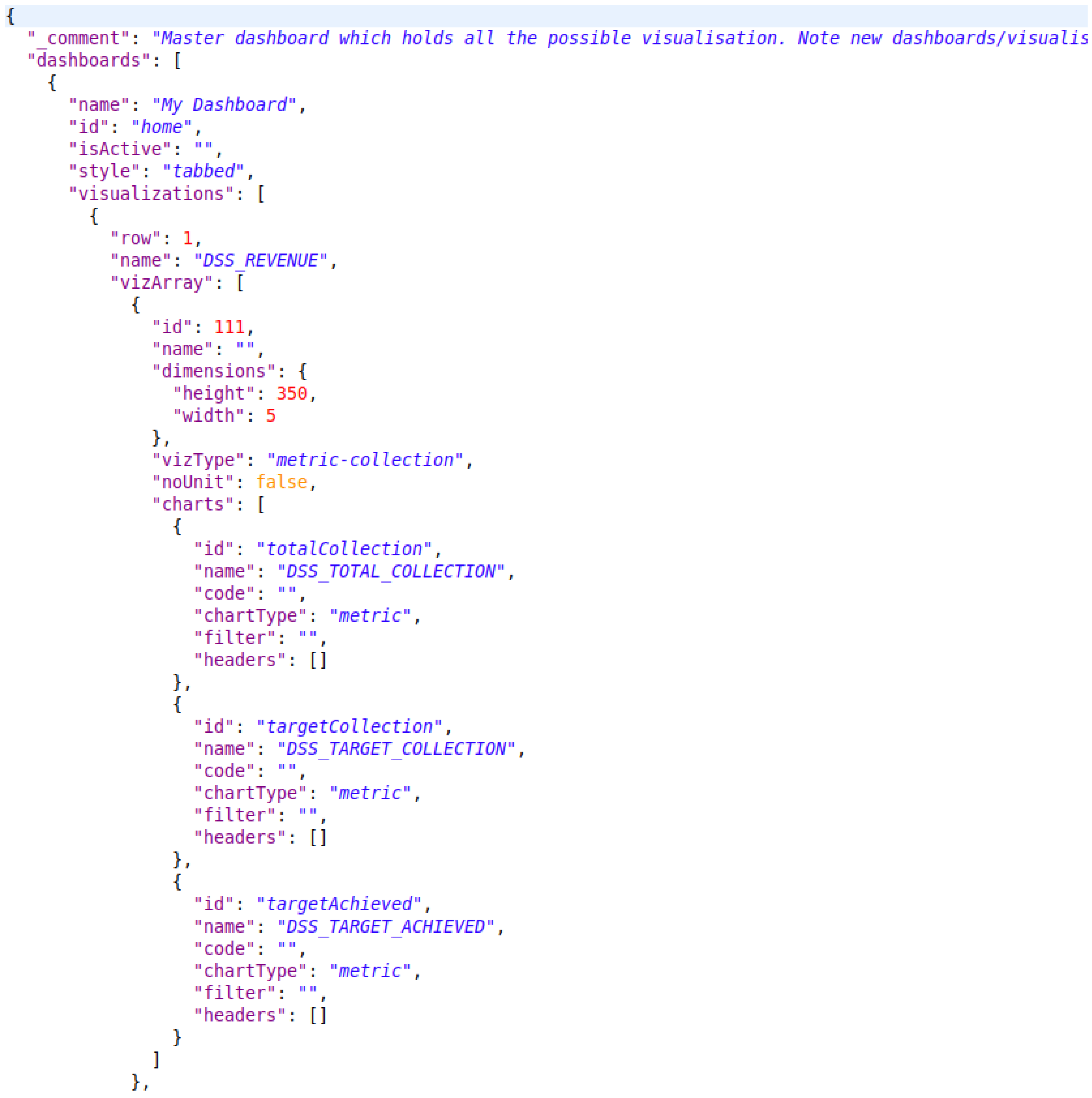
	In order to maintain what is the date field in which index, we have this configured in this configuration parameter
chartType	As there are different types of visualizations, this field defines as what is the type of chart / visualization that this data should be used to represent. **Chart types available are**: **metric** - this represents the aggregated amount/value for records filter by the aggregate es query **pie** - this represents the aggregated data on grouping. This is can be used to represent any line graph, bar graph, pie chart or donuts **line** - this graph/chart is data representation on date histograms or date groupings **perform -** this chart represents groping data as performance wise. **table** - represents a form of plots and value with headers as grouped on and list of its key, values pairs. **xtable -** represents a advanced feature of table, it has addition capabilities for dynamic adding header values.
valueType	In any case of data, the values which are sent to plot, might be a percentage, sometimes an amount and sometimes it is just a count. In order to represent them and differentiate the numbers from amount from percentage, this field is used to indicate the type of value that this Visualization will be sending.
action	Some of the visualizations are not just aggregation on data source. There might be some cases where we have to do a post aggregation computation. For Example, in the case of Top 3 Performing ULBs, the Target and Total Collection is obtained and then the percentage is calculated. In these kind of cases, what is the action that has to be performed on that data obtained is defined in this parameter.
documentType	The type of document upon which the query has to be executed is defined here.
drillChart	If there is a drill down on the visualization, then the code of the Drill

	Down Visualization is added here. This will be used by Client Service to manage drill downs
aggregationPaths	All the queries will be having Aggregation names in it. In order to fetch the value out of each Aggregation Responses, the name of the aggregation in the query will be an easy bet. These aggregation paths will have the names of Aggregation in it.
_comment	In order to display information on the “i” symbol of each visualization, Visualization Information is maintained in this field.

Master Dashboard Configuration

Master Dashboard Configuration is the main configuration which defines as which are the Dashboards which are to be painted on screen.

It includes all the Visualizations, their groups, the charts which come within them and even their dimensions as what should be their height and width.



```json
{
  "_comment": "Master dashboard which holds all the possible visualisation. Note new dashboards/visualis
  "dashboards": [
    {
      "name": "My Dashboard",
      "id": "home",
      "isActive": "",
      "style": "tabbed",
      "visualizations": [
        {
          "row": 1,
          "name": "DSS_REVENUE",
          "vizArray": [
            {
              "id": 111,
              "name": "",
              "dimensions": {
                "height": 350,
                "width": 5
              },
              "vizType": "metric-collection",
              "noUnit": false,
              "charts": [
                {
                  "id": "totalCollection",
                  "name": "DSS_TOTAL_COLLECTION",
                  "code": "",
                  "chartType": "metric",
                  "filter": "",
                  "headers": []
                },
                {
                  "id": "targetCollection",
                  "name": "DSS_TARGET_COLLECTION",
                  "code": "",
                  "chartType": "metric",
                  "filter": "",
                  "headers": []
                },
                {
                  "id": "targetAchieved",
                  "name": "DSS_TARGET_ACHIEVED",
                  "code": "",
                  "chartType": "metric",
                  "filter": "",
                  "headers": []
                }
              ]
            },
```

Figure 7

Click here for an example configuration

Parameter Name	**Description**
name	Name of the Dashboard which has to be displayed as Page Heading

id	Unique Identifier of the Dashboard which should be used later for Querying each of these Visualizations
isActive	Active Indicator which can be used to quickly disable a dashboard if required.
style	Style of the Dashboard. Whether it should be a linear one or tabbed one. This information is maintained in this parameter.
visualizations	The list of visualizations which are to be displayed in the Dashboard are listed out here.
visualizations.row	The row identifier for each Visualization are mentioned here
visualizations.name	The name of an individual visualization is added here
visualizations.vizArray	The list of Charts within the Visualization is specified in this list.
visualizations.vizArray.id	Group of Charts is given an ID to have a placement on the Dashboard. This unique identifier is maintained in this field.
visualizations.vizArray.name	Group of Charts is given a name which can be displayed on the group on Dashboard in that row.
visualizations.vizArray.dimensions	Each of these group of charts are given a dimension based on which they are placed in a specific row in a dashboard
visualizations.vizArray.vizType	As there are multiple charts grouped into one visualization, the type of Visualization needs to be specified in order to indicate to the client application as what goes inside each of these visualizations and charts inside them vizType used for any other dashboards:- metric-collection, chart, performing-metric metric-collection:- Used to specify the type as single or group of metric chart type

	2. performing-metric:- Used perform chart type 3. chart:- Used chart type for pie, donut, table, bar, horizantalbar, line vizType used for Home page:- collection, module collection: used in UI style as full width 2. module: used in UI style for specific width.
visualizations.vizArray.noUnit visualizations.vizArray.isCollapsible visualizations.vizArray.ref	The value types of these charts are different. Some are numbers, some are amounts, some are percentage. In the case of amounts, there is a requirement as to display in Lakhs, Crores and Units. In order to indicate the client application as whether to display these units or not, we have this boolean to control that The value type is for card/visualisation collapsible as boolean values. This object contains url (as mandatory), logoUrl (optional), type(optional).
visualizations.vizArray.charts	The list of individual charts inside a Visualization Group is maintained in this array list
visualizations.vizArray.charts.id	Individual Chart Number Identifier to indicate the uniqueness of Charts
visualizations.vizArray.charts.name	Name of the Chart which can be a header label for Charts within a Visualization
visualizations.vizArray.charts.code	Code of the Chart which is the indicator which has to be sent to Server Side to get the data for representing the Visualization.

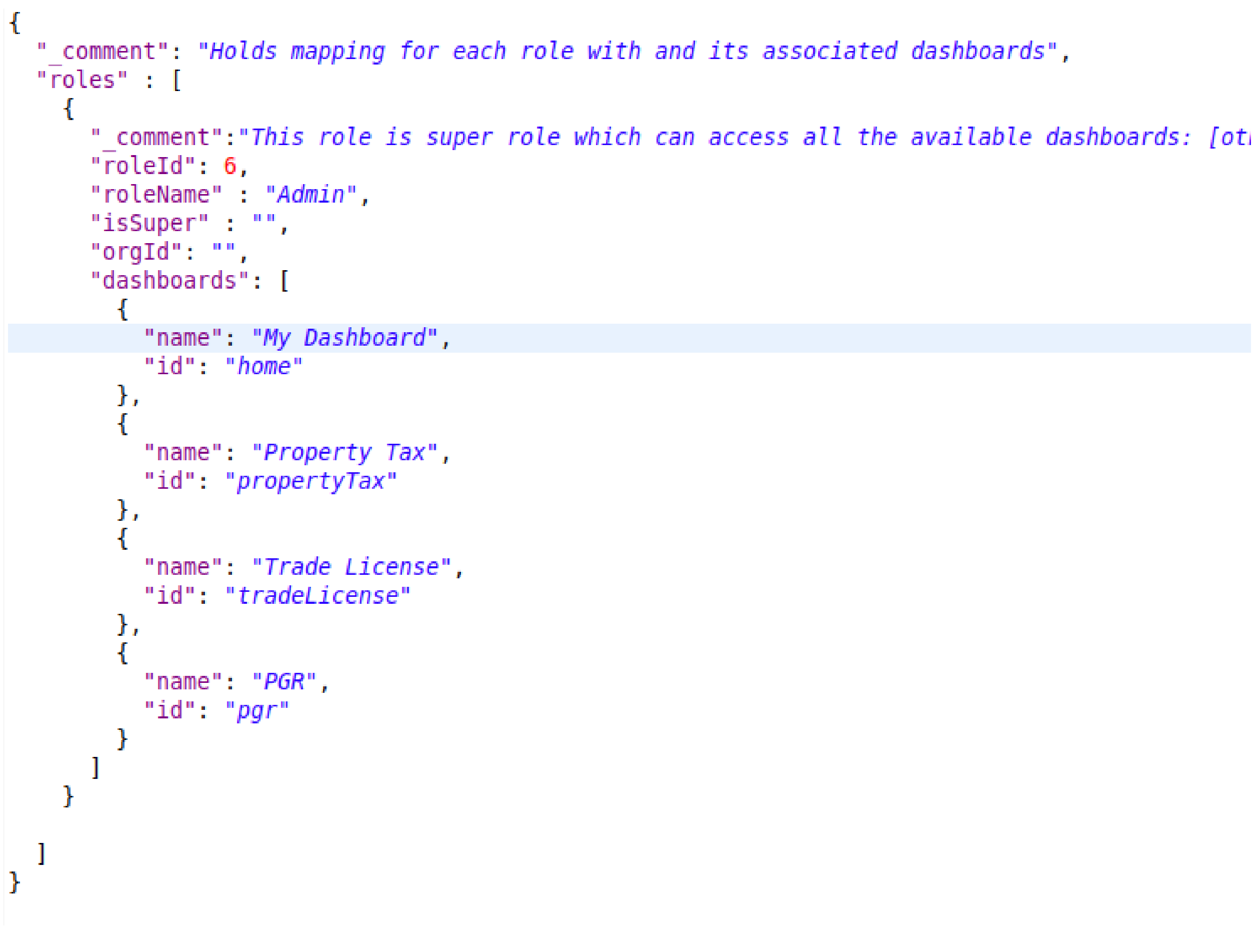
visualizations.vizArray.charts.chartType	Type of Chart which has to represent the data result set which is obtained is specified here chartType:- bar, horizontalBar, line, donut, pie, metric, table
visualizations.vizArray.charts.filters	Filters which can be applied on the Visualization and what are the fields which are filterable are mentioned here.
visualizations.vizArray.charts.headers	In some cases, there are headers which can be a title or an additional information for the Chart Data which gets represented. This field is kept open to accommodate those information which can be sent along with the Chart Data in itself.

Role Dashboard Mappings Configuration

Master Dashboard Configuration which was explained earlier hold the list of Dashboards which are available.
Given the instance where Role Action Mapping is not maintained in the Application Service, this configuration will act as Role - Dashboard Mapping Configuration

In this, each Role is mapped against the Dashboard which they are authorized to see

This was used earlier when the Role Action Mapping of eGov was not integrated.
Later, when the Role Action Mapping started controlling the Dashboards to be seen on the client side, this configuration is just used to enable the Dashboards for viewing.



```
{
  "_comment": "Holds mapping for each role with and its associated dashboards",
  "roles" : [
    {
      "_comment":"This role is super role which can access all the available dashboards: [ot
      "roleId": 6,
      "roleName" : "Admin",
      "isSuper" : "",
      "orgId": "",
      "dashboards": [
        {
          "name": "My Dashboard",
          "id": "home"
        },
        {
          "name": "Property Tax",
          "id": "propertyTax"
        },
        {
          "name": "Trade License",
          "id": "tradeLicense"
        },
        {
          "name": "PGR",
          "id": "pgr"
        }
      ]
    }

  ]
}
```

Figure 8

Click here for an example configuration

Parameter Name	**Description**
roles	List of Roles which are available in the system
roles._comment	Role Description and a comment on why does this role has an entry in this configuration and sums up the summary as what are the things which are to be enabled.
roles.roleId	Unique Identifier of the Role for which Access is being given
roles.roleName	Name of the Role for which the access is being given
roles.isSuper	Boolean flag which defines as whether the Role is a Super User or not

roles.orgId	Organization to which the Role belongs to
roles.dashboards	List of Dashboards that are enabled for the Role
roles.dashboards.name	Name of the individual Dashboard which has been enabled
roles.dashboards.id	Identifier of the individual Dashboard which has been enabled

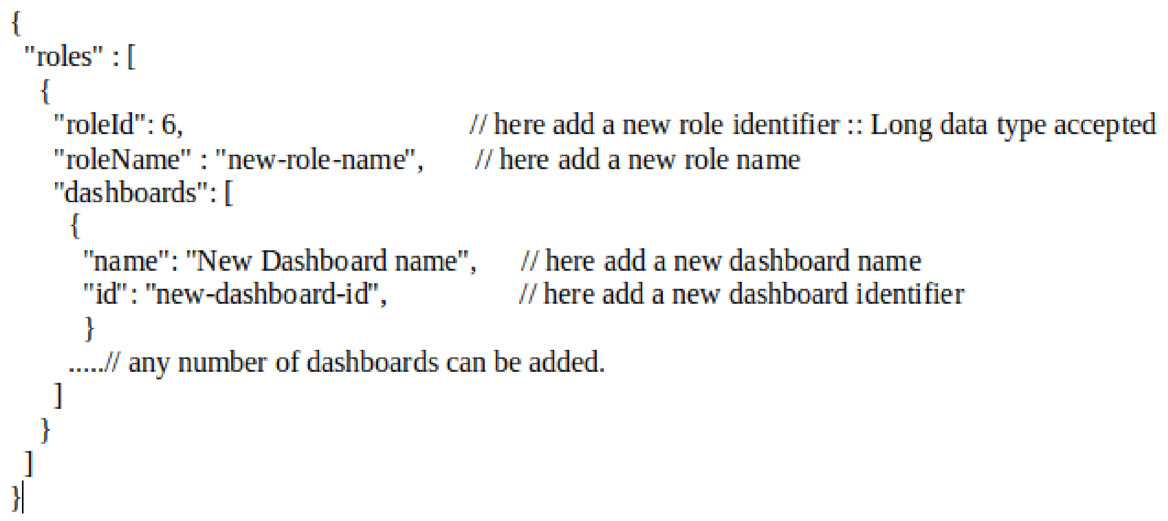
Adding Configurations

Adding Roles and Dashboards:

To add new role, RoleDashboardMappingsConf.json (**roles** node) configuration file has to be modified as below

Note: Any number of roles & dashboards can be added

Below as in Figure 9 is a sample to add new role object, new dashboard object

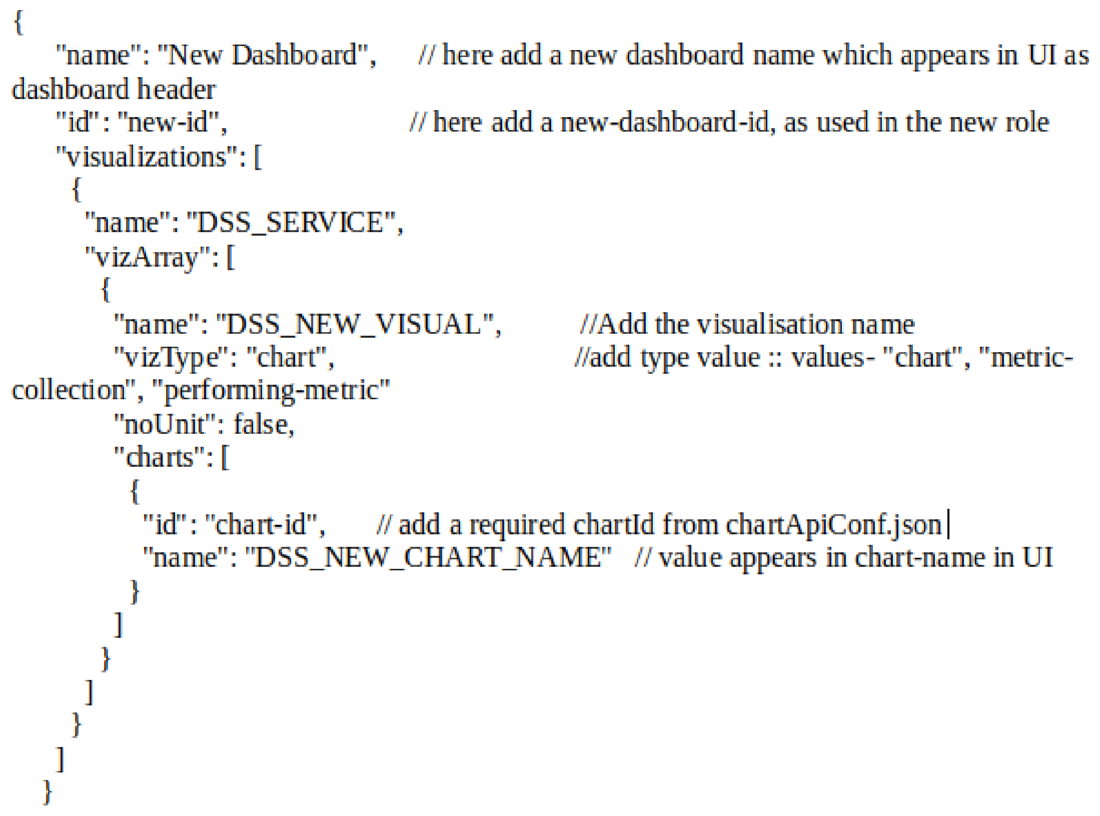


```
{
 "roles" : [
   {
    "roleId": 6,                          // here add a new role identifier :: Long data type accepted
    "roleName" : "new-role-name",      // here add a new role name
    "dashboards": [
      {
       "name": "New Dashboard name",      // here add a new dashboard name
       "id": "new-dashboard-id",           // here add a new dashboard identifier
       }
     .....// any number of dashboards can be added.
    ]
   }
  ]
}
```

Figure 9

To add new dashboard, MasterDashboardConfig.json (**dashboards** node) has to be modified as below in Figure 10.

Note: dashboards array add a new dashboard as given below



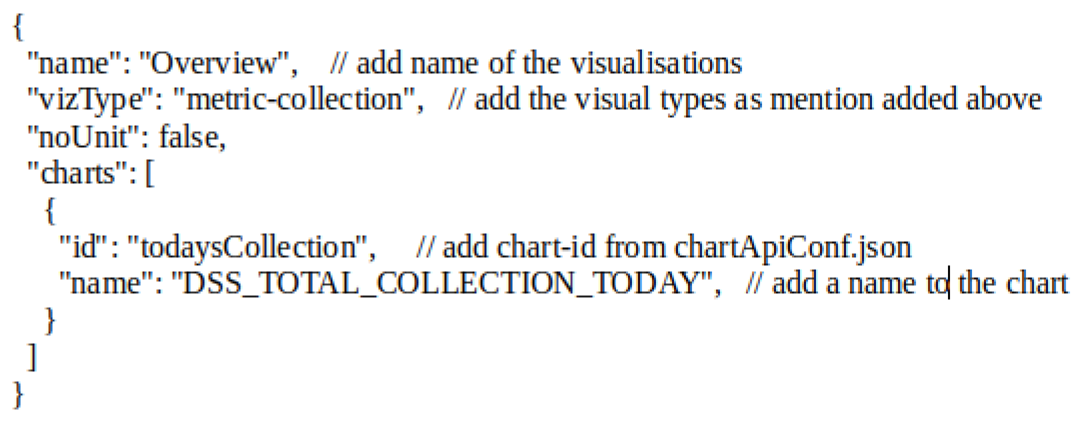
```
{
    "name": "New Dashboard",     // here add a new dashboard name which appears in UI as
dashboard header
    "id": "new-id",                    // here add a new-dashboard-id, as used in the new role
    "visualizations": [
      {
        "name": "DSS_SERVICE",
        "vizArray": [
          {
            "name": "DSS_NEW_VISUAL",          //Add the visualisation name
            "vizType": "chart",                          //add type value :: values- "chart", "metric-
collection", "performing-metric"
            "noUnit": false,
            "charts": [
              {
                "id": "chart-id",      // add a required chartId from chartApiConf.json
                "name": "DSS_NEW_CHART_NAME"   // value appears in chart-name in UI
              }
            ]
          }
        ]
      }
    ]
  }
```

Figure 10

Adding Visualizations in existing Dashboard

To add new visualisations, again MasterDashboardConfig.json (**vizArray** node) has to be modified as below as shown in Figure 11.

Note: vizArray is to hold multiple visualizations



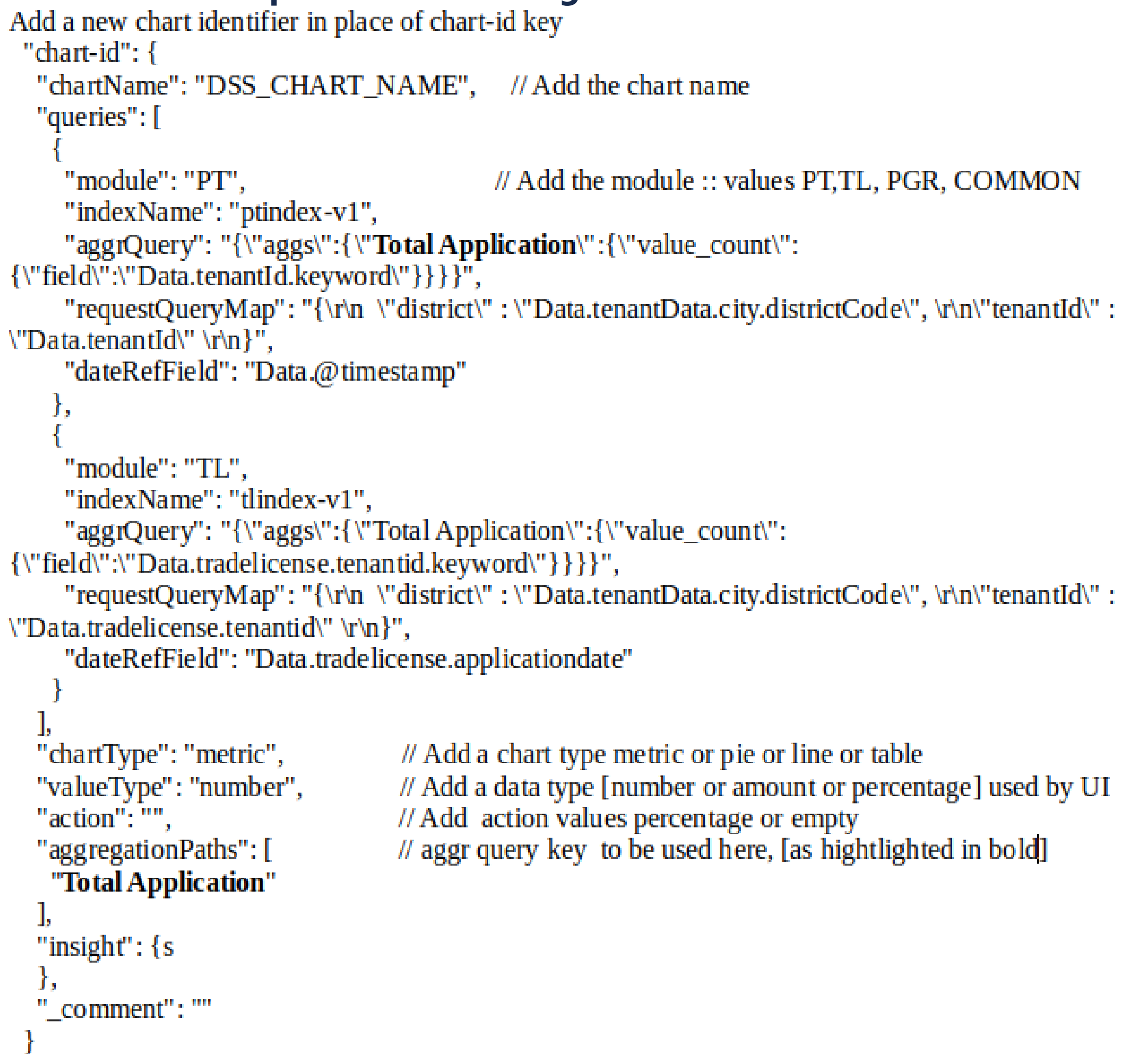
```
{
  "name": "Overview",    // add name of the visualisations
  "vizType": "metric-collection",   // add the visual types as mention added above
  "noUnit": false,
  "charts": [
    {
      "id": "todaysCollection",     // add chart-id from chartApiConf.json
      "name": "DSS_TOTAL_COLLECTION_TODAY",   // add a name to the chart
    }
  ]
}
```

Figure 11

Adding charts for visualizations

To add new chart, chartApiConf.json has to be modified as shown below. A new chartid has to be added with chart node object.

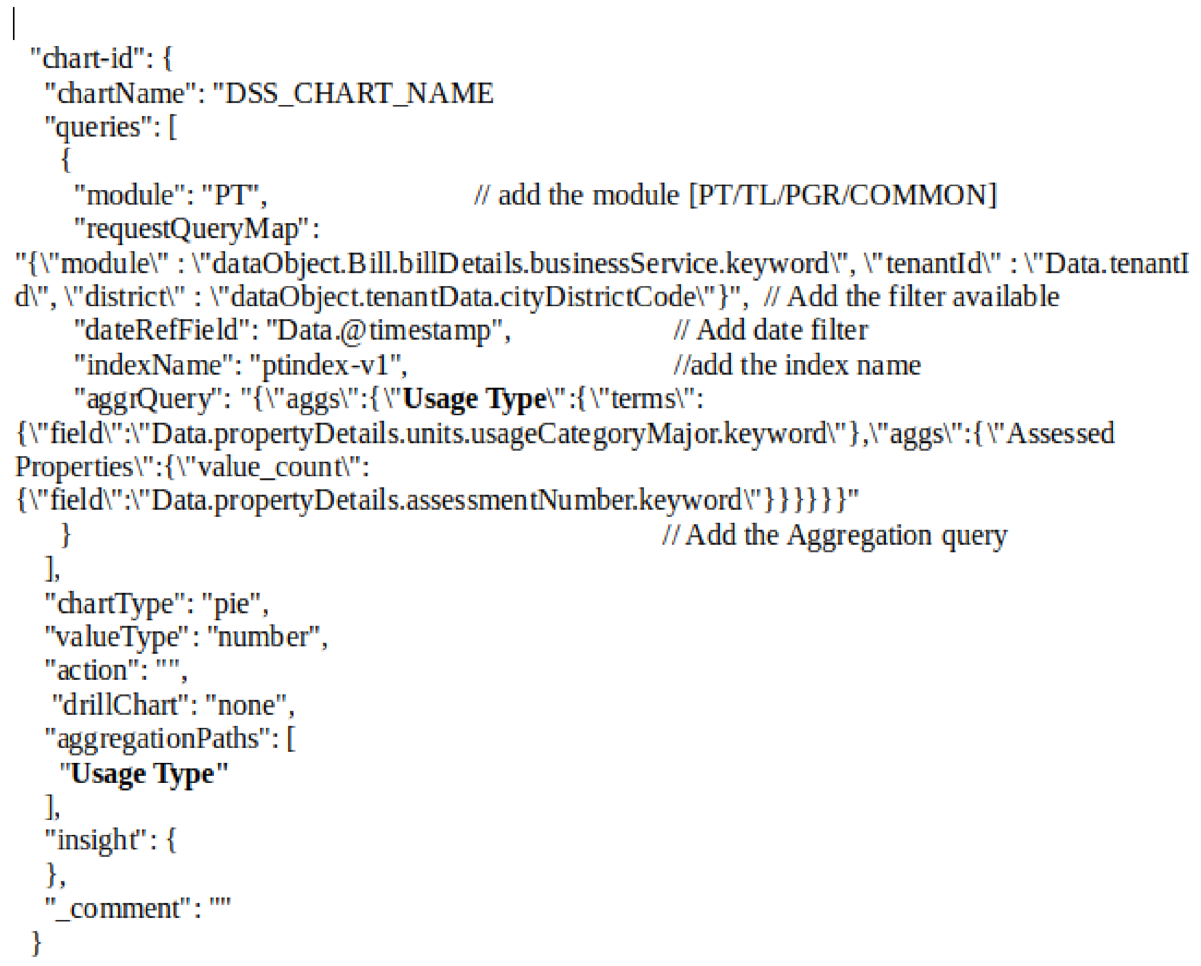
Metric chart Sample as shown in Figure 12



```
Add a new chart identifier in place of chart-id key
  "chart-id": {
    "chartName": "DSS_CHART_NAME",     // Add the chart name
    "queries": [
      {
        "module": "PT",                                // Add the module :: values PT,TL, PGR, COMMON
        "indexName": "ptindex-v1",
        "aggrQuery": "{\"aggs\":{\"Total Application\":{\"value_count\":
{\"field\":\"Data.tenantId.keyword\"}}}}",
        "requestQueryMap": "{\r\n  \"district\" : \"Data.tenantData.city.districtCode\", \r\n\"tenantId\" :
\"Data.tenantId\" \r\n}",
        "dateRefField": "Data.@timestamp"
      },
      {
        "module": "TL",
        "indexName": "tlindex-v1",
        "aggrQuery": "{\"aggs\":{\"Total Application\":{\"value_count\":
{\"field\":\"Data.tradelicense.tenantid.keyword\"}}}}",
        "requestQueryMap": "{\r\n  \"district\" : \"Data.tenantData.city.districtCode\", \r\n\"tenantId\" :
\"Data.tradelicense.tenantid\" \r\n}",
        "dateRefField": "Data.tradelicense.applicationdate"
      }
    ],
    "chartType": "metric",              // Add a chart type metric or pie or line or table
    "valueType": "number",              // Add a data type [number or amount or percentage] used by UI
    "action": "",                       // Add  action values percentage or empty
    "aggregationPaths": [               // aggr query key  to be used here, [as hightlighted in bold]
      "Total Application"
    ],
    "insight": {s
    },
    "_comment": ""
  }
```

Figure 12

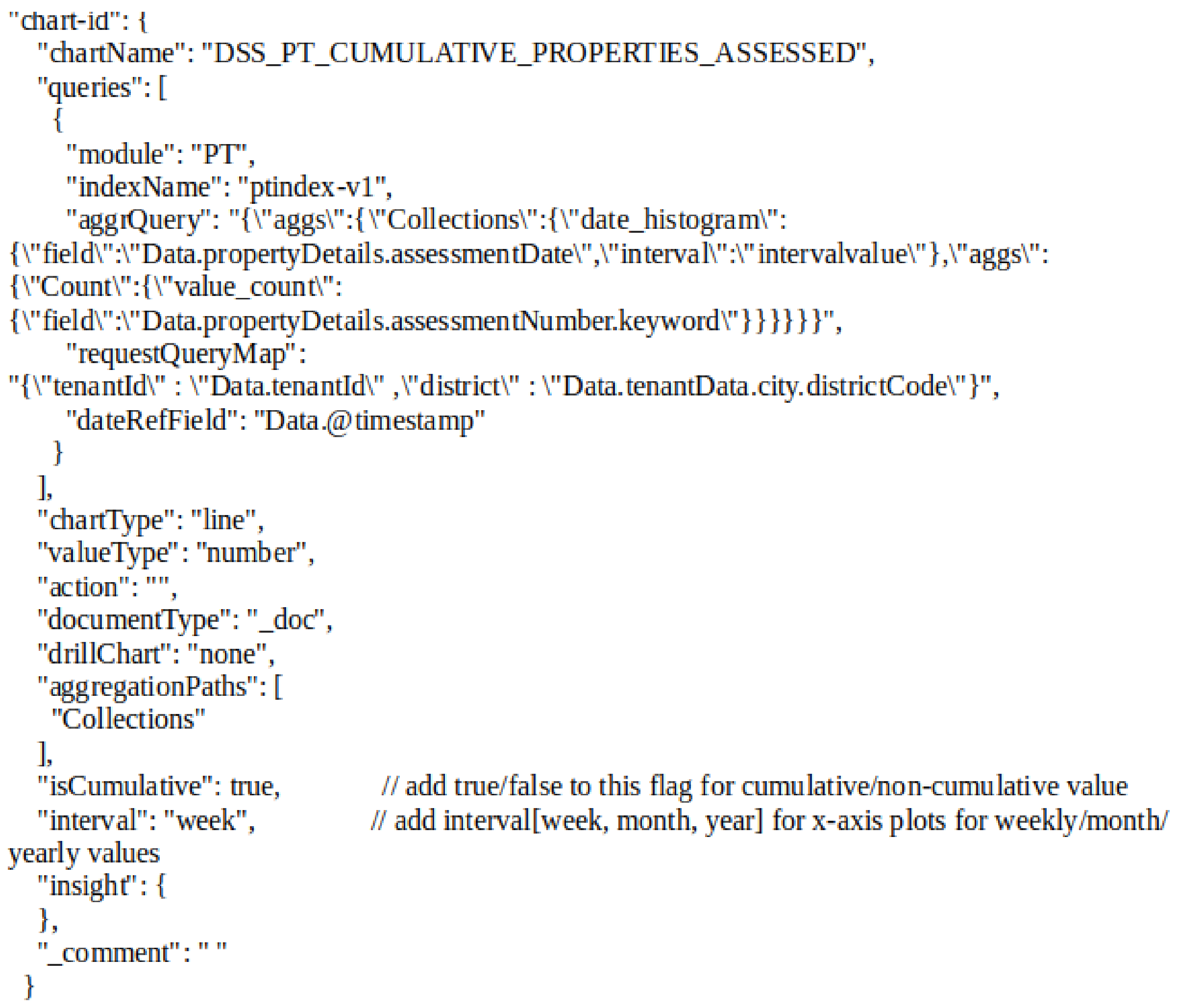
Pie chart Sample as shown in Figure 13



```
  "chart-id": {
    "chartName": "DSS_CHART_NAME
    "queries": [
      {
        "module": "PT",                          // add the module [PT/TL/PGR/COMMON]
        "requestQueryMap":
"{\"module\" : \"dataObject.Bill.billDetails.businessService.keyword\", \"tenantId\" : \"Data.tenantI
d\", \"district\" : \"dataObject.tenantData.cityDistrictCode\"}",  // Add the filter available
        "dateRefField": "Data.@timestamp",                    // Add date filter
        "indexName": "ptindex-v1",                            //add the index name
        "aggrQuery": "{\"aggs\":{\"Usage Type\":{\"terms\":
{\"field\":\"Data.propertyDetails.units.usageCategoryMajor.keyword\"},\"aggs\":{\"Assessed
Properties\":{\"value_count\":
{\"field\":\"Data.propertyDetails.assessmentNumber.keyword\"}}}}}}"
      }                                                        // Add the Aggregation query
    ],
    "chartType": "pie",
    "valueType": "number",
    "action": "",
     "drillChart": "none",
    "aggregationPaths": [
      "Usage Type"
    ],
    "insight": {
    },
    "_comment": ""
  }
```

Figure 13

Line chart Sample as shown in Figure 14

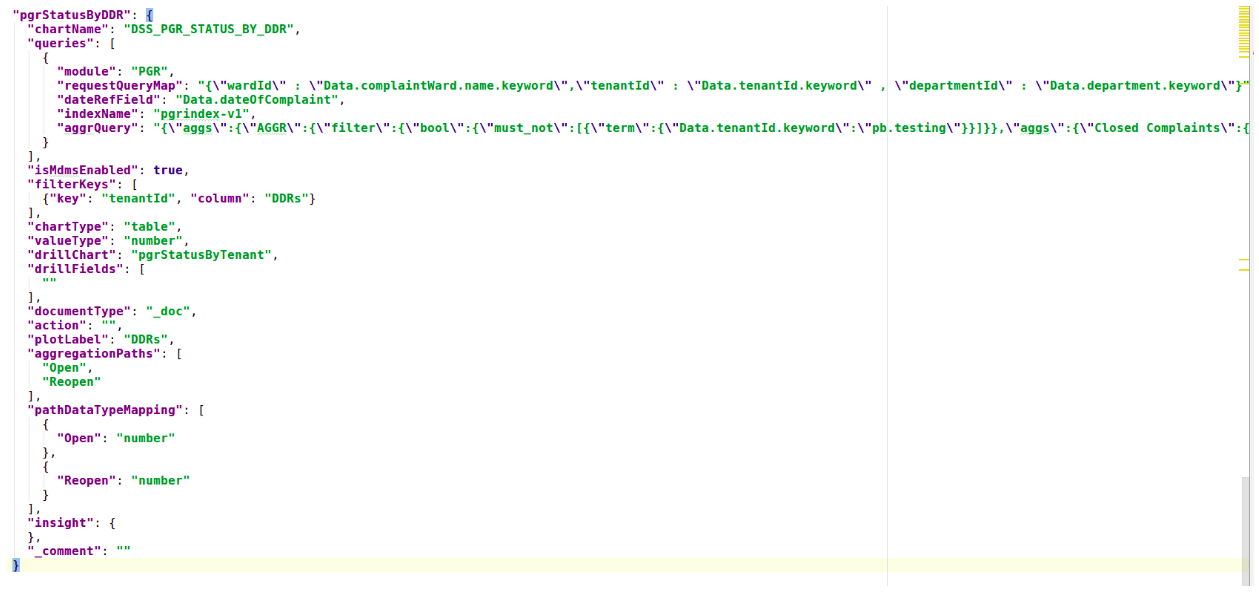


```
"chart-id": {
   "chartName": "DSS_PT_CUMULATIVE_PROPERTIES_ASSESSED",
   "queries": [
     {
       "module": "PT",
       "indexName": "ptindex-v1",
       "aggrQuery": "{\"aggs\":{\"Collections\":{\"date_histogram\":
{\"field\":\"Data.propertyDetails.assessmentDate\",\"interval\":\"intervalvalue\"},\"aggs\":
{\"Count\":{\"value_count\":
{\"field\":\"Data.propertyDetails.assessmentNumber.keyword\"}}}}}}",
       "requestQueryMap":
"{\"tenantId\" : \"Data.tenantId\" ,\"district\" : \"Data.tenantData.city.districtCode\"}",
       "dateRefField": "Data.@timestamp"
     }
   ],
   "chartType": "line",
   "valueType": "number",
   "action": "",
   "documentType": "_doc",
   "drillChart": "none",
   "aggregationPaths": [
     "Collections"
   ],
   "isCumulative": true,           // add true/false to this flag for cumulative/non-cumulative value
   "interval": "week",             // add interval[week, month, year] for x-axis plots for weekly/month/
yearly values
   "insight": {
   },
   "_comment": " "
 }
```

Figure 14

Table chart Sample: This chart comes with 2 kind, table and xtable.

table (as shown in Figure 15.) type allows to added aggregated fields added as available in the query keys, hence to extract the values based on the key, **aggegationPaths** needs to add along with their data type as in **pathDataTypeMapping**.

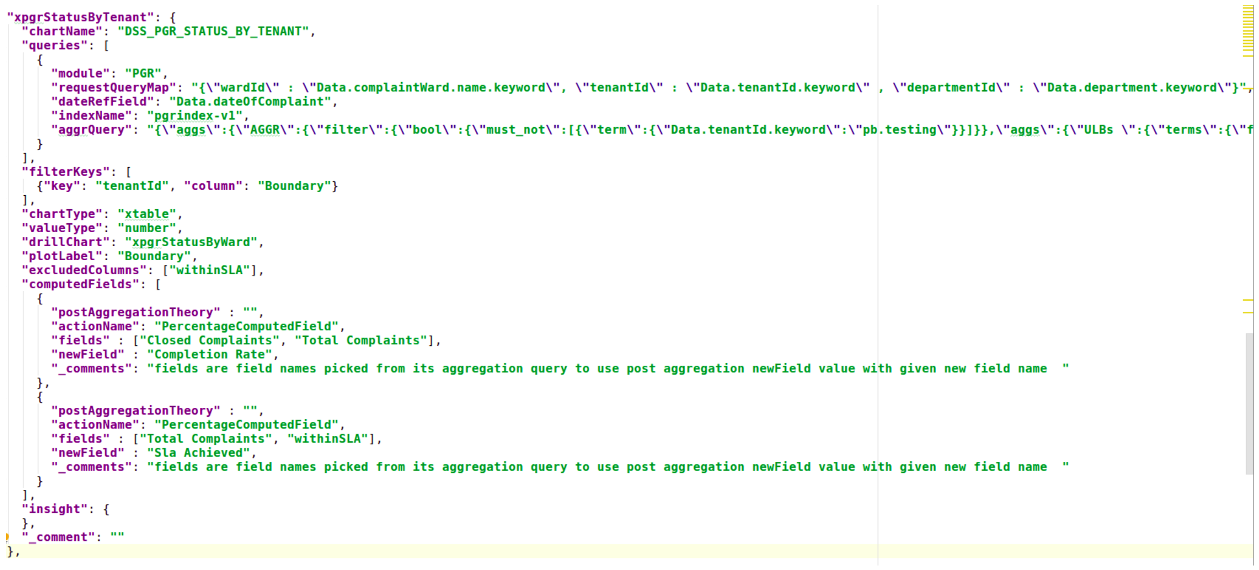


```
"pgrStatusByDDR": {
  "chartName": "DSS_PGR_STATUS_BY_DDR",
  "queries": [
    {
      "module": "PGR",
      "requestQueryMap": "{\"wardId\" : \"Data.complaintWard.name.keyword\",\"tenantId\" : \"Data.tenantId.keyword\" , \"departmentId\" : \"Data.department.keyword\"}"
      "dateRefField": "Data.dateOfComplaint",
      "indexName": "pgrindex-v1",
      "aggrQuery": "{\"aggs\":{\"AGGR\":{\"filter\":{\"bool\":{\"must_not\":[{\"term\":{\"Data.tenantId.keyword\":\"pb.testing\"}}]}},\"aggs\":{\"Closed Complaints\":{
    }
  ],
  "isMdmsEnabled": true,
  "filterKeys": [
    {"key": "tenantId", "column": "DDRs"}
  ],
  "chartType": "table",
  "valueType": "number",
  "drillChart": "pgrStatusByTenant",
  "drillFields": [
    ""
  ],
  "documentType": "_doc",
  "action": "",
  "plotLabel": "DDRs",
  "aggregationPaths": [
    "Open",
    "Reopen"
  ],
  "pathDataTypeMapping": [
    {
      "Open": "number"
    },
    {
      "Reopen": "number"
    }
  ],
  "insight": {
  },
  "_comment": ""
}
```

Figure 15

xtable(as shown in Figure 16.) type allows to add multiple computed fields with the aggregated fields dynamically added.

To add multiple computed columns, **computedFields** [] where actionName (IComputedField<T> interface), fields [] names as in exist in query key, newField as name to appear for computation must be defined.



```
"xpgrStatusByTenant": {
  "chartName": "DSS_PGR_STATUS_BY_TENANT",
  "queries": [
    {
      "module": "PGR",
      "requestQueryMap": "{\"wardId\" : \"Data.complaintWard.name.keyword\", \"tenantId\" : \"Data.tenantId.keyword\" , \"departmentId\" : \"Data.department.keyword\"}",
      "dateRefField": "Data.dateOfComplaint",
      "indexName": "pgrindex-v1",
      "aggrQuery": "{\"aggs\":{\"AGGR\":{\"filter\":{\"bool\":{\"must_not\":[{\"term\":{\"Data.tenantId.keyword\":\"pb.testing\"}}]}},\"aggs\":{\"ULBs \":{\"terms\":{\"f
    }
  ],
  "filterKeys": [
    {"key": "tenantId", "column": "Boundary"}
  ],
  "chartType": "xtable",
  "valueType": "number",
  "drillChart": "xpgrStatusByWard",
  "plotLabel": "Boundary",
  "excludedColumns": ["withinSLA"],
  "computedFields": [
    {
      "postAggregationTheory" : "",
      "actionName": "PercentageComputedField",
      "fields" : ["Closed Complaints", "Total Complaints"],
      "newField" : "Completion Rate",
      "_comments": "fields are field names picked from its aggregation query to use post aggregation newField value with given new field name  "
    },
    {
      "postAggregationTheory" : "",
      "actionName": "PercentageComputedField",
      "fields" : ["Total Complaints", "withinSLA"],
      "newField" : "Sla Achieved",
      "_comments": "fields are field names picked from its aggregation query to use post aggregation newField value with given new field name  "
    }
  ],
  "insight": {
  },
  "_comment": ""
},
```

Figure 16

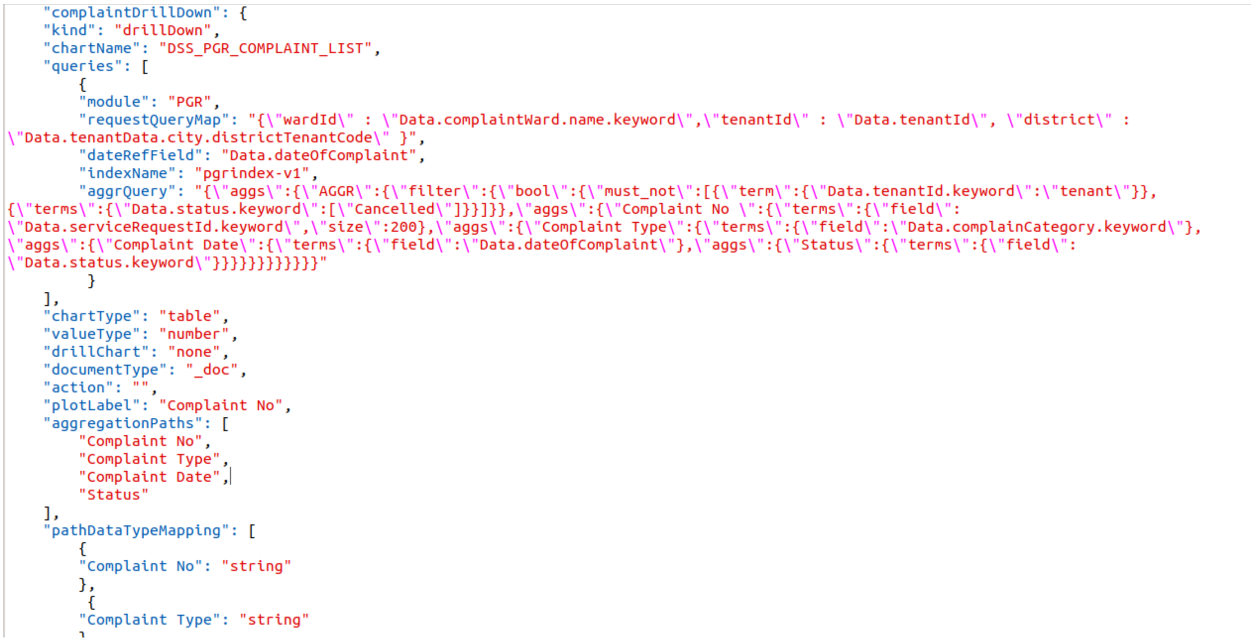
https://github.com/egovernments/configs/blob/master/egov-dss-dashboards/dashboard-analytics/ChartApiConfig.json - Click on this link for the detailed configuration.

Steps to create charts and visualizations are:

1. Create/Add a chart in chartApiConf.json
2. Add a visualization for existing dashboard in MasterDashboardConfig.json as defined above.
3. Or in order to create/add a new dashboard create dashboard in MasterDashboardConfig.json and create a role in RoleDashboardConfig.json

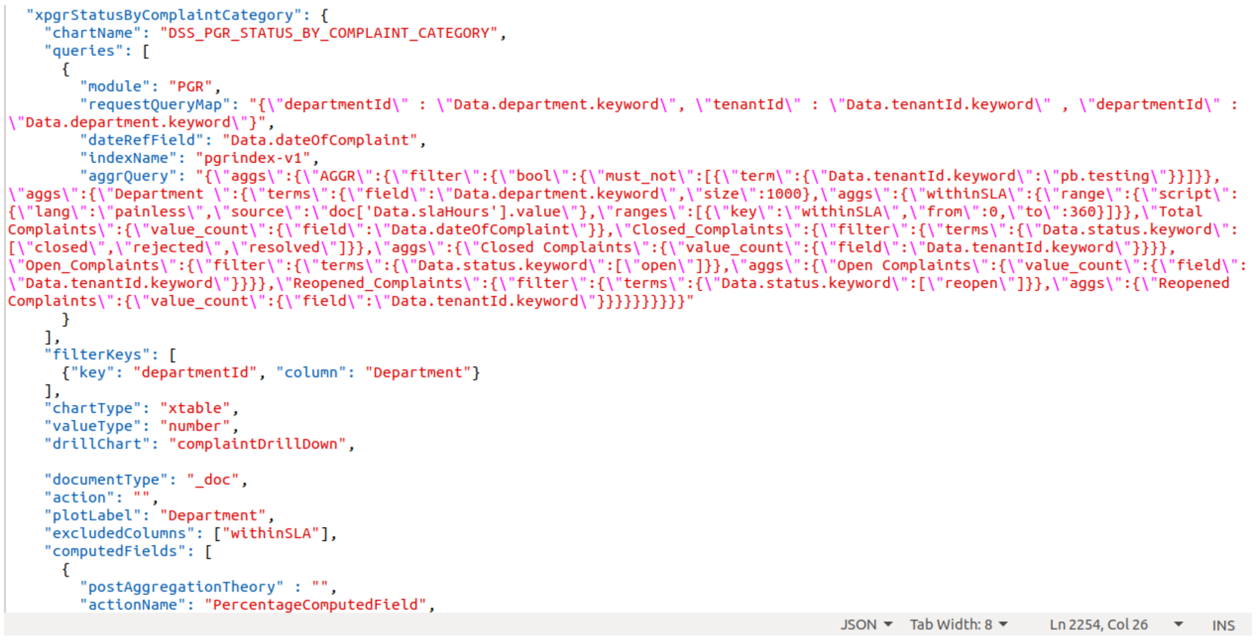
Configuration Changes for DrillThrough

1. Example Drill through in Ward table in Property Dashboard. wardDrillDown is the visualization code for PT Drill Down. 'Kind' is the attribute that shows the type of visualization code. Apart from two things all the attributes are common.



```
    "complaintDrillDown": {
    "kind": "drillDown",
    "chartName": "DSS_PGR_COMPLAINT_LIST",
    "queries": [
        {
        "module": "PGR",
        "requestQueryMap": "{\"wardId\" : \"Data.complaintWard.name.keyword\",\"tenantId\" : \"Data.tenantId\", \"district\" :
\"Data.tenantData.city.districtTenantCode\" }",
        "dateRefField": "Data.dateOfComplaint",
        "indexName": "pgrindex-v1",
        "aggrQuery": "{\"aggs\":{\"AGGR\":{\"filter\":{\"bool\":{\"must_not\":[{\"term\":{\"Data.tenantId.keyword\":\"tenant\"}},
{\"terms\":{\"Data.status.keyword\":[\"Cancelled\"]}}]}},\"aggs\":{\"Complaint No \":{\"terms\":{\"field\":
\"Data.serviceRequestId.keyword\",\"size\":200},\"aggs\":{\"Complaint Type\":{\"terms\":{\"field\":\"Data.complainCategory.keyword\"},
\"aggs\":{\"Complaint Date\":{\"terms\":{\"field\":\"Data.dateOfComplaint\"},\"aggs\":{\"Status\":{\"terms\":{\"field\":
\"Data.status.keyword\"}}}}}}}}}}}"
         }
    ],
    "chartType": "table",
    "valueType": "number",
    "drillChart": "none",
    "documentType": "_doc",
    "action": "",
    "plotLabel": "Complaint No",
    "aggregationPaths": [
        "Complaint No",
        "Complaint Type",
        "Complaint Date",
        "Status"
    ],
    "pathDataTypeMapping": [
        {
        "Complaint No": "string"
        },
         {
        "Complaint Type": "string"
```

2. Example Drill through in ComplaintList table in PGR Dashboard. complaintDrillDown is the visualization code for PGR Drill Down.



```
  "xpgrStatusByComplaintCategory": {
    "chartName": "DSS_PGR_STATUS_BY_COMPLAINT_CATEGORY",
    "queries": [
      {
        "module": "PGR",
        "requestQueryMap": "{\"departmentId\" : \"Data.department.keyword\", \"tenantId\" : \"Data.tenantId.keyword\" , \"departmentId\" :
\"Data.department.keyword\"}",
        "dateRefField": "Data.dateOfComplaint",
        "indexName": "pgrindex-v1",
        "aggrQuery": "{\"aggs\":{\"AGGR\":{\"filter\":{\"bool\":{\"must_not\":[{\"term\":{\"Data.tenantId.keyword\":\"pb.testing\"}}]}},
\"aggs\":{\"Department \":{\"terms\":{\"field\":\"Data.department.keyword\",\"size\":1000},\"aggs\":{\"withinSLA\":{\"range\":{\"script\":
{\"lang\":\"painless\",\"source\":\"doc['Data.slaHours'].value\"},\"ranges\":[{\"key\":\"withinSLA\",\"from\":0,\"to\":360}]}},\"Total
Complaints\":{\"value_count\":{\"field\":\"Data.dateOfComplaint\"}},\"Closed_Complaints\":{\"filter\":{\"terms\":{\"Data.status.keyword\":
[\"closed\",\"rejected\",\"resolved\"]}},\"aggs\":{\"Closed Complaints\":{\"value_count\":{\"field\":\"Data.tenantId.keyword\"}}}},
\"Open_Complaints\":{\"filter\":{\"terms\":{\"Data.status.keyword\":[\"open\"]}},\"aggs\":{\"Open Complaints\":{\"value_count\":{\"field\":
\"Data.tenantId.keyword\"}}}},\"Reopened_Complaints\":{\"filter\":{\"terms\":{\"Data.status.keyword\":[\"reopen\"]}},\"aggs\":{\"Reopened
Complaints\":{\"value_count\":{\"field\":\"Data.tenantId.keyword\"}}}}}}}}}}"
      }
    ],
    "filterKeys": [
      {"key": "departmentId", "column": "Department"}
    ],
    "chartType": "xtable",
    "valueType": "number",
    "drillChart": "complaintDrillDown",

    "documentType": "_doc",
    "action": "",
    "plotLabel": "Department",
    "excludedColumns": ["withinSLA"],
    "computedFields": [
      {
        "postAggregationTheory" : "",
        "actionName": "PercentageComputedField",
```

The above complaintDrillDown visualization code called in the drill chart parameter.