LTI

Let's Solve

# Whitepaper

## Reporting Framework using Micro-services

Author: Manish Ranjan

A Larsen & Toubro Group Company

# Introduction

Data extraction and its representation in the form of reports (MS Excel, PDF and data files such as .csv, .psf, .txt, fixed length etc) has been an essential part of systems. Organizations are paying heavy license fees for reporting tools such as Business Object, Actuate, Cognos, Jasper etc. We observed that in most of the requirements clients need reports in simple formats with little or no modifications, but are still paying a lot for maintaining expensive reporting tools. Java-based HTML reports were an option from the very beginning but developing such frameworks in Java is still a very expensive process and maintenance is another overhead.

In the last few years, Java has bounced back with its new avatar – Micro-services. Micro-services though can be written in Java or in any other tool that can be packaged together as a service and can be deployed independently. This gives a unique advantage over traditional ways of developing software and products.

Let's revisit our age old reporting application but with a new Micro-services based perspective.

## How Micro-services can help us resolve our problem?

There is no one-to-one mapping between what a typical reporting framework needs and what the Micro-service world can match. We have to look at this from a service-based approach perspective. Let us list features of service-based architecture that can help us define a reporting framework:

**i. Evolution is a natural phenomenon:** Service-based architecture supports evolution of services over time. There is no need to develop the complete solution in one go, which usually is an expensive and impractical approach. Micro-services give us the flexibility to develop multiple versions of the same service: old, new and transition state all with mutual coexistence. This gives us time to evolve and migrate as and when users/clients are ready.

**ii. Small is good:** Package one functionality and just focus on that. In service-based approach we can package common services into one executable that can be launched as a service. For example, a query service that takes list of columns as input and returns data as output in JSON format. We can create one service per business object to create a DAO layer that is accessible over HTTP though APIs. Imagine how far we can go in using such services for our master data management, data extraction, and central data repository related requirements.

**iii. Adaptability is the real intelligence:** Service-based approach is equipped with a very strong orchestration layer that manages sequence of call of services. With dynamic or rule based orchestration uncountable permutations can be formed that gives the flexibility to adapt to new requirements without affecting existing ones. For example, we can have a service like validation (business validation, functional validation, data validation). Within this data validation we can have different type of validations such as XML and CSV,

with different versions or types of validations under XML variation. All these can co-exist and work simultaneously without affecting each other as the essence of micro-services is "Independence" and "Adaptability".

**iv. Demand-based scaling:** Demand driven scale up and down is the new normal. We monitor the performance of and load on each service, and then based on that decide the instances that would be required to provide the best possible performance. Since this is all automated, we are revisiting the possibilities Micro-service offers to provide solutions to long-standing issues. Isn't that exciting!

## Reporting Framework – A Reverse Engineering Approach

The industry has progressed so much so that we do not need to talk about different reporting and analytical platforms. We can start with basic tabular report to highly complex advanced analytics based report but at the core it is "data" that plays a very important role. Let's see how we can re-engineer the existing reporting framework with new knowledge of Micro-services.

Basic format of reporting means reports that are tabular in nature and exported in PDF, Excel, HTML and other data-extraction formats.

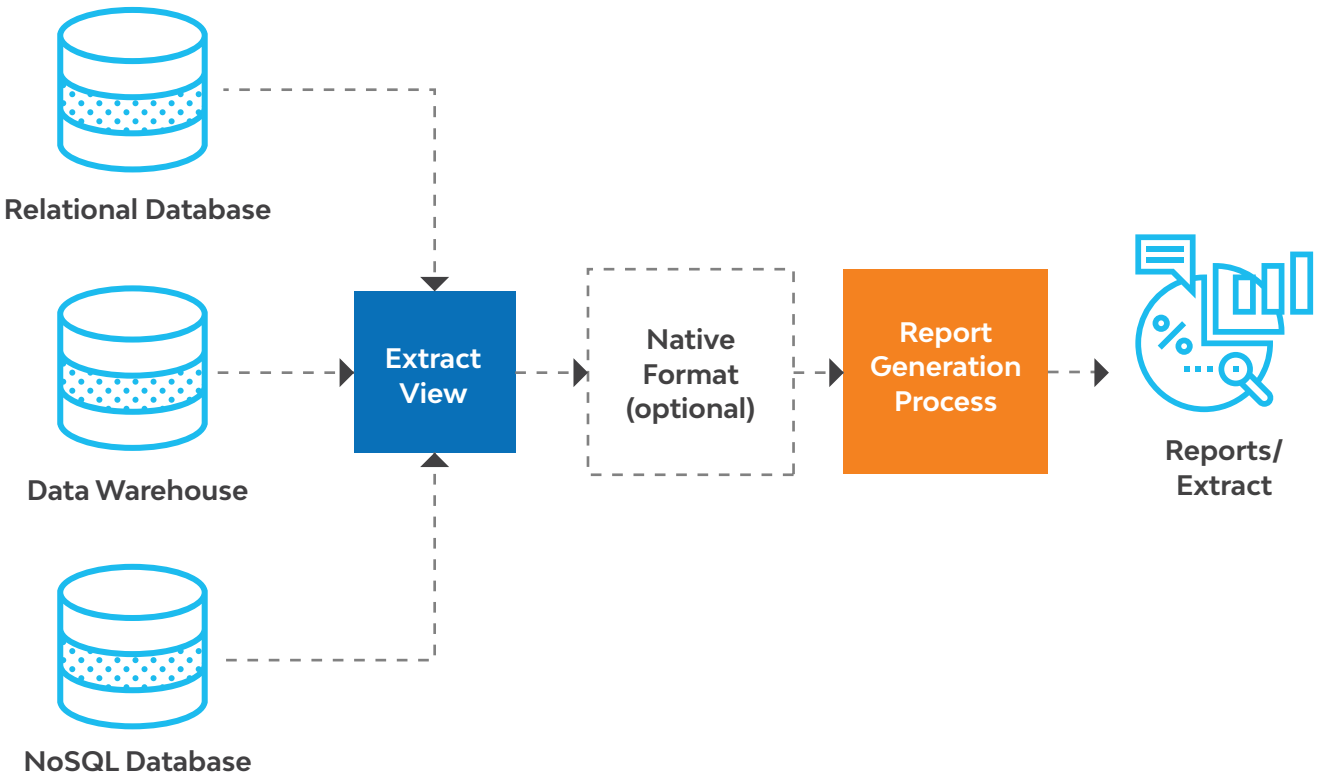



Fig 1: A traditional report generation process using any traditional tool.

Let's visualize how it will look using Micro-services. We are not changing the data source for simplicity. Albeit, Micro-services can work as a typical Enterprise Service Bus (ESB) and can fetch data or file from any source using any standard protocol such as MQ, JDBC, FTP, Web-services etc. The flexibility to provide integration with any internal (on premise or on cloud) or any set of external applications through APIs or any connectors, makes Micro-services much more powerful than traditional reporting applications. Although traditional tools provide connectivity through FTP and MQ, implementing that would require a lot of custom coding, which is as good as writing a new Micro-service.

In this approach, we can come up with a simple Data Extraction layer that will represent each business object. For example, Account is a business object. All the data in the Account table will be exposed through ms_account service with API as end point. Wherever we need data from Account we can use this API. There will be similar Micro-services or API end points for each business object and the cluster would work as the Data Access layer.

The Data Extraction layer would fetch data using sets of pre-defined/ad hoc selected fields and logic. Then, by using structural components, which are represented as "Helper Service", the report generation service can generate the report and can pass that to the front end as HTML, Excel, PDF or a text file.

The diagram below illustrates theis approach:

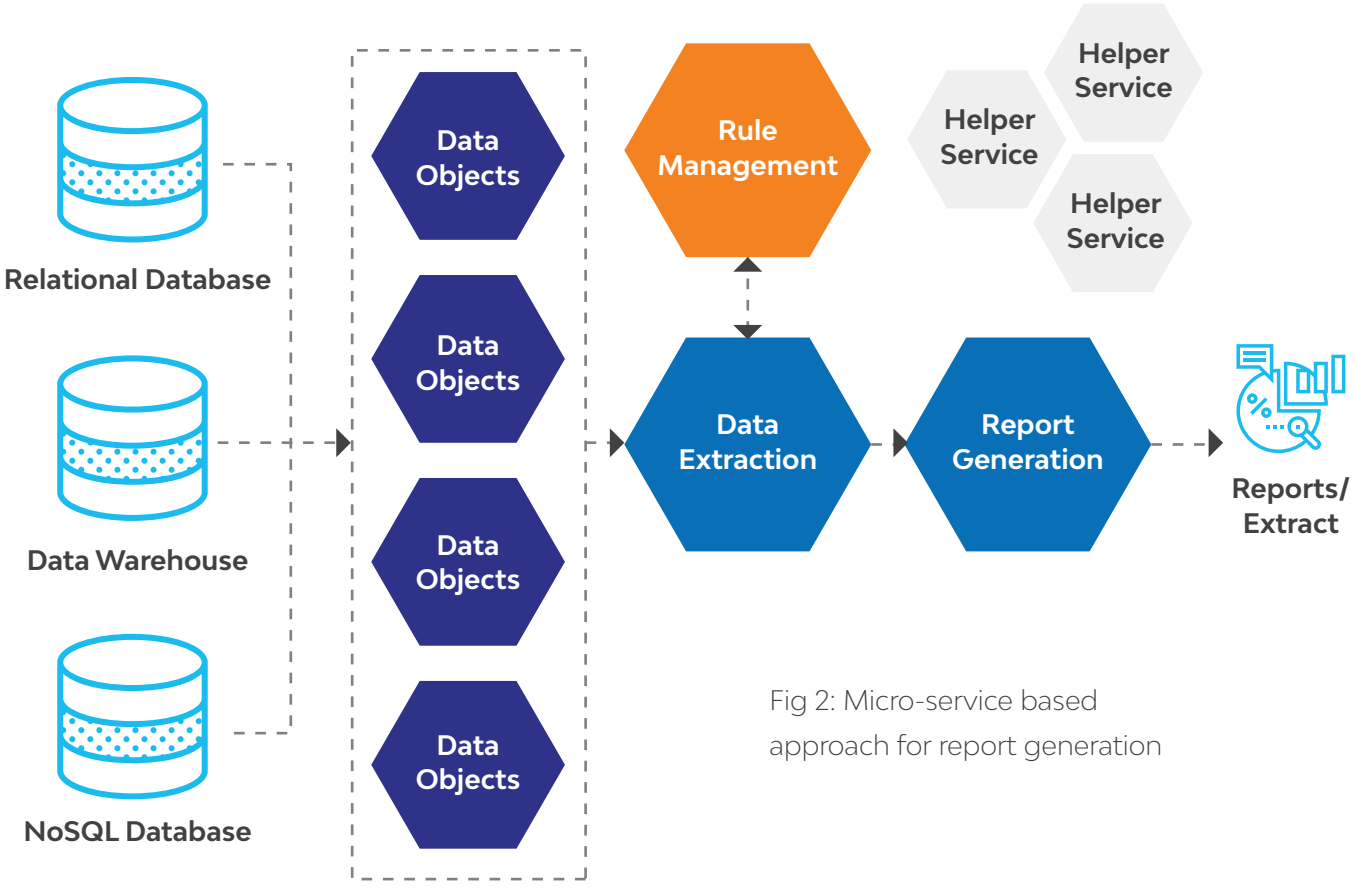


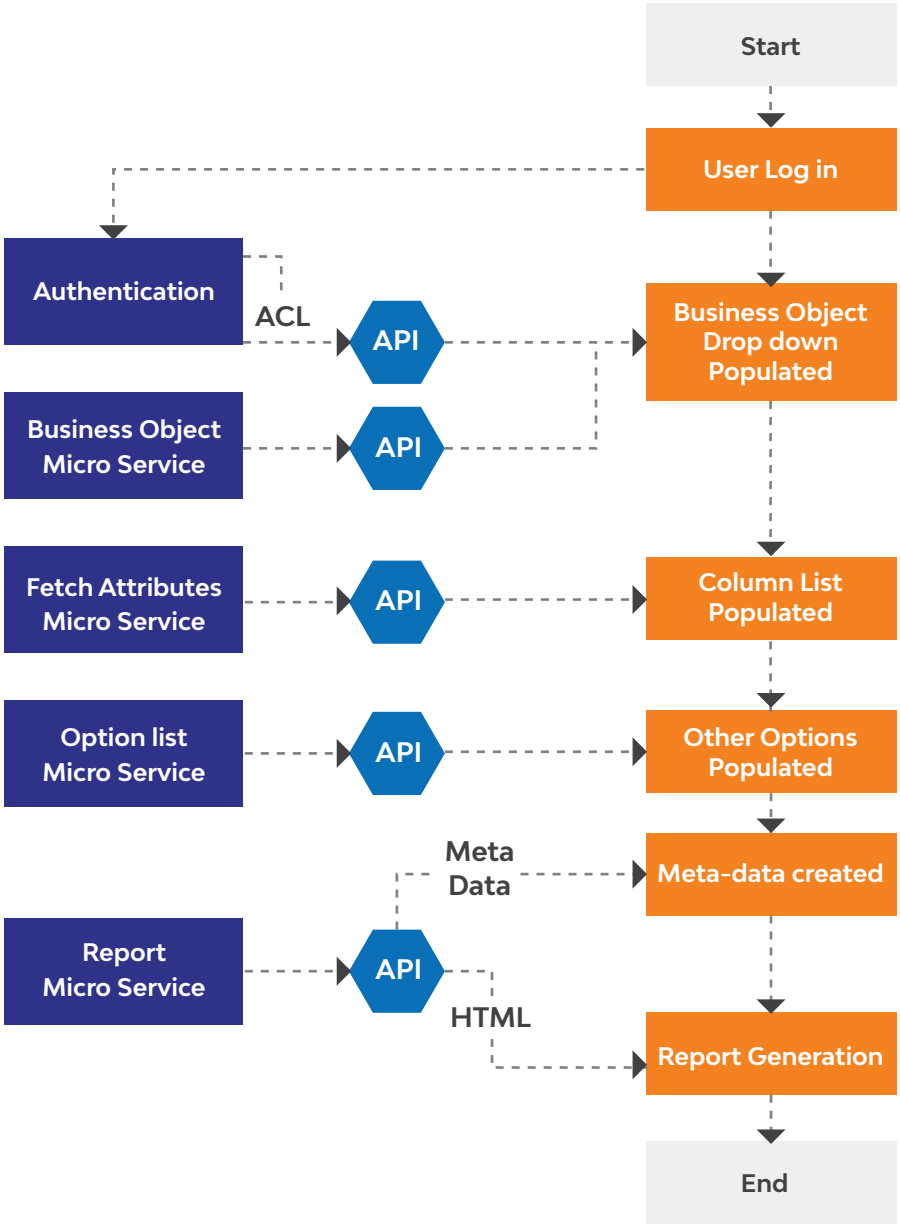

Fig 2: Micro-service based approach for report generation

Let's divide this into different framework components are see how can we implement it.

**Business Objects as APIs:** Entities can expose their data through APIs with all standard methods of GET, POST, PUT, PATCH and DELETE. We can fetch data from one table or from a group of tables. Business entity or object is the set of data that can exist independently, with no dependency on other objects (correlation can exists). We can make API call parameterized so only the filtered data will be returned as JSON object.

**Data Extraction Service:** This layer will call APIs for business object as per definition of the report that can be configured in Rule Engine or Template configurator. Let's consider the UI angle. We provide an UI screen where the user can choose business object, set of columns, sequence of appearance, groups, total required or not, and output format. The data model can be exposed as JSON that can be used by UI code – JSP or Angular to create such screens.

The logical flow of report generation would look like this:



**Helper Micro-service/Report Factory Service:** This is the most complex but interesting part of our story. The framework will be based on how skilfully we are creating this module. At a very high level this module will provide the required backbone of the reporting framework. This will capture all the attributes of the reporting meta-data. To design this, we could have used an XML-based approach. we designed this as per relational norm of data modelling. Let's see how this meta-data data model will look like.

| Flow_ID (PK) |
| --- |
| Flow_Name |
| Flow_type |

| Grp_After_ID (PK) |
| --- |
| Grp_After_Name |
| Grp_key |
| BGColor |
| Height |
| Weight |
| Grp_total |

| Grp_Before_ID (PK) |
| --- |
| Grp_Before_Name |
| Grp_key |
| BGColor |
| Height |
| Weight |

| Table_Footer_ID (PK) |
| --- |
| Table_Footer_Name |
| BG_Color |
| Height |
| Weight |
| Font |
| Grp_total |

| Table_Header_ID (PK) |
| --- |
| Table_Header_Name |
| BG_Color |
| Height |
| Weight |
| Font |

| Report_Header_ID (PK) |
| --- |
| Report_Header_Name |
| Height |
| Weight |
| Image |
| BG Color |
| Logo |
| Logo_Position |

| Report_Footer_ID (PK) |
| --- |
| Report_Footer_Name |
| Height |
| Weight |
| Image |
| BG Color |
| Logo |
| Logo_Position |
| Foot_Note |

Fig 3: Table Structure – Meta Data

1. Flow: Decides how representation will be done, i.e. –vertically (report will expand from top to bottom, records will be added vertically), or horizontally (records will be added horizontally and report will expand from left to right).

2. Report Header: Appears first in the report and can contain the logo, report's name and background image or color.

3. Report Footer: Appears last in the report, and can contain the logo, footnote and background image or color.

4. Table Header: These will be the columns displayed in the report.

5. Table Footer: This may contain total fields if selected.

6. Group Header: This field gets populated when the group by is used, and is displayed when the group key changes.

7. Group Footer: For total or group separator purpose.

8. olumns: This table stores column related meta-data.

9. Business_Objects: This table contains lists of all business objects.

**Report Generation Service**

Using the report factory services, the Report Generation Service captures a user's preference for each reporting instance and then stores that in the Instance table. If the user schedules the report then this Instance definition is used to re-create the report. The report generation service will fetch the meta-data from the Instance table and will call individual factory/helper services and will get HTML snippets as return. Finally it will use selected/predefined templates to make the layout of the report, followed by the final HTML report with all embedded features that will render on the user's screen.

There will be connecting tables which will work as collection for the reporting instance. For example, Column_List table will list down all selected columns (by user) with respective preferences of position, font, and color. This will be mapped to the instance table and at run time, it will fetch all the columns with its meta-data selected by user. This meta-data will be used to render the HTML snippet and column name will be used to form the query that will fetch the record from the exposed JSON data through APIs.

| |
|---|
| Instance_ID (PK) |
| Customer_Report_Name |
| USer_ID |
| Business_Object_ID |
| Column_list_ID |
| Report_Before_ID |
| Report_After_ID |
| Table_Header_ID |
| Table_Footer_ID |
| Grp_Before_ID |
| Grp_After_ID |
| Report_Format |

## End-to-End Flow

Let us visualize how this framework that is explained above in bits and pieces will stitch together to work as a working solution. Let's look at the requirement for a proper context. At high level our requirement is (keeping graph related requirement out of scope for now)

i. Application should generate standard and custom reports.

ii. Standard reports are fixed format reports for which layout is defined at design time.

iii. Custom reports are dynamic reports which are defined at run time by users.

iv. For custom reports, user will choose business objects in the drop down and then corresponding columns will be populated in the column drop down.

v. User will choose columns and then will choose position of columns.

vi. If any grouping is required then the user will choose group key or multiple group keys.

vii. User will select sorting option and will choose column on which sorting is required.

viii. User will choose total column if total is required.

ix. User will choose one among list of templates

and output format such as Excel, PDF, HTML available for the report.

x. User can save the custom report with a custom name that will be accessible to him when he will log in again.

xi. User will have the option to schedule the report.

Now we have to see how above requirement can be implemented using what we have been discussing so far.

**Standard Report Flow:** Standard reports are fixed format reports which are pre-defined. For each business objects one standard report can be defined. The Data Extract service will fetch all the records in the database – we can apply the default filter of showing first 1000 records only or can make this parameterized – for that business object and will call predefined – columns set with pre-defined template to generate a report. This report will be a simple tabular report without any grouping or sub-total.

For example, the Account Detail report lists down all the columns of Account_Detail business object and then shows all the data that is available including filtered data.

At a high level, the flow can be illustrated as shown below:

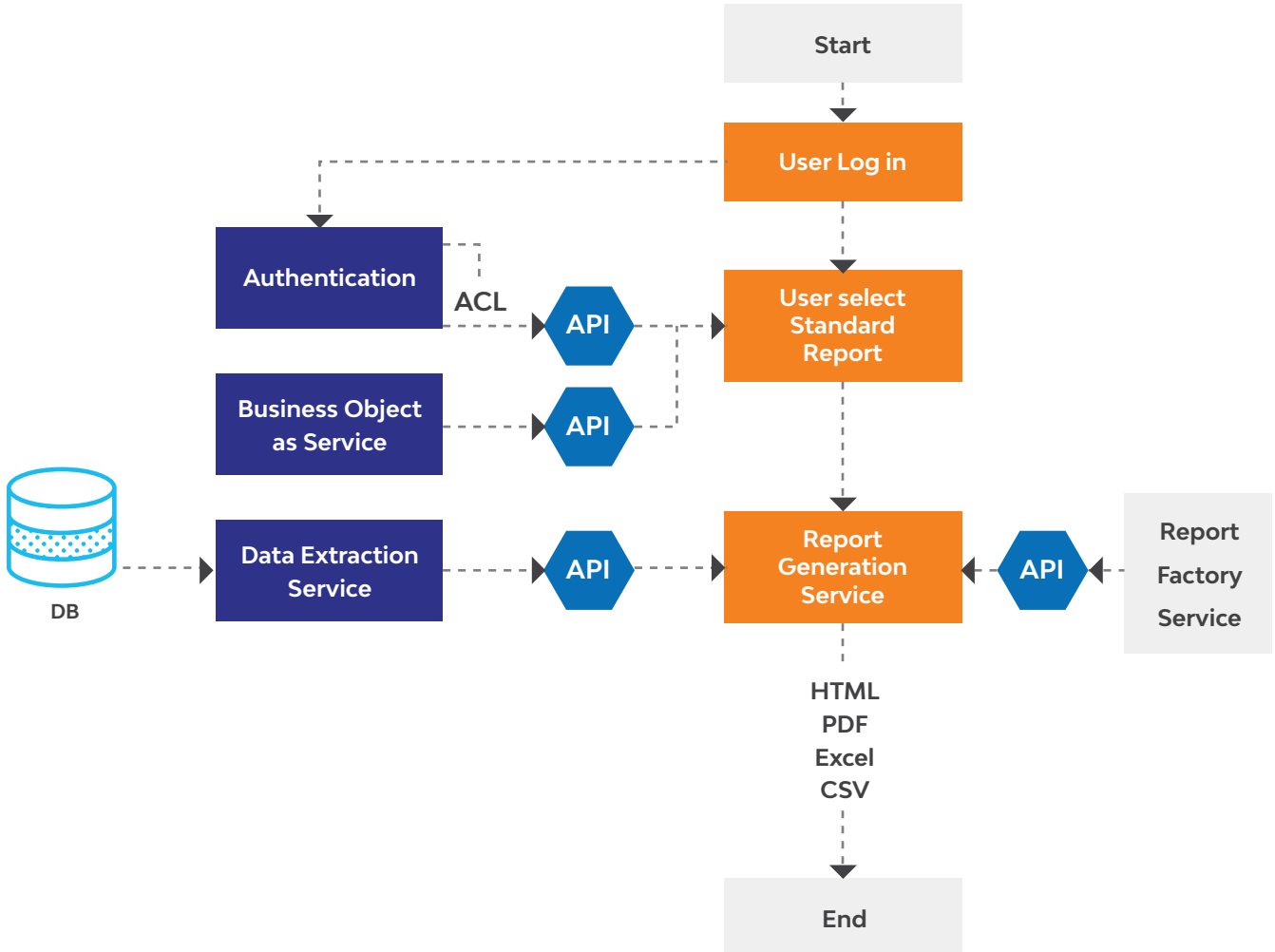



Fig 4: Standard Report

**Custom Reports:** Custom reports are the real challenge for this framework. Standard reports are more or less static in nature so creating simple HTML-based reports with little coding in the JSP/Servlet is very easy. The achievement is to provide a platform that can generate dynamic reports as per each customer's choice without any code change.

Let us solve this step-by-step:

1. User logs in and his credentials are authenticated.

2. As per pre-defined mapping (the Access Control List) Business objects will be loaded on the UI screen for Custom Report.

a. Business_Object will fetch all the business objects from the metadata table that is accessible to the user. Values will be returned through API in JSON format.

b. Angular/JSP code can call API for Business Object list.

3. User selects one Business Object at UI that calls API which in turn calls ms_Column_List micro-service. This Micro-service will fetch all the columns that are mapped to the given Business Object.

4. User selects few columns that will be passed as input parameters for the next API call that is to fetch grouping option.

5. Group by option will appear on the screen and then if the user is selecting this option then all columns that are selected in step 4 will be displayed to be selected as group key. If the user selects any grouping then the selection preference will go into group table (mentioned above).

6. Other options such as sorting options, total options and export options will be displayed on the screen and the selection will be stored in the corresponding meta-data table.

7. Once all the selections are complete, the user will be asked to assign a unique name to the custom report. The report name will be stored against the user ID. Meta-data about the custom report will be stored in the instance table.

8. The user will have an option to run the report immediately or will have the option to schedule it.

9. If the user selects "Run" then

| Instance_ID (PK) |
|---|
| Customer_Report_Name |
| USer_ID |
| Business_Object_ID |
| Column_list_ID |
| Report_Before_ID |
| Report_After_ID |
| Table_Header_ID |
| Table_Footer_ID |
| Grp_Before_ID |
| Grp_After_ID |
| Report_Format |

a. A query will fetch all the mappings/meta-data from the Instance table.
b. As we can see in the table structure, all other details about the selection will be retrieved as well as the selected Business Object, selected column list ID (list of columns will be stored in a separate table and reference will be stored in interface table against Column_list_ID), other preferences such as

report before, after, report before, after, group key etc.

c. All this data for the custom report will be available through one API.

d. Now, the Report Generation Engine will call Helper Micro-services (Factory services) to create HTML snippets for the report. For example, for the header part – corresponding Micro-service will take the parameters stored in the DB, such as logo file location, background color and a custom report title, and then will return the HTML snippet for the report header. Similarly, for the footer and other sections the HTML snippet will be returned by respective Micro-services.

e. The Report Generation Engine will use a pre-defined template to arrange HTML snippets to create reports. Once reports are created, it will be rendered on the user's screen.

10. For scheduled reports, the Reporting Engine will use the instance table to find out all the reporting preferences of the user and will form the layout of the report at every trigger of the schedule. We can use any standard scheduling tool/features.

## Advantages of Using Micro-service for Reporting Framework

Now let's come back to our original topic: Why micro-services?

1. They are scalable
2. Multiple versions can co-exist
3. Evolutionary in nature – add features without affecting existing ones
4. Cloud Ready
5. Java-based hence has endless possibilities
6. Technology agnostic
7. Strong performance and usability monitoring options
8. Service-based architecture

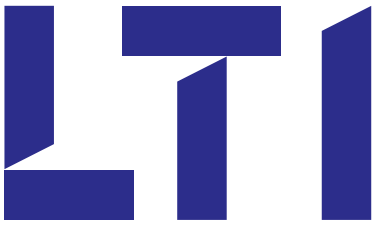
## Conclusion

Hopefully this article has triggered some imagination and design thinking in the reader's mind. The very purpose of this article is to encourage application managers to revisit their strategy for reporting requirements. With little investment we can develop a reporting application that can serve the purpose and can save huge licensing costs. Apart from that this framework can provide a platform that can be extended to handle NoSQL data, transient data, log data and data from RDBMS – all within one application for us to integrate and to extract the best of what we can!

# About the Author

**Manish Ranjan**
Senior Technical Architect, LTI Whitefield, Bangalore, India

Manish has around 16 years of experience in Analytics and Enterprise Integration. He is extensively working on API led micro-service architecture and helping clients in their APIfication journey.



Let's Solve

LTI (NSE: LTI, BSE: 540005) is a global technology consulting and digital solutions Company helping more than 300 clients succeed in a converging world. With operations in 30 countries, we go the extra mile for our clients and accelerate their digital transformation with LTI's Mosaic platform enabling their mobile, social, analytics, IoT and cloud journeys. Founded in 1997 as a subsidiary of Larsen & Toubro Limited, our unique heritage gives us unrivaled real-world expertise to solve the most complex challenges of enterprises across all industries. Each day, our team of more than 25,000 LTItes enable our clients to improve the effectiveness of their business and technology operations, and deliver value to their customers, employees and shareholders. Find more at www.Lntinfotech.com or follow us at @LTI_Global

info@Lntinfotech.com

A Larsen & Toubro Group Company