

Backend Assignment Design Specification Report

Submitted By: Ankit Kumar
Email: ankitanand2909@gmail.com
Contact No: +919024933694

Overview:	2
Technology Used:	2
System Architecture Explanation:	2
Form-MicroService:	3
Data-Export-Microservice:	4
SMS-MicroService:	5
Error-Data-MicroService:	6
API-Gateway:	7
Eureka-Server:	7
Form-Load-balancer:	8
Database Schema	8
Demonstration of Google use case:	9
Demonstration of SMS use case:	9
Problem Statement and their solution:	9
Google sheet use case:	9
SMS use case:	9
Error Handling use case:	9
How we handle millions of requests:	10
Plug and play:	10
Single interface:	10

Overview:

As the problem statement suggested, we mainly need to care about implementing the features and a clean bug-free architecture to handle millions of requests. This system should also have the ability to interact with third-party services like Google sheet API, SMS service, and more. I considered breaking every use case into different microservices and implemented the horizontal scaling concept to solve architecture problems. I have used the Java Spring framework for server-side coding.

Technology Used:

- Java
- Spring framework
- REST
- Docker (For MySQL Database instance)

System Architecture Explanation:

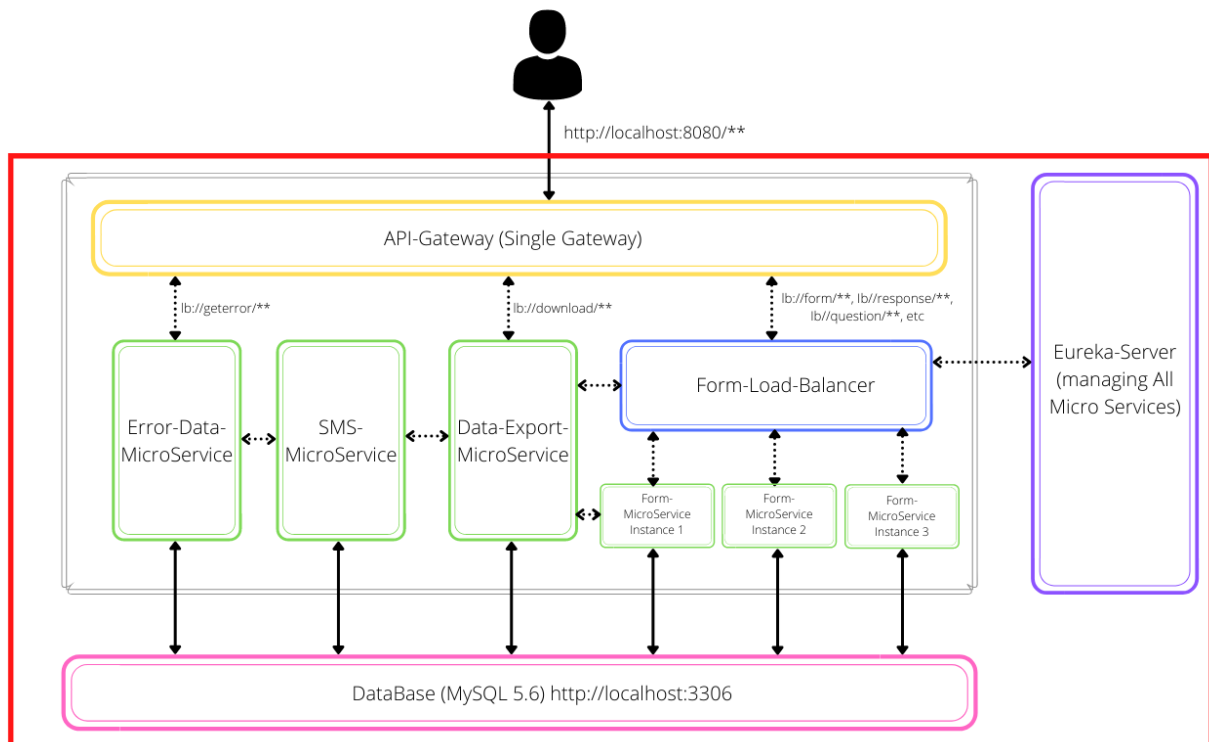


Fig 1: System Design

I followed the microservice architecture to solve the problem because in this architecture we can easily plug in the service instance and in case of a high number of requests we can add more instances of the same service using different ports.

Let See one by one the different microservices and their use:

Form-MicroService:

This microservice handles all the queries related to the Form, Response, Question, and answer. There are multiple instances running at the same time to support the load balancing. All the different instances which running on different ports are managed by the eureka server. **We can add the new instance whenever we want and the eureka server discovery client gets registered automatically like in plug and plays fashion.**

All the REST endpoints for Form related queries are mentioned here.

BASE_URL : <http://localhost>

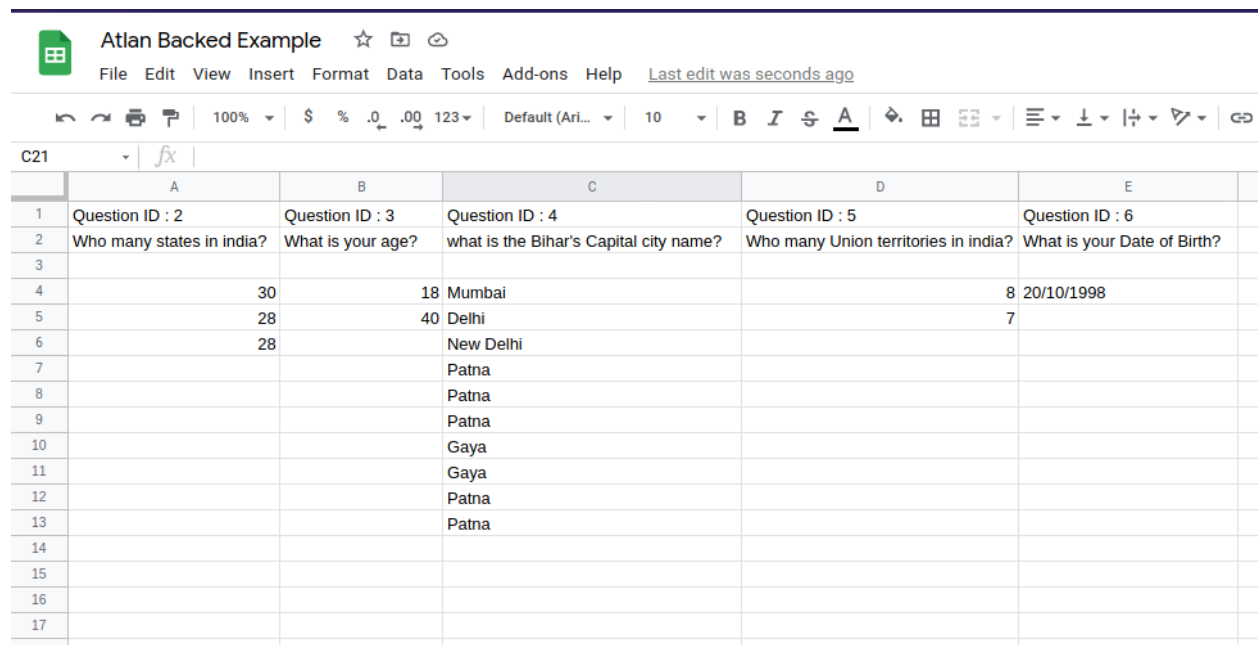
S.No.	Method	Endpoint	Request Body	Response Body
1	GET	/forms	--	Return the list of Form Body Object along with a list of questions associated with the form.
2	GET	/form/{id}	--	Return the single form
3	POST	/form	{ "title": "name_of_form" }	Return the newly created Form object along with ID.
4	GET	/questions	--	Return the list of all the questions along with their responses.
5	POST	/question	{ "value": "question", "answer" : "expected answer", "form": "form_id" }	Return the newly created question object along with ID.
6	GET	/responses	--	Return the list of all the

				responses
7	POST	/responses	{ "Value": "answer_String", "question": "question_id" }	Return the newly created response object

Data-Export-Microservice:

This microservice handles all the export-related works like **Google-Sheet use case implementation and downloading the resources in CSV files**. This microservice also uses Google Sheet API to create and upload the data to the sheet. For uploading the data to google sheet we need a credential.json file and a google account.

Here is the Screenshot of the initial data present on the server:



	A	B	C	D	E
1	Question ID : 2	Question ID : 3	Question ID : 4	Question ID : 5	Question ID : 6
2	Who many states in india?	What is your age?	what is the Bihar's Capital city name?	Who many Union territories in india?	What is your Date of Birth?
3					
4	30	18	Mumbai	8	20/10/1998
5	28	40	Delhi	7	
6	28		New Delhi		
7			Patna		
8			Patna		
9			Patna		
10			Gaya		
11			Gaya		
12			Patna		
13			Patna		
14					
15					
16					
17					

Fig 2: Uploaded Sheet

All the REST endpoints for Export related queries are mentioned here.

S. No	Method	Endpoint	Request Body	Response body
1	GET	/download/google	--	Google Sheet shareable link
2	GET	/download/forms	--	A downloadable CSV file with all the information
3	GET	/download/forms?form={id}	--	A downloadable CSV file with data related to the form requested.

SMS-MicroService:

This microservice handles SMS service. Whenever the response is recorded, the backend sends the SMS to the registered mobile number. **I used the Twilio SMS API to send the SMS.**

Here is the screenshot of the message when we submit the response:



Fig 3: Response SMS

This Service doesn't contain any public endpoint because this service can only be used by the other microservices. Although it may contain a private IP address but not public.

Error-Data-MicroService:

This microservice is used for recording all the error messages thrown by other microservices during uptime. **It stores the location of the error produced along with the message in the database and can easily retrieve it via GET request for further study for the cause of failure.** We can also add other rules of validation as project demand and add them to the table. Here is the Screenshot of the table:

```
mysql> select * from error;
```

id	class_name	date_time	instance_port	message
2	com.atlan.formService.Controller.Impl.FormControllerImpl	Sun Sep 19 11:25:21 IST 2021	8088	Error caused by nullPointer Exception at line 5
3	com.atlan.formService.Controller.Impl.FormControllerImpl	Sun Sep 19 11:25:26 IST 2021	8090	Id not found

```
2 rows in set (0.00 sec)

mysql>
```

Fig 4: Error data table

Here is the Rest Endpoint for queries related to error data.

S.No.	Method	Endpoint	Request Body	Response Body
1	GET	/error	--	List of all the error Data

API-Gateway:

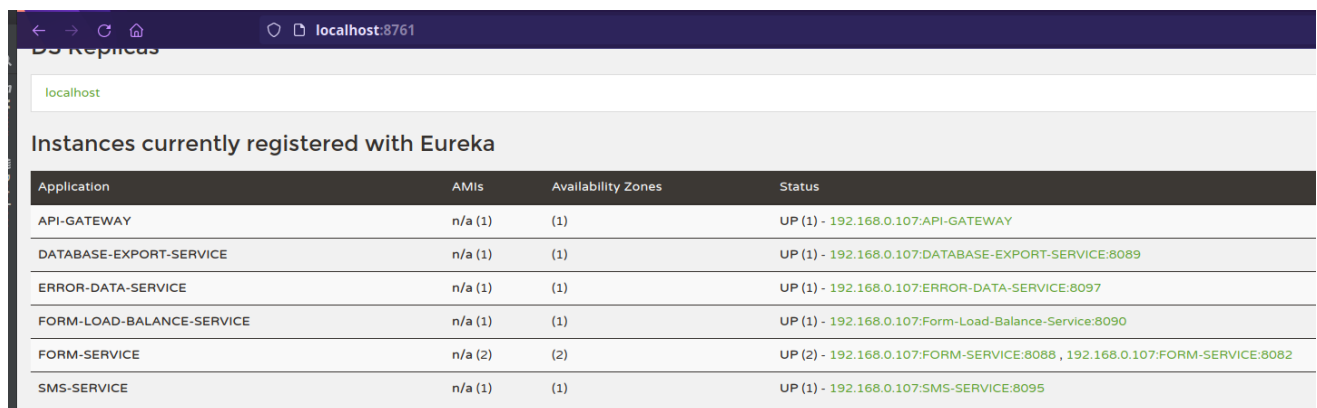
This microservice is used to accept the request from the frontend and redirect the request to the other microservices. By using this, **we can access our backend with a single public IP address and users need not worry about different services present inside the backend.**

BASE_URL : <http://localhost/>

Eureka-Server:

This server is used for managing all the microservices running on the system. Eureka knows about all the services currently running and their instance. It helps us in knowing the status of different services.

Here is the screenshot of the eureka server dashboard:



The screenshot shows the Eureka Server Dashboard in a web browser. The browser's address bar displays 'localhost:8761'. The dashboard has a header with 'localhost' and a main section titled 'Instances currently registered with Eureka'. Below this title is a table with four columns: 'Application', 'AMIs', 'Availability Zones', and 'Status'. The table lists six applications: API-GATEWAY, DATABASE-EXPORT-SERVICE, ERROR-DATA-SERVICE, FORM-LOAD-BALANCE-SERVICE, FORM-SERVICE, and SMS-SERVICE. Each row provides details on the number of instances (AMIs), the number of availability zones, and the current status (UP) along with the specific URLs of the instances.

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - 192.168.0.107:API-GATEWAY
DATABASE-EXPORT-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.107:DATABASE-EXPORT-SERVICE:8089
ERROR-DATA-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.107:ERROR-DATA-SERVICE:8097
FORM-LOAD-BALANCE-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.107:Form-Load-Balance-Service:8090
FORM-SERVICE	n/a (2)	(2)	UP (2) - 192.168.0.107:FORM-SERVICE:8088 , 192.168.0.107:FORM-SERVICE:8082
SMS-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.107:SMS-SERVICE:8095

Fig 5: Eureka Server Dashboard

In the above fig, we can see that the currently running service and no of their instances. Not only monitoring but it also provides the mapping of URL.

Eureka Server: <http://localhost:8761>

Form-Load-balancer:

This server is used to redirect the request to the different form-microservices instances so that we can handle the bulk requests. There are different types of algorithms used to select which instance gets which request. Load-balancer solves the problem of over-requesting a single instance of service and divides the request into others.

Database Schema

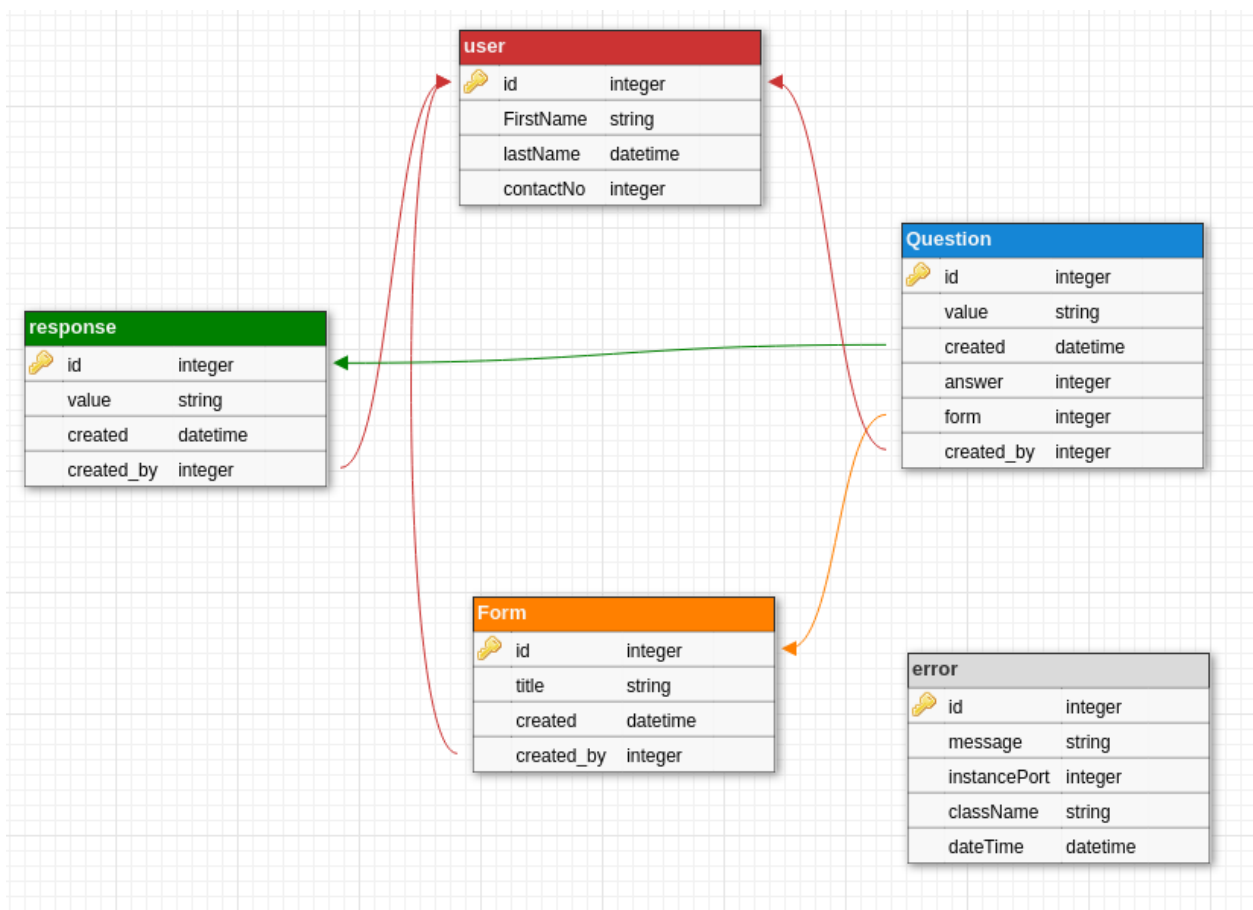


Fig 6: Database Schema

For this project assignment, we used a very simple schema to demonstrate the work:

There are total 5 tables present in the database:

- user
- response
- form
- error
- question

Demonstration of Google use case:

Here is the Drive link of a video:

https://drive.google.com/file/d/14JqSqfze2NR59K5auNTJB_xAeRhHwRMJ/view?usp=sharing

Demonstration of SMS use case:

Here is the Drive link of a video:

https://drive.google.com/file/d/1SH_3S1brbjNlf3KBDcyaBYPXFebZ2JCF/view?usp=sharing

Problem Statement and their solution:

- **Google sheet use case:**

Created a microservice that handles the export of data from database to google sheet via Google SHeet API. With only one Get request, the backend creates a new Google sheet and transfers all the data to it.

- **SMS use case:**

Created a microservice that handles our SMS functionality. We took the help of Twilio API to request the third-party service to send the SMS to the mentioned phone no. We can also define the rule that how and when the SMS will be sent to which user.

- **Error Handling use case:**

Created a microservice that handles all our error and unexpected rule-breaking data storage so that developers can get these data in one place. Every time the backend

throws an exception or any rule break, this microservice records the data and stores it into the database.

- How we handle millions of requests:

Created the load balancer that can divide the incoming request into different available server instances. We can create the instance as per our need at different ports.

- Plug and play:

We created a eureka server that manages all our microservices. If we want to add a new instance of any service during the run time we have to simply run the application at a different port and eureka discovery automatically registers the microservice.

- Single interface:

We created an API Gateway service that manages all the incoming requests to the backend. This provides a single public interface and diverts the request to different microservices based on the resources requested.

I have set up logs everywhere to track the flow, service health can be monitored by the eureka server. Also, the error microservice stores all the exception and rule break cases.

--Thank you--