Backend Various Approach and their Pros and Cons

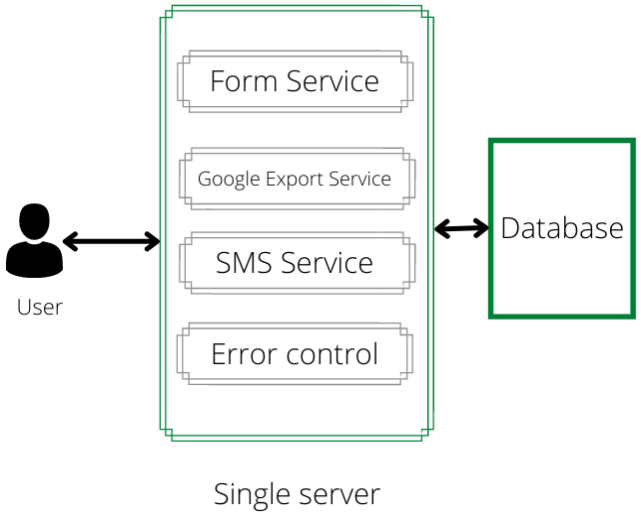
Submitted By: Ankit kumar
Email: ankitanand2909@gmail.com
Contact no, +919024923695

System architecture approaches:

Monolithic approach:

In this architecture, a mono service is created which contains all the other sub-services. A single instance of all services integrated into one server load.

A single interface that handles all requests.



Pros:

- Easy to handle.
- Less resources are required and easy to deploy.
- Easy to manage.

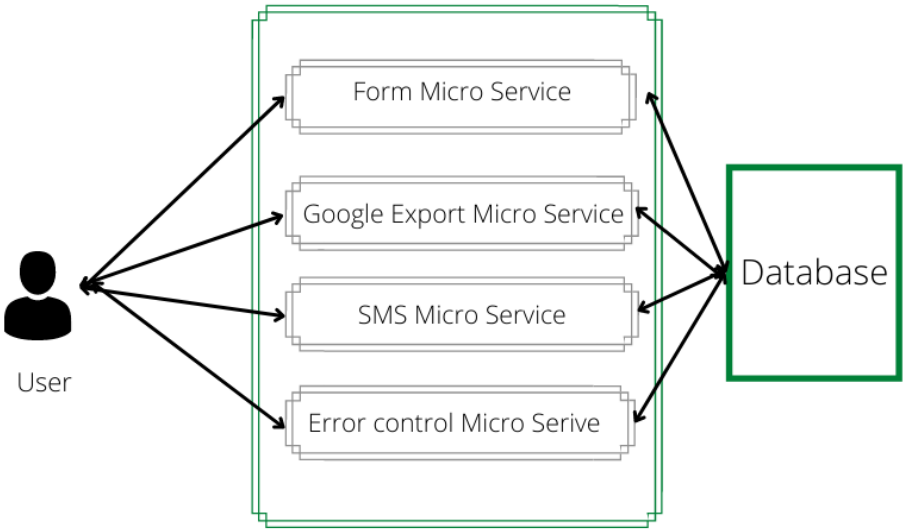
Cons:

- The whole system fails by encountering one single error.
- Monolithic architecture can't handle millions of requests.
- We can't scale the system.

Microservice approach:

In this approach, every service behaves like a microservice. They are independent of other microservice but can communicate with others. This approach solves the problem of a single point of failure. If one service gets down, it won't affect the other service.

Every microservice has its own interface.



Pros:

- Easy to handle
- Solve the problem of a single point of failure.
- Debugging is easy compared to monolithic.

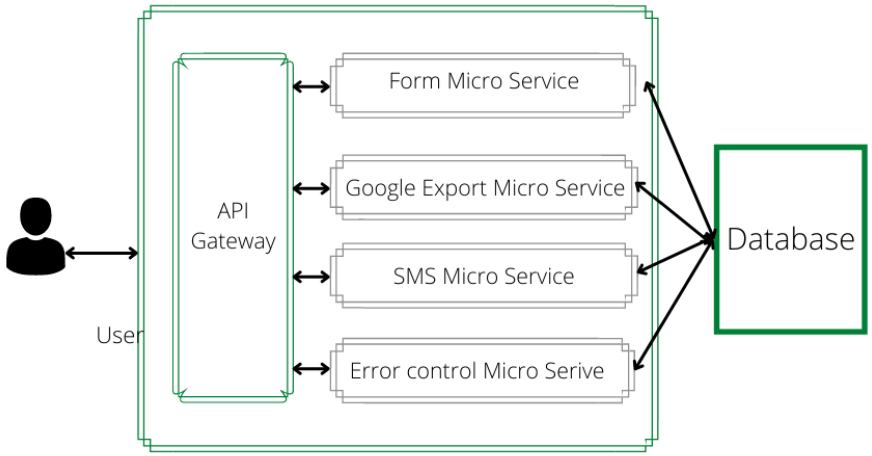
Cons:

- Still can't handle millions of requests.
- Every microservice has its own interface.
- Managing the different microservice which run on different IP addresses is hard.

API Gateway approach:

In this approach, a single gate is provided for all service access. Users know only one single gateway of access. This will solve the problem of multiple interfaces.

A single interface that handles all requests.



Pros:

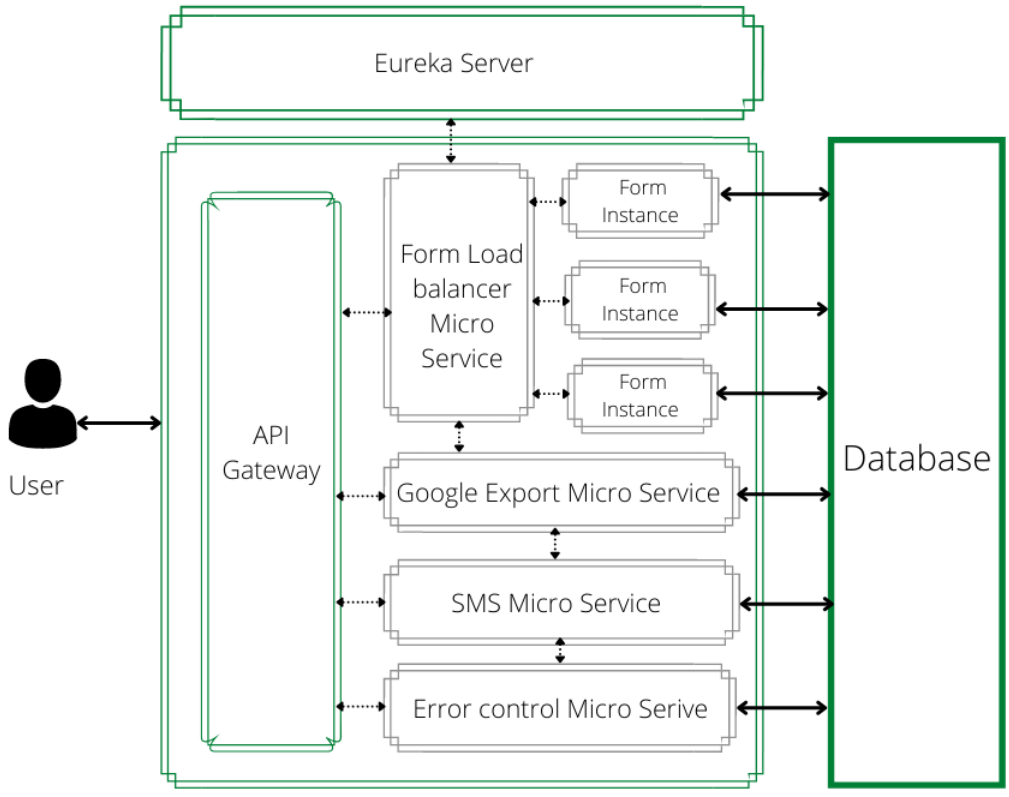
- Microservice functionality with a single gateway.
- Easy debugging.

Cons:

- Still can't handle millions of requests.
- Managing the microservice is hard. And if one service gets down that service is completely unavailable.

Microservice with Load-balancer and eureka server (My approach):

In this approach, we are using a load balancer to handle and divide the request into the different current running instances of the same service. Eureka server manages the microserver and keeps tracking of status. In this approach, we are free to launch multiple instances of service to divide the request. This approach can handle millions of requests.



Pros:

- Provide a single public IP address because of API Gateway.
- Due to the load balancer, we can handle millions of requests.
- Eureka's server can manage all our microservices.

- Debugging becomes easy due to the eureka server.
- If the system encounters failure we can create a new instance of that service.
- Plug the service at runtime.

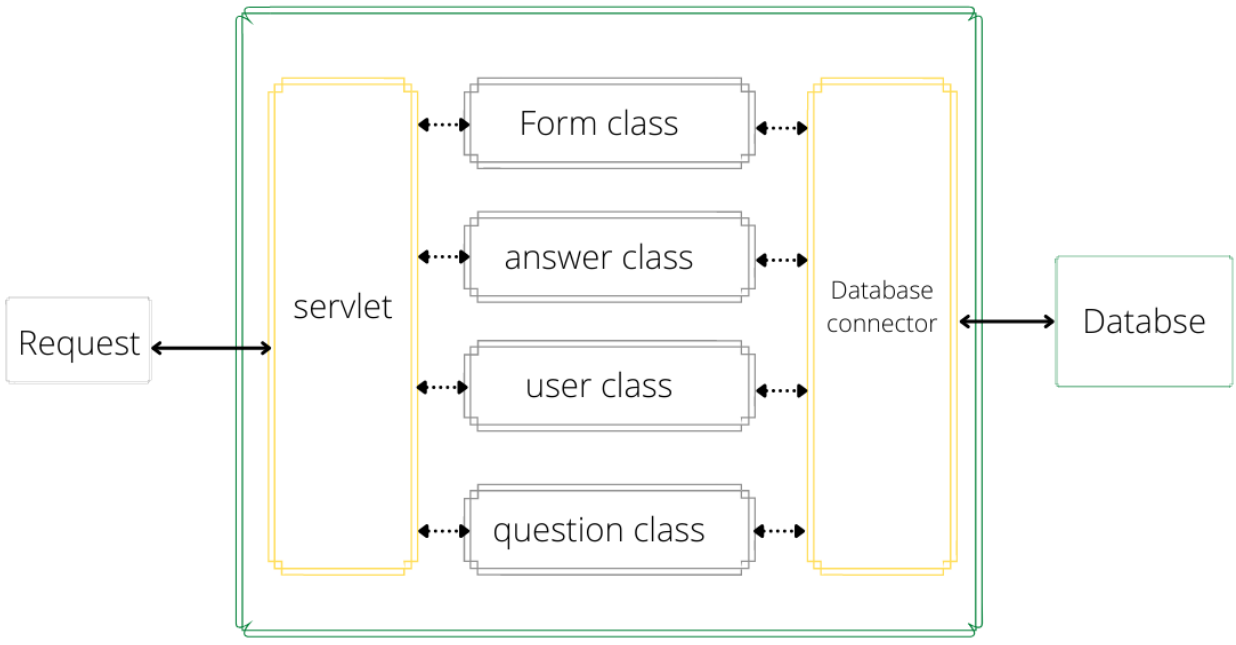
Cons:

- Hard to handle such a complex architecture.
- The cost of running is high compared to other approaches.

Software design pattern:

Normal approach:

In this approach, for every method, we define a single class. We handle all the network, business, service logic in a single place. This makes the code easy to handle in one place but this is not a good approach. **Single class for all work.**



Pros:

- Easy to handle.
- All code in one place.
- Less resources are required.

Cons:

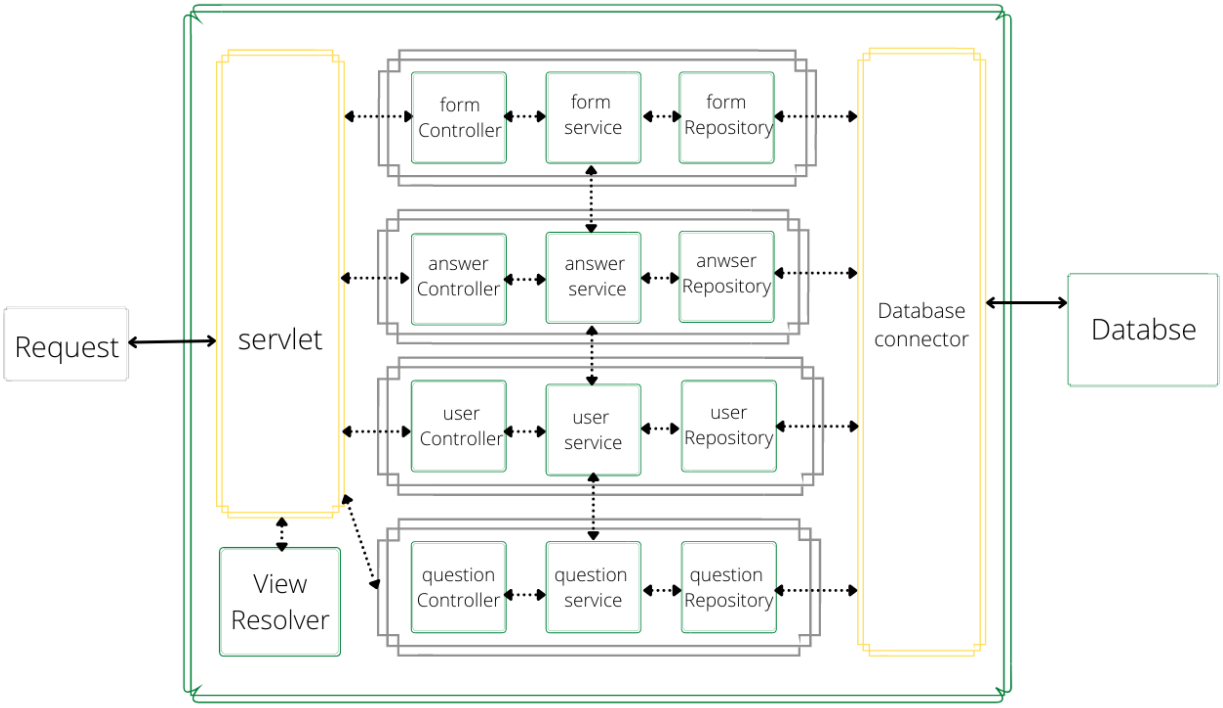
- Hard to debug.
- Complex to understand after some time.

Model View Controller (My Approach):

In this architecture, we divide our code into different groups based on their behavior. **This pattern mainly contains three-part: Model, View, Controller.**

Model: Storing the data and defining how the data is structured.
View: defining the response body;
Controller: control everything and handle all the flow and user-defined rules.

Here we defined Controller, service, and repository architecture.



Pros:

- Easy to debug.
- Separate class for every logic.
- Clean code.

Cons:

- Little hard to understand.
- Long codebase.