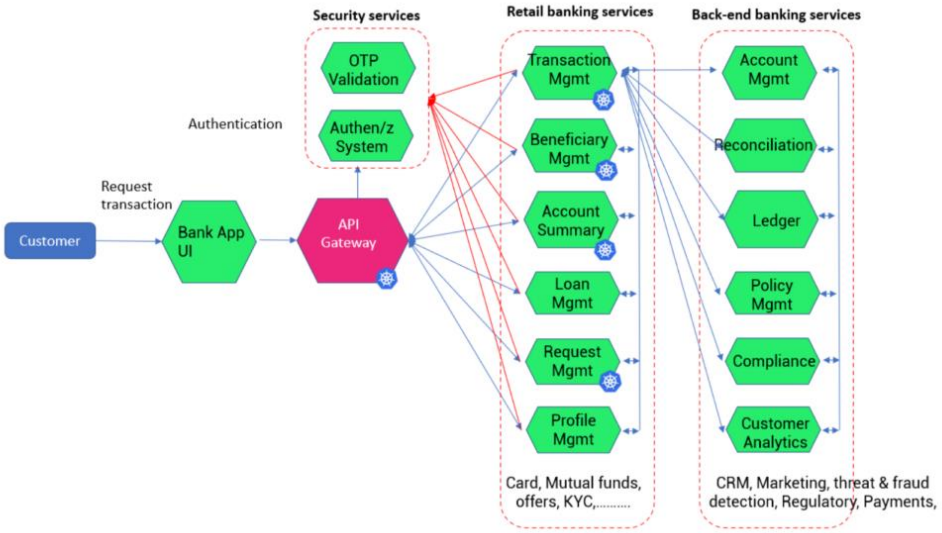
7)API Gateway, Service Discovery, and Load Balancer

1) API Gateway

API Gateway can be summarized as a central management structure that combines all APIs within the application under a single roof. When we think of an application, there are separate APIs for each microservice, and different requests are made to use these services. With the use of API gateway, the request first comes to the gateway and is transmitted to the desired structure from there. API Gateway usage for the bank can be exemplified below: The request from the application created for the bank first goes to the API Gateway, and from there it is forwarded to the microservices required for the application.



There are many advantages that the use of gateway brings to the system, these are.

- **Central management:** Since the system is gathered at a single point, it becomes easier to examine and manage it.
- **Security**: Since all requests pass through a single point, security steps are installed on the API gateway, and there is no need to establish a separate security layer for each API. In the example above, this security step is exemplified, and the authentication steps are connected to this gateway.
- **Scalability:** Gateway contributes to the establishment of a scalable application as it can increase or decrease the running instances. For a shopping site, service discovery allows for scaling up the number of instances

during peak periods like Black Friday, effectively managing increased traffic and preventing system crashes. Similarly, during high-demand events like course selection periods in universities, scaling up the system capacity can ensure smooth operation. Depending on the system design, scaling can be achieved either vertically or horizontally. Vertical scaling involves increasing the capacity of individual instances, while horizontal scaling involves accommodating a higher number of instances as needed.

- **Monitoring:** Since all incoming requests pass through a single point, it becomes much easier to analyze system load and performance.
- **Traffic Control:** Since incoming traffic can be examined, it allows restricting or blocking incoming traffic at certain points. While you can speed up the system by allowing cash to be made; Can be used to block requests that pose security risks.
- **Protocol Translations:** Since it is situated between the user and the services, the protocols utilized by the user and the services may vary. For instance, while the user employs HTTPS, services may communicate using HTTP.

2) **Service Discovery**

In an application, there are multiple services, and there may be more than one instance for each service. These structures and their instances can change dynamically. Configuration is necessary for these structures to communicate, and while it can be done manually, in a complex and dynamic structure, this approach poses numerous challenges. Service discovery offers a solution to this problem by configuring structures dynamically.

The service discovery structure comprises various components for managing these settings. Firstly, there's a service registry dedicated to storing service information. Service registration is employed to create entries within this service registration. Additionally, service lookup and health checking functionalities are provided to ensure system continuity and maintain its health.

Service discovery holds a significant role in contemporary application designs. With services evolving and changing continuously, this structure facilitates adjustments without the need for manual intervention. Moreover, in scenarios where demand fluctuates, the ability to scale instances up or down allows for seamless integration of newly added or removed structures. By automating

settings, service discovery greatly simplifies the implementation of a microservice architecture, enabling easy division into smaller components.

In essence, service discovery eliminates the need for manual processes and empowers the establishment of a dynamic structure.

3) Load Balancer

As outlined in the preceding sections, modern architecture often involves numerous instances performing the same function. A load balancer plays a crucial role in ensuring the equitable distribution of incoming requests. By efficiently distributing traffic across different servers or instances, this setup establishes a robust structure and mitigates the risk of system crashes by preventing excessive load on any single system.

The load balancer can distribute the load based on predetermined algorithms or criteria, such as bandwidth density, application turnaround time, or number of connections. The choice of metric can vary depending on the system's architecture or the preferences of the software developer. For instance, a platform like Netflix, which deals with streaming media, may prioritize bandwidth minimization, while an online shopping system like Amazon may prioritize minimizing turnaround time to enhance user experience.