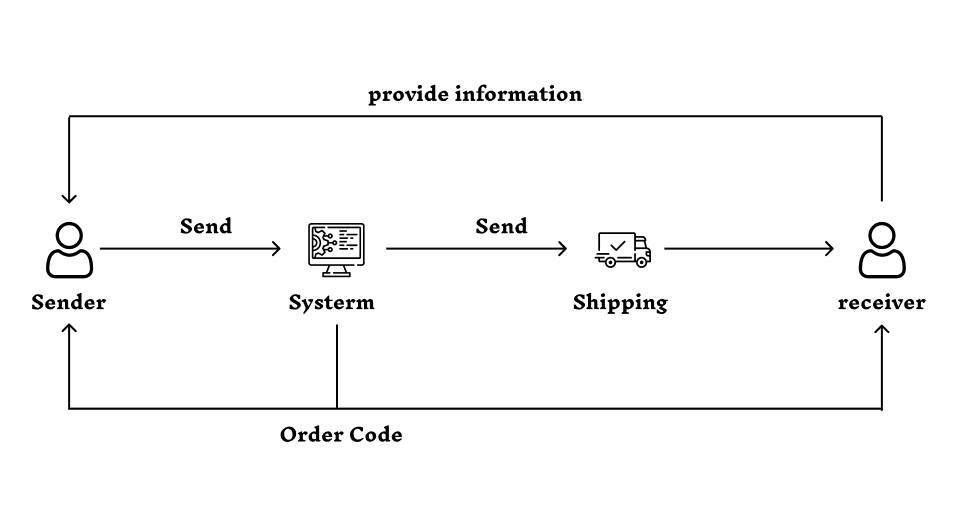
**Order Status Tracking System**

# **1. System Architecture**

# **Order Processing**

First, for each order, we need to know the sender and receiver information to ensure the most accurate processing possible. Additionally, each order should have a unique order code that is sent to the sender’s and the receiver's phone number for retrieval status of shipping. The system process is illustrated in Figure 1.

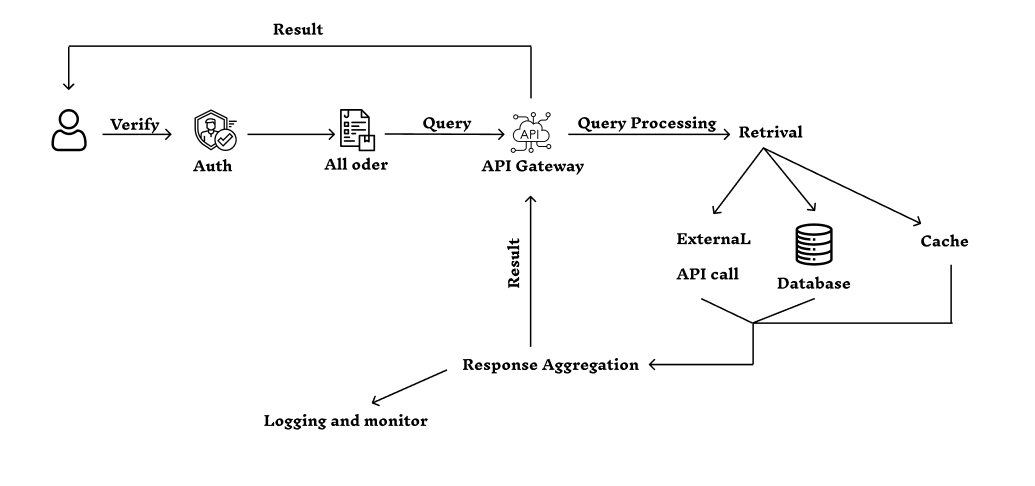


**Fig 1. Order Processing and Delivery Process**

The reason why the order code needs to be sent to both the sender and receiver is that not everyone registers an account on the system, especially online shoppers. Requiring users to create an account just to check the order status is inconvenient. Instead, they can simply use the order code to check the status of their order.

# **Retrieval processing**

For retrieval processing i propose two ways for query data, one for user have account and one for user only want search information of status shipping. Reason why i think so is because for some people they need an account to see all their orders, especially saler, some people only want easy to look for status shipping because they don't have many orders.



**Fig 2. Retrieval system with a registered account**

In **Fig 2** the graph is a system description for those who have registered an account with a phone number, users need to authenticate their accounts through their username and password.

First once authenticated, the user needs to verify an account to sign in the system and see all information of account and shipping order, **JWT** will take care of this.

User’s query through API Gateway like **AWS**, **KONG** will be requested, which processes their requests.

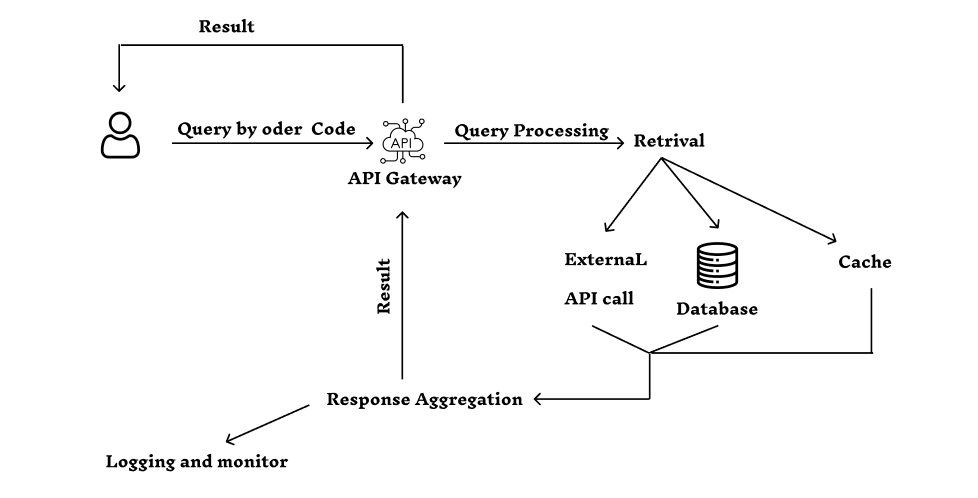
The query is then sent to the system and processed to find the result. Response aggregation will gather all responses from these sources, organize and format the data, and then return it to the user as a single, easy-to-understand result.

Later retrieve data to respond to the user through the API gateway and logging, monitoring the activity.

The user can retrieve with any order code they have, and order with their phone number if they have registered an account.

**Fig 3** for the user who wants to search status shipping by order code. At first, I thought about whether searching by order code would ensure information security. Order information could be leaked if someone knew the code. Is there a need for a security system for searching by order code?

However, after careful consideration, I think sending the code via phone number is already a secure method and furthermore, if I add a security system here, it will make the system time-consuming to check the sending status. Instead, with each sending, I will remind customers to avoid revealing the code to prevent information theft.



**Fig 3. Retrieval system without a registered account**

For users without an account they can query by order code of order provided. The system will return the order corresponding to the code they have. This will make it more convenient for those who do not need to register an account but still want to check their orders.

The system becomes simple for users who only want search status shipping and don't want to create an account.

**2. Components and Their Roles**

**Verify:**

Function: Validating API keys, tokens, or credentials before forwarding requests.

Responsibilities:

* User Authentication: Confirm the identity of users by validating credentials such as usernames and passwords, API keys, or OAuth tokens.
* Token verify: Check validity token to ensure they are authentic and not tampered without a database of data shipping.

**All order:**

Function: Show all orders which relate to the phone number registered.

Responsibilities:

* Retrieve order: Fetch all orders associated with the provided phone number from the database.
* Data Privacy: Ensure that only the orders linked to the verified phone number are shown to maintain data security and privacy.

**API Gateway:**

Function: Serves as the entry point for all user queries across different channels.

Responsibilities:

* Routing Requests to Appropriate Services: Directing incoming API calls to the correct microservice.
* SSL Termination: Handling the decryption of SSL/TLS traffic to offload the overhead from backend services.
* Rate Limiting: Implementing mechanisms to control the number of requests a user or client can make to prevent abuse.
* Load Balancing: Distributing incoming traffic evenly across multiple service instances for optimal resource use and high availability.

**Database:**

Function: Database will help me to retrieve data quickly through order code.

Responsibilities:

* Data Storage: Efficiently store structured and unstructured data for easy retrieval and management.
* Data Retrieval: Provide mechanisms for querying and retrieving data based on various criteria, ensuring fast access to information.
* Database Management: Oversee database configuration, performance tuning, backup, and recovery processes to ensure high availability and reliability.

**External API call:**

Function: retrieve data from out of source.

Responsibilities:

* Integration with External Services: Establish and maintain connections to external APIs for data exchange and service interaction, enabling integration with third-party services or platforms.

**Cache:**

Function: Implementing caching mechanisms to improve response times and reduce server load.

Responsibilities:

* Data Storage: Store frequently accessed data in memory to improve retrieval speed and reduce latency for users and applications.
* Read and Write Operations: Optimize read and write operations to the cache, ensuring that data is quickly available for requests while maintaining consistency with the primary data source.

**Response Aggregation:**

Function: Response aggregation will gather all responses from these sources, organize and format the data, and then return it to the user as a single, easy-to-understand result.

Responsibilities:

* Combine information: Combine information from multi retrieval sources into one response and return it to the user, this reduces the number api call and response completely for the user.
* Optimizer system: By minimizing the number of calls over the network, this can reduce bandwidth and resource costs .

**Logging:**

Function: Logging helps store information about request, response, error and other events and everything related to data, so that it can be traced back when we need it.

Responsibilities:

* Request Logging: Record details of each incoming request, including timestamps, request method, URL, headers, and body (if necessary).
* Response Logging: Log information about the response sent back to the client, such as status codes, response times, and content if required.
* Error Logging: Capture and store details about any errors or exceptions that occur, including error messages, stack traces, and the context in which they happened.
* Event Logging: Track significant events within the system, such as authentication successes or failures, user activity, and configuration changes.

**Monitor:**

Func: assesses the performance and health of the system to ensure all components are operating as expected, facilitating quick detection of issues, performance degradation

Responsibilities:

* Health checks: Regularly check the status of services and components to confirm they are operational.
* Create alerts based on predefined thresholds for performance metrics, ensuring prompt notification of issues.

# **3. Technical**

API gateway: AWS, Kong.

Auth: JWT.

Backend microservices: Python, Javascript

Database: MySQL for data retrieval and Mongodb for logging.

Monitor: Prometheus.

External API call: RESTful.

Cache: Redis.

# **4. Response**

**Functional :**

* User multi query: web, app mobile.
* Real time update.
* Multiprocessing query real time.
* Handle query smoothly.
* Provide a view for all previous orders.

**Non Functional :**

* Security
* High availability
* Low latency
* High consistency
* convenient