**Interview Introduction**

"Hello, my name is **Yogesh Sanap**, and I'm thrilled to have the opportunity to discuss my experiences as a Java backend developer with you.

Over the past four years, I've been deeply immersed in developing robust applications using Java/J2EE technologies, particularly leveraging the Spring framework. I've extensively utilized components like **Spring Core, Spring Cloud, Spring Security, Spring Data JPA, and Spring Boot** to not only meet business requirements but also enhance functionality.

One aspect I'm particularly proud of my proficiency in implementing RESTful web services using **JSON**, ensuring **efficient communication between client and server components**. Additionally, I've keen understanding of design patterns such as **Singleton, Strategy, Factory, Abstract Factory, Builder, Façade, Proxy, Chain of responsibility** to improve code structure, scalability and reusability.

My proficiency isn't limited to backend technologies alone. I have extensive **hands-on experience in Core Java, adeptly utilizing the Collections API, Java 8, and SOLID principles**. Moreover, I possess substantial expertise in **monolithic** applications and hold a proficient understanding of **Microservices architecture**.

In terms of development methodologies, I've engaged in the entire **software development life cycle**, from requirement analysis and design to testing, troubleshooting and debugging.

Collaboration is key in any development environment, and I've had the privilege of working closely with cross-functional teams in Agile settings. Tools like **Jira and Git** have been instrumental in our **project management and version control processes.**

My experience extends beyond just coding. I've **managed MySQL** databases, **ensuring schema design** aligns precisely with application requirements while adhering to fundamental ACID properties for transactional reliability.

Documentation and testing are paramount in ensuring quality deliverables. I've documented **APIs using Swagger**, ensuring clear and comprehensive **documentation for developers and stakeholders.** Additionally, I've implemented unit tests using **JUNIT** to verify functionality and behavior, ensuring reliability and stability of the codebase.

Furthermore, my experience extends to working with various tools and technologies, including **Redis for efficient data storage and retrieval, Zipkin for distributed tracing and performance monitoring**, and **AWS Cloud** Services such as Amazon EC2 and S3.

In summary, my four years of experience as a Java backend developer have equipped me with a diverse skill set and a proven track record of delivering high-quality, scalable solutions. I'm excited about the opportunity to bring my expertise to [Company Name] and contribute to its success."

**Project Description - MNX Global Logistics**

"The MNX Global Logistics software project aims to develop a comprehensive logistics management solution tailored specifically for MNX Global, a leading provider of time-critical transportation and logistics services. The software will integrate advanced features such as real-time shipment tracking, route optimization, warehouse management, and customizable reporting tools. By leveraging cutting-edge technology and industry best practices, the project seeks to enhance MNX Global's operational efficiency, optimize resource utilization, and deliver exceptional service to their clients worldwide."

**Project Modules Names :-**

* **Order Management: ( I was working on this module )**
* **Route Planning and Optimization**
* **Carrier Management**
* **Vehicle Management:**

**Project microservices:-**

* **Freight Booking ( I was working on this microservice )**

**Entities ;**

**Shipment(** Contains details such as origin, destination, cargo information, and special requirements.**)**

**Carrier – (**Contains information about the carrier's services, capabilities, and contact details.**)**

**Booking – (**Contains details such as the shipment ID, carrier ID, pickup and delivery dates, and any special instructions.**)**

**Tracking – (**Contains details such as the current location of the cargo, expected delivery time, and any status updates.**)**

**Customer – (**Contains information about the customer's contact details, billing address, and any preferences or requirements.**)**

**Location -** **(**Contains details such as the address, coordinates, and any relevant contact information.**)**

* **Cargo Consolidation and Containerization**
* **Port Operations**
* **Customs Clearance**
* **Vessel Selection and Chartering**
* **Routing and Scheduling:**
* **Risk Management**
* **Monitoring and Tracking**

**API’s in Freight Booking Microservice:-**

* **Create Shipment:**
  + Endpoint: **POST /shipments**
  + Method: Create a new shipment with details such as origin, destination, cargo information, and any special requirements.
* **Get Shipment Details:**
  + Endpoint: **GET /shipments/{shipment\_id}**
  + Method: Retrieve detailed information about a specific shipment using its unique identifier.
* **Update Shipment:**
  + Endpoint: **PUT /shipments/{shipment\_id}**
  + Method: Update the details of an existing shipment, such as changing the delivery date or modifying cargo information.
* **Cancel Shipment:**
  + Endpoint: **DELETE /shipments/{shipment\_id}**
  + Method: Cancel a booked shipment, removing it from the system.
* **List Shipments:**
  + Endpoint: **GET /shipments**
  + Method: Retrieve a list of all shipments, possibly with filtering options (e.g., by status, date, or location).
* **Quote and Pricing:**
  + Endpoint: **POST /quotes**
  + Method: Obtain a price quote for a shipment based on the provided details.
* **Booking Confirmation:**
  + Endpoint: **POST /bookings**
  + Method: Confirm the booking of a shipment, indicating that it is ready for transportation.
* **Tracking Information:**
  + Endpoint: **GET /tracking/{tracking\_id}**
  + Method: Retrieve real-time tracking information for a shipment using its tracking identifier.
* **Carrier Information:**
  + Endpoint: **GET /carriers/{carrier\_id}**
  + Method: Retrieve information about a specific carrier, including their services, capabilities, and contact details.
* **Authentication:**
  + Endpoint: **POST /authenticate**
  + Method: Authenticate users or systems to ensure secure access to the freight booking API.

**Exception occurred in project : -**

* **BeanCreationException:** This exception typically occurs during the initialization of Spring's application context when it encounters an error while trying to create or initialize a bean.

**occur due to** - Circular dependencies

**how to handle** – For carefully review the bean definitions and associated configuration to identify and resolve any errors.

* **NoSuchBeanDefinitionException:** This exception occurs when you attempt to retrieve a bean from the Spring application context, but Spring cannot find a bean definition matching the specified bean name or type.

**occur due to** - Typo in bean name , Incorrect bean scope,

**how to handle** – ensure that the bean name or type used for retrieval matches the defined bean names and types in the application context.

**Design Patterns**

1. **Singleton Pattern**:
   * Use Case: When you want to ensure that a class has only one instance and provide a global point of access to that instance.
   * Example: Database connection pool, logger classes, configuration settings.
2. **Strategy Pattern**:
   * Use Case: When you want to define a family of algorithms, encapsulate each one, and make them interchangeable.
   * Example: Payment processing with different strategies for payment gateways, sorting algorithms in a system with various sorting strategies.
3. **Factory Pattern**:
   * Use Case: When you want to create objects without exposing the instantiation logic and allow subclasses to alter the type of objects that will be created.
   * Example: Creating instances of different database drivers (MySQL, PostgreSQL, etc.) through a common interface without exposing the instantiation details.
4. **Builder Pattern**:
   * Use Case: When you want to construct complex objects step by step and provide a clear separation between the construction and representation.
   * Example: Building SQL queries dynamically, creating complex objects with many optional parameters such as GUI components.
5. **Facade Pattern**:
   * Use Case: When you want to provide a unified interface to a set of interfaces in a subsystem, thus simplifying the usage for clients.
   * Example: Providing a simplified interface to a complex subsystem such as an API for a third-party service.
6. **Proxy Pattern**:
   * Use Case: When you want to control access to an object, provide additional functionality, or defer the instantiation of the object until it is needed.
   * Example: Virtual proxy for lazy loading of large objects, security proxy for access control, remote proxy for accessing objects on remote servers.
7. **Chain of Responsibility Pattern**:
   * Use Case: When you want to avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request.
   * Example: Request processing pipelines, logging systems with multiple loggers handling different log levels.