Node.js Senior Developer Challenge

Objective

Develop a backend application using **Node.js** that manages an order system and processes events asynchronously. The solution must expose RESTful endpoints and be fully functional, testable via **Postman**.

Technical Requirements

1. **Database**: Use **MongoDB** to store order information and their statuses.
2. **Cache**: Implement **Redis** to accelerate queries for recent orders.
3. **Messaging**: Use **RabbitMQ** to manage asynchronous event processing related to orders.
4. **Logging and Search**: Store logs in **Elasticsearch** for easier event tracking and debugging.
5. **Containerization**: Provide a **docker-compose.yml** file to launch the entire solution (service and dependencies) with a single command.

Functionality

1. Create an Order

- **Endpoint**: `POST /orders`

Input:
json
Copy code

```
{
  "userId": "12345",
  "products": [
    { "productId": "p1", "quantity": 2 },
    { "productId": "p2", "quantity": 1 }
  ]
}
```

-

Output (Example):
json
Copy code

```
{
  "orderId": "o123",
  "status": "PENDING"
```

```
}
```

-
- **Action**:
    - Save the order in **MongoDB** with the status `PENDING`.
    - Publish a message to **RabbitMQ** for a worker to process the order.

---

2. Process an Order

- Implement a **worker** that consumes messages from a RabbitMQ queue:
    - Change the order status to `PROCESSING`, and then to either `COMPLETED` or `FAILED`.
    - Log the processing details into **Elasticsearch**.

---

3. Retrieve an Order

- **Endpoint**: `GET /orders/:orderId`

Output:
json
Copy code

```
{
  "orderId": "o123",
  "userId": "12345",
  "products": [
    { "productId": "p1", "quantity": 2 },
    { "productId": "p2", "quantity": 1 }
  ],
  "status": "COMPLETED"
}
```

-
- **Action**:
    - Query **Redis** first to check for the order.
    - If not found in Redis, fetch it from **MongoDB** and store the result in Redis.

---

4. Order Logs

- **Endpoint**: `GET /logs/orders/:orderId`
- **Output**:
    - Return processing logs for the order from **Elasticsearch**.

Deliverables

1. Source code of the project.
2. A `docker-compose.yml` file to launch:
    - The Node.js application.
    - Instances of **RabbitMQ**, **Redis**, **MongoDB**, and **Elasticsearch**.
3. A `README.md` file with:
    - Instructions on how to run the project.
    - Description of the endpoints.
    - Example requests (e.g., Postman collections).

Evaluation Criteria

1. **Code Structure**:
    - Use of best practices (Clean Code, Clean Architecture).
    - Modularization and proper separation of concerns.
2. **Technology Usage**:
    - Efficient and functional implementation of RabbitMQ, Redis, MongoDB, and Elasticsearch.
3. **Documentation**:
    - Clear and detailed `README.md` file.
4. **Functionality**:
    - Correct behavior of the endpoints.
    - Proper error handling.
5. **Performance**:
    - Efficient use of Redis for caching queries.
    - Asynchronous handling with RabbitMQ.
6. **Containerization**:
    - The project should be easy to launch and test on any machine.