Gojek

techical questions
- problem 1
- problem 2
- problem 3
- problem 4
- problem 5
- problem 6
- problem 7
- problem 8
- problem 9

bar rasier(final round)
- problem 1
- problem 2
- problem 3(low priority)
- problem 4
- problem 5
- problem 6(low priority)
- problem 7
- problem 8(not yet)
- problem 9(not yet)
- problem 10(not yet)
- problem 11
- problem 12
- problem 13
- problem 14(not yet)
- problem 15(not yet)
- problem 16(not yet)
- problem 17

techical questions

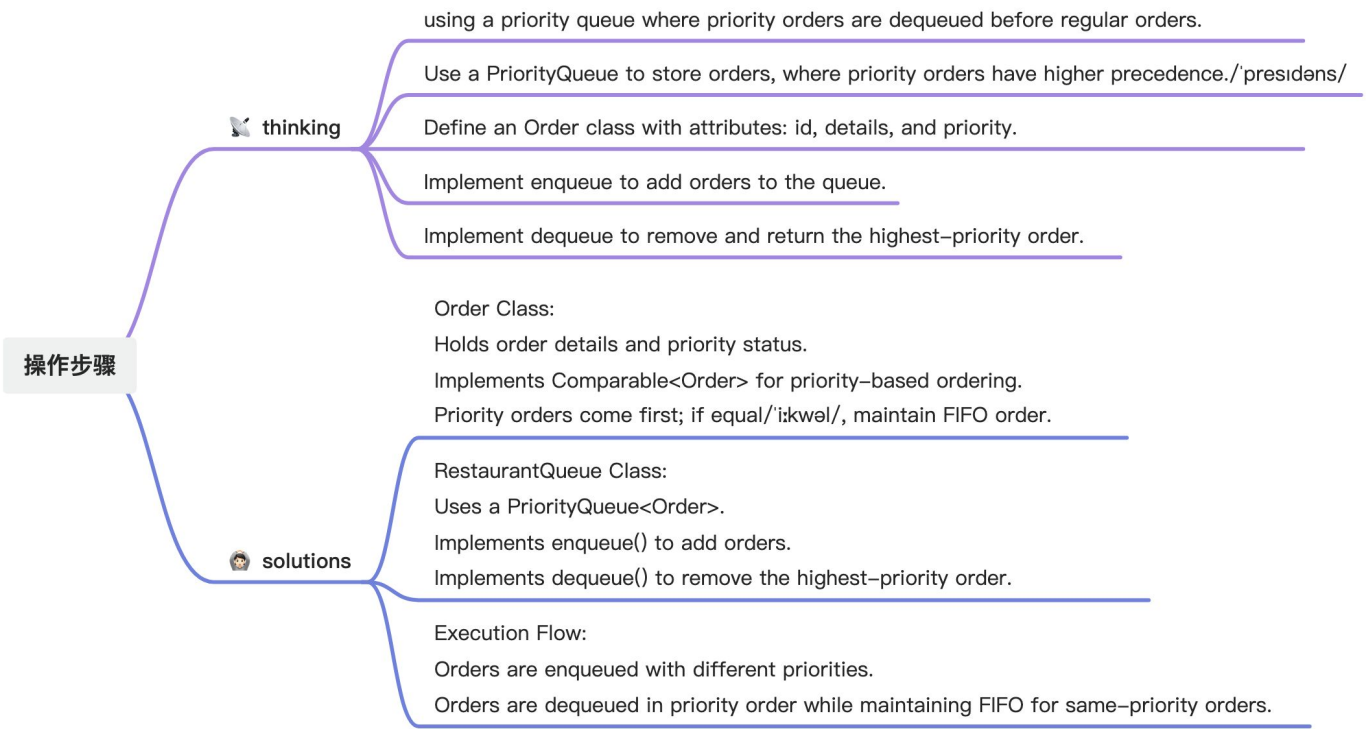
problem 1

> Problem 1: “Design a queue system for restaurant orders where some orders are marked priority. Implement enqueue and dequeue operations.”
>
> - Difficulty: Medium–Hard
> - Topics: Queue Implementation, Priority Queue
> - Time Limit: 45 minutes

Java

```java
import java.util.PriorityQueue;

class Order implements Comparable<Order> {
    private static int counter = 0; // Auto-incrementing ID
    private final int id;
    private final String details;
    private final boolean isPriority;

    public Order(String details, boolean isPriority) {
        this.id = counter++;
        this.details = details;
        this.isPriority = isPriority;
    }

    public int getId() {
        return id;
    }

    public String getDetails() {
        return details;
    }

    public boolean isPriority() {
        return isPriority;
    }

    @Override
    public int compareTo(Order o) {
        // Priority orders should come first; if same, maintain FIFO order
        if (this.isPriority && !o.isPriority) return -1;
        if (!this.isPriority && o.isPriority) return 1;
        return Integer.compare(this.id, o.id);
    }

    @Override
    public String toString() {
        return "Order ID: " + id + ", Details: " + details + ", Priority: " + isPriority;
    }
}

class RestaurantQueue {
    private final PriorityQueue<Order> queue;

```

```
44      public RestaurantQueue() {
45          this.queue = new PriorityQueue<>();
46      }
47
48      public void enqueue(String details, boolean isPriority) {
49          queue.offer(new Order(details, isPriority));
50      }
51
52      public Order dequeue() {
53          return queue.poll();
54      }
55
56      public boolean isEmpty() {
57          return queue.isEmpty();
58      }
59  }
60
61  public class RestaurantOrderSystem {
62      public static void main(String[] args) {
63          RestaurantQueue orderQueue = new RestaurantQueue();
64
65          orderQueue.enqueue("Burger", false);
66          orderQueue.enqueue("Pasta", true);
67          orderQueue.enqueue("Pizza", false);
68          orderQueue.enqueue("Steak", true);
69
70          while (!orderQueue.isEmpty()) {
71              System.out.println(orderQueue.dequeue());
72          }
73      }
74  }
75
```



problem 2

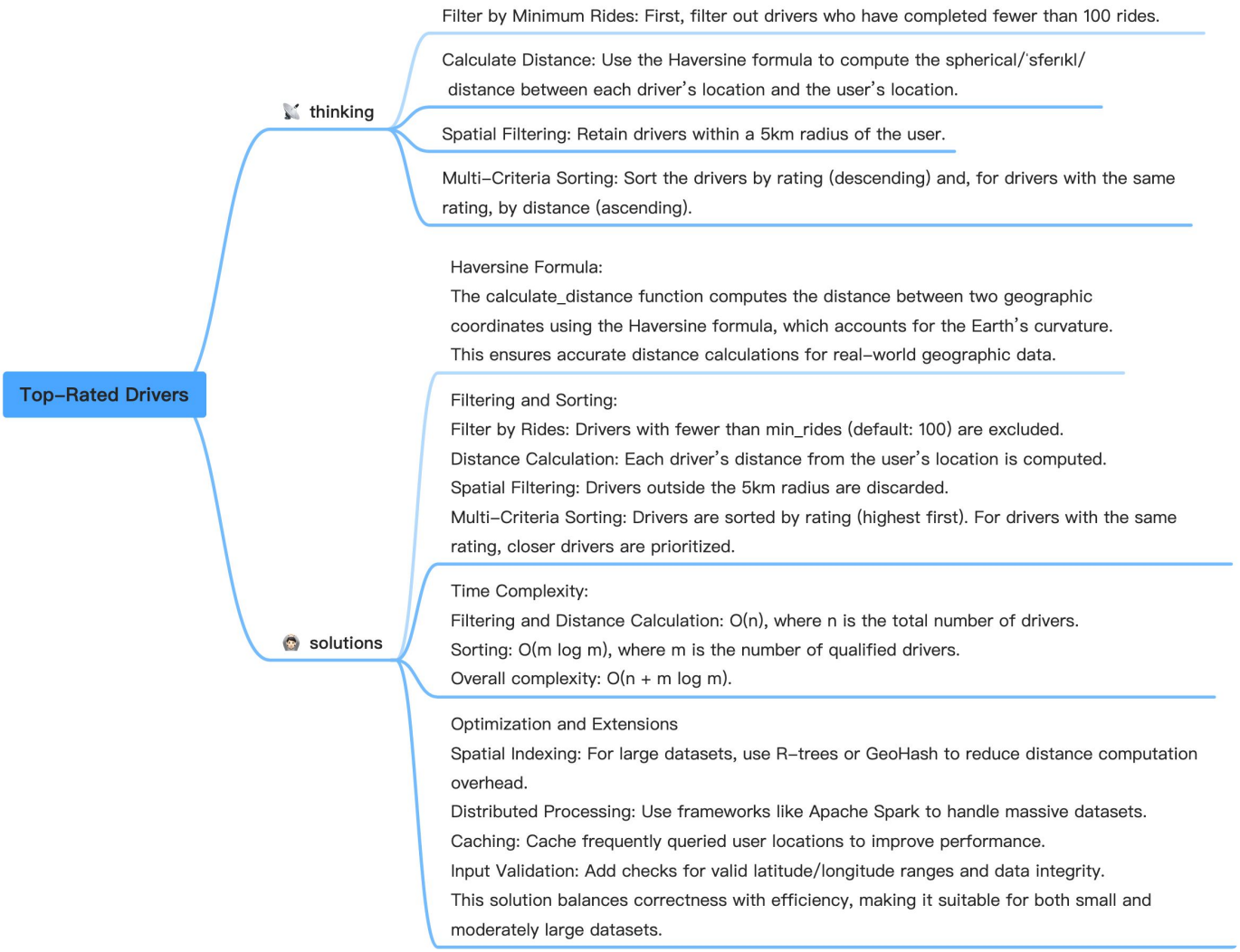
> Problem 2: “Given a list of driver locations and their ratings/ˈreɪtɪŋ/, find the top-rated drivers within a 5km radius/ˈreɪdiəs/ who have completed at least 100 rides.”

- Difficulty: Medium
- Topics: Sorting, Spatial/ˈspeɪʃl/ Data Structures
- Time Limit: 45 minutes

Python

```
Driver class:
    String id;
    double latitude;
    double longitude;
    double rating;
    int completedRides;

import math

def calculate_distance(lat1: float, lon1: float, lat2: float, lon2: float) -> float:
    """Calculate the distance between two points using the Haversine formula (in kilometers)."""
    R = 6371  # Earth's average radius in kilometers
    # Convert degrees to radians
    lat1_rad = math.radians(lat1)
    lon1_rad = math.radians(lon1)
    lat2_rad = math.radians(lat2)
    lon2_rad = math.radians(lon2)

    # Differences in coordinates
    dlat = lat2_rad - lat1_rad
    dlon = lon2_rad - lon1_rad

    # Haversine formula components
    a = math.sin(dlat / 2) ** 2 + math.cos(lat1_rad) * math.cos(lat2_rad) * math.sin(dlon / 2) ** 2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    return R * c

def find_top_drivers(
    user_location: tuple[float, float],
    drivers: list[dict],
    min_rides: int = 100,
    radius_km: float = 5
) -> list[dict]:
    """Find and return the list of top-rated drivers within a specified radius who meet the minimum ride count."""
    user_lat, user_lon = user_location
    qualified_drivers = []

    # Step 1: Filter drivers with at least 'min_rides'
    for driver in drivers:
        if driver.get("rides", 0) >= min_rides:
```

```
            # Step 2: Calculate distance to user
            distance = calculate_distance(
                user_lat, user_lon,
                driver["lat"], driver["lon"]
            )
            # Step 3: Keep drivers within the radius
            if distance <= radius_km:
                # Copy driver data and add distance for sorting
                driver_copy = driver.copy()
                driver_copy["distance"] = distance
                qualified_drivers.append(driver_copy)

    # Step 4: Sort by rating (descending), then distance (ascending)
    qualified_drivers.sort(key=lambda x: (-x["rating"], x["distance"]))

    # Remove the temporary 'distance' field (optional)
    for driver in qualified_drivers:
        driver.pop("distance", None)

    return qualified_drivers


# Sample Input
drivers = [
    {"lat": 40.7128, "lon": -74.0060, "rating": 4.9, "rides": 150},
    {"lat": 40.7130, "lon": -74.0061, "rating": 4.9, "rides": 200},
    {"lat": 40.7580, "lon": -73.9855, "rating": 4.8, "rides": 120},
    {"lat": 40.7128, "lon": -74.0160, "rating": 5.0, "rides": 90},
]

# User location (New York City center)
user_loc = (40.7128, -74.0060)

# Execute the function
result = find_top_drivers(user_loc, drivers)

# Expected Output:
# 1. Driver 1 (rides=150, rating=4.9, distance=0 km)
# 2. Driver 2 (rides=200, rating=4.9, distance≈0.11 km)
```


Top-Rated Drivers
thinking
Filter by Minimum Rides: First, filter out drivers who have completed fewer than 100 rides.
Calculate Distance: Use the Haversine formula to compute the spherical/ˈsferɪkl/ distance between each driver's location and the user's location.
Spatial Filtering: Retain drivers within a 5km radius of the user.
Multi-Criteria Sorting: Sort the drivers by rating (descending) and, for drivers with the same rating, by distance (ascending).
solutions
Haversine Formula:
The calculate_distance function computes the distance between two geographic coordinates using the Haversine formula, which accounts for the Earth's curvature.
This ensures accurate distance calculations for real-world geographic data.
Filtering and Sorting:
Filter by Rides: Drivers with fewer than min_rides (default: 100) are excluded.
Distance Calculation: Each driver's distance from the user's location is computed.
Spatial Filtering: Drivers outside the 5km radius are discarded.
Multi-Criteria Sorting: Drivers are sorted by rating (highest first). For drivers with the same rating, closer drivers are prioritized.
Time Complexity:
Filtering and Distance Calculation: O(n), where n is the total number of drivers.
Sorting: O(m log m), where m is the number of qualified drivers.
Overall complexity: O(n + m log m).
Optimization and Extensions
Spatial Indexing: For large datasets, use R-trees or GeoHash to reduce distance computation overhead.
Distributed Processing: Use frameworks like Apache Spark to handle massive datasets.
Caching: Cache frequently queried user locations to improve performance.
Input Validation: Add checks for valid latitude/longitude ranges and data integrity.
This solution balances correctness with efficiency, making it suitable for both small and moderately large datasets.

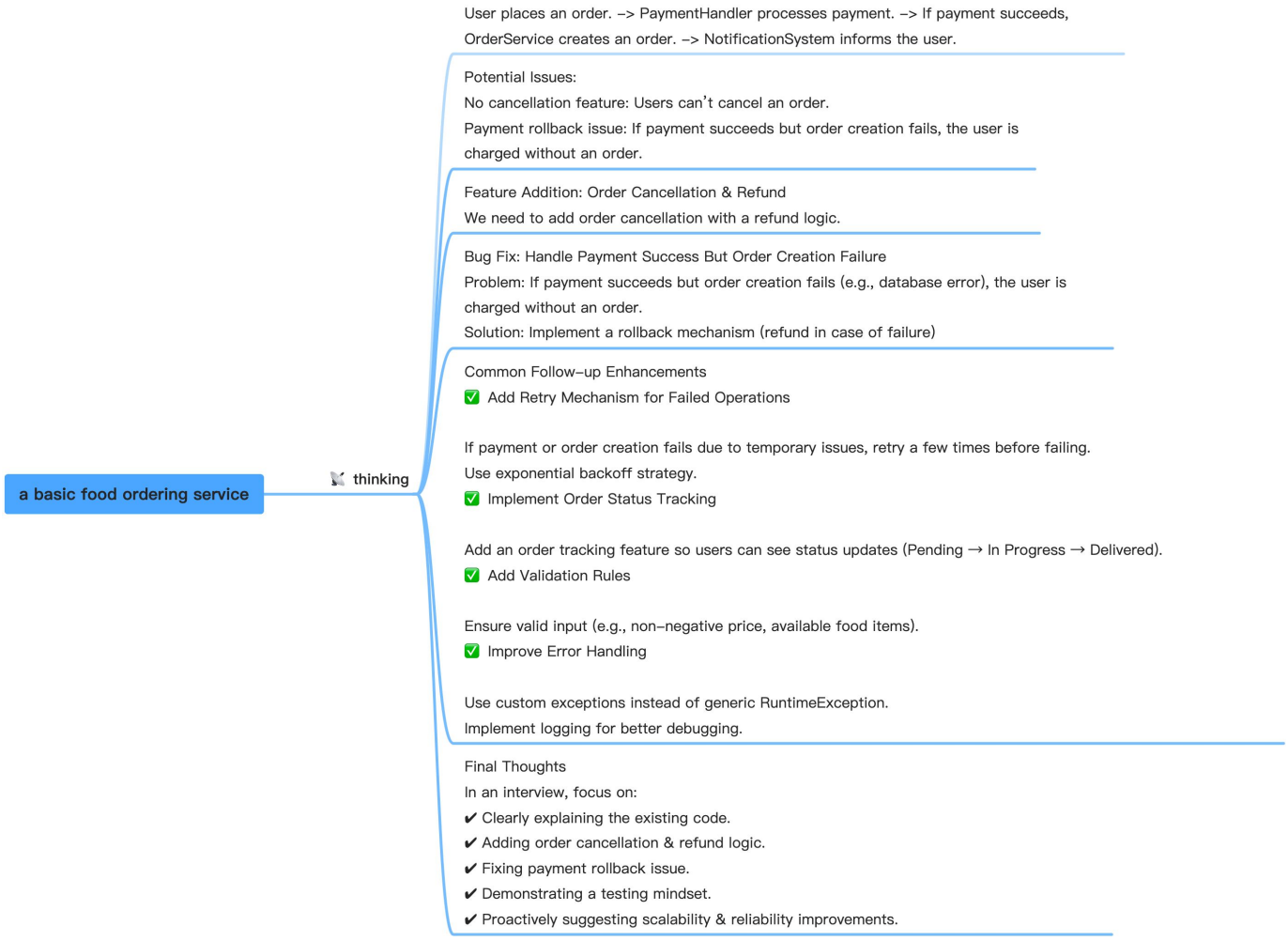
problem 3

```
Given a basic food ordering service code to review and enhance:

Base Program Structure:
- Order Service: Handles order creation and status management
- Payment Handler: Basic payment processing
- Notification System: Simple order updates
Tasks During Interview:

Code Walkthrough: Explain the existing implementation
Feature Addition: Add order cancellation with refund logic
Bug Fix: Handle edge case where payment succeeds but order creation fails
Testing: Write test cases for the new feature
What’s Evaluated:

Code comprehension
Problem-solving approach
Testing mindset
Communication skills
Common Follow-up Tasks:

Add retry mechanism for failed operations
Implement order status tracking
Add validation rules
Improve error handling
```

Java

```java
import java.util.*;

class Order {
    enum Status { PENDING, COMPLETED, CANCELED }

    String orderId;
    String userId;
    String item;
    double amount;
    Status status;

    public Order(String orderId, String userId, String item, double amount) {
        this.orderId = orderId;
        this.userId = userId;
        this.item = item;
        this.amount = amount;
        this.status = Status.PENDING;
    }

    public void setStatus(Status status) {
        this.status = status;
    }
}

class OrderService {
    private PaymentHandler paymentHandler;
    private NotificationSystem notificationSystem;
    private Map<String, Order> orders = new HashMap<>();

    public OrderService(PaymentHandler paymentHandler, NotificationSystem notificationSystem) {
        this.paymentHandler = paymentHandler;
        this.notificationSystem = notificationSystem;
    }

    public Order createOrder(String userId, String item, double amount) {
        String orderId = UUID.randomUUID().toString();
        boolean paymentSuccess = paymentHandler.processPayment(userId, amount);

        if (paymentSuccess) {
            Order order = new Order(orderId, userId, item, amount);
            orders.put(orderId, order);
```

```
            notificationSystem.sendNotification(userId, "Order placed successfully.");
            return order;
        } else {
            throw new RuntimeException("Payment failed. Order not created.");
        }
    }

    public boolean cancelOrder(String orderId) {
        Order order = orders.get(orderId);
        if (order != null && order.status == Order.Status.PENDING) {
            paymentHandler.refund(order.userId, order.amount);
            order.setStatus(Order.Status.CANCELED);
            notificationSystem.sendNotification(order.userId, "Order canceled. Refund issued.");
            return true;
        }
        return false;
    }
}

public Order createOrder(String userId, String item, double amount) {
    String orderId = UUID.randomUUID().toString();

    boolean paymentSuccess = paymentHandler.processPayment(userId, amount);
    if (!paymentSuccess) {
        throw new RuntimeException("Payment failed. Order not created.");
    }

    try {
        Order order = new Order(orderId, userId, item, amount);
        orders.put(orderId, order);
        notificationSystem.sendNotification(userId, "Order placed successfully.");
        return order;
    } catch (Exception e) {
        paymentHandler.refund(userId, amount); // Refund the payment
        throw new RuntimeException("Order creation failed. Payment refunded.");
    }
}

```



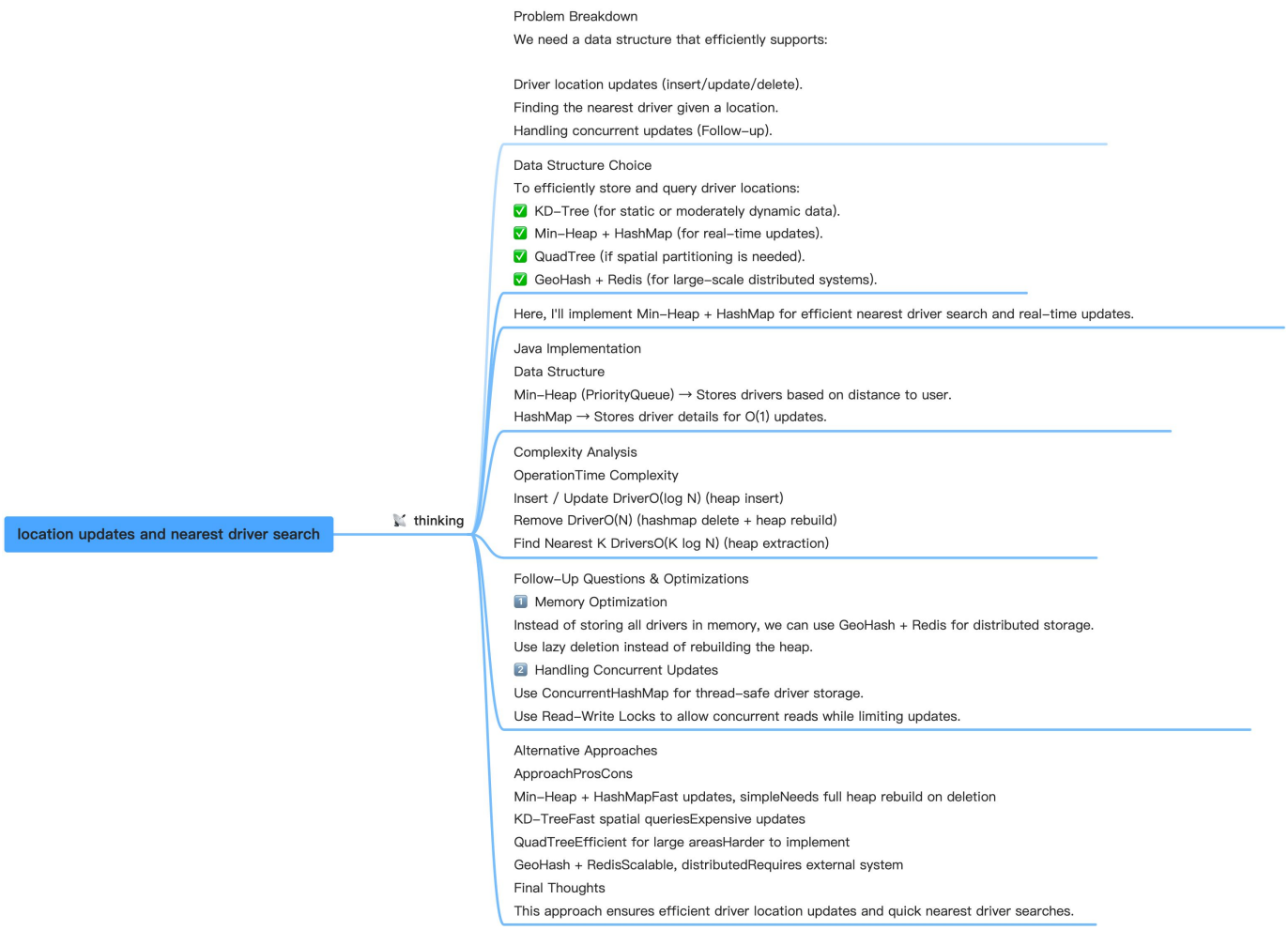
problem 4

> DSA Question: "Design a data structure for efficient driver location updates and nearest driver search."
>
> - Topics: Heap
> - Follow-ups: Memory optimization, concurrent updates

Java

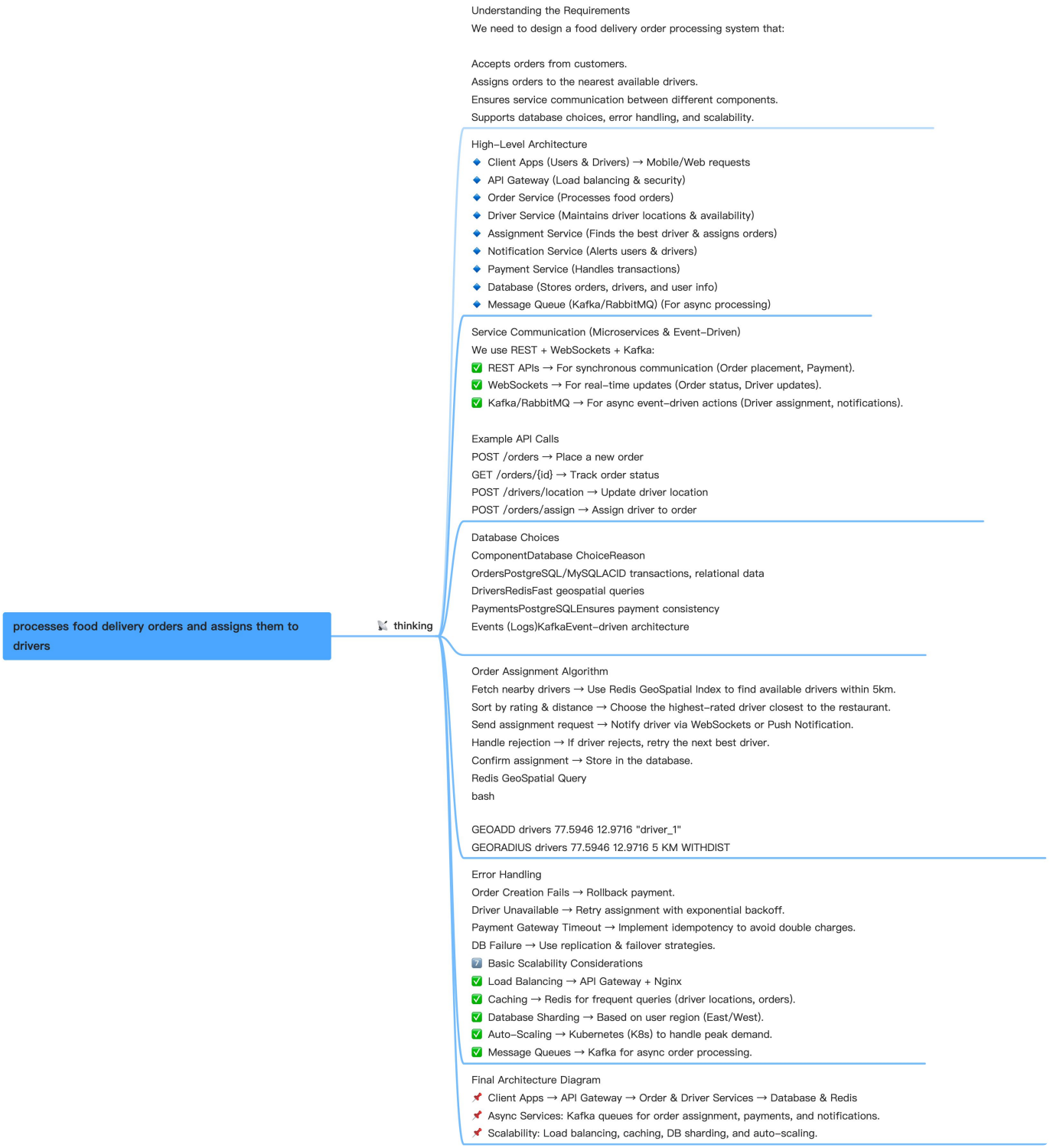
```java
import java.util.*;

class Driver {
    String id;
    double latitude;
    double longitude;

    public Driver(String id, double latitude, double longitude) {
        this.id = id;
        this.latitude = latitude;
        this.longitude = longitude;
    }
}

class DriverLocator {
    private Map<String, Driver> driverMap = new HashMap<>();
    private PriorityQueue<Driver> minHeap;
    private double userLat, userLon;

    public DriverLocator(double userLat, double userLon) {
        this.userLat = userLat;
        this.userLon = userLon;
        this.minHeap = new PriorityQueue<>(Comparator.comparingDouble(this::haversineDistance));
    }

    // Haversine formula to calculate distance between two points (lat/lon)
    private double haversineDistance(Driver driver) {
        final double R = 6371.0; // Earth radius in KM
        double dLat = Math.toRadians(driver.latitude - userLat);
        double dLon = Math.toRadians(driver.longitude - userLon);
        double a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
                   Math.cos(Math.toRadians(userLat)) * Math.cos(Math.toRadians(driver.latitude)) *
                   Math.sin(dLon / 2) * Math.sin(dLon / 2);
        double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
        return R * c;
    }

    // Add or Update a Driver's Location
    public void updateDriver(String id, double lat, double lon) {
        Driver driver = new Driver(id, lat, lon);
        driverMap.put(id, driver);
```

```
        minHeap.add(driver);
    }

    // Remove a Driver
    public void removeDriver(String id) {
        driverMap.remove(id);
        rebuildHeap();
    }

    // Get Nearest N Drivers
    public List<Driver> getNearestDrivers(int k) {
        List<Driver> result = new ArrayList<>();
        PriorityQueue<Driver> tempHeap = new PriorityQueue<>(minHeap);
        while (!tempHeap.isEmpty() && k-- > 0) {
            result.add(tempHeap.poll());
        }
        return result;
    }

    // Rebuild Heap (Needed after removals)
    private void rebuildHeap() {
        minHeap.clear();
        minHeap.addAll(driverMap.values());
    }

    public static void main(String[] args) {
        DriverLocator locator = new DriverLocator(12.9716, 77.5946);
        locator.updateDriver("D1", 12.9656, 77.5836);
        locator.updateDriver("D2", 12.9721, 77.6012);
        locator.updateDriver("D3", 12.9500, 77.5800);
        locator.updateDriver("D4", 12.9740, 77.6090);

        List<Driver> nearest = locator.getNearestDrivers(2);
        System.out.println("Nearest Drivers:");
        for (Driver d : nearest) {
            System.out.println("Driver ID: " + d.id);
        }
    }
}
```



problem 5

> System Design Question: “Design a service that processes food delivery orders and assigns them to drivers”
>
> Key Points –
>
> - Service communication
> - Database choices
> - Error handling
> - Basic scalability



problem 6

System Design Deep Dive (60 minutes)

"Design Gojek's wallet system handling transactions across services"

Cross-Service Transactions

🛰️ thinking

Understanding the Problem
Gojek's wallet system must:
✓ Handle transactions (payments, transfers, refunds) across multiple services.
✓ Ensure consistency (ACID for financial transactions).
✓ Be highly available & scalable (support millions of users).
✓ Prevent fraud (double spending, unauthorized access).
✓ Support service communication (between merchants, drivers, and users).

High-Level Architecture
Key Components
User Service — Manages user accounts & wallet details.
Transaction Service — Processes transactions (debit, credit, refunds).
Balance Service — Maintains real-time wallet balance.
Ledger Service — Immutable record of all transactions.
Fraud Detection Service — Monitors & flags suspicious activities.
Notification Service — Sends payment updates to users.
Settlement & Reconciliation Service — Ensures correct settlements between banks, merchants, and Gojek.
📌 Communication Style: Event-driven (Kafka) + Synchronous (REST/gRPC) APIs.

Database & Storage Choices
ComponentDatabaseJustification
User AccountsPostgreSQL / MySQLStrong consistency for user profiles.
Wallet BalancesRedis / DynamoDBFast reads/writes with eventual consistency.
Transaction HistoryCassandra / PostgreSQLImmutable, high-write throughput.
Ledger (Audit Logs)Apache Kafka + S3Immutable event store for all transactions.
Fraud DetectionElasticsearchReal-time anomaly detection.

API Design
1. User Wallet APIs
POST /wallets/{userId}/topup
POST /wallets/{userId}/withdraw
GET /wallets/{userId}/balance

2. Transaction APIs
POST /transactions/pay
POST /transactions/refund
GET /transactions/{txnId}

3. Ledger APIs
GET /ledger/{userId}

Transaction Flow & ACID Guarantees
📌 Steps for a Wallet Payment
User initiates payment → Calls POST /transactions/pay.
Balance Check → Ensure user has sufficient funds (via Redis).
Transaction Locking → Use a distributed lock (e.g., Redisson in Redis) to prevent double spending.
Deduct Balance → Update balance in Redis & trigger an event to Kafka.
Persist Transaction → Store it in PostgreSQL & Ledger Service.
Notify User & Merchant → Send success notification.
Eventual Consistency → Reconcile & confirm transaction via Kafka.

Error Handling & Edge Cases
✅ Double Spending Prevention
Use Redis-based locking (SETNX) to prevent multiple transactions on the same balance.
Use Idempotency Keys to ensure the same request is not processed twice.
✅ Transaction Failures (Partial Commit)
Distributed Transactions (Saga Pattern):
Step 1: Debit Wallet → Step 2: Credit Merchant → Step 3: Mark as Success.
If Step 2 fails, rollback Step 1 (refund user wallet).
✅ Eventual Consistency Handling
If Kafka fails to publish a transaction, retry using a Dead Letter Queue (DLQ).
Use background reconciliation jobs to check inconsistencies.

Scaling the System
✔ Horizontal Scaling
Use API Gateway + Load Balancers for high traffic.
Scale Wallet, Transaction, and Ledger services independently.
✔ Distributed Caching (Redis)
Cache wallet balance to reduce DB load.
Expiry-based invalidation to sync with DB.
✔ Event-Driven Architecture (Kafka)
Kafka ensures asynchronous transaction processing for scale.
Multiple consumers (fraud detection, reconciliation, analytics).
✔ Sharding & Partitioning
Shard Transactions by User ID (PostgreSQL/Cassandra).
Partition Kafka Topics by User ID to parallelize processing.
8️⃣ Security & Fraud Prevention

✅ Multi-Factor Authentication (MFA) for high-value transactions.
✅ Rate Limiting & API Throttling to prevent abuse.
✅ Anomaly Detection System (Elasticsearch) to detect fraudulent transactions.
✅ End-to-End Encryption for transaction payloads.

9️⃣ Final Architecture Diagram
Client Apps → API Gateway → Wallet Service, Transaction Service, Ledger Service → Redis, PostgreSQL, Kafka, Elasticsearch

💡 Final Thoughts
This design ensures:
✔ High Availability (via caching & event-driven processing).
✔ Scalability (via microservices, sharding, and Kafka).
✔ Security & Fraud Prevention (via MFA, encryption, anomaly detection).
✔ Consistency (via Redis locks, Kafka, & eventual reconciliation).

problem 7

> Architecture Round (60 minutes) Focus areas:
>
> Microservices patterns
>
> API design principles
>
> Data consistency
>
> Failure handling

Architecture Round

thinking

Key Focus Areas
Microservices Patterns — Design principles, communication, service boundaries.
API Design Principles — RESTful/gRPC, versioning, authentication.
Data Consistency — Transactions, distributed consistency models.
Failure Handling — Resilience, retries, circuit breakers.

Microservices Architecture Patterns
1.1 Service Decomposition
Domain-Driven Design (DDD) to define bounded contexts.
Split services based on business capabilities:
Order Service → Manages food orders.
Payment Service → Handles transactions.
Driver Service → Manages driver availability.
Notification Service → Sends SMS/push/email.
1.2 Communication Patterns
Synchronous (REST/gRPC): For critical API calls (e.g., placing an order).
Asynchronous (Kafka/RabbitMQ): For event-driven flows (e.g., order status updates).
1.3 Service Registry & Discovery
Service Mesh (Istio, Consul, Eureka) for automatic service discovery.
API Gateway to expose services externally & handle authentication.

API Design Principles
2.1 RESTful API Best Practices
✔ Use Resource-Oriented URLs

yaml
Copy
Edit
GET /orders/{orderId}
POST /payments
✔ HTTP Status Codes for Error Handling

Status CodeMeaning
200 OKSuccess
201 CreatedResource created
400 Bad RequestValidation failure
404 Not FoundResource not found
500 Internal Server ErrorUnexpected failure
✔ Versioning for Backward Compatibility

Use URL-based (/v1/orders) or Header-based (Accept-Version: v1).
✔ Pagination & Filtering for Large Datasets

yaml
Copy
Edit
GET /orders?limit=10&offset=20&status=delivered
2.2 gRPC for Internal Microservice Communication
Faster & efficient than REST for inter-service communication.
Supports strong typing & contract enforcement using Protocol Buffers.
Example gRPC Definition (order.proto):

proto
Copy
Edit
service OrderService {
  rpc CreateOrder (OrderRequest) returns (OrderResponse);
}

Data Consistency in Microservices
3.1 Strong vs. Eventual Consistency
ApproachUse Case
ACID (Strong Consistency)Payments, transaction processing
Eventual Consistency (BASE)Order status updates, analytics
3.2 Distributed Transactions Handling
✅ Saga Pattern (Compensating transactions)

Choreography → Each service listens for events & reacts accordingly.

Orchestration → Centralized service (e.g., Order Orchestrator) coordinates steps.
✅ Outbox Pattern (Ensuring atomicity in event publishing)

Store event in DB before publishing to Kafka to avoid inconsistencies.
3.3 Example: Order Creation with Saga
Order Service → Creates order in pending state.
Payment Service → Deducts amount, emits event (payment_success).
Order Service → Marks order as confirmed.
If Payment Fails → Compensation workflow cancels order.
4️⃣ Failure Handling & Resilience
4.1 Retry Mechanisms
Exponential Backoff (Increase retry interval on failure).
Idempotency Keys (Prevent duplicate requests).
4.2 Circuit Breaker Pattern
Use Netflix Hystrix or Resilience4j to prevent cascading failures.
If a service is down, fallback logic kicks in.
Example Circuit Breaker in Spring Boot:

java
Copy
Edit
@Retryable(value = {RemoteServiceException.class}, maxAttempts = 3)
public PaymentResponse processPayment(PaymentRequest request) {
    return paymentClient.charge(request);
}
4.3 Dead Letter Queues (DLQ)
Failed Kafka messages are sent to a DLQ for manual inspection & retry.
📌 Final Architecture
API Gateway → Routes traffic, handles security (JWT, OAuth).
Microservices → Order, Payment, Notification, Driver.
Event Bus (Kafka) → Asynchronous communication.
Databases
Orders → PostgreSQL
Payments → MySQL
Drivers → Redis (Geospatial Queries)
Events → Kafka + S3
Resilience Features
Circuit Breakers, Retries, Timeouts
Dead Letter Queues (DLQ) for failed events
💡 Final Thoughts
✅ Microservices Best Practices — DDD, Service Discovery, Event-driven.
✅ API Design — RESTful, gRPC, Pagination, Versioning.
✅ Data Consistency — Sagas, Outbox, Eventual vs. Strong Consistency.
✅ Failure Handling — Circuit Breakers, Retries, Dead Letter Queues.

problem 8

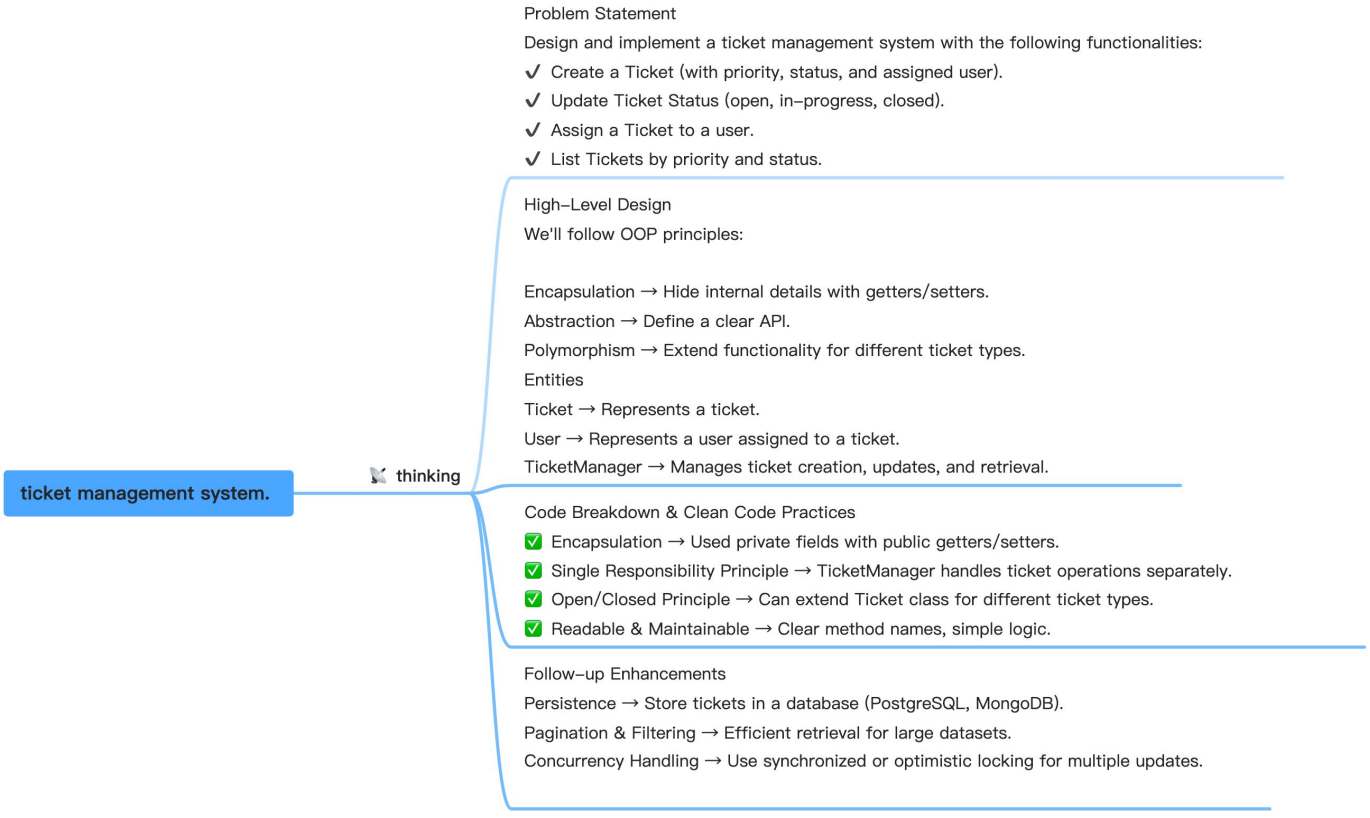
> Problem-solving on a task like building a ticket management system. Tips: Showcase clean code, OOP principles, and clear communication.

Java

```java
import java.util.*;

enum Status {
    OPEN, IN_PROGRESS, CLOSED
}

class Ticket {
    private static int idCounter = 1;
    private final int ticketId;
    private String title;
    private String description;
    private int priority;
    private Status status;
    private User assignedUser;

    public Ticket(String title, String description, int priority) {
        this.ticketId = idCounter++;
        this.title = title;
        this.description = description;
        this.priority = priority;
        this.status = Status.OPEN;
    }

    public int getTicketId() { return ticketId; }
    public Status getStatus() { return status; }
    public int getPriority() { return priority; }
    public User getAssignedUser() { return assignedUser; }

    public void assignUser(User user) {
        this.assignedUser = user;
    }

    public void updateStatus(Status status) {
        this.status = status;
    }

    @Override
    public String toString() {
        return "Ticket{" +
        "id=" + ticketId +
        ", title='" + title + '\'' +
        ", priority=" + priority +
        ", status=" + status +

```

```
            ", assignedTo=" + (assignedUser != null ? assignedUser.getName()
: "None") +
            '}';
    }
}

class User {
    private final int userId;
    private final String name;

    public User(int userId, String name) {
        this.userId = userId;
        this.name = name;
    }

    public String getName() { return name; }
}

class TicketManager {
    private final List<Ticket> tickets = new ArrayList<>();

    public Ticket createTicket(String title, String description, int prio
rity) {
        Ticket ticket = new Ticket(title, description, priority);
        tickets.add(ticket);
        return ticket;
    }

    public void assignTicket(int ticketId, User user) {
        for (Ticket ticket : tickets) {
            if (ticket.getTicketId() == ticketId) {
                ticket.assignUser(user);
                return;
            }
        }
        System.out.println("Ticket not found!");
    }

    public void updateTicketStatus(int ticketId, Status status) {
        for (Ticket ticket : tickets) {
            if (ticket.getTicketId() == ticketId) {
                ticket.updateStatus(status);
                return;
            }
        }
        System.out.println("Ticket not found!");
    }
```

```java
    public List<Ticket> listTicketsByPriority() {
        tickets.sort(Comparator.comparingInt(Ticket::getPriority).reversed());
        return tickets;
    }
}

public class TicketSystem {
    public static void main(String[] args) {
        TicketManager manager = new TicketManager();
        User alice = new User(1, "Alice");
        User bob = new User(2, "Bob");

        Ticket t1 = manager.createTicket("Bug in UI", "Fix button alignment", 2);
        Ticket t2 = manager.createTicket("Server Crash", "High CPU usage", 1);

        manager.assignTicket(t1.getTicketId(), alice);
        manager.updateTicketStatus(t2.getTicketId(), Status.IN_PROGRESS);

        System.out.println("All Tickets:");
        manager.listTicketsByPriority().forEach(System.out::println);
    }
}
```



problem 9

> Problem-solving on a task like building a ticket management system. Tips: Showcase clean code, OOP principles, and clear communication.

bookmyshow

bar rasier(final round)

> This round was focused on leadership and team collaboration, assessing the candidate's ability to mentor junior engineers and manage cross-functional teams.

problem 1

> Tell us about a time you led a project. How did you manage technical debt and deliver the project on time?
>
> tell me about a challenging project that you are responsible for?

This question sought to understand how the candidate balances technical excellence with real-world constraints.

Plain Text

Describe a challenging project that you worked on. Why was it challengin g? What was your role in the project? How did you deal with the various di fficulties of the project?

So throughout my software engineering career, I've certainly worked on a l ot of challenging projects. One that comes to mind here is a particularly challenging project that related to the machine learning platform that I c urrently working on.

I’m the main developer on our team—the one with the most experience. So, n aturally, the lead handed this project over to me and put me in charge of writing the core functionality. We’ve got two other backend devs on the te am: one is an intern who doesn’t have much work experience yet, and the ot her has less experience than I do. In the past, I’ve always been the go-t o person for leading development tasks, so it makes sense that the lead tr usted me to take the reins on this project. Besides us backend folks, ther e are also two frontend developers and a tester. On the backend, our job i s mainly focused on server-side development.

there are a lot of reasons for it to be challenging. First of all, it was a large project. The surface area of the project was two times, that of an y other project I'd worked on. I worked on a back-end team, and so a good way to quantify that was how many new APIs I need to design does a projec t involve, for front-end, that is how many new pages that involved, And th is project involved basically twice the amount of existing pages we had o n our UI. So it was a very large project. which means, the more pages, th e more APIs I need to design.

On top of that, we had very little time to complete this project. So not o nly was it large, but we had very little time to put things into perspecti ve. Most previous projects that I had worked on, we had at least two quart ers/ˈkwɔːrtər/, maybe two and a half quarters, to work on them. Here, thi s project was announced to us at the very end of Q4 in 2023, so like earl y December, and we had to launch the product or the feature at the end of Q1 or early Q2 of 2024. So we basically only had like a quarter, a little over a quarter, to complete this project.

And then the final thing that was really hard about this project was the f act that we were only three, three and a half, because we had an intern co ming in on my team. And with only three and a half engineers, we would nev er have been able to do this project, or three engineers if you don't coun t the intern who hadn't even joined the team when we first heard about thi s project. So we were going to need other back-end engineers. And it so ha ppened that Our lead temporarily brought in someone from another team to s

upport us. He’s based in a different city, so all our communication with h
im has been online. and he was going to join us. But that meant that we ha
d to onboard a brand new team member whom we had never met before and not
familiar, So a lot of complications, a very, very challenging project.

My role in the project was to lead it, basically acting as the project lea
d or tech lead. The way I approached it was by first scoping out the entir
e project. Whenever I attended meetings to learn more about the project—wh
ether it was from the product managers or other people involved—I made sur
e to take notes on everything I heard. Then, I started mapping out the wor
k we’d need to do. I took a ton of notes and put together this huge runnin
g document that basically had everything we needed to know about the featu
re. I also made it a point throughout the week to ask every single questio
n I thought might come up later on.

You know, I kind of pulled from my experience on previous projects, rememb
ered what are things that you tend to run into, as an engineer, when you'r
e developing a feature. So I figured, let's just clarify all those things
right now, that way we have full visibility into everything. And it turne
d out to be pretty successful.

And so, when it came time to actually distribute the work and get starte
d, what I did was try to break things down as granularly as possible. I ta
lked with the front-end developers, and for each page, we split it into sm
aller modules. For each module, we figured out how many APIs we needed to
design and defined the data formats that should be returned. It was all ab
out logically separating the sub-features of the entire feature. Anything
that required a lot of context from our codebase, I assigned to the engine
ers who had been working on it for a long time. And anything that didn’t n
eed as much context, I gave to the engineers who were newer to the codebas
e.

Then I also leveraged/'levərɪdʒ/ the fact that I was hosting an intern/ɪn
'tɜːrn , 'ɪntɜːrn/. I mentioned that before. Actually, you never want to g
ive an intern like critical work. we picked the sub features that were les
s critical. That way we were kind of killing two birds with one stone, yo
u know, providing my intern with a great experience and a good project, bu
t also a project that happened to be relevant to this, you know, greater c
hallenging project.

And then otherwise, apart from that, So I did all of the kind of administr
ative work/əd'mɪnɪstreɪtɪv wɜːrk/ and the work of communicating requiremen
ts and progress and everything across all the teams. I made the sacrifice
of being the person communicating with every deveopers, you know, in a wee
kly meeting. So I would stay pretty late at the office and make sure every
thing goes well.

In the end, the project was a complete success. I believe the key to our success was the reasonable planning we put into the project, as well as the decisive prioritization of tasks that seemed unlikely to be completed on time. We also set clear success metrics, identifying which tasks were essential for the project's success and which were secondary (such as the tasks I assigned to the interns). This clarity in planning allowed the project to progress very smoothly, ultimately becoming a highly successful case. It's fair to say that this is one of the projects I'm most proud of in my career.

在我的软件工程职业生涯中，我确实参与了许多具有挑战性的项目。其中一个让我印象深刻的项目，是与我目前正在开发的机器学习平台相关的特别困难的项目。

我是团队中的主要开发者——也是经验最丰富的一个。所以很自然地，领导把这个项目交给了我，并让我负责编写核心功能。我们团队还有另外两名后端开发人员：一个是实习生，没有什么工作经验；另一个虽然有一定经验，但比我资历浅。在过去的工作中，我一直都是负责主导开发任务的人，所以领导信任我来负责这个项目是很合理的。除了我们后端团队，还有两名前端开发人员和一名测试人员。在后端方面，我们的工作主要集中在服务器端开发。

这个项目之所以特别具有挑战性，有很多原因。首先，这是一个非常庞大的项目。项目的规模是我之前参与过的任何项目的两倍。我在后端团队工作，一个很好的衡量方法是看需要设计多少个新的 API，而对于前端来说，则是看需要开发多少个新页面。而这个项目涉及的页面数量基本上是我们现有 UI 页面的两倍。因此，这是一个非常大的项目，这意味着页面越多，需要设计的 API 也就越多。

除此之外，我们完成这个项目的时间非常有限。不仅项目规模大，而且我们几乎没有时间去从容应对。我之前参与的大多数项目，通常至少有两个季度，甚至两个半季度的时间来完成。然而，这个项目是在 2023 年第四季度末（大约是 12 月初）宣布的，而我们必须在 2024 年第一季度末或第二季度初推出产品或功能。也就是说，我们只有大约一个季度多一点的时间来完成这个项目。

最后，这个项目最具挑战性的一点是，我们团队人手不足。当时团队里只有三个人，或者说是三个人半，因为我们有一个实习生即将加入。即使算上实习生，我们也只有三个半工程师，而在项目刚宣布时，实习生甚至还没有正式加入团队。光靠我们这几个人，根本无法完成这个项目。所以我们需要其他后端工程师的支持。碰巧的是，领导临时从另一个团队调派了一名工程师来协助我们。他来自另一个城市，所以我们只能通过线上沟通合作。尽管他会加入我们，但这意味着我们需要快速让一个完全陌生的新成员融入团队，而这带来了许多复杂性。这是一个非常、非常具有挑战性的项目。

我在项目中的角色是领导整个项目，基本上就是担任项目负责人或技术负责人。我的处理方式是从全面界定项目范围开始。在每次参加会议时，无论是从产品经理还是其他相关人员那里了解更多项目信息，我都会把听到的内容记录下来。然后，我开始规划我们需要完成的工作范围。我做了大量的笔记，并整理出了一份巨大的文档，里面基本包含了关于这个功能的所有信息。我还特别注意在每周的工作中，尽可能提前提出所有我认为未来可能会遇到的问题。

你知道，我借鉴了之前项目的经验，回想了一下作为一名工程师在开发功能时经常会遇到哪些问题。于是我想，不如现在就把这些问题都澄清清楚，这样我们就能对所有情况一目了然。结果证明，这种

方法实际上非常成功。

当真正需要分配工作并开始执行的时候，我所做的就是尽可能做到细致入微。我和前端开发人员沟通，针对每个页面，我们将其划分为更小的模块。对于每个模块，我们明确了需要设计多少个 API，并定义了需要返回的数据格式。这其实就是将整个功能逻辑上分解为子功能模块。任何需要大量代码库上下文的任务，我都分配给了那些长期参与代码库工作的工程师；而那些不需要太多上下文的任务，我则交给了对代码库不太熟悉的新人。

此外，我还利用了实习生的存在（我之前提到过）。实际上，你永远不应该把关键任务交给实习生。我们挑选了一些不太重要的子功能交给实习生。这样一来，我们可以说是一举两得：既为实习生提供了一个良好的体验和适合他的项目，同时也让他参与到这个更大的、具有挑战性的项目中。

除此之外，我还承担了所有的行政类工作，以及在各个团队之间沟通需求、进度和其他事项的任务。我主动承担了与每位开发者沟通的工作，比如在每周的会议上。我会经常在办公室待到很晚，确保一切顺利进行。

最终，这个项目取得了圆满成功。我认为成功的关键在于我们对项目进行了合理的规划，并对那些看起来无法按时完成的任务进行了果断的优先级调整。我们还设定了明确的成功指标，明确了哪些任务是项目成功所必须完成的，哪些是次要的（例如我分配给实习生的任务）。这种清晰的规划使得项目进展得非常顺利，最终成为了一个非常成功的案例。可以说，这是我职业生涯中最让我自豪的项目之一。

problem 2

> How do you approach code reviews for a senior team member?

The candidate needed to show how they handle sensitive situations in code reviews, focusing on mentoring rather than just pointing out mistakes.

Plain Text

```
Here’s a concise English summary of the approach to code reviews for senio
r team members, focusing on constructive feedback and mentorship:

---

### **Approach to Code Reviews for Senior Team Members**
1. **Focus on Collaboration, Not Criticism/ˈkrɪtɪsɪzəm/**
   - **Objective**: Emphasize improving code quality and reducing risks, n
ot judging individual ability.
   - **Example**: "Let’s ensure this code aligns with team standards and o
ptimizes performance, not just meet basic requirements."

2. **Use Constructive Feedback Frameworks**
   - **Structure Feedback**:
     - **Observation**: Highlight specific code patterns (e.g., "This loo
p makes three redundant database calls").
     - **Impact**: Explain potential issues (e.g., "Risk of performance bo
ttlenecks or data inconsistency").
     - **Suggestion**: Offer actionable improvements (e.g., "Extract a hel
per method or add caching").
   - **Balance Critique/krɪˈtiːk/ with Praise**: Start with认可 (e.g., "Th
e logic here is well-structured and highly readable") before suggesting ch
anges.

3. **Respect Experience and Guide Thinking**
   - **Ask Questions, Don’t Lecture**:
     - "Have we considered concurrency risks in this high-traffic scenari
o?"
     - "How flexible is this design if we need to extend the API later?"
   - **Share Context and Data**:
     - Reference team guidelines, performance metrics, or past issues (e.
g., "Similar patterns caused a 5% error rate last week").

4. **Leverage Automation**
   - **Streamline Routine Checks**: Use tools (e.g., linters, formatters)
to handle style or syntax issues automatically.
   - **Focus on Critical Issues**: Prioritize complex logic, architectur
e, or risk mitigation over fixable syntax errors.

5. **Follow-Up and Knowledge Sharing**
   - **Summarize Learnings**:
     - "This optimization improved maintainability—should we document thi
s as a best practice?"

```

```
   - **Maintain Relationships**: Regularly check in with seniors to ensure feedback is constructive and valued.

---

### **Example Scenario**
**Issue**: Unhandled exceptions in critical code.
- **Feedback**:
  "The `fetchData()` call here (line 23) lacks error handling. If the API fails, users might see a blank screen."
  "Last week's logs show similar issues caused 5% errors. Adding a `try-catch` block and friendly error messaging could help."
- **Guided Question**:
  "Do you think this approach fits the current design, or should we explore alternatives?"

---

### **Key Takeaways**
- **Mentorship Over Micromanagement**: Treat reviews as collaborative learning opportunities.
- **Empower Ownership**: Encourage seniors to reflect on their choices while providing data-driven guidance.
- **Sustain Team Morale**: Foster a culture where feedback is seen as growth-oriented, not personal.

This approach balances respect for senior experience with constructive inp
```

~~problem 3(low priority)~~

> Have you worked in an Agile environment? How do you ensure that backend development aligns with product goals in a fast-paced setting?

This question tested the candidate's experience working in Agile sprints and collaborating with product and front-end teams.

Notes: 💡 The candidate gave detailed examples of mentoring junior developers and handling high-pressure situations. They highlighted their experience in leading sprints and managing client expectations.

Plain Text

```
Im Young,
```

problem 4

> How do you handle situations where you disagree with a colleague about a technical solution?

The interviewer wanted to see the candidate's communication style, their ability to collaborate, and resolve conflicts in a professional manner.

Plain Text

Describe a time when there was a conflict within your team.

How did you help resolve the conflict?

Did you do anything to prevent it in the future?

So as a matter of fact, there was a time when there was a conflict within my team.

And basically, my team had been working on a very large project, a huge feature, and we were getting ready to launch it in production environment.

We were about two weeks away from the release day, maybe even one week away,

And the important thing to note here is that right around a week before launching, called a code freeze.Basically, any code that's merged into the main repository, the main branch, release branch, will not get pushed or deployed to production for about two weeks until after launching.
This is meant to ensure that there are no bugs in production during the production because there's so many teams launching so many features.

And so we were about a little over a week from the release day when the code freeze was going to happen.I think we were one day away from the code freeze.Now, we had finished our feature. Everything was ready. We were ready to ship,We were very excited and it had been a pretty stressful time leading up to it since it was such a big project.

But one of the other engineers on my team sent out a pull request with a lot of code changes and one file.And these code changes didn't add any functionality. They were not, you know, logic changes.So they were purely syntactic, purely stylistic, But so he sent out this pull request and the other engineer on my team was reviewing the pull request and just basically like flat out rejected it, you know, left a comment, a pretty quick comment, like, "We're not doing this right now," and just rejected it.

And I remember this led to a pretty like tense argument. basically these two other engineers, like the one who sent the pull request and the one who reviewed it, just weren't happy with each other and, few comments were exchanged and it just wasn't good.

Clearly there was a conflict bottling up and rumbling.

And so I'll tell you how I handled that situation.

But first, I guess I'll tell you just how I handle any conflict at work, especially.

Two things that I think about.First of all, when there's a conflict, I always try to assume good intent in the other party or in this case in the two parties involved in the conflict.

Most of the time at work, when someone is disagreeing with somebody else, when someone has a firm position and a firm disagreement with somebody else, they have good intent.Typically, they just genuinely think that, you know, their way is the right way for themselves, for the team, for the company, for the organization.

And so it's important to assume good intent and not bad intent.

The person that you're arguing with or that you're having a conflict with is not out to get you or to screw you over, so to speak.They're just they're they're just disagreeing with you or having a conflict with you because they want what's best for the team.

That's number one.

The second thing is always try to find common ground when there's a conflict because more often than not, since both since both people or every party involved in a conflict have good intent, there is some sort of common ground.

And typically like both parties want the same thing.They just have different ways of getting to it or of accomplishing it.And so here what I did is I grabbed, you know, both of the engineers and pulled them in a room, obviously, metaphorically speaking. We went in a conference room and I kind of walked them through this mental exercise of let's try to assume good intent in each other and let's try to figure out what was the actual intent of both of your actions.

And it turns out, you know, the intent of the person who sent the pull request was just to improve the code base, right, to make the code base better because this codes was kind of objectively poorly written. Everybody agreed that on that. Important.This is where we get to the common ground. And that was that was that person's intent.

The intent of the other person was also good.It was just let's minimize the risk of introducing a bug into production right before the launching when we will not be able to fix it afterwards because once the code freez

e happens, you cannot, merge any new code or deploy any new code to production.

And so both of them clearly had good intent and they suddenly saw that once we kind of calmed down and talked it out. And then both of them found common ground in the sense that they both cared about improving the code base.They both agreed that the change that one person made was important just not necessarily at that time. And they also both didn't want the launch to be ruined by a bug.

And they agreed that, you know, not merging the pull request at that time was better to prevent the bug, but merging it right after a few days later when the code freeze had begun and, that that code would no longer be deployed to production.That would be fine.

So suddenly, you know, it turns out like they were in agreement and the conflict could have just been avoided. And so as far as what I did to try to help them, you know, prevent such a conflict from happening in the future and the lesson that I took from it because I could have been part of that conflict was just number one, always go through this mental exercise.

These two things, Assume good intent and try to find common ground.

And also the second thing is try to make sure that your communication is on point because this conflict that I'm describing here could have been avoided by just better communication.

If the engineer who rejected the pull request had just said, "Hey, I agree with this change. I think this is great. Thanks so much for doing this, but I really think right now is not the good time to merge it into the main repository because, you know, the code freeze is about to happen and we have to ensure that there's absolutely no risk of a bug getting into production."

If he had said that, then, you know, communicated more properly, then the conflict could have maybe been avoided.

So those were the two things that we kind of all left that conference room with.

And that's how I handled this conflict and how I would handle any other conflict in the workplace.

描述一个团队内部发生冲突的时刻

你是如何帮助解决这个冲突的?

你做了什么来防止它在未来再次发生?

事实上，确实有一次我的团队内部发生了冲突。

当时，我的团队正在开发一个非常庞大的项目，一个巨大的功能，并准备将其发布到生产环境中。

距离发布日期大约还有两周的时间，甚至可能更短，大概一周多一点。
这里需要注意的是，在发布前大约一周，公司会进行所谓的“代码冻结”。
也就是说，任何合并到主分支或发布分支的代码都不会在接下来的两周内被部署到生产环境中。这是为了确保在发布期间生产环境中不会出现任何错误，因为有太多团队在发布各种功能。

我们距离发布日期大约还有一周多一点的时间，正好是代码冻结即将开始的时候。我记得我们离代码冻结只有一天了。
那时，我们的功能已经完成了，一切都准备就绪，可以发布了。
我们非常兴奋，但之前的压力也非常大，因为这是一个非常大的项目。

然而，团队中的一位工程师发出了一个包含大量代码更改的拉取请求（pull request），涉及一个文件。
这些代码更改并没有添加任何新功能，也没有涉及逻辑上的改动。
它们纯粹是语法和风格上的调整。
于是，他发出了这个拉取请求，而团队中的另一位工程师负责审查这个请求，结果直接拒绝了它，并留下了一个简短的评论，比如“我们现在不能这样做”，然后就直接拒绝了。

我记得这件事引发了一场相当激烈的争论。发送拉取请求的工程师和审查它的工程师彼此都不满意，双方交换了几条评论，气氛变得非常糟糕。
显然，冲突正在酝酿并逐渐升级。

下面我来谈谈我是如何处理这种情况的。
不过在此之前，我想先分享一下我通常是如何处理工作中的任何冲突的。

处理冲突的两个关键点：
1. **假设对方怀有善意**：
    每当发生冲突时，我总是尽量假设对方怀有善意。在这个案例中，就是假设冲突双方都怀有善意。
    大多数时候在工作中，当有人与别人意见不一致、持有坚定立场或强烈反对某事时，他们的出发点通常是好的。
    他们通常只是真诚地认为自己的方式才是对团队、公司或组织最有利的。
    因此，重要的是要假设对方怀有善意，而不是恶意。
    和你争论或发生冲突的人并不是针对你，也不是想“害”你。
    他们只是与你意见不同或发生冲突，因为他们希望团队能取得最好的结果。

2. **寻找共同点**：
    第二件事是，始终尝试找到冲突双方的共同点。因为大多数情况下，既然双方都怀有善意，那么总会有一些共同点。
    通常，双方的目标其实是一致的，只是实现目标的方式不同而已。

具体做法：

```
107 当时，我把两位工程师叫到了一个会议室（当然，这里是比喻意义上的）。
    我引导他们进行了一次心理练习：让我们假设对方怀有善意，并试着弄清楚双方行为背后的真正意图
108 是什么。
109
    结果发现，发送拉取请求的工程师的意图很简单，就是想改进代码库，让代码变得更好，因为那段代
110 码确实写得很差，大家都同意这一点。这是重要的共同点。
111 这就是他的意图。
    而另一位工程师的意图也同样很好。他只是想尽量减少在发布前引入生产环境错误的风险，因为在代
    码冻结后，我们无法再修复这些问题。一旦代码冻结生效，就不能再合并或部署任何新代码到生产环
112 境中。
113
114 因此，显然两人都怀有善意，当我们冷静下来并深入讨论后，他们也意识到了这一点。
115 接着，他们在某种程度上找到了共同点：他们都关心改进代码库。
116 他们都同意那位工程师所做的更改很重要，只是不一定适合在那个时候进行。
117 同时，他们都不希望发布因错误而受到影响。
    他们达成共识，认为在当前时间点不合并拉取请求是为了避免潜在的错误，但在代码冻结开始几天
118 后，也就是代码不再会被部署到生产环境时，再合并是可以接受的。
119
120 最终，他们意识到彼此其实是站在同一阵线上的，这场冲突本可以避免。
121
122 **未来如何预防类似冲突：**
    至于我做了什么来帮助他们避免未来再次发生类似的冲突，以及我从中学到了什么（毕竟我也可能成
123 为冲突的一部分），主要有以下两点：
124
125 1. **始终进行心理练习**：
126     这就是我提到的两点——假设善意和寻找共同点。
127
128 2. **确保沟通清晰到位**：
129     我描述的这场冲突本来可以通过更好的沟通来避免。
        如果审查拉取请求的工程师当时说：“嘿，我同意这个更改，这很棒，非常感谢你这么做。但我
    真的认为现在不是合并到主分支的好时机，因为代码冻结即将开始，我们必须确保没有任何错误进入
130 生产环境。”
131     如果他这样说了，沟通得更清楚一些，那么这场冲突可能就不会发生了。
132
133 **总结：**
134 我们带着这两点共识离开了会议室：
135 1. 假设善意，寻找共同点。
136 2. 确保沟通清晰明确。
137
```

problem 5

> Tell me about a time you had to manage multiple competing priorities. How did you ensure deadlines were met?
>
> How do you prioritize tasks when multiple projects are competing for your attention?

This question assessed time management and the ability to prioritize tasks effectively.

Plain Text

Hey everybody, welcome to Algo Expert. In this video, we're going to answer the following behavioral interview question.

How would you go about distributing work for a project across a team of software engineers?

If you've led a project in the past, describe what you did.

So, I think that the first thing I would do, which is the first thing that you probably should do if you're trying to distribute work across a team of engineers, is to scope out the work.

It's very important for you to understand what it is exactly that you're building and to have a clear picture of the various logical chunks that the project or the work can be divided into that engineers on your team can then take ownership of.

So, you want to divide the work into these sensible chunks, and here it's kind of half art, half science, because you don't want to be too vague, like you don't want to just say, "Oh, the chunk of work is that we have to build an API that's kind of too vague."

But on the other hand, you probably don't want to go too deep or too granular.

For instance, if we're talking about front-end engineering and you're building, you know, a brand new product or feature that's got five different pages, and one of those pages is a form, you probably don't want to divide it into each individual input field.

That might be something for the engineer who's going to be taking charge of that page. That might be for them to do in their own design doc of that page or what have you.

So, point is, you want to scope out the work without being too deep or too, you know, far removed.

And here, my guideline typically would be, you know, if we're talking about back-end, maybe scope out the work with, you know, the various API endpoints that you're going to be building or the various, you know, logical pieces of functionality of a feature that you're going to be supporting.

For the front-end, maybe the different pages that you're going to be building out.

Once you've done that, you have to be able to balance a few things at the same time.

So, first of all, you have to make sure that the work gets done, right?

And so, work that is highly critical and that is very difficult has to be prioritized.

And oftentimes, it has to be assigned to the most capable person.

If you've got maybe a senior engineer on the team who would be most qualified to complete this very complicated part of the project and who would be the least risky person to assign this part of the project to, then that might just have to happen.

Now, this actually can go against one of the other things that you need to balance, which is people's happiness, right?

If you're on a team, you want to make sure that your engineers are happy.

And sometimes, you know, some people have preferences.

Maybe another engineer is going to say, "Hey, I would have really wanted to... I would have really liked to work on this part of the feature."

But you have to, you know, use your judgment to see if that would make sense.

And sometimes, those priorities, like the critical work, and assigning it to someone who's able to complete it, is going to, you know, take precedence over someone's preference of what they want to work on.

And then the last thing that you want to balance is your engineers' career trajectories, because... and this might be, I guess, more relevant for managers, but even as a peer, if you're, let's say, you know, the team lead of a project, or the project lead of a project, you want to look out for your peers, and you want to make sure that they're best equipped for their career, to grow in their career, and same for yourself, to be honest.

And so, if you've got one engineer on your team who really needs to focus on proving their technical... their ability to build out technically complex features, then ideally, you can assign to them technically complex features.

If you've got someone who needs to work a little bit more with other teams and other stakeholders, then maybe it might make sense to put them on a

part of the project that is going to involve, you know, meetings with back end engineers, or with product managers, or UX designers.

Point is, you have to balance all of these things, right?

The criticality, if that's a word, of parts of the project to ensure that stuff gets done, people's personal preferences, just to make sure that people are happy, and also people's career trajectories, and making sure that you're setting them up for success.

Now, one thing that is sometimes overlooked, that's also important to keep in mind, is just logistical issues.

So sometimes, you know, you'll have some people who have to travel at one point during the quarter, or some people who, for whatever reason, you know, have a commitment that they can't get out of, and that's going to prevent them from attending certain meetings, or they won't be able to attend meetings with other engineers who are in a different time zone at a specific time.

When you've got these logistical issues, you have to also keep them in mind in the way that you distribute the work.

If someone is not going to be able to attend meetings with engineers who are in a completely different time zone late in the day, then maybe, you know, somebody else should do the work that is going to require attending those meetings.

And so, everything that I'm describing here stems from, well, stems from a lot of different experiences, but I'm specifically thinking about one, which was the last project that I worked on when I was a software engineer at Google.

I was leading the project, I was kind of the tech lead of the project, and this was a very large project.

We had engineers in a different time zone in South Korea, so 14 hours ahead.

Lots of work, new engineers, existing engineers, people of different levels.

And so, I basically applied everything that I'm describing here.

You know, I scoped out the work, not too granular, but not too high level, assigned it in a way that made most sense.

```
So, parts of the project that didn't require too much context in the code base were given to some of these new engineers that were just joining the team.

Parts of the project that were very critical and that needed a lot of context were given to existing engineers who had already been on the team.

Even if they were not necessarily going to benefit from having those technically complex projects under their belt, maybe they would have been better given to a more junior engineer who would have really been able to make use of that, had to make some trade-offs there.

And then finally, the time zone thing.

I made sure to assign work in a way that made sense and respected people's preferences with when they like to work, whether or not they're going to be able to go to meetings with engineers in a different time zone, and
```

~~problem 6(low priority)~~

> Describe a project where you made a significant impact on the user experience. What steps did you take to ensure the product was user-centric?

This helped gauge the candidate's focus on user experience and their ability to optimize frontend performance.

Notes: 💡 The candidate shared several insightful stories about team dynamics, emphasizing their proactive approach to problem-solving and prioritizing user needs.

Plain Text

```
Im Young,
```

problem 7

> show your interest in GoTo

Plain Text

```
Why do you wanna work at insert company name?

So quick pause before I answer this question.

I'm gonna answer this question as if I were currently working at Google and applying to Facebook.

So the question would be, why do I wanna work at GoTo?

So I wanna work at GoTo for three main reasons.

The first one is from a purely technical point of view.

I feel that GoTo is a big tech company, works in a lot of technologies that really excite me, technologies that I've worked a lot on in personal projects. also I realy like interacting with people, especial foreigners, having a tendency to work in multinational company. yes Goto is a good choice.


So basically, all of or a lot of GoTo's technology and products really excites me and interests me.

Now, putting aside the technical stuff for a bit, I'm also really interested or excited about GoTo's culture. like

Three Zeros Commitments
1. Zero Emissions:Focusing on decarbonization efforts for direct and indirect emissions
2.Zero Waste:Accelerate the reusing, repurposing, recycling and processing of waste
3.Zero Barriers:Focuses on reducing barriers to socioecomonic growth for driver and merchant partners in our ecosystem


I've heard about the, I don't know if it's an unofficial or official motto, move fast and break things.

I don't know if that's kind of an old motto or a recent one or still applicable today, but the point is, I have heard that Facebook has a very unique culture of moving fast, and that's something that really excites me.

Having worked at Google for two years now, even though I've had incredible opportunities, I've worked with incredible people in a very engineering
```

```
excellence driven company, one thing that has been frustrating at times has been the slowness of some of the engineering projects I've worked on, and from a few peers I've known who worked at Google and now have gone to Facebook, apparently this difference in culture and the velocity with which you can ship out code and ship out projects, launch projects and features is pretty dramatic, the difference between the two companies.

And so I'm really excited about that at Facebook.

Finally, the third thing that is particularly attracting me to Facebook is Facebook's bootcamp experience.

So most of the other companies I've looked at, when you apply to them, you have to apply to a specific team, or for example at Google, you don't apply to a specific team in most cases, but once you pass the interviews, you go through their team matching process and you have to commit to a team before your first day on the job.

At Facebook, I've read a lot about the bootcamp experience and my recruiter told me about it, I'm really excited about being able to try out, actually work on multiple different teams before having to commit to one.

That's something that is incredibly unique, I haven't heard about another company doing that, it's something that really is appealing to me.

And on that note, at the end of the interview, if we have time, I'd love to ask you what your experience was with bootcamp, what you liked about it and how it worked
```

introduce GoTo

Plain Text

So what exactly is GoTo, and how did it get so big?

GoTo is the largest digital ecosystem in Indonesia, with a mission to 'empower progress' by offering technology infrastructure and solutions that help everyone to access and thrive in the digital economy. The GoTo ecosystem provides a wide range of services including mobility, food delivery, groceries and logistics, as well as payments, financial services, and technology solutions for merchants. The ecosystem also provides e-commerce services through Tokopedia and banking services through its partnership with Bank Jago.

GoTo Group is an Indonesian tech company formed in a blockbuster merger between two of the country’s largest start-ups: Gojek and Tokopedia.
The main product of the merger, a powerful super app you can use to book a ride,
buy something or make a payment, is today used by tens of millions.

From day one, we offered multiple different services, like a ride-hailing service or a delivery service.

There’s a saying that if you want to go fast, you go alone; if you want to go far, you go together. So GoTo, basically, is go far, go together.

In May 2021, Two of Indonesia’s largest tech start-ups have announced one of the biggest mergers ever in the world. These two are famously known as Gojek, a ride-hailing and payment services firm, Tokopedia, an e-commerce firm. The two giants made up what is now known as the GoTo Group and it is one of Southeast Asia’s most valuable start-ups ever. At the time of the merger, the company got a public big market valuation.

what GoTo really is. As mentioned earlier, it is a merger of Indonesia’s biggest tech giants which are Gojek and Tokopedia.

Gojek, the first company that is a multi-service platform with products and services such as ride-hailing, food delivery, and entertainment services, video streaming platform, and financial services. Gojek is one of the top companies that is changing the world. It is a company that operates in most of the countries in Southeast Asia and is offering a wide array of services.

Tokopedia is an e-commerce platform, fintech, logistics,
The Tokopedia Platform and fulfillment business. Like Gojek, Tokopedia is also a billion-dollar business.

```
22
23  These two together combined makeup one of  the biggest internet companies
    of Southeast Asia
24
```

problem 8(not yet)

> Topics covered:
>
> - Distributed systems
> - Caching strategies
> - Database design
> - System reliability
>
> System Design Components
>
> - Load balancing
> - Caching
> - Database scaling
> - API design

problem 9(not yet)

> **Behavioral Rounds:** Questions focus on leadership experiences, teamwork, and problem-solving in real-world scenarios.
>
> **Bar Raiser Round:** A final culture fit and strategic vision evaluation by senior team members.
>
> Questions like, “Explain a challenging project and how you managed it” aim to assess strategic thinking and cultural fit.
>
> Strategic decision-making in high-pressure situations.
>
> **Leadership and Management**
>
> 1. **Cross-Functional Team Leadership**:
>
> Describe a situation where you led a cross-functional team to deliver a complex data project, and how you managed stakeholder expectations.
>
> **Key Preparation:** Focus on communication, breaking down silos, and aligning technical execution with business goals.
>
> 2. **Mentorship**:

> How do you approach mentoring and developing junior data engineers on your team?
>
> **Key Preparation:** Share examples of mentoring, creating learning pathways, and fostering a culture of innovation.
>
> 3. **Difficult Technical Decisions**:
>
> Share an instance where you had to make a difficult technical decision regarding data infrastructure, and how you arrived at the solution.
>
> Key Preparation: Include a scenario involving trade-offs, risk assessment, and collaboration.
>
> 4. **Project Prioritization**:
>
> How do you prioritize data engineering projects based on business needs and technical considerations?
>
> Key Preparation: Discuss frameworks like RICE (Reach, Impact, Confidence, Effort) or weighted scoring models.
>
> **Cultural Fit**: Be ready for bar-raiser questions to assess your alignment with Gojek's values and collaborative culture.

problem 10(not yet)

> Tell me about a time when you faced a major technical challenge. How did you overcome it?

Notes: 💡 The candidate's responses highlighted their ability to handle complex situations, collaborate with teams, and maintain high standards in their work. They demonstrated a growth mindset, which is crucial for a company like Gojek that is always evolving.

problem 11

> Strong disagreement with your co-workers

Plain Text

Hey everybody, welcome to Algo Expert. In this video we're going to answer the following question.

Describe a time when you strongly disagreed with a co-worker about an engineering decision.

How did you go about making the final decision? What did you do after the decision was made?

So there have been many times when I've disagreed with co-workers. When I was at Google, when I was at Facebook, and especially now that I'm running Algo Expert full-time.

And oftentimes these have been pretty big disagreements. Especially now on Algo Expert because the engineering disagreements that we usually have have big product ramifications.

So they end up being product disagreements as well and even, you know, business disagreements, if that makes sense. Now I do think that it's less meaningful for me to focus on a particular disagreement that I've had as much as it is to focus on the general system or process that I follow when we have disagreements. So that's what I'm going to focus on here. And the process that I follow really has three steps. And they're actually kind of inspired by some of the leadership principles at Amazon, if you're familiar with those. So the first step, which is one of the Amazon leadership principles, I think, is to dig deep. To really try to understand every person who's part of the disagreement, all of their ideas, their arguments, and their stances.

You have to understand why someone you're disagreeing with, why they came to the conclusion that they came to. And they have to understand why and how you came to your conclusion. As an example, let's say someone wants to change a piece of code and you don't want to change that piece of code, you disagree with their change. You have to understand why are they changing that piece of code. Is it because they think it'll be better? Okay, probably. But why is it going to be better? Is it because it'll be stylistically better? Okay. Is it because it'll be more performant? Okay, that's another option. Is it because it'll be something that they just prefer? You know, they have a personal preference. Is it because it'll be more consistent with the code base?

You see, there are a lot of different reasons for which someone might think a piece of code is going to be better. You have to understand that. And then once you get to that answer, you continue asking why. You continue di

ssecting the idea. Well, why do we want to be consistent with the rest of our code base? Why do we want to be more performant? Do we want to be more performant? And again, here in this trivial example, this might seem kind of silly, but in a larger, you know, feature or in a larger decision or disagreement, this might make a lot of sense because someone might have a completely different idea precisely because they think that performance doesn't matter for this particular feature. So super important, dissect people's ideas, try to find loopholes in their arguments just because that'll lead you to a better decision.

And that might sway other people one way or another. Now, one very important aspect of this process here is that you do not want to attack people. You want to attack ideas. Attacking people is toxic, it's unhealthy, it's not professional. You don't want to do that. You want to attack ideas. And what that means is that you have to make sure you're not attacking people, but you also have to make sure that when people are attacking your ideas, you don't take that personally. So these are kind of important, important things that every individual in the disagreement has to kind of internalize for this to actually work. But so that was the first step. You dig deep, really try to understand why people are coming to the conclusions that they're coming to. The second step is to take a step back and to kind of see things from a higher level, from a higher perspective, because oftentimes the environment that you're in, the ecosystem that you're in, maybe the company's current stance or point in time, that might influence the engineering decision.

Now, there have been many times on Algo Expert where we had a disagreement on an engineering decision. If you purely looked at the arguments from both sides without looking at the bigger picture, one person was clearly right. Like, okay, this is, it's better not to do this feature because from an engineering point of view, it'll take far too much time. It'll worsen other parts of the code base. We'll have to use another third party service that we really shouldn't be using, blah, blah, blah. And we agree that that's the better decision. But if you take a step back and you realize that our business's livelihood depends on doing this feature, then well, suddenly, you know, there's an art, there's an additional argument for doing that feature. So got to take a step back sometimes and realize that your organization or your team might have priorities that trump some of the arguments at the purely engineering level.

And then finally, the third part of this process or system that I've got in place, and I think this one is again based on a Amazon leadership principle, or not necessarily based, but I think they have a leadership principle that's similar, is that you want to strongly disagree. You want to do everything that I said, dissect each other's ideas, all of that. But once it comes to making a decision, you have to commit to the decision that gets made. Once the decision is made, rather, you have to commit to that deci

```
sion. You can't hold grudges, especially if the decision doesn't go your w
ay. If it goes your way, then obviously you'll be happy. But if it does
n't go your way, you have to be able to accept that. You have to commit t
o the decision that was made, and you have to move forward as if that had
been your decision to make sure that the product and the team and everythi
ng just goes smoothly.

And of course, I guess here I skipped over one step, which is like, how d
o you make the decision? Because sometimes someone will change their min
d, and that's great. You'll make the decision. Everybody will be happy. Bu
t other times, despite all of the arguing and dissecting people's ideas an
d all that, you still will have a big disagreement. People will still not
agree to agree with each other. And in that case, you really have two opti
ons. Either you do a majority vote. You vote to see what decisions should
be made, and that can work very well. Or you escalate the decision to a de
cision maker. Typically, that will be someone who's previously appointed,
like a manager or your manager's manager, maybe your team lead, tech lea
d, whatever you want to call it, and they have to make the decision.

If you do escalate to a decision maker, you have to make sure if they were
n't part of the disagreement, that they get the full context. You have to
present them with all the arguments and everything to make sure that they
make a sound decision. But once the decision is made, like I said, commit
to the decision. And yeah, this is how I've handled most, if not all, of m
y disagreements at work, and how I plan to continue handling disagreement
```

problem 12

> tough feedback

Plain Text

Sure! Here’s the extracted English text from the provided content:

Hey everybody, welcome to Algo Expert. In this video, we're going to answer the following behavioral interview question. How do you think about receiving and giving feedback? Describe a time when you received tough feedback and/or a time when you gave tough feedback. How did you react to it? How did you give it? So I'm a firm believer in giving constructive feedback by way of radical transparency. Now I realize that the term or the expression "radical transparency" is a bit of a buzzword, but I do think that it's a very useful one, a very applicable one here in the context of giving feedback. You want to be very prompt and direct in giving your feedback or in receiving feedback. There is no better way to improve as a human being and of course as a software engineer, if we're talking about this in the context of software engineering, than by receiving quality constructive feedback that is direct, that doesn't put things vaguely or fluff things up, and that is also prompt, meaning it doesn't come two months, three months after the fact. It comes one week or one day after something has occurred that deserves feedback. Now one additional thing that I want to add here is that I think it's very important when giving feedback to and receiving feedback to give or ask for actionable steps to improve. In other words, it doesn't really help a person who's receiving feedback if you just tell them that they're doing something wrong or doing something poorly without telling them how they can improve, what they can do, what steps they can take to improve. So that's very important, but really I think that this concept of radical transparency is super useful and it's very different from being overly harsh and just mean to people, which is obviously not good and not useful, and very different from being overly nice and hiding things from people just to protect them, so to speak. That doesn't help them. But so as far as examples of when I've received and given feedback, one particular example that comes to mind for when I received feedback was near the start of my career as a software engineer at Google. I think I had been at Google for about two or three months, and one of my peers following a pull request that I had sent out with code basically came up to me and told me, "Hey, Clement, you know, this might sound harsh, but I just want to tell you I think you're writing certain front-end components the wrong way, or not necessarily the wrong way, but in a suboptimal way." And of course, you know, as with all kinds of constructive feedback, which is effectively negative feedback but meant to improve you, I reacted a little bit, you know, had a bit of a tense reaction. You always feel you're kind of like, "Ugh, am I being attacked?" or whatever, but you quickly realize like this is how you improve. And so I had a good conversation with him and, you know, we talked about how I could actually, you know, why were thes

```
e front-end components that I was writing suboptimal, what was bad, or, yo
u know, what could be improved about them. And, you know, I learned a lot
from that. We talked about, you know, stateless components versus statefu
l components, all that stuff, all that good front-end engineering stuff. A
nd I really improved following that conversation. I noticed it, you know,
when I would write future pull requests and future components that the thi
ngs that this person had told me were true, and I was now applying them, a
nd it was just a great experience. Now, for the time that I gave feedback
to someone, this was one of the first interns that I hosted while at Googl
e. And I remember this was a really bright person, really excited, really
talented, very good software engineer. One thing that he would do that was
n't particularly good, that could have been improved, was that, especiall
y in our one-on-one meetings, and even in group meetings, he would often,
you know, start talking, start asking questions, or start, you know, givin
g ideas. But he, it kind of felt like he hadn't prepared his thoughts, an
d he would start rambling on for pretty, like, noticeable periods of tim
e. And so, I told him after I saw him do that, I think, like, three time
s, so pretty promptly, I told him in one of our one-on-one meetings, "He
y, like, this is something where I think you could improve, that you coul
d improve on." You know, as far as actionable steps, I told him, "Just tr
y to prepare yourself and gather your thoughts a little bit more before yo
u take the microphone, so to speak." And again, as with any time that you
give feedback or receive feedback, when it's constructive and it effective
ly feels like it's negative, you kind of tense up, I could see it in his f
ace, but afterwards, he totally, you know, accepted it, acknowledged it, a
nd at our next one-on-one, he told me that he really appreciated the feedb
ack, he really took it to heart, and what do you know, by the end of the i
nternship, he had completely improved in this regard. And not only had he
continued to be a great public speaker, but he was really honing his skill
s of not, you know, rambling on, not being prepared before speaking. So th
at was another great example of where direct, radically transparent promp
t feedback is very useful.

---

If you need further assistance, feel free to ask!
```

problem 13

> production outage

Plain Text

Hey everybody, welcome to Algo Expert. In this video we're going to answer the following behavioral interview question.

Describe a time when you had to deal with an outage at work. How did you handle the situation? What steps did you take after the issue was resolved?

So, throughout my entire software engineering career, I would say that I've experienced about a handful of real production outages.

Now the one that's most memorable, the one that I want to talk about here, is actually the very first real production outage that I ever experienced.

It happened back in 2017. I had been at Google as a software engineer for a little over six months.

And one afternoon, it was kind of late in the afternoon, I was the last person on my team, or at least the last engineer on my team, at the office.

And I happened to go on the UI, the user interface, of my team's product. I worked on Google Cloud Platform on the front end of a particular Google Cloud Platform product.

And so I go on the website, on the UI, and one of our primary pages was just blank. It wasn't loading, nothing on the screen, just completely blank.

So obviously for a split second, I kind of panicked, "What's going on?" Then I composed myself, and here's how I handled this, what turned out to be a pretty bad production outage.

The first thing that I did is, I wanted to confirm that this was actually a real bug in production and not just something with my computer or my account or something like that.

So I asked a couple people who were around me, these were not my teammates, because like I said, I was the last person from my team at the office,

but a couple of other people who were in proximity to me to go on the link of the page and just to see if they were also experiencing the same thing on their computers, on their accounts.

And yes, they were. So clearly this was a real production issue.

So the second thing that I decided to do was, before alerting anybody else, and I suppose here maybe I should take a step back and say,

at the time, my team didn't have an on-call rotation set up, because if we had had an on-call rotation, the first thing to do here would have been, once I confirmed that this was a real bug,

the first thing to do would have been to flag the bug or the issue to the primary on-call. Just at least have them be aware of what was going on.

We didn't have an on-call rotation at that point in time. We created our on-call rotation a little bit later on.

And this was, by the way, a relatively new Google Cloud Platform product at the time, which is why we didn't have our on-call rotation.

Given that, the first thing that I said I was going to do, or the second thing rather, was I was going to spend five to ten minutes just trying to see if I could debug the issue.

I wanted to see, like, can I fix this without having to alert my teammates who are no longer in the office, without having to alert my manager, who was, I don't know where he was, probably at home.

I spent five to ten minutes trying to debug the issue. Turns out I wasn't able to figure out what the cause of the issue was.

It was a very weird bug, blank page, nothing in the Chrome console. There had been nothing out of the ordinary that day or in the previous days.

So it was really confusing and I just wasn't able to figure it out. So at that point, I did alert the rest of my team after, I would say, about ten minutes.

Because, you know, I think this is one of my philosophies for these types of issues. You never want to be, you don't want to play the hero, right?

If you can, try to figure out the issue within five minutes, but after a certain point, there's no point in hiding this from other people.

The key thing is to fix the issue, not for you to be the hero or for people not to be bothered, if that makes sense.

So after five to ten minutes, alerted my team and fortunately, one of my teammates was actually still in the office, just upstairs getting dinner or something.

So he came down and we started pair programming to try to figure out this bug.

Now, one of the key things also here that we did, which is something that I am a very big proponent of, is even though the situation was pretty stressful,

literally one of the primary pages of our product was down, not working, which meant that the entire product was basically not usable or only half usable.

But so despite the stressful situation, you have to stay calm and you have to also, you know, not point fingers at anybody, not put any blame on anybody.

Because obviously when you're debugging in the moment and everything, it's like, oh, well, this file was poorly written. Why did someone do that?

So I'm not going to go into the bugs here. It's very easy to potentially point fingers and blame people.

I didn't want to do that. And that's just unhealthy in a team in general.

But so we pair programmed, we documented everything that we were doing, keeping our team in the loop.

At that point, my manager was also in the loop. We followed the process that we knew to follow.

Unfortunately, we didn't have some sort of playbook to follow line by line for an outage. And this was a lesson that we learned from this. I'll get into that in a little bit.

But we just documented everything. And after a while, you know, I would say probably after 30 minutes, we decided with my manager, who was chatting with us at this point,

to create an official what's called an OMG at Google, which is basically like an incident report that actually gets flagged kind of at the engineering company level.

And so we did that. We kept the incident report sort of up to date.

81
And this kind of flags people who are on call and all of Google Cloud Pla
82 tform and the rest of the company and just followed the procedure there.
83
Eventually, we figured out what the bug was. It turned out it was somethi
84 ng really, really convoluted with Angular, the front end framework.
85
But that was it. And then we fixed the bug. We had this OMG, this inciden
86 t report kind of in progress. We eventually closed it.
87
And, you know, everything was done after that. By the next morning, you k
88 now, the new sort of production release or deployment, the bug was fixed.
89
Now, as for what we did afterwards, we did something that's very common a
90 t Google, which I've continued to do even since I've left Google.
91
And we do it a lot at my current job on Algo Expert, which is to write a
92 postmortem, which is kind of like a document explaining what happened,
93
the issue, detailing all the steps that were taken to resolve the issue.
And most importantly, we answered two questions that I think are crucial
94 to answer after production outage.
95
Number one, how did we get lucky? And this is a good question to answer b
ecause it allows you to determine where there are holes or flaws in your
96 general processes and systems.
97
For us, we got lucky in the sense that I happened to be at the office lat
98 e and happened to go on the UI and fall on that page.
99
Nothing in our systems at that point in time would have flagged that the
100 page was down. And that was clearly an issue.
101
102 We had gotten lucky there that we caught the outage quickly and randomly.
103
And the lesson from that and the thing that we took away from that was th
104 at we needed end to end integration tests.
105
Like I said, we were a fairly new product at the time. We didn't have int
egration tests, but we obviously, you know, made it a point to create the
106 m after this outage.
107
That was one of our lessons from the postmortem. The second question tha
t I really like to answer in postmortems, which we did, was what could w
108 e have done better?
109

```
And the thing that we could have done better there was and this was more
of a preventative measure was we could have had a written out playbook fo
r how to deal with these kinds of production outages, whom to contact, ho
w to create an incident report, because that was the first time we had al
l done it.

Or at least me and the engineer pair programming with me. And so, again,
another lesson that we took from that it was we need a playbook. And in t
he couple of weeks following the issue, we actually wrote a, you know, wh
at do you do if you're at work and there's a production outage or what d
o you do when there's a production outage in general and you have to fix
it?
```

problem 14(not yet)

> After the technical rounds, it was time for the behavioral interview. This round was meant to assess cultural fit and collaboration skills, which are vital in a fast-paced, cross-functional environment like Gojek.
>
> - **Question 1**: “Tell me about a time when you had to work with cross-functional teams. How did you ensure smooth collaboration?” I wanted to hear about their experience in a collaborative setting and how they manage different perspectives.
> - **Question 2**: “How do you handle multiple priorities with tight deadlines?” This was aimed at assessing their ability to manage time and prioritize effectively in a high-pressure environment.
> - **Question 3**: “Tell us about a time you received constructive feedback. How did you handle it?” This was to understand their growth mindset and openness to feedback.
>
> The candidate was well-spoken and gave examples of teamwork, highlighting their experience collaborating with product managers and designers. They emphasized how they approach challenges with an open mind and adapt quickly to feedback.

problem 15(not yet)

> We focused on cultural alignment and teamwork. The candidate was asked:
>
> - Tell me about a time you worked on a team with conflicting opinions
> - How do you handle tight deadlines with limited resources?
>
> Their responses reflected a collaborative approach and effective communication, showing they could thrive in Gojek’s dynamic environment.

problem 16(not yet)

> The final round was more strategic and focused on leadership. Key questions included:
>
> - How do you prioritize tasks in a fast-paced work environment?
> - What excites you about working at Gojek?
>
> The candidate shared valuable insights on balancing technical depth and business goals, which aligned well with Gojek's mission.

problem 17

> comfort zone

Plain Text

```
Hey everybody, welcome to Algo Expert. In this video, we're going to answer the following behavioral interview question.

Describe a time when you went out of your comfort zone.

Why did you do it? What lessons did you learn from the experience?

So one time that I went out of my comfort zone was back during the summer of 2018.

I had decided to host for the very first time a pair of engineering interns at Google.

This was my very first managerial experience.

And I was very excited about it because it was something that I'd always wanted to do, but it was something very new to me.

So it was out of my comfort zone in that I had never been in a managerial position before.

And I had been warned by my manager, by other people who had hosted interns in the past, that this was going to be a non-trivial endeavor.

It wasn't going to be easy.

And also, I chose to host what are called engineering practicum interns at Google, where these are interns who are a bit younger.

They are, I think, freshmen, incoming sophomores or incoming juniors.

So a bit earlier on in their career, and they come in pairs, not just a single intern.

So it was definitely going to stretch me out of my comfort zone.

And the reason I did it was because not only did I want to do it, like I said, I had always wanted to try this out, to try to be a manager, to see if that was something that I was going to really enjoy.

But also, I'm a firm believer that one of the best ways to grow is to push yourself out of your comfort zone, like to challenge yourself with new things, things that you haven't done before.

```

It's a great way to test, to see if you like certain things.

It's a great way to just learn new skills and grow.

And so that's why I wanted to do it.

Now, what I learned from it, I think probably two big lessons that I that I drew from the experience.

The first one was that management isn't easy.

I know that engineers especially have a tendency to kind of look down upon management to say that management is a bit easy.

It's it's sometimes useless.

That's that's the feeling that I felt from a lot of engineers that I've met.

But truthfully, being in a managerial position, I realized that management isn't easy.

And here I'll quote something that my my manager, my current manager, told me, which is that engineering is easy.

People are hard.

So management, by definition or by extension, is hard.

And one of the reasons that it's hard is you have people who are dependent on you and your actions are almost directly going to determine multiple people, in my case, the two interns I was hosting, multiple people's experiences at work and basically happiness at work.

And one last thing I'll say about this is it was particularly difficult when I had to do, let's say, one on one meetings with with my interns.

It was particularly difficult to to deal with sometimes my own like bad days.

Like, for example, if I had a bad day as an engineer, I could just go to my computer, put my headphones on and code.

And I didn't really have to talk to anybody except maybe in a couple of meetings or in a pair programming session.

But when you're a manager, you can't have you can't have a bad day and go to a meeting in a one on one with a direct report and and show that you're having a bad day.

Right. Because you have to serve them.

You can't kind of leak your bad mood on them.

So that was something that was pretty difficult.

But anyway, I did the gate. The second big lesson that I learned from the experience was that you you quickly realize that there's no adult in the other room to save you.

So what I mean by that is when you work at a big company, especially a big tech company, you often have this feeling of security, at least as a as an engineer or as just an individual contributor.

You often have this feeling of of mental security where if something goes wrong or if you don't have the answer to something or if you don't know what to do, somebody else will be there to save you.

Somebody else will be there to give you a parachute to give you the answer that you're looking for.

When you put yourself in a managerial position, at the very least when you're when you're the manager of of an intern or of a pair of interns, there is nobody else behind you to pick up your messes.

If I messed up the projects that I assigned to my interns, if I messed up the one on ones with my intern, there wasn't anybody else who was responsible for them.

And that's a pretty that's a pretty frightening realization.

It's also a very empowering one.

And I've just remembered it ever since.

And now whenever when I kind of see my own managers now and leaders in companies in general in a bit of a different light, and I realize the difficulty that they might be experiencing when they're making decisions, because oftentimes