SuperDial Backend Engineer Interview Preparation Guide

Comprehensive Solutions & Strategies

November 9, 2025

Contents

1 Introduction

1.1 About SuperDial

SuperDial is a voice AI platform revolutionizing healthcare communication. Founded in 2021, they automate patient interactions, appointment scheduling, and administrative tasks using advanced voice AI technology. As a fast-growing startup with approximately 15 employees, they're focused on HIPAA-compliant voice automation for medical practices.

1.2 Interview Process

Based on similar healthcare AI startups:

1. **Recruiter Screen** - Culture fit, passion for healthcare (30 min)
2. **Technical Interview** - LeetCode medium + system design (60 min)
3. **Take-Home Assignment** - Voice API or healthcare data processor (2-3 hours)
4. **Team Interview** - Technical deep dive (60 min)
5. **Founder Interview** - Vision alignment (30 min)

1.3 Tech Stack

SuperDial likely uses:

- **Backend**: Python/Node.js, RESTful APIs
- **Voice AI**: Speech-to-text, NLP, conversational AI
- **Healthcare**: HIPAA compliance, secure PHI handling
- **Real-time**: WebSockets, streaming data
- **Cloud**: AWS/GCP for scalable infrastructure

1.4 Key Skills to Demonstrate

- Strong fundamentals in data structures and algorithms
- System design for real-time, high-reliability systems
- Understanding of healthcare data sensitivity (HIPAA, PHI)
- API design and microservices architecture
- Passion for healthcare and AI/ML systems

2 Problem 1: Priority Rate Limiter

2.1 Problem Statement

SuperDial handles healthcare voice calls that require prioritization based on urgency. Implement a rate limiter that processes API requests with priority levels while enforcing rate limits.

Requirements:

- Limit to `max_requests` per `time_window` seconds
- Higher priority requests (lower number = higher priority) should be processed first when possible
- Return `True` if request allowed, `False` if rate limited
- Clean up old requests outside the time window

Example:

```
limiter = PriorityRateLimiter(max_requests=3, time_window=1.0)
limiter.allow_request(priority=1)  # True (urgent)
limiter.allow_request(priority=2)  # True
limiter.allow_request(priority=2)  # True
limiter.allow_request(priority=3)  # False (rate limit reached)

# After 1 second
limiter.allow_request(priority=1)  # True (window reset)
```

2.2 Solution

```
import time
from collections import deque

class PriorityRateLimiter:
    def __init__(self, max_requests: int, time_window: float):
        self.max_requests = max_requests
        self.time_window = time_window
        self.requests = deque()  # (timestamp, priority)

    def allow_request(self, priority: int) -> bool:
        current_time = time.time()

        # Remove requests outside the time window
        while self.requests and current_time - self.requests[0][0] >=
    self.time_window:
            self.requests.popleft()

        # Check if we can accept the request
        if len(self.requests) < self.max_requests:
            self.requests.append((current_time, priority))
            return True

        return False
```

2.3 Complexity Analysis

- **Time Complexity**: O(n) where n is the number of requests in the window (amortized O(1) for cleanup)

- **Space Complexity**: O(max_requests)

2.4 Key Insights

1. Use a deque for efficient removal of old requests from the front
2. Store timestamps with requests to track the time window
3. For healthcare systems, priority is critical but rate limiting applies to all requests
4. In production, consider distributed rate limiting with Redis

2.5 Follow-up Questions

1. How would you implement this in a distributed system with multiple servers?
2. How would you handle time synchronization across servers?
3. What if we wanted to give priority requests a separate, higher rate limit?

3 Problem 2: Streaming Buffer Management

3.1 Problem Statement

SuperDial processes real-time voice transcription data in chunks. Implement a circular buffer that efficiently manages streaming data with fixed capacity.

Requirements:

- Fixed-size buffer with efficient write/read operations
- Support for writing chunks of data
- Support for reading all available data
- Handle buffer overflow (old data overwritten)
- Track number of bytes available

Example:

```
buffer = StreamingBuffer(capacity=10)
buffer.write("Hello")       # Returns 5 (bytes written)
buffer.write(" World")      # Returns 6
buffer.read()               # Returns "Hello Worl" (last 10 bytes)
buffer.available()          # Returns 10
```

3.2 Solution

```
class StreamingBuffer:
    def __init__(self, capacity: int):
        self.capacity = capacity
        self.buffer = ""

    def write(self, data: str) -> int:
        # Add new data to buffer
        self.buffer += data

        # Keep only the last 'capacity' bytes (circular behavior)
        if len(self.buffer) > self.capacity:
            self.buffer = self.buffer[-self.capacity:]

        return len(data)

    def read(self) -> str:
        return self.buffer

    def available(self) -> int:
        return len(self.buffer)
```

3.3 Complexity Analysis

- **Time Complexity**: O(n) for write where n is data size; O(1) for read and available
- **Space Complexity**: O(capacity)

3.4 Key Insights

1. Simple string-based approach works for small buffers
2. For production voice streaming, use byte arrays or ring buffers
3. Circular buffers prevent memory growth while maintaining recent data
4. Real-time systems need low-latency buffer operations

3.5 Optimizations

- Use `bytearray` for better performance with binary data
- Implement true circular indexing to avoid string slicing
- Add watermark notifications (buffer 80% full)
- Consider thread-safe operations for concurrent access

4 Problem 3: Patient Data Deduplication

4.1 Problem Statement

SuperDial's healthcare platform needs to detect duplicate patient records. Implement a system to identify potential duplicates using fuzzy matching on names and exact matching on identifiers.

Requirements:

- Detect duplicates based on similar names (Levenshtein distance)
- Exact match on phone numbers or patient IDs
- Return groups of potential duplicates
- Handle edge cases (missing data, empty fields)

Example:

```python
patients = [
    {"id": "P1", "name": "John Smith", "phone": "555-1234"},
    {"id": "P2", "name": "Jon Smith", "phone": "555-1234"},
    {"id": "P3", "name": "Jane Doe", "phone": "555-5678"},
]

detector = PatientDeduplicator()
duplicates = detector.find_duplicates(patients)
# Returns: [[0, 1]] (indices 0 and 1 are duplicates)
```

4.2 Solution

```python
from typing import List, Dict

class PatientDeduplicator:
    def __init__(self, name_threshold: int = 3):
        self.name_threshold = name_threshold

    def find_duplicates(self, patients: List[Dict[str, str]]) -> List[
    List[int]]:
        n = len(patients)
        parent = list(range(n))  # Union-Find

        def find(x):
            if parent[x] != x:
                parent[x] = find(parent[x])
            return parent[x]

        def union(x, y):
            px, py = find(x), find(y)
            if px != py:
                parent[px] = py

        # Compare all pairs
        for i in range(n):
            for j in range(i + 1, n):
                if self.is_duplicate(patients[i], patients[j]):
                    union(i, j)

        # Group duplicates
        groups = {}
        for i in range(n):
            root = find(i)
            if root not in groups:
                groups[root] = []
            groups[root].append(i)

        # Return groups with more than 1 patient
        return [group for group in groups.values() if len(group) > 1]

    def is_duplicate(self, p1: Dict[str, str], p2: Dict[str, str]) -> bool:
        # Same ID
        if p1.get("id") == p2.get("id") and p1.get("id"):
            return True

        # Same phone number
        if p1.get("phone") == p2.get("phone") and p1.get("phone"):
            return True

        # Similar names
        name1 = p1.get("name", "")
        name2 = p2.get("name", "")
        if name1 and name2:
            distance = self.levenshtein_distance(name1.lower(), name2.lower())
            if distance <= self.name_threshold:
                return True

        return False

    def levenshtein_distance(self, s1: str, s2: str) -> int:
        m, n = len(s1), len(s2)
        dp = [[0] * (n + 1) for _ in range(m + 1)]

        for i in range(m + 1):
            dp[i][0] = i
        for j in range(n + 1):
            dp[0][j] = j

        for i in range(1, m + 1):
            for j in range(1, n + 1):
                if s1[i-1] == s2[j-1]:
                    dp[i][j] = dp[i-1][j-1]
                else:
                    dp[i][j] = 1 + min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1])

        return dp[m][n]
```

4.3 Complexity Analysis

- **Time Complexity**: $O(n^2 \times L^2)$ where n is number of patients, L is average name length
- **Space Complexity**: $O(n + L^2)$ for Union-Find and DP table

4.4 Key Insights

1. Union-Find efficiently groups duplicates
2. Levenshtein distance catches typos and minor variations
3. Multiple matching criteria (ID, phone, name) increase accuracy
4. Healthcare data requires careful handling of missing/null values
5. HIPAA compliance: de-identify before testing/development

4.5 Healthcare Considerations

- **PHI Protection**: Patient data must be encrypted and access-controlled
- **Audit Logging**: Track all access to patient records
- **Data Quality**: Healthcare data often has inconsistencies
- **False Positives**: Manual review process for suggested duplicates

5 Problem 4: Call Routing System

5.1 Problem Statement

SuperDial needs to route incoming voice calls to available AI agents. Implement a load balancer that distributes calls efficiently across agents while considering capacity and specialization.

Requirements:

- Route calls to least loaded available agent
- Support agent specialization (e.g., ”cardiology”, ”general”)
- Track active calls per agent
- Handle agent availability (online/offline)
- Return None if no agents available

Example:

```
router = CallRouter()
router.add_agent("agent1", capacity=5, specialization="cardiology")
router.add_agent("agent2", capacity=3, specialization="general")

router.route_call("call1", specialty="cardiology")  # Returns "agent1"
router.route_call("call2", specialty="general")     # Returns "agent2"
router.end_call("call1")                             # Frees "agent1"
```

5.2 Solution

```
from typing import Optional, Dict, Set

class CallRouter:
    def __init__(self):
        self.agents = {}  # agent_id -> {capacity, specialization,
    online, calls}
        self.call_to_agent = {}  # call_id -> agent_id

    def add_agent(self, agent_id: str, capacity: int, specialization:
    str) -> None:
        self.agents[agent_id] = {
            "capacity": capacity,
            "specialization": specialization,
            "online": True,
            "calls": set()
        }

    def route_call(self, call_id: str, specialty: str) -> Optional[str
    ]:
        best_agent = None
        min_load = float('inf')

        # Find least loaded agent with matching specialty
        for agent_id, agent in self.agents.items():
            if not agent["online"]:
                continue
            if agent["specialization"] != specialty:
                continue
            if len(agent["calls"]) >= agent["capacity"]:
                continue

            current_load = len(agent["calls"])
            if current_load < min_load:
                min_load = current_load
                best_agent = agent_id

        if best_agent:
            self.agents[best_agent]["calls"].add(call_id)
            self.call_to_agent[call_id] = best_agent
            return best_agent

        return None

    def end_call(self, call_id: str) -> None:
        if call_id in self.call_to_agent:
            agent_id = self.call_to_agent[call_id]
            self.agents[agent_id]["calls"].discard(call_id)
            del self.call_to_agent[call_id]

    def set_agent_status(self, agent_id: str, online: bool) -> None:
        if agent_id in self.agents:
            self.agents[agent_id]["online"] = online
```

5.3 Complexity Analysis

- **Time Complexity**: O(n) for routing where n is number of agents; O(1) for end call and status change
- **Space Complexity**: O(a + c) where a is agents, c is active calls

5.4 Key Insights

1. Load balancing prevents agent overload
2. Specialization matching ensures proper expertise for calls
3. Tracking active calls enables accurate capacity management
4. Agent status management handles maintenance/breaks

5.5 Production Enhancements

- **Priority Queue**: Use heap for O(log n) best agent selection
- **Metrics**: Track call duration, success rate per agent
- **Failover**: Route to "general" if specialty not available
- **Sticky Routing**: Route follow-up calls to same agent
- **Geographic**: Route based on agent location/timezone

6 System Design Topics

6.1 Voice AI Architecture

Key components of a voice AI system:

1. **Speech-to-Text (STT)**: Convert audio to text
2. **Natural Language Understanding (NLU)**: Extract intent and entities
3. **Dialog Management**: Manage conversation flow
4. **Text-to-Speech (TTS)**: Generate voice responses
5. **Backend API**: Business logic and data access

6.2 HIPAA Compliance Requirements

Critical for healthcare applications:

- **Encryption**: At-rest and in-transit (TLS 1.2+)
- **Access Control**: Role-based access, MFA
- **Audit Logging**: Track all PHI access
- **Data Retention**: Secure deletion after retention period
- **Business Associate Agreements**: With all third-party services

6.3 Real-time System Design

Considerations for voice calls:

- **Low Latency**: ¡300ms for natural conversation
- **High Availability**: 99.99% uptime for healthcare
- **Scalability**: Handle thousands of concurrent calls
- **Error Handling**: Graceful degradation, fallback to human
- **Monitoring**: Real-time alerts for system health

7 Behavioral Interview Prep

7.1 SuperDial Values

- **Mission-Driven**: Passionate about improving healthcare
- **Startup Mentality**: Comfortable with ambiguity, rapid iteration
- **Technical Excellence**: High-quality code despite fast pace
- **Patient-Centric**: Always considering patient impact

7.2 Common Questions

1. Why are you interested in healthcare technology?
2. Tell me about a time you built something with limited resources
3. How do you handle competing priorities?
4. Describe your experience with AI/ML systems
5. What's your approach to ensuring code quality under pressure?

7.3 Questions to Ask

- What are the biggest technical challenges you're facing?
- How do you balance speed and quality in development?
- What's your approach to HIPAA compliance and security?
- How does the voice AI system handle edge cases and errors?
- What's the team structure and growth plans?

8 Additional Resources

8.1 Technical Learning

- **Algorithms**: "Cracking the Coding Interview" - LeetCode medium problems
- **System Design**: "Designing Data-Intensive Applications" by Martin Kleppmann
- **Voice AI**: Google Cloud Speech-to-Text, AWS Transcribe documentation
- **Healthcare**: HIPAA compliance guides, HL7/FHIR standards

8.2 Practice Platforms

- **LeetCode**: Focus on medium difficulty, strings, queues, graphs
- **System Design**: Grokking the System Design Interview
- **Mock Interviews**: Pramp, Interviewing.io

8.3 Healthcare Tech Blogs

- Healthcare IT News
- HIMSS Healthcare Blog
- Health Tech Insider

9 Final Tips

1. **Healthcare Passion**: Show genuine interest in improving patient care
2. **System Thinking**: Discuss trade-offs, not just solutions
3. **Startup Fit**: Demonstrate adaptability and ownership mentality
4. **Ask Questions**: Show curiosity about their technology and mission
5. **Follow Up**: Send thank-you notes, reiterate interest

Good luck with your SuperDial interview!