ThinkRealty Backend Developer Assessment - Lead Management System Implementation

Time Allocation: 4 Days
Tech Stack: FastAPI + PostgreSQL + Docker + Redis

Background Context

You are building the lead management system for ThinkRealty, a Dubai-based real estate platform. This system must handle lead capture from multiple sources (Bayut, PropertyFinder, Dubizzle, website forms, walk-ins), intelligent lead distribution among agents, and comprehensive lead analytics.

The system must support the UAE real estate market specifics, including lead scoring based on buyer nationality, budget ranges in AED, property type preferences, and agent performance tracking with Arabic language support.

Assessment Tasks

Task 1: Database Schema Design

Design and implement the complete database schema for the lead management system.

Requirements:

1. Core Tables to Design:

```
-- Starter schema (you need to complete this)
CREATE TABLE leads (
    lead_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    source_type VARCHAR(50) NOT NULL, -- 'bayut', 'dubizzle', 'propertyFi
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(255),
    phone VARCHAR(20) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -- Complete the rest...
);

CREATE TABLE agents (
```

```
    agent_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    full_name VARCHAR(200) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    phone VARCHAR(20) NOT NULL,
    -- Add more fields...
);

-- You need to design these tables completely:
-- lead_assignments
-- lead_activities
-- lead_scoring_rules
-- follow_up_tasks
-- lead_property_interests
-- lead_sources
-- agent_performance_metrics
-- lead_conversion_history
```

2. Business Rules to Implement in Schema:

- Lead phone numbers must be unique per source (same person can exist from different sources)
- Each lead must be assigned to exactly one agent at a time
- Lead scoring must be between 0-100
- Follow-up tasks cannot be overdue by more than 30 days without escalation
- Agent workload cannot exceed 50 active leads
- Lead status progression must follow: new → contacted → qualified → viewing_scheduled → negotiation → converted/lost
- All lead activities must be timestamped and auditable

3. Sample Data Requirements:

Create sample data for testing:

- 100+ leads from various sources
- 10+ agents with different specializations
- 50+ lead activities and follow-ups
- 20+ property interests linked to leads
- Various lead statuses and conversion outcomes

Deliverables:

- Complete SQL schema with all tables, constraints, and indexes
- Database migration scripts
- Sample data insertion scripts
- Data validation triggers

Task 2: Core API Endpoints

Implement the following FastAPI endpoints with complete business logic:

2.1 Lead Capture Endpoint

```
POST /api/v1/leads/capture
```

Request Body:

```
{
  "source_type": "bayut|propertyFinder|dubizzle|website|walk_in|referral"
  "lead_data": {
    "first_name": "Ahmed",
    "last_name": "Al Mansouri",
    "email": "ahmed@email.com",
    "phone": "+971501234567",
    "nationality": "UAE",
    "language_preference": "arabic|english",
    "budget_min": 800000,
    "budget_max": 1500000,
    "property_type": "apartment|villa|townhouse|commercial",
    "preferred_areas": ["Downtown Dubai", "Marina", "JBR"]
  },
  "source_details": {
    "campaign_id": "spring_campaign_2024",
    "referrer_agent_id": "uuid", // if referral
    "property_id": "uuid", // if from property inquiry
    "utm_source": "google_ads"
  }
}
```

Response Structure:

```
{
  "success": true,
  "lead_id": "uuid",
  "assigned_agent": {
    "agent_id": "uuid",
    "name": "Sarah Johnson",
    "phone": "+971501234568"
  },
```

```
  "lead_score": 85,
  "next_follow_up": "2024-01-16T10:00:00Z",
  "suggested_properties": ["uuid1", "uuid2", "uuid3"]
}
```

Business Logic Requirements:

- Automatically score lead based on predefined rules
- Assign lead to best available agent using round-robin with workload balancing
- Generate initial follow-up task
- Detect duplicate leads across sources

2.2 Agent Dashboard Endpoint

```
GET /api/v1/agents/{agent_id}/dashboard
```

Query Parameters:

- `date_range` : "7d|30d|90d|custom"
- `status_filter` : "all|active|converted|lost"
- `source_filter` : "all|bayut|propertyFinder|dubizzle|website"

Response Structure:

```
{
  "agent_summary": {
    "total_active_leads": 23,
    "overdue_follow_ups": 5,
    "this_month_conversions": 8,
    "average_response_time": "4.5 hours",
    "lead_score_average": 67
  },
  "recent_leads": [
    {
      "lead_id": "uuid",
      "name": "Ahmed Al Mansouri",
      "phone": "+971501234567",
      "source": "bayut",
      "status": "qualified",
      "score": 85,
      "last_activity": "2024-01-15T14:30:00Z",
      "next_follow_up": "2024-01-16T10:00:00Z"
    }
```

```
  ],
  "pending_tasks": [
    {
      "task_id": "uuid",
      "lead_name": "Fatima Hassan",
      "task_type": "call|email|whatsapp|viewing",
      "due_date": "2024-01-16T10:00:00Z",
      "priority": "high|medium|low"
    }
  ],
  "performance_metrics": {
    "conversion_rate": 15.5,
    "average_deal_size": 1200000,
    "response_time_rank": 3
  }
}
```

2.3 Lead Update Endpoint

```
PUT /api/v1/leads/{lead_id}/update
```

Request Body:

```
{
  "status": "new|contacted|qualified|viewing_scheduled|negotiation|conver
  "activity": {
    "type": "call|email|whatsapp|viewing|meeting|offer_made",
    "notes": "Lead showed interest in 2BR apartment in Marina",
    "outcome": "positive|negative|neutral",
    "next_follow_up": "2024-01-18T14:00:00Z"
  },
  "property_interests": [
    {
      "property_id": "uuid",
      "interest_level": "high|medium|low"
    }
  ]
}
```

Task 3: Business Logic Implementation

3.1 Lead Scoring Engine

```python
class LeadScoringEngine:
    async def calculate_lead_score(
        self,
        lead_data: Dict[str, Any],
        source_details: Dict[str, Any]
    ) -> int:
        """
        Calculate lead score based on multiple factors:
        - Budget range (higher budget = higher score)
        - Source quality (Bayut=90, PropertyFinder=85, Website=80, etc.)
        - Nationality (UAE nationals = +10, GCC = +5)
        - Property type preference
        - Response time to initial contact
        - Referral source bonus
        """
        pass

    async def update_lead_score(
        self,
        lead_id: UUID,
        activity_data: Dict[str, Any]
    ) -> int:
        """
        Update lead score based on activities:
        - Positive interactions = +5 points
        - Property viewings = +10 points
        - Offer made = +20 points
        - No response for 7 days = -10 points
        """
        pass
```

3.2 Lead Assignment Manager

```python
class LeadAssignmentManager:
    async def assign_lead(
        self,
        lead_data: Dict[str, Any]
    ) -> UUID:
        """
        Smart lead assignment based on:
        - Agent specialization (property type, area)
        - Current workload (max 50 active leads)
```

```
        - Language preference matching
        - Performance metrics
        - Round-robin with weighted distribution
        """
        pass

    async def reassign_lead(
        self,
        lead_id: UUID,
        reason: str,
        new_agent_id: Optional[UUID] = None
    ) -> UUID:
        """
        Reassign lead to different agent:
        - Auto-reassign if no response in 24 hours
        - Manual reassignment by supervisor
        - Workload balancing reassignment
        """
        pass
```

Task 4: Advanced Query Implementation

Write complex PostgreSQL queries for these scenarios:

4.1 Lead Performance Analytics Query

```
-- Generate comprehensive lead performance report including:
-- - Lead conversion rates by source and agent
-- - Average time to conversion by property type
-- - Monthly lead volume trends
-- - Agent performance rankings
-- - Revenue attribution by lead source
```

4.2 Lead Quality Analysis Query

```
-- Analyze lead quality patterns:
-- - High-scoring leads that didn't convert (identify issues)
-- - Low-scoring leads that converted (identify opportunities)
-- - Source quality comparison over time
-- - Optimal follow-up timing analysis
```

4.3 Agent Workload Optimization Query

```
-- Optimize agent assignments:
-- - Current workload distribution
-- - Agents approaching maximum capacity
-- - Specialized vs general agent performance
-- - Lead response time correlation with conversion
```

Task 5: Error Handling & Edge Cases

Implement comprehensive error handling for:

1. **Duplicate Lead Detection**: Same phone number from multiple sources within 24 hours
2. **Agent Overload**: Assignment when agent reaches maximum lead capacity
3. **Invalid Lead Data**: Missing required fields or invalid formats
4. **Follow-up Conflicts**: Multiple follow-ups scheduled at same time
5. **Status Transition Violations**: Invalid status changes (e.g., new → converted)
6. **External API Failures**: Property suggestion service unavailable

Technical Specifications

Docker Setup Required

```
FROM python:3.11-slim
# Complete the Dockerfile for the application
```

Environment Configuration

```
DATABASE_URL=postgresql://user:pass@postgres:5432/thinkrealty_leads
REDIS_URL=redis://redis:6379/1
```

Evaluation Criteria

Code Quality (25%)

- Clean, readable, well-documented code
- Proper separation of concerns
- Consistent naming conventions
- Comprehensive error handling

Database Design (25%)

- Proper normalization and relationships
- Efficient indexing strategy
- Constraint implementation
- Sample data quality

Business Logic (30%)

- Correct implementation of scoring rules
- Proper lead assignment logic
- Edge case handling
- Data validation accuracy

API Design (20%)

- RESTful API principles
- Proper HTTP status codes
- Input validation and sanitization
- Response structure consistency

Submission Requirements

1. **Complete Source Code**
    - All Python files with FastAPI implementation
    - Database migration scripts
    - Docker configuration files
    - Requirements.txt with dependencies
2. **Database Schema**
    - Complete SQL schema file
    - Migration scripts (up/down)
    - Sample data insertion scripts
3. **Documentation**
    - API documentation (FastAPI auto-docs)
    - Database schema documentation
    - Business rules explanation
    - Setup and running instructions

Prohibited Resources

- No AI assistants (ChatGPT, Copilot, etc.)
- No direct code copying from online sources
- You may use official documentation only
- Standard libraries and framework docs are allowed

Good luck!