Django Developer Assignment

Vendor Management System with Performance Metrics

Objective

Develop a Vendor Management System using Django and Django REST Framework. This system will handle vendor profiles, track purchase orders, and calculate vendor performance metrics.

Core Features

1. Vendor Profile Management:

- Model Design: Create a model to store vendor information including name, contact details, address, and a unique vendor code.
- API Endpoints:
  - POST /api/vendors/: Create a new vendor.
  - GET /api/vendors/: List all vendors.
  - GET /api/vendors/{vendor_id}/: Retrieve a specific vendor's details.
  - PUT /api/vendors/{vendor_id}/: Update a vendor's details.
  - DELETE /api/vendors/{vendor_id}/: Delete a vendor.

2. Purchase Order Tracking:

- Model Design: Track purchase orders with fields like PO number, vendor reference, order date, items, quantity, and status.
- API Endpoints:
  - POST /api/purchase_orders/: Create a purchase order.
  - GET /api/purchase_orders/: List all purchase orders with an option to filter by vendor.
  - GET /api/purchase_orders/{po_id}/: Retrieve details of a specific purchase order.
  - PUT /api/purchase_orders/{po_id}/: Update a purchase order.
  - DELETE /api/purchase_orders/{po_id}/: Delete a purchase order.

3. Vendor Performance Evaluation:

- Metrics:
  - On-Time Delivery Rate: Percentage of orders delivered by the promised date.
  - Quality Rating: Average of quality ratings given to a vendor's purchase orders.
  - Response Time: Average time taken by a vendor to acknowledge or respond to purchase orders.
  - Fulfilment Rate: Percentage of purchase orders fulfilled without issues.
- Model Design: Add fields to the vendor model to store these performance metrics.
- API Endpoints:
  - GET /api/vendors/{vendor_id}/performance: Retrieve a vendor's performance metrics.

Data Models

1. Vendor Model

This model stores essential information about each vendor and their performance metrics.

- Fields:
    - name: CharField - Vendor's name.
    - contact_details: TextField - Contact information of the vendor.
    - address: TextField - Physical address of the vendor.
    - vendor_code: CharField - A unique identifier for the vendor.
    - on_time_delivery_rate: FloatField - Tracks the percentage of on-time deliveries.
    - quality_rating_avg: FloatField - Average rating of quality based on purchase orders.
    - average_response_time: FloatField - Average time taken to acknowledge purchase orders.
    - fulfillment_rate: FloatField - Percentage of purchase orders fulfilled successfully.

2. Purchase Order (PO) Model

This model captures the details of each purchase order and is used to calculate various performance metrics.

- Fields:
    - po_number: CharField - Unique number identifying the PO.
    - vendor: ForeignKey - Link to the Vendor model.
    - order_date: DateTimeField - Date when the order was placed.
    - delivery_date: DateTimeField - Expected or actual delivery date of the order.
    - items: JSONField - Details of items ordered.
    - quantity: IntegerField - Total quantity of items in the PO.
    - status: CharField - Current status of the PO (e.g., pending, completed, canceled).
    - quality_rating: FloatField - Rating given to the vendor for this PO (nullable).
    - issue_date: DateTimeField - Timestamp when the PO was issued to the vendor.
    - acknowledgment_date: DateTimeField, nullable - Timestamp when the vendor acknowledged the PO.

3. Historical Performance Model

This model optionally stores historical data on vendor performance, enabling trend analysis.

- Fields:
    - vendor: ForeignKey - Link to the Vendor model.
    - date: DateTimeField - Date of the performance record.
    - on_time_delivery_rate: FloatField - Historical record of the on-time delivery rate.
    - quality_rating_avg: FloatField - Historical record of the quality rating average.
    - average_response_time: FloatField - Historical record of the average response time.
    - fulfillment_rate: FloatField - Historical record of the fulfilment rate.

These models form the backbone of the Vendor Management System, enabling comprehensive tracking and analysis of vendor performance over time. The performance metrics are updated based on interactions recorded in the Purchase Order model

Backend Logic

Backend Logic for Performance Metrics

On-Time Delivery Rate:
- Calculated each time a PO status changes to 'completed'.
- Logic: Count the number of completed POs delivered on or before delivery_date and divide by the total number of completed POs for that vendor.

Quality Rating Average:
- Updated upon the completion of each PO where a quality_rating is provided.
- Logic: Calculate the average of all quality_rating values for completed POs of the vendor.

Average Response Time:
- Calculated each time a PO is acknowledged by the vendor.
- Logic: Compute the time difference between issue_date and acknowledgment_date for each PO, and then find the average of these times for all POs of the vendor.

Fulfilment Rate:
- Calculated upon any change in PO status.
- Logic: Divide the number of successfully fulfilled POs (status 'completed' without issues) by the total number of POs issued to the vendor.

API Endpoint Implementation

- Vendor Performance Endpoint (GET /api/vendors/{vendor_id}/performance):
    - Retrieves the calculated performance metrics for a specific vendor.
    - Should return data including on_time_delivery_rate, quality_rating_avg, average_response_time, and fulfillment_rate.
- Update Acknowledgment Endpoint:
    - While not explicitly detailed in the previous sections, consider an endpoint like POST /api/purchase_orders/{po_id}/acknowledge for vendors to acknowledge POs.
    - This endpoint will update acknowledgment_date and trigger the recalculation of average_response_time.

Additional Technical Considerations

- Efficient Calculation: Ensure that the logic for calculating metrics is optimised to handle large datasets without significant performance issues.
- Data Integrity: Include checks to handle scenarios like missing data points or division by zero in calculations.
- Real-time Updates: Consider using Django signals to trigger metric updates in real-time when related PO data is modified.

Technical Requirements

- Use the latest stable version of Django and Django REST Framework.
- Adhere to RESTful principles in API design.
- Implement comprehensive data validations for models.
- Utilise Django ORM for database interactions.
- Secure API endpoints with token-based authentication.
- Follow PEP 8 style guidelines for Python code.
- Document each API endpoint thoroughly.

Deliverables

- Complete source code in a public Git repository.
- A README file with setup instructions and details on using the API endpoints.
- A test suite demonstrating the functionality and reliability of the endpoints.

Submission Guidelines

- Host the code on a platform like GitHub or GitLab.
- Ensure the README is clear for easy setup and testing.
- Include instructions on how to run the test suite.

This assignment tests your ability to create a functional Django-based system for vendor management, integrating aspects of data handling, API development, and basic performance metric calculations.