Technical Task Definition: Creating Endpoints using FastAPI and Clean Architecture

Overview

Develop a set of API endpoints for managing the registration of buyers (Bayers) using FastAPI and adhering to clean architecture principles. The registration involves collecting detailed information about the buyer's preferences and conditions for trade.

Design database

Comprehensive Database Design Requirements

Functional Requirements

1. **Buyer Management**:
   - Store buyer details including unique identifiers, preferences, and profile information.
2. **Chat Management**:
   - Store chat sessions between buyers.
   - Track participants in each chat session.
3. **Message Management**:
   - Store messages within each chat session.
   - Support both text messages and media files (images, documents).
4. **Media Storage**:
   - Efficiently store and retrieve media files associated with messages and buyer profiles.

Non-Functional Requirements

1. **Scalability**:
   - The database should handle high volumes of data efficiently.
   - Support sharding or partitioning if necessary.
2. **Performance**:
   - Optimize read and write operations for buyer data, chat messages, and media.
   - Support indexing for fast retrieval of recent chats, messages, and buyer profiles.
3. **Security**:
   - Ensure data encryption at rest and in transit.
   - Implement access controls and authentication mechanisms.

4. **Reliability**:
    - Ensure data integrity and consistency.
    - Implement backup and recovery procedures.

Requirements

Buyer Registration Details

1. **Price Range**:
    - Minimum price
    - Maximum price
2. **Location**:
    - Country
    - City
3. **Import Countries**:
    - List of countries the buyer imports from
4. **Product Segment Category**:
    - Segment (e.g., luxury, mass market, both)
5. **Commission Rate**:
    - Minimum percentage
    - Maximum percentage
6. **Delivery Options**:
    - Delivery included (boolean)
    - Delivery not included (boolean)
    - Delivery negotiable (boolean)
7. **Payment Options**:
    - Cryptocurrency
    - Card payment
    - On delivery
    - 100% prepayment
8. **Prepayment Percentage**:
    - Percentage value

Endpoint Specifications

1. Register Buyer

- **Endpoint**: `/register_buyer`
- **Method**: POST
- **Description**: Registers a new buyer with the specified details.
- **Request Body**:

```
{
```

```
"price_range": {
  "min": float,
  "max": float
},
"location": {
  "country": str,
  "city": str
},
"import_countries": [str],
"product_segment": str,
"commission_rate": {
  "min": float,
  "max": float
},
"delivery_options": {
  "included": bool,
  "not_included": bool,
  "negotiable": bool
},
"payment_options": {
  "cryptocurrency": bool,
  "card_payment": bool,
  "on_delivery": bool,
  "prepayment_100": bool
},
"prepayment_percentage": float
```

```
}
```

- **Response**:
```
{
  "status": "success",
  "buyer_id": int
}
```

2. Get Buyer Information

- **Endpoint**: `/buyer/{buyer_id}`
- **Method**: GET
- **Description**: Retrieves the information of a registered buyer.
- **Response**:
```
{
  "buyer_id": int,
  "price_range": {
    "min": float,
    "max": float
  },
  "location": {
    "country": str,
    "city": str
  },
  "import_countries": [str],
  "product_segment": str,
  "commission_rate": {
    "min": float,
    "max": float
  },
  "delivery_options": {
    "included": bool,
    "not_included": bool,
    "negotiable": bool
  },
  "payment_options": {
    "cryptocurrency": bool,
    "card_payment": bool,
    "on_delivery": bool,
    "prepayment_100": bool
  },
  "prepayment_percentage": float
}
```

3. Update Buyer Information

- **Endpoint**: `/buyer/{buyer_id}`
- **Method**: PUT
- **Description**: Updates the information of an existing buyer.
- **Request Body**: Same as the request body for the `Register Buyer` endpoint.
- **Response**:
  ```
  {
    "status": "success",
    "buyer_id": int
  }
  ```

4. Delete Buyer

- **Endpoint**: `/buyer/{buyer_id}`
- **Method**: DELETE
- **Description**: Deletes a registered buyer from the system.
- **Response**:
  ```
  {
    "status": "success",
    "message": "Buyer deleted successfully."
  }
  ```

Implementation Notes

1. **Clean Architecture**: Ensure that the implementation follows clean architecture principles, separating concerns into distinct layers such as presentation, application, domain, and infrastructure.
2. **Data Validation**: Utilize Pydantic models for request body validation to ensure data integrity and proper error handling.
3. **Error Handling**: Implement comprehensive error handling to manage scenarios like invalid input data, resource not found, and internal server errors.
4. **Security Considerations**:
   - Ensure endpoints are secured using appropriate authentication and authorization mechanisms.
   - Use HTTPS to encrypt data in transit.
5. **Testing**: Write unit and integration tests to verify the functionality of each endpoint. Employ mock objects where necessary to isolate tests.
6. **Documentation**: Use tools like Swagger (integrated with FastAPI) to auto-generate API documentation. Ensure that all endpoints, request parameters, and responses are well-documented.

Example Models and Schemas

Pydantic Models

Buyer Model

```
from pydantic import BaseModel, Field

from typing import List



class PriceRange(BaseModel):

  min: float

  max: float



class Location(BaseModel):

  country: str

  city: str



class CommissionRate(BaseModel):

  min: float

  max: float



class DeliveryOptions(BaseModel):

  included: bool

  not_included: bool

  negotiable: bool



class PaymentOptions(BaseModel):
  cryptocurrency: bool

  card_payment: bool

  on_delivery: bool

  prepayment_100: bool




class Buyer(BaseModel):

  price_range: PriceRange

  location: Location

  import_countries: List[str]

  product_segment: str

  commission_rate: CommissionRate

  delivery_options: DeliveryOptions

  payment_options: PaymentOptions

  prepayment_percentage: float
```

Chat Feature Requirements

Functional Requirements

1. **Send Messages**:
    - Buyers can send text messages.
    - Buyers can send media files (images, documents).
2. **Receive Messages**:
    - Buyers can receive text messages.
    - Buyers can receive media files (images, documents).
3. **List Chats**:
    - List all chat conversations for a buyer.
4. **List Messages in a Chat**:
    - List all messages in a specific chat conversation.

Non-Functional Requirements

1. **Scalability**:

    - The system should be able to handle a high volume of messages and media files.
    - Use a message broker like RabbitMQ or Kafka for handling real-time messaging at scale.
2. **Storage**:
    - Use cloud storage (e.g., AWS S3) to store media files.
    - Use a scalable database (e.g., MongoDB, PostgreSQL) to store chat messages.
3. **Security**:
    - Ensure messages and media files are encrypted in transit using HTTPS.