Data Transfer Object (DTO) under Database Design

1. User Authentication DTOs (Login/Registration)

1.1 User Registration DTO

This DTO is used when a new user registers an account.

```
class UserRegistrationDTO:
    """DTO for user registration requests"""
    def __init__(self, email: str, password: str, mfa_enabled: bool = False):
        self.email = email
        self.password = password
        self.mfa_enabled = mfa_enabled
```

1.2 User Login DTO

This DTO is used for the login process, where the user submits their credentials.

```
class UserLoginDTO:
    """DTO for user login requests"""
    def __init__(self, email: str, password: str, mfa_code: Optional[str] = None):
        self.email = email
        self.password = password
        self.mfa_code = mfa_code
```

1.3 User Authentication Response DTO

This DTO is used when returning the authentication token after login.

```
class AuthResponseDTO:
    """DTO for authentication responses"""
    def __init__(self, access_token: str, refresh_token: str, expires_in: int, token_type: str = "bearer",
mfa_required: bool = False):
        self.access_token = access_token
        self.refresh_token = refresh_token
        self.token_type = token_type
        self.expires_in = expires_in
        self.mfa_required = mfa_required
```

1.4 Token Information DTO

This DTO is used store the information of tokens.

```
class TokenInfoDTO:
    """DTO for token information"""
    def __init__(self, scopes: List[str], expires_at: datetime, created_at: datetime):
        self.scopes = scopes
        self.expires_at = expires_at
        self.created_at = created_at
```

2. Device Management DTOs

2.1 Device Location DTO

This DTO represents the real-time IoT data sent by devices.

```
class DeviceLocationDTO:
    """DTO for device location coordinates"""
    def __init__(self, latitude: float, longitude: float):
        self.latitude = latitude
        self.longitude = longitude
```

2.2 Device Registration DTO

This DTO is used to register a new IOT device in the system.

```
class DeviceRegistrationDTO:
    """DTO for registering new devices"""
    def __init__(self, name: str, device_type: str,  # 'temperature', 'pressure', or 'motion'
        location: Optional[DeviceLocationDTO] = None
    ):
        self.name = name
        self.type = device_type
        self.location = location
```

2.3 Device Response DTO

This DTO contains complete information about a registered device.

```
class DeviceResponseDTO:
    """DTO for device information responses"""
    def __init__(self, device_id: UUID, user_id: UUID, name: str, device_type: str, registered_at: datetime,
location: Optional[Dict[str, float]] = None, status: str = "active"):
        self.device_id = device_id
        self.user_id = user_id
        self.name = name
        self.type = device_type
        self.registered_at = registered_at
        self.location = location
        self.status = status
```

2.4 Device List DTO

This DTO lists all the registered device with pagination.

```
class DeviceListDTO:
    """DTO for listing devices with pagination"""
    def __init__(self, devices: List[DeviceResponseDTO], total_count: int):
        self.devices = devices
        self.total_count = total_count
```

3. Sensor Data DTOs

3.1 Sensor Stream DTO

This DTO contains stream information for sensors.

```
class SensorStreamDTO:
    """DTO for sensor stream information"""
    def __init__(self, stream_id: UUID, device_id: UUID, metric_name: str, sampling_rate: int, created_at:
datetime):
        self.stream_id = stream_id
        self.device_id = device_id
        self.metric_name = metric_name
        self.sampling_rate = sampling_rate
        self.created_at = created_at
```

3.2 Sensor Data Point DTO

This DTO represents a single sensor reading with timestamp.

```
class SensorDataPointDTO:
    """DTO for individual sensor data points"""
    def __init__(self, timestamp: datetime, value: float, normalized_value: Optional[float] = None):
        self.timestamp = timestamp
        self.value = value
        self.normalized_value = normalized_value
```

3.3 Sensor Data Point DTO

This DTO contains a collection of sensor reading for visualization.

```
class SensorDataResponseDTO:
    """DTO for sensor data responses"""
    def __init__(self, stream: SensorStreamDTO, data: List[SensorDataPointDTO], time_range: Dict[str,
datetime], stats: Dict[str, float]):
        self.stream = stream
        self.data = data
        self.time_range = time_range
        self.stats = stats
```

3.4 Bulk Data Upload DTO

This DTO represent for bulk sensor data uploads.

```
class BulkDataUploadDTO:
    """DTO for bulk sensor data uploads"""
    def __init__(self, device_id: UUID, metric_name: str, readings: List[Dict[datetime, float]]):
        self.device_id = device_id
        self.metric_name = metric_name
        self.readings = readings
```

4. System DTO (CSV/JSON Export)

4.1 Correlation Result DTO

This DTO contains the correlation analysis result.

```
class CorrelationResultDTO:
    """DTO for correlation analysis results"""
    def __init__(self, correlation_id: UUID, stream_a: UUID, stream_b: UUID, coefficient: float, window_start:
datetime, window_end: datetime, algorithm: str):
        self.correlation_id = correlation_id
        self.stream_a = stream_a
        self.stream_b = stream_b
        self.coefficient = coefficient
        self.window_start = window_start
        self.window_end = window_end
        self.algorithm = algorithm
```

4.2 Anomaly Detection DTO

This DTO contains anomaly detection results.

```
class AnomalyDetectionDTO:
    """DTO for anomaly detection results"""
    def __init__(self, anomaly_id: UUID, stream_id: UUID, detected_at: datetime, anomaly_type: str,
raw_value: float, confidence_score: float, status: str = "pending"):
        self.anomaly_id = anomaly_id
        self.stream_id = stream_id
        self.detected_at = detected_at
        self.anomaly_type = anomaly_type
        self.raw_value = raw_value
        self.confidence_score = confidence_score
        self.status = status
```

5. System DTOs

5.1 File Upload Request DTO

This DTO represents requests for file uploading.

```
class FileUploadRequestDTO:
    """DTO for file upload requests"""
    def __init__(self, file_format: str,  # 'csv', 'json', or 'xlsx'
        expected_columns: Optional[List[str]] = None
    ):
        self.format = file_format
        self.expected_columns = expected_columns
```

5.2 File Upload Response DTO

This DTO represents response for file uploading.

```
class FileUploadResponseDTO:
    """DTO for file upload responses"""
    def __init__(self, file_id: UUID, status: str, upload_url: Optional[str] = None, fields: Optional[Dict[str, str]] = None):
        self.file_id = file_id
        self.status = status
        self.upload_url = upload_url
        self.fields = fields
```

5.3 File Upload Response DTO

This DTO represents information for system status.

```
class SystemStatusDTO:
    """DTO for system status information"""
    def __init__(self, database: bool, storage: bool, analytics_engine: bool, last_updated: datetime):
        self.database = database
        self.storage = storage
        self.analytics_engine = analytics_engine
        self.last_updated = last_updated
```

6. Chatbot Interaction DTOs

6.1 Chat Message DTO

This DTO represents the messages exchanged with the chatbot.

```
class ChatMessageDTO:
    """DTO for chat messages"""
    def __init__(self, message_id: UUID, content: str, is_bot: bool, sent_at: datetime):
        self.message_id = message_id
        self.content = content
        self.is_bot = is_bot
        self.sent_at = sent_at
```

6.2 Chat Session DTO

This DTO represents the chat session.

```
class ChatSessionDTO:
    """DTO for chat sessions"""
    def __init__(self, session_id: UUID, user_id: Optional[UUID], started_at: datetime, messages: List[ChatMessageDTO], ended_at: Optional[datetime] = None, escalation_level: int = 0):
        self.session_id = session_id
        self.user_id = user_id
        self.started_at = started_at
        self.ended_at = ended_at
        self.messages = messages
        self.escalation_level = escalation_level
```

7. Error Handling DTOs

7.1 Error Response DTO

This DTO represents the error messages returned from the system.

```
class ErrorResponseDTO:
   """DTO for error responses"""
   def __init__(self, error_code: str, message: str, details: Optional[Dict[str, Any]] = None, timestamp:
datetime = datetime.now()):
      self.error_code = error_code
      self.message = message
      self.details = details
      self.timestamp = timestamp
```

8. Conclusion

The **DTOs (Data Transfer Objects)** outlined here ensure that data is transferred efficiently and securely between the frontend and backend layers of the **IoT Web Application**. The DTOs outlined in this document provide a structured approach to data transfer between system components. Each DTO has a clearly defined purpose and usage context, includes type hints for better code reliability, contains detailed field descriptions, matches the database schema requirements, and supports both required and optional fields. These DTOs ensure consistent data representation across the application layers while maintaining flexibility for future enhancements.