**ATM Monitoring System - Technical Design Document**

**1. System Design**

**Objective**

The ATM Monitoring System monitors the status and behavior of ATMs in real-time across a bank’s network. This application supports:

* Authentication and authorization for access control
* Customer transaction logging (e.g., deposits, withdrawals, balance inquiries)
* Failure detection (e.g., hardware and software failures)
* Downloading camera images/videos for specified time ranges

**Technology Stack**

* **Backend:** Java, Spring Boot
* **Database:** MySql (primary data storage), H2 (optional for local testing)
* **API Documentation:** OpenAPI 3.1

**Architecture**

* **RESTful API**: To provide a structured way for other services to interact with ATM monitoring data.

**2. Component Design**

**Components Overview**

1. **Transaction Service**
   * Records customer transactions, with capabilities for filtering transactions by type (e.g., deposits, withdrawals).
2. **Failure Logging Service**
   * Tracks and logs hardware/software failures and sends alerts for critical issues.
3. **Media Management Service**
   * Manages access to ATM camera images and video, allowing filtered access by date and time ranges.
4. **ATM Monitoring Controller**
   * Central controller that handles requests and routes them to the appropriate services for execution.

**Service Interactions**

* **Client applications** interact with the ATM Monitoring API through secured endpoints.
* **Database services** handle transaction logging, failure logs, and media storage.
* **External services** (e.g., notification systems) can be integrated for alerting based on failure logs.

**3. API Design**

**API Endpoints**

| **Endpoint** | **Method** | **Description** | **Authentication Required** |
| --- | --- | --- | --- |
| /api/transactions | GET | Retrieves transactions in the last 24 hours | Yes |
| /api/transactions/{type} | GET | Retrieves transactions filtered by type (deposit, withdrawal) | Yes |
| /api/failures | GET | Lists system or device failures | Yes |
| /api/media/download | GET | Downloads camera media by specified time range | Yes |
| /api/transactions/count | GET | Returns count of unique customers in the last 24 hours | Yes |

**Sample API Requests**

* **Retrieve Transactions**:
  + **Request**: GET /api/transactions?dateFrom=2024-10-01&dateTo=2024-10-02
  + **Response**: [{ "type": "deposit", "amount": 100, "timestamp": "2024-10-01T10:00:00" }]
* **Download Camera Media**:
  + **Request**: GET /api/media/download?start=2024-10-01T09:00:00&end=2024-10-01T11:00:00
  + **Response**: Media file (e.g., MP4)

**4. Activity Flow Diagrams**

**Login and Authentication Flow**

**1. User Login → /api/atm/... (no /login endpoint needed).**

**2. Send Credentials with Each Request: Users access the API endpoints by including the admin username and admin password in the HTTP Basic Authentication header.**

**Transaction Retrieval Flow**

1. **User Authenticates** → **Uses Token** → /api/transactions → **Fetch Transaction Data** → **Return Response**

**Failure Log and Alerting Flow**

1. **Error Detected in ATM** → **Log Failure in MySql** → **Check Severity** → **Trigger Notification (if critical)**

**Media Download Flow**

1. **User Authenticates** → **Requests Media** → /api/media/download → **Fetch Media Data** → **Return Media File**

**5. Data Model Design**

**Collections and Schema**

1. **Transaction Collection**
   * Stores transaction details, including transaction type, amount, timestamp, and ATM ID.

json

Copy code

{

"transactionId": "string",

"atmId": "string",

"type": "string", // e.g., "deposit", "withdrawal"

"amount": "double",

"timestamp": "datetime"

}

1. **Failure Log Collection**
   * Logs failures, including a description, timestamp, ATM ID, and error type (hardware, software).

json

Copy code

{

"failureId": "string",

"atmId": "string",

"type": "string", // e.g., "hardware", "software"

"description": "string",

"timestamp": "datetime"

}

1. **Media Collection**
   * Holds metadata for camera media (e.g., start and end times, file paths).

json

Copy code

{

"mediaId": "string",

"atmId": "string",

"mediaPath": "string",

"timestamp": "datetime"

}

**6. Data Model Examples**

**Example Documents**

* **Transaction Example**

json

Copy code

{

"transactionId": "txn123",

"atmId": "ATM001",

"type": "deposit",

"amount": 500.00,

"timestamp": "2024-10-01T10:00:00"

}

* **Failure Log Example**

json

Copy code

{

"failureId": "fail456",

"atmId": "ATM002",

"type": "hardware",

"description": "Cash dispenser error",

"timestamp": "2024-10-01T10:30:00"

}

* **Media Example**

json

Copy code

{

"mediaId": "media789",

"atmId": "ATM001",

"mediaPath": "/media/2024/10/01/video1.mp4",

"timestamp": "2024-10-01T09:45:00"

}

**7. API Documentation**

To document these APIs, configure **Swagger** in Spring Boot:

1. **Add Swagger Dependency in pom.xml:**

xml

Copy code

<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>

<version>2.0.2</version>

</dependency>

1. **Access API Documentation:**
   * Run the application and visit http://localhost:8080/swagger-ui.html to view and interact with the API documentation.