**Build a User Transaction Service for a hypothetical financial application**

**Overview**

The service will handle user transactions, specifically allowing users to:  
- Create new accounts.  
- Make transactions (credits and debits).  
- Fetch transaction history.  
- Fetch account balance.  
- Ensure concurrency safety when multiple transactions are processed for the same account.  
- Support pagination and filtering for transaction history.  
- Implement a basic fraud detection mechanism.  
  
**System Requirements**

Tech Stack:  
- Java 17+ (or Java 21+ if familiar).  
- Spring Boot (as the framework for building the microservice).  
- PostgreSQL/MySQL.  
- [optional] Docker.  
- [optional] JUnit.  
  
**Functional Requirements**

Account Management:  
- POST /accounts: Create a new account with an initial balance.  
- GET /accounts/{accountId}: Get the account details and balance.  
- PUT /accounts/{accountId}: Edit the account details only.  
- DELETE /accounts/{accountId}: Suspend the account (do not remove it physicaly).

Transaction Management:  
- POST /accounts/{accountId}/transactions: Make a transaction.  
The transaction should have details like amount, type (in or out), and timestamp.  
Ensure the account balance cannot go negative.  
Handle concurrent transactions safely.

Transaction History:  
-GET /accounts/{accountId}/transactions: Get a paginated list of transactions, with optional filters (e.g., date range, type).

Fraud Detection:  
-Implement basic fraud detection logic: if a user tries to withdraw more than $10,000 in one day, flag it for review.

**Non-Functional Requirements**- Ensure thread safety when multiple transactions are processed on the same account.  
- Paginate large transaction history requests.  
- Design the service for high availability and fault tolerance (e.g., retry failed transactions).

**Bonus Challenges**  
- [optional] Use Docker to package the service for easy deployment.  
- [optional] Security: Secure the API using JWT-based authentication.  
- [optional] Testing: Write unit tests and integration tests to cover various edge cases (e.g., concurrent transactions, fraud detection).

**Deliverables**  
- A fully functional Spring Boot microservice.  
- Postman collection for testing API endpoints.  
- [optional] Dockerfile and instructions to run the service.  
- [optional] Unit tests covering the main functionality.