Senior Backend Engineer – Technical Assessment

Task: Secure & Scalable Payment Withdrawal Module

Objective

Design and implement a secure, scalable, and concurrency-safe withdrawal module similar to real-world financial systems (e.g., Google Pay / Paytm). The focus is on backend architecture, data integrity, concurrency handling, security, and robustness — not UI.

Tech Stack (Mandatory)

- **Backend:** Node.js (latest LTS)
- **Database:** MongoDB
- **ORM/ODM:** Mongoose (or equivalent ODM)
- **Payment Gateway:** Mocked / bypassed (no external API integration required)

Functional Requirements

1. **User Wallet & Balance**
   - Each user has a wallet balance stored in the database
   - Balance accuracy must be guaranteed under high concurrency
   - Wallet balance must never go negative
2. **Withdrawal Flow**
   - User initiates a withdrawal request (user_id, withdrawal_amount, destination)
   - Validate user status and sufficient balance
   - Process withdrawal atomically (balance deduction + transaction record)
   - Status flow: pending → processing → success / failed
3. **Concurrency & Race Condition Handling**
   - Handle thousands of simultaneous withdrawal requests
   - Prevent double spending and lost updates
   - Ensure idempotency where applicable
   - Multiple concurrent requests for the same user must be safely handled
4. **Security Requirements**
   - SQL Injection prevention (and injection-safe query practices for MongoDB)
   - Mass assignment protection / input whitelisting
   - Replay / duplicate requests prevention
   - Data tampering prevention (amount, user_id)
   - Secure state transitions and transaction integrity
5. **Database Design**

Design appropriate collections such as users, wallets, withdrawals, and transaction_logs. Ensure proper indexing, financial precision correctness, and normalization/structure suited to MongoDB.

6. **Transaction Logging (Mandatory)**

- Maintain a detailed transaction log for every balance-affecting operation
- Logs must be immutable (insert-only)
- Include before & after balance, transaction type, status, timestamp, and reference ID
- Logs must support auditing, debugging, and reconciliation

7. **Scalability & Reliability**

- Design for horizontal scalability
- Support background processing (queues/jobs)
- Safe retry mechanisms for failures
- Extendable for real payment gateway integration

Non-Functional Requirements

- Clean, production-grade code
- Proper separation of concerns (Controller, Service, Repository, etc.)
- Logging of critical events
- Graceful error handling with zero partial updates

Deliverables

- Code ZIP file containing the complete Node.js project
- Database setup scripts (and/or migration equivalents if used)
- Any seeders or setup scripts required to run the project locally
- README.md explaining architecture, concurrency handling, security decisions, and assumptions

Evaluation Focus

- Concurrency & transaction safety
- Security depth and threat modeling
- Locking / transactional strategy (MongoDB transactions, optimistic concurrency, etc.)
- Code structure and maintainability
- Ownership and system-level thinking

Time Expectation

Recommended: 6–10 hours

Focus on correctness, safety, and scalability — not UI or polish.

Note

This task mirrors real-world financial system complexity. There is no single correct solution — engineering depth and decision-making matter most.