Non-Functional Requirements (NFRs)

For a retail chain operating across 3000 stores in multiple countries, the system must meet high standards in scalability, availability, performance, security, and maintainability.

Below are the expected NFRs and how the proposed architecture addresses them.

1. Scalability

- **Requirement**
  - The system must handle:
    - Pricing feeds from 3000 stores
    - High concurrent users
    - Large CSV uploads
    - Growth in product catalog
- **Design Approach**
  - Stateless Node.js backend (horizontal scaling possible)
  - Load balancer in front of API servers
  - MongoDB cluster with replication and sharding
  - Indexed search fields (storeId, sku, date)
  - Dockerized deployment for auto-scaling
- **Result**
  - System can scale horizontally without redesign.

2. Performance

- **Requirement**
  - Fast search response (<2 seconds)
  - Efficient bulk CSV processing
  - Minimal UI load time
  - Low latency across countries
- **Design Approach**
  - Angular SPA for responsive UI
  - REST API with optimized queries
  - MongoDB indexing strategy
  - Stream-based CSV parsing (avoids memory overload)
  - CDN-ready frontend deployment
- **Result**
  - Optimized read/write operations even with high data volume.

3. Availability & High Availability (HA)

- **Requirement**
  - 24/7 availability
  - No single point of failure
  - Support multiple regions
- **Design Approach**
  - MongoDB Replica Set
  - Load-balanced Node.js instances
  - Cloud-based deployment
  - Containerization with Docker
- **Result**
  - Failure of one server does not impact overall system availability.

4. Reliability & Data Integrity

- **Requirement**
  - No data corruption during CSV upload
  - Accurate pricing records
  - Validation before persistence
- **Design Approach**
  - Backend validation using express-validator
  - Frontend validation using Angular Reactive Forms
  - Centralized error handling middleware
  - Transaction-safe update operations
- **Result**
  - High data integrity and consistent records.

5. Security

- **Requirement**
  - Protect pricing data
  - Prevent unauthorized access
  - Secure file uploads
  - Prevent injection attacks
- **Design Approach**
  - Input validation at API layer
  - CORS configuration
  - Environment variables via dotenv
  - File size limits on uploads
  - Sanitization of request inputs

    - HTTPS deployment (recommended)
- **Future enhancements:**
    - JWT authentication
    - Role-based access control
    - Rate limiting

6. Maintainability

- **Requirement**
    - Easy to update features
    - Clean separation of concerns
    - Easy onboarding for developers
- **Design Approach**
    - Layered backend architecture (Controller → Service → Model)
    - Angular Standalone modular components
    - Clean folder structure
    - RESTful API standards
- **Result**
    - Codebase is modular and easy to extend.

7. Observability & Monitoring

- **Requirement**
    - Track API usage
    - Debug production issues
    - Monitor failures
- **Design Approach**
    - Morgan for HTTP logging
    - Centralized error middleware
- **Result**
    - Operational transparency and traceability.

8. Usability

- **Requirement**
    - Easy CSV upload process
    - Simple search filters
    - Editable pricing grid
    - Responsive UI
- **Design Approach**
    - Angular SPA
    - Reactive Forms

    - Pagination support
    - Clear error messages
- **Result**
    - Improved user productivity across stores.

9. Compliance & Auditability

- **Requirement**
    - Track pricing changes
    - Support audit requirements
- **Design Approach**
    - Timestamps (createdAt, updatedAt)
    - Logging for updates
    - Extendable audit trail collection