Design Decisions

1. Frontend Architecture Decision

- **Decision:** Use Angular (Standalone SPA architecture) for the frontend.
- **Reason**:
    - Single Page Application ensures fast user experience.
    - Standalone components reduce NgModule complexity.
    - Reactive Forms support dynamic validation and editing.
    - Strong TypeScript typing improves maintainability.
    - Built-in HttpClient simplifies REST API integration.

2. Backend Framework Decision

- **Decision**: Use Node.js with Express.js.
- **Reason**:
    - Non-blocking I/O model suitable for handling large CSV uploads.
    - Lightweight and flexible.
    - Easy middleware integration.
    - High concurrency support.
    - Matches existing MEAN stack expertise.

3. Database Selection

- **Decision**: Use MongoDB (NoSQL).
- **Reason**:
    - Flexible schema for pricing feeds.
    - High write throughput (important for bulk CSV uploads).
    - Horizontal scaling via sharding.
    - Built-in replication for high availability.
    - Suitable for multi-country distributed deployment.

4. CSV Upload Handling

- **Decision**: Use multer for file upload and csv-parser for stream-based parsing.
- **Reason**:
    - Stream-based processing avoids loading entire file into memory.
    - Supports large file uploads.
    - Improves performance and scalability.

5. Validation Strategy

- **Decision**: Use express-validator for backend validation and Angular Reactive Forms for frontend validation.
- **Reason**:
  - Double-layer validation improves data integrity.
  - Prevents malicious or malformed data.
  - Ensures clean data before persistence.

6. Logging & Monitoring

- **Decision**: Use morgan for HTTP logging and centralized error handling middleware.
- **Reason**:
  - Tracks incoming requests.
  - Helps debugging in production.
  - Supports audit trails.

7. API Design Pattern

- **Decision**: Follow RESTful API design principles.
- **Endpoints**:
  - POST /api/pricing/upload
  - GET /api/pricing
  - PUT /api/pricing/:id
- **Reason**:
  - Standardized communication.
  - Stateless design.
  - Easy frontend-backend integration.
  - Supports horizontal scaling.

8. Data Indexing Strategy

- **Decision**: Create indexes on:
  - storeId
  - sku
  - date
  - Compound index (storeId + sku + date)
- **Reason**:
  - Optimized search queries.
  - Faster filtering across 3000 stores.
  - Improved query performance.

9. Scalability Design

- **Decision**: Design backend as stateless service.
- **Reason**:
  - Enables horizontal scaling.
  - Works behind load balancer.
  - No session dependency.

10. Error Handling Strategy

- **Decision**: Centralized error-handling middleware in Express.
- **Reason**:
  - Uniform error responses.
  - Cleaner controller logic.
  - Easier debugging.

11. Security Considerations

- **Decisions**:
  - Enable CORS properly.
  - Input validation on all APIs.
  - Environment variable configuration via dotenv.
  - Limit file upload size.
- **Future Enhancements**:
  - JWT authentication
  - Role-based access control
  - Rate limiting

12. Folder Structure Decision (Backend)

- **Decision**: Use layered separation:
  - controllers/
  - services/
  - models/
  - routes/
  - middlewares/
- **Reason**:
  - Separation of concerns
  - Better maintainability
  - Testability