MongoDB Inventory Management System Documentation

Database Overview

The system uses MongoDB, a NoSQL document database, to manage inventory data. The primary collection is `products` within the `inventory_db` database.

Database Connection

- Connection is established via MongoDB Atlas (cloud-based)
- Connection string: `mongodb+srv://HelloBeta:IV7wqssYdgwijpQx@cluster0.nnecnay.mongodb.net/inventory_db`
- Environment variables are supported via `dotenv` for configuration flexibility

Collection Schema

The `products` collection has a flexible schema, but typically includes:

- `_id`: Unique identifier (MongoDB ObjectId)
- `name`: Product name (string)
- `price`: Product price (number)
- `stock`: Available quantity (number)
- `category`: Product category (string)
- Optional additional fields as needed

API Endpoints

CRUD Operations

Endpoint	Method	Description
`/products`	POST	Create single or multiple products
`/products`	GET	Retrieve all products
`/products/category/<category>`	GET	Get products by category
`/products/price`	GET	Get products within price range
`/products/search`	GET	Search products by name (fuzzy)
`/products/paginated`	GET	Get paginated product results
`/products/<product_id>`	PATCH	Update a specific product
`/products/<product_id>`	DELETE	Delete a specific product

Aggregation Operations

Endpoint	Method	Description
`/products/total-value`	GET	Calculate total inventory value
`/products/category-count`	GET	Count products per category
`/products/average-price`	GET	Average price per category
`/products/most-expensive-category`	GET	Category with highest total value
`/products/value-category`	GET	Value of all categories (sorted)
`/products/value-category-less-than`	GET	Products in category below price threshold
`/products/low-stock`	GET	Products with stock below threshold
`/products/group-by-category`	GET	Products grouped by category

Implementation Details

Key Features Implemented

1. **Flexible Document Structure**
   - MongoDB's schema-less design allows for product variations
   - Additional fields can be added without schema migrations
2. **Pagination Support**
   - Implemented with `skip()` and `limit()` for handling large datasets
   - Adjustable page size and current page parameters
3. **Search Functionality**
   - Regex-based search with case insensitivity (`$regex` with `$options: "i"`)
   - Dedicated index on `name` field for performance optimization

4. **Advanced Aggregation Pipelines**
   - Total inventory value calculation using `$multiply` and `$sum`
   - Category-based grouping and analysis
   - Price-based filtering with thresholds
5. **Database Indexing**
   - Dedicated endpoint (`/setup-indexes`) to create performance-optimizing indexes
   - Indexes on frequently queried fields: `name`, `category`, and `status`
6. **Bulk Operations**
   - Supports bulk insertion of multiple products in a single request
   - Efficient for batch data processing
7. **ObjectId Serialization**
   - Helper function `serialize_objectid()` to convert MongoDB ObjectIds to strings
   - Ensures proper JSON serialization across all endpoints

Implementation Choices

1. **MongoDB Selection**
   - Selected for flexibility in modeling product variations
   - Schema-less design makes it easier to adapt to changing inventory requirements
   - Powerful aggregation framework for inventory analytics
2. **Flask Framework**
   - Lightweight web framework that pairs well with MongoDB
   - Easy routing and request handling for API endpoints
   - Simple to extend with additional middleware as needed
3. **RESTful Design**
   - Clear endpoint structure following REST conventions
   - Appropriate HTTP methods (GET, POST, PATCH, DELETE)
   - Consistent response formatting with JSON
4. **Error Handling**
   - HTTP status codes used appropriately (201 for creation, etc.)
   - JSON responses with descriptive messages
5. **Performance Optimization**
   - Strategic indexing for frequently queried fields

- Aggregation pipelines used for complex queries instead of application-level computation
- Pagination to handle large datasets efficiently

Additional Notes

- The system implements both basic CRUD operations and advanced analytics capabilities
- MongoDB's aggregation framework is leveraged for complex inventory analysis
- The API is designed to be extensible for additional features
- Security considerations: The connection string should be properly secured in production environments