# WareFlow - Technical Documentation

## Table of Contents

1. [System Architecture Overview](https://www.google.com/search?q=%231-system-architecture-overview)
2. [Frontend Architecture](https://www.google.com/search?q=%232-frontend-architecture)
3. [Backend Architecture](https://www.google.com/search?q=%233-backend-architecture)
4. [Database Schema](https://www.google.com/search?q=%234-database-schema)
5. [API Endpoints](https://www.google.com/search?q=%235-api-endpoints)
6. [Authentication Flow](https://www.google.com/search?q=%236-authentication-flow)
7. [Data Flow Examples](https://www.google.com/search?q=%237-data-flow-examples)
8. [Security Features](https://www.google.com/search?q=%238-security-features)
9. [Deployment Considerations](https://www.google.com/search?q=%239-deployment-considerations)
10. [Performance Optimizations](https://www.google.com/search?q=%2310-performance-optimizations)
11. [Future Enhancements](https://www.google.com/search?q=%2311-future-enhancements)
12. [Appendix: Key Technologies](https://www.google.com/search?q=%23appendix-key-technologies)

## 1. System Architecture Overview

WareFlow is a full-stack warehouse management system built with a **MERN stack (MongoDB, Express, React, Node.js)** architecture, featuring a clear separation between frontend and backend services.

### Architecture Layers

#### CLIENT LAYER (Frontend - Port 3000)

* React Application
* React Router for navigation
* Redux Toolkit for state management
* Tailwind CSS for styling

#### SERVER LAYER (Backend - Port 5000)

* Express.js Application
* RESTful API endpoints
* JWT authentication middleware
* Multer for file uploads
* Express-validator for input validation

#### DATABASE & STORAGE LAYER

* **MongoDB Atlas**
  + Users collection
  + Warehouses collection
  + Products collection
* **File Storage**
  + Cloudinary CDN (User profile images)
  + Local File System (Product images)

## 2. Frontend Architecture

### Technology Stack

* **Framework:** React 18
* **Routing:** React Router v6
* **State Management:** Redux Toolkit
* **Styling:** Tailwind CSS
* **HTTP Client:** Fetch API
* **Icons:** Lucide React

### Directory Structure

src/  
├── components/  
│ ├── auth/  
│ ├── layout/  
│ ├── product/  
│ ├── profile/  
│ ├── shared/  
│ └── warehouse/  
├── pages/  
│ ├── Home.jsx  
│ ├── Login.jsx  
│ ├── Signup.jsx  
│ └── Dashboard.jsx  
├── redux/  
│ ├── api/  
│ ├── slices/  
│ └── store.js  
├── utils/  
│ ├── api.js  
│ ├── constants.js  
│ └── validators.js  
├── App.jsx  
├── index.js  
└── index.css

### Redux State Management

The Redux store is structured into three main slices: auth, warehouse, and product.

* **auth:** Manages user data, token, and authentication status.
* **warehouse:** Manages the list of warehouses and the currently selected warehouse.
* **product:** Manages the list of products for the selected warehouse.

## 3. Backend Architecture

### Technology Stack

* **Runtime:** Node.js
* **Framework:** Express.js 5.x
* **Database:** MongoDB with Mongoose ODM
* **Authentication:** JWT (JSON Web Tokens)
* **Validation:** Express-validator
* **File Upload:** Multer
* **Image Storage:** Cloudinary (profiles) + Local (products)
* **Password Hashing:** bcryptjs

### Directory Structure

backend/  
├── config/  
├── controllers/  
├── middlewares/  
├── models/  
├── routes/  
├── utils/  
├── uploads/  
├── .env  
├── server.js  
└── package.json

## 4. Database Schema

### User Model

* username: String, required, unique
* email: String, required, unique
* password: String, required, hashed
* profileImageUrl: String
* profileImagePublicId: String
* timestamps: true

### Warehouse Model

* name: String, required
* address: String
* user: ObjectId, ref: 'User'
* timestamps: true

### Product Model

* name: String, required
* category: String, enum, required
* inStock: Boolean, default: true
* quantity: Number, required
* pricePerUnit: Number, required
* totalValue: Number (calculated)
* imageUrl: String, required
* warehouse: ObjectId, ref: 'Warehouse'
* timestamps: true

## 5. API Endpoints

### Authentication

|  |  |  |
| --- | --- | --- |
| **Method** | **Endpoint** | **Description** |
| POST | /api/signup | Register new user |
| POST | /api/login | Authenticate user |
| GET | /api/users/profile | Get current user (protected) |
| PUT | /api/users/profile | Update user profile |
| DELETE | /api/users/delete | Delete user account |

### Warehouses

|  |  |  |
| --- | --- | --- |
| **Method** | **Endpoint** | **Description** |
| POST | /api/warehouses | Create warehouse (protected) |
| GET | /api/warehouses | Get user's warehouses |
| PUT | /api/warehouses/:id | Update warehouse |
| DELETE | /api/warehouses/:id | Delete warehouse |

### Products

|  |  |  |
| --- | --- | --- |
| **Method** | **Endpoint** | **Description** |
| POST | /api/products | Create product (protected) |
| GET | /api/products?warehouseId=... | Get products by warehouse |
| PUT | /api/products/:id | Update product |
| DELETE | /api/products/:id | Delete product |

## 6. Authentication Flow

1. **Registration:** User signs up, password is hashed with bcryptjs, and data is saved to MongoDB.
2. **Login:** User logs in, credentials are verified, and a JWT token (10-day expiration) is generated.
3. **Storage:** The client stores the token in localStorage.
4. **Authorization:** For protected routes, the token is sent in the Authorization: Bearer <token> header.
5. **Verification:** A middleware on the backend verifies the JWT token before allowing access to the protected resource.

## 7. Data Flow Examples

### Creating a Product

1. **User Action:** User fills and submits the product creation form.
2. **Frontend:** Client-side validation runs, a FormData object is created, and a Redux action calls the POST /api/products endpoint.
3. **Backend:** The JWT is verified, Multer handles the file upload, express-validator validates input, and the controller creates the new product document in MongoDB.
4. **Frontend Update:** The Redux store is updated with the new product, causing the UI to re-render and a success notification to be shown.

## 8. Security Features

* **Authentication:** JWT tokens with bcryptjs password hashing.
* **Authorization:** Middleware-protected routes and user-specific data isolation.
* **Input Validation:** Server-side (express-validator) and client-side validation.
* **CORS:** Configured to only allow requests from the client URL.
* **File Uploads:** Restrictions on file type and size via Multer.
* **Environment Variables:** Sensitive keys and connection strings stored in a .env file.

## 9. Deployment Considerations

* **Frontend:** Build with npm run build and hosted on Vercel.
* **Backend:** Hosted on a service Render.
* **Database:** Used cloud-hosted service like MongoDB Atlas.
* **File Storage:** User profiles are on Cloudinary CDN; product images are served via Express static middleware but should be migrated to a CDN for production.

## 10. Performance Optimizations

* **Frontend:** Code splitting, lazy loading, pagination, debounced search, and state persistence.
* **Backend:** Mongoose lean queries, database indexing, connection pooling, and response compression.
* **Database:** Compound indexes and proper use of document references.

## 11. Future Enhancements

* **Analytics Dashboard:** Add charts and reports for inventory trends.
* **Export Features:** Allow exporting product lists to CSV, Excel, or PDF..
* **Advanced Search:** Integrate Elasticsearch for full-text and faceted search.
* **Email Notifications:** Send emails for low stock alerts, reports, and password resets.
* **OAuth2.0:** allowing user to continue with google.
* **Real-time Updates:** Integrate WebSockets (Socket.io) for live inventory updates.

## Appendix: Key Technologies

|  |  |
| --- | --- |
| **Category** | **Technology** |
| **Frontend** | React 18, React Router, Redux Toolkit, Tailwind CSS |
| **Backend** | Node.js, Express, Mongoose, JWT, bcryptjs, Multer |
| **Database** | MongoDB |
| **Dev Tools** | VS Code, Postman, MongoDB Compass, Git, npm |