A blue and white logo

AI-generated content may be incorrect.

OmniStock Inventory Tracker

Software Requirements Specification

*Version 2*

|  |  |
| --- | --- |
| Team Number: | 7 |
| Project Manager: | Brandon Tatum |
| Mentor: | Sandeep Ginna |
| Team Members: | Brandon Tatum, Bethany Hill, Serina Rodriguez, Dillon Hollis, Esteban Spadea |

# Revisions

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Primary Author(s)** | **Description of Version** | **Date  Completed** |
| 1.0 | Brandon Tatum, Serina Rodriguez, Dillon Hollis, Esteban Spadea, Bethany Hill | Initial draft submission (SRS v1) including project description and requirements | 03/16/2025 |
| 2.0 | Brandon Tatum, Serina Rodriguez, Dillon Hollis, Esteban Spadea, Bethany Hill | Finalized SRS combining v1 and v2. Added high-level design and resolved instructor feedback. | 04/06/2025 |

# Review History

|  |  |  |
| --- | --- | --- |
| **Reviewer** | **Version Reviewed** | **Date** |
| Professor Hong Cui | 1.0 | 03/17/2025 |

# Table of Contents

1. **Introduction**  
     1.1 Project Objectives  
     1.2 Project Scope  
     1.3 Project Overview
2. **Project Description**  
     2.1 Project Features / Functions  
     2.2 User Stories  
     2.3 Use Cases  
     2.4 Project Assumptions and Dependencies
3. **Project Collaboration and Documentation**
4. **Project Management**
5. **Requirements Specification**  
     5.1 Business Requirements  
     5.2 User Requirements  
     5.3 Functional Requirements  
     5.4 Non-Functional Requirements  
     5.5 Implementation Requirements
6. **High-Level Design**  
     6.1 Security  
       6.1.1 User Authentication (OAuth 2.0, JWT, 2FA)  
       6.1.2 Data Protection Measures  
       6.1.3 Security Testing Strategy  
       6.1.4 Summary of Security Strategy  
     6.2 Hardware  
       6.2.1 Server Requirements  
       6.2.2 RAM and Storage  
       6.2.3 Platform Compatibility  
       6.2.4 Summary of Hardware Architecture  
     6.3 User Experience (UX)  
       6.3.1 User Flow Mapping  
       6.3.2 Feedback & Error Handling  
       6.3.3 Wireframes  
       6.3.4 UX Summary  
     6.4 System Architecture  
       6.4.1 Client-Server Architecture  
       6.4.2 Architecture Diagram  
       6.4.3 Layer Responsibilities  
       6.4.4 Summary  
     6.5 Database & ERD  
       6.5.1 Entities and Attributes  
       6.5.2 ERD Diagram  
       6.5.3 Relationship Description  
       6.5.4 Schema Summary  
     6.6 Top-Level Classes  
       6.6.1 Class Identification  
       6.6.2 Class Diagram  
       6.6.3 Class Attributes & Methods  
       6.6.4 Summary  
     6.7 Data Flow and System States  
       6.7.1 Data Flow Diagram  
       6.7.2 State Transitions  
       6.7.3 Summary  
     6.8 Reports  
       6.8.1 Report Structure  
       6.8.2 Report Access  
       6.8.3 Reporting Summary  
     6.9 Internal Interfaces (Optional)  
     6.10 External Interfaces (Optional)  
     6.11 Other Outputs (Optional)  
     6.12 Configuration Data (Optional)  
     6.13 Training (Optional)

# Glossary

|  |  |
| --- | --- |
| **Term** | **Definition** |
| *2FA (Two-Factor Authentication)* | An additional layer of security requiring users to provide a one-time code (usually from an authenticator app) in addition to their password. |
| *Access Token* | A short-lived credential (JWT) that allows a user to access protected resources in the system. |
| *AES-256* | Advanced Encryption Standard with 256-bit keys; used for encrypting sensitive data at rest. |
| *Alert* | A notification generated when a specific event or threshold is triggered (e.g., low stock). |
| *API (Application Programming Interface)* | A set of rules and endpoints allowing communication between the frontend and backend systems. |
| *AWS (Amazon Web Services)* | A cloud platform used to host OmniStock services such as EC2, RDS, and S3. |
| *Backend* | The server-side logic layer responsible for processing data, handling business logic, and interfacing with the database. |
| *Barcode Scanner* | A hardware device or mobile camera used to scan barcode inputs for inventory items. |
| *bcrypt* | A secure password-hashing function that protects stored passwords using a salt. |
| *CI/CD (Continuous Integration / Continuous Deployment)* | A development practice where code changes are automatically tested and deployed. |
| *CORS (Cross-Origin Resource Sharing)* | A security feature controlling which domains are allowed to access server resources. |
| *CSRF (Cross-Site Request Forgery)* | A type of attack where unauthorized commands are transmitted from a user’s browser; prevented with CSRF tokens. |
| *CSV (Comma-Separated Values)* | A plain-text format used to export structured data like inventory reports. |
| *Dashboard* | The main interface users see after login, summarizing inventory health, alerts, and activity logs. |
| *Data Flow Diagram (DFD)* | A visual or text-based representation of how data moves through the system from input to storage and output. |
| *Database* | The persistent storage layer where all inventory, user, scan, and alert data is stored. |
| *Docker* | A platform used to containerize applications for consistent deployment across environments. |
| *Elastic Load Balancer (ELB)* | An AWS service that distributes incoming traffic across multiple instances for scalability. |
| *Endpoint* | A specific URL route in the API that performs a particular function (e.g., /login, /inventory). |
| *Entity* | A real-world concept or object (like User or InventoryItem) represented as a table or class in the system. |
| *ERD (Entity-Relationship Diagram)* | A visual or text description showing how data entities relate to each other. |
| *Frontend* | The user-facing part of the app built using React.js, responsible for displaying UI and handling user interaction. |
| *HTTPS (Hypertext Transfer Protocol Secure)* | A secure version of HTTP that encrypts data in transit between the client and server. |
| *InventoryItem* | An individual product being tracked in OmniStock, including its quantity, threshold, and barcode. |
| *JWT (JSON Web Token)* | A compact token format used for securely transmitting user identity and authorization information. |
| *ORM (Object-Relational Mapping)* | A method for querying and manipulating data using object-oriented code instead of SQL. |
| *RBAC (Role-Based Access Control)* | A permission model where users are assigned roles (e.g., admin, standard user) that determine their access level. |
| *ScanEvent* | A record created each time a user scans an item, used to track scan history and detect changes. |
| *SAST (Static Application Security Testing)* | Automated source code analysis that detects vulnerabilities before the application runs. |
| *DAST (Dynamic Application Security Testing)* | Security testing performed on a running application to find runtime vulnerabilities. |
| *Threshold* | The user-defined minimum acceptable quantity for an item. When inventory drops below this level, an alert is generated. |
| *Token* | A secure, encoded string (typically a JWT) that authenticates a user's session and permissions. |
| *Wireframe* | A low-fidelity layout that shows the structure of a user interface screen without full design elements. |

# 1.0 Introduction

**1.1 Project Objectives**

OmniStock aims to simplify inventory management for small businesses and individual users by providing a cloud-based system that supports efficient tracking, monitoring, and analysis of stock. The key objectives are:

* Enable users to scan and register inventory items via barcode.
* Provide real-time visibility into stock levels.
* Automatically alert users when inventory falls below thresholds.
* Support inventory organization through CRUD operations.
* Generate data-driven reports on stock trends to aid business decisions.

**1.2 Project Scope**

**Included Features:**

* **User Authentication** – Users will securely register and log in using encrypted credentials and optional two-factor authentication.
* **Barcode Scanning** – Items can be quickly added or updated using USB scanners or mobile device cameras.
* **Stock Alerts** – The system will notify users when item quantities fall below a customizable threshold.
* **CRUD Operations** – Users can create, view, update, and delete inventory items.
* **Analytics & Reporting** – Inventory activity trends and stock usage reports will be available for download.

**Excluded from Scope:**

* **Full ERP Functionality** – OmniStock is not intended to replace comprehensive ERP systems.
* **Role-Based Access Control (RBAC)** – Advanced permission tiers will be considered for future versions but are not part of the current release.

**1.3 Project Overview**

OmniStock is being developed as a responsive, cross-platform inventory management application. The development process follows Agile and Scrum methodologies in three-week sprint cycles, with weekly team meetings and sprint reviews.

Key milestones include:

* **Sprint 1**: Requirements gathering, use case modeling, and UI wireframes.
* **Sprint 2**: Backend and database structure setup.
* **Sprint 3**: Security, architecture, and class design implementation.
* **Sprint 4**: Testing, documentation finalization, and project presentation.

The expected outcome is a fully functional prototype hosted in a cloud environment (AWS), supporting up to 1,000 users with scan logging, alert notifications, and downloadable reports. Resources used include GitHub (CI/CD), Jira (task management), and Microsoft Teams (collaboration).

# 2.0 Project Description

**2.1 Project Features / Functions**

OmniStock provides the following key features, each tailored to improve the speed, accuracy, and usability of inventory tracking:

* **Secure User Authentication:** Users create accounts and securely log in using encrypted credentials, with optional two-factor authentication for enhanced security.
* **Barcode Scanning for Quick Inventory Entry:** Users can scan barcodes using a mobile camera or USB scanner to add or update inventory items with minimal manual input.
* **Stock Level Monitoring with Real-Time Alerts:** When an item's quantity drops below its predefined threshold, users receive immediate alert notifications.
* **Data Analytics and Usage Trends:** The system tracks and analyzes scan patterns and inventory usage to help users make data-driven restocking decisions.
* **Import/Export Functionality for Inventory Data:** Users can upload or download inventory in CSV format, making it easy to transition between systems or generate external reports.

**2.2 User Stories**

* As a user, I want to create an account and securely log in using my email and password to access my inventory.
* As a user, I want to scan barcodes quickly so that they can be added to the inventory system.
* As a user, I want to easily view detailed information about a product after scanning its barcode.
* As a user, I want to receive alert notifications when an item’s inventory count falls too low so I can restock in time.
* As a user, I want to view a list of recently scanned items and recent inventory activity.
* As a user, I want to check the current stock levels of my items so I can know immediately what is available.
* As a user, I want to add, view, update, or delete items in my inventory so I can manage my stock effectively.

# 2.3 Use Cases

**Use Case 1: Scan and Add Item**  
The user logs in and scans a barcode. If the item exists, the system updates its quantity; otherwise, the user is prompted to add a new item.

**Use Case 2: Receive Stock Alert**  
When stock falls below the threshold, the system generates an alert which is shown in the dashboard and via a notification badge.

**Use Case 3: View Inventory Report**  
User accesses the “Reports” section and filters results by date, then downloads a CSV or PDF showing recent scan events, updates, and low-stock alerts.

**A diagram of a product

AI-generated content may be incorrect.**

**2.4 Project Assumptions and Dependencies**

* Users must have internet access to use the application.
* The system depends on barcode scanner APIs for barcode recognition.
* The server infrastructure assumes stable deployment on AWS services.

# 3. Project Collaboration and Documentation

The following tools will be used consistently throughout the project for communication, version control, and task tracking:

* **GitHub** will be used to store all project code and documentation. It will also manage branches and pull requests, enabling collaborative development and code reviews. GitHub Actions will handle CI/CD automation for builds and deployments.
* **Jira** will be used to manage the product backlog, create sprints, and assign user stories to team members. Tasks are broken into epics and tracked using story points for sprint planning.
* **Microsoft Teams** will be used for weekly meetings, instant messaging, and shared file access to ensure all team members are updated and aligned on priorities.

# 4. Project Management

The project is managed using Agile methodologies, specifically the **Scrum framework**, to support iterative development, team alignment, and continuous delivery.

**Scrum Implementation:**

The project follows **four 2-week sprints**, each with clearly defined goals and deliverables. Scrum ceremonies help guide the development process and maintain team focus.

* **Sprint Planning:** Conducted at the beginning of each sprint to outline tasks, assign responsibilities, and define sprint goals.
* **Weekly Sync-Ups:** Live meetings are held once per week to review progress, discuss any blockers, and realign priorities if needed.
* **Sprint Reviews:** Held at the end of each sprint to demonstrate completed features and gather feedback from the team or instructor.
* **Sprint Retrospectives:** Reflective sessions after each sprint to evaluate what went well, identify challenges, and improve team processes.

**Project Management Tools:**

* **Jira** is used for sprint planning, backlog tracking, and task assignment through user stories and epics.
* **GitHub** manages all code repositories, supports branching and pull requests, and automates deployments with GitHub Actions. Jira and GitHub are linked for better task/code traceability.

This structure supports flexible, incremental development while ensuring the team remains aligned and productive throughout the semester.

# 5. Requirements Specifications

**5.1 Business Requirements**

| **Requirement ID** | **Requirement Description** | **MOSCOW** |
| --- | --- | --- |
| BR1 | The system must improve supplier communication through a supplier portal that tracks real-time stock levels, reducing delivery delays by at least 20%. | M |
| BR2 | The system must implement AI-driven forecasting to analyze demand trends and automatically adjust inventory levels with 90%+ accuracy. | M |
| BR3 | The system should automate order tracking and real-time notifications to maintain at least 95% on-time deliveries and improve customer satisfaction. | S |

**5.2 User Requirements**

| **Requirement ID** | **Requirement Description** | **MOSCOW** |
| --- | --- | --- |
| UR1 | Users must be able to create an account and securely log in using their email and password. | M |
| UR2 | Users must be able to scan product barcodes and immediately view relevant product details. | M |
| UR3 | Users must receive and view low stock alert notifications when item quantities fall below thresholds. | M |
| UR4 | Users must be able to see a list of recently scanned items to monitor recent inventory activity. | M |
| UR5 | Users must be able to view inventory usage trends for scanned items, including frequency and restock patterns. | C |
| UR6 | Users must be able to check the current stock levels of all items within the inventory database. | M |
| UR7 | Users must be able to add, view, update, or delete inventory items within the system. | M |

**5.3 Functional Requirements**

| **Requirement ID** | **Requirement Description** | **MOSCOW** |
| --- | --- | --- |
| FR1 | The system must allow users to register, log in, and securely manage their account credentials. | M |
| FR2 | CRUD operations must be available for inventory items, enabling full inventory lifecycle control. | M |
| FR3 | The system must notify users with alerts when item stock levels fall below a predefined threshold. | S |
| FR4 | The system must support importing and exporting inventory data in CSV format for offline access and backups. | C |
| FR5 | The system must support role-based access control, distinguishing between administrators and standard users. | S |

**5.4 Non-Functional Requirements**

| **Requirement ID** | **Requirement Description** | **MOSCOW** |
| --- | --- | --- |
| NFR1 | The system must handle up to 1,000 concurrent users with minimal latency and no degradation in responsiveness. | M |
| NFR2 | The system must implement encrypted password storage, follow OWASP security guidelines, and support 2FA. | M |
| NFR3 | The system must scale horizontally to support growing inventory sizes and higher usage loads. | S |
| NFR4 | The system must provide an accessible and responsive UI that works across desktops, tablets, and smartphones. | S |
| NFR5 | The system should aim for 99.9% uptime during active hours, with regular backups and failover mechanisms. | S |

**5.5 Implementation (Performance) Requirements**

| **Requirement ID** | **Requirement Description** | **MOSCOW** |
| --- | --- | --- |
| IR1 | The frontend must be built with React.js, optimized for accessibility and responsive design across platforms. | M |
| IR2 | The backend must be developed using Node.js with Express, achieving API response times under 200ms for at least 95% of all requests. | M |
| IR3 | The database must run on AWS RDS PostgreSQL with nightly backups and read replicas for high availability and scalability. | M |
| IR4 | Authentication must use OAuth 2.0 and JWT, with enforced role-based access control for multi-user security. | M |
| IR5 | Testing must include Jest for unit testing and Cypress for integration testing. CI/CD pipelines will automate deployment and updates. | M |

# 6. High-Level Design Structure

# 6.1 Security Design

**6.1.1 User Authentication (OAuth 2.0, JWT, 2FA)**

OmniStock will implement OAuth 2.0 as the foundation for its authentication and authorization system. This industry-standard protocol provides secure delegation and supports modern login flows. The implementation will follow a Password Grant Flow for user credentials and issue JWT (JSON Web Tokens) for session handling.

**Key Features:**

* Access Token: Short-lived JWT (expires in 15 minutes)
* Refresh Token: Long-lived JWT (expires in 7 days) to request new access tokens
* Token Payload: Encoded with user ID, role, permissions, and expiration metadata
* Signature: Tokens signed using HS256 algorithm with a secret key

Refresh tokens are stored securely and invalidated immediately upon logout or password reset using a Redis-backed token blacklist. Access and refresh tokens are rotated every 7 days, with sliding expiration for active sessions.

**2FA (Two-Factor Authentication):** Optional for added security. If enabled, users must enter a one-time password (OTP) generated through a TOTP-compatible app like Google Authenticator. The 2FA secret will be stored securely in encrypted form and only validated server-side during login attempts.

**Re-authentication Enforcement:** For critical operations (e.g., deleting inventory or changing threshold limits), the system enforces session re-validation within a 3-minute window to prevent token hijack misuse.

**Security Measures:**

* Login throttling and rate limiting
* IP logging on login events
* Re-authentication required for sensitive actions

**6.1.2 Data Protection Measures (Encrypted Passwords, HTTPS)**

Security extends beyond login, and data protection is implemented at multiple layers:

* Passwords are hashed using bcrypt with a salt factor of 12. Raw passwords are never stored or logged.
* HTTPS is enforced sitewide using SSL certificates issued by Let's Encrypt.
* CORS policies are strictly defined to prevent cross-origin API abuse.
* CSRF tokens are embedded in all authenticated forms and AJAX requests.
* Sensitive fields like user email, IP logs, and barcode metadata are encrypted at rest using AES-256.
* All endpoints are protected with middleware that verifies the JWT and checks authorization scopes for each role.

**Secret Management Policy:** All cryptographic secrets, JWT signing keys, and API credentials are stored in a secure parameter store (e.g., AWS Secrets Manager). No sensitive config is committed to source control.

**6.1.3 Security Testing Strategy**

Security testing will be continuous and integrated into our CI/CD process. The strategy consists of the following layers:

**Static Code Analysis (SAST):**

* Tools: SonarQube, ESLint with security rules
* Frequency: On every PR or merge to main
* Targets: Input validation, token handling, query sanitization

**Dynamic Testing (DAST):**

* Tool: OWASP ZAP (Zed Attack Proxy)
* Test Scope: All public endpoints and user flows
* Automated scans scheduled weekly, with manual testing once per sprint

**Threat Modeling:**

* Method: STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege)
* Timing: During Sprint 4 planning
* Output: Threat matrix identifying vulnerabilities with mitigation plans

**Manual Testing:**

* Checklist-based security audits before every major release
* Broken Access Control and IDOR scenarios manually tested

**6.1.4 Summary of Security Strategy**

The security strategy is designed to comply with modern best practices and is proactive in addressing threats. Authentication and data transmission are secured with token-based flows and HTTPS. Sensitive information is encrypted and access to system components is regulated based on clearly defined roles and token scopes. Testing is ongoing, using automated tools and manual methods to catch common vulnerabilities before they affect end users.

OmniStock is moving toward a zero-trust security model, with future features like device fingerprinting, geo-aware login alerts, and session anomaly detection.

# 6.2 Hardware Design

This section defines the minimum hardware, memory, storage, and platform compatibility requirements necessary to support the development, deployment, and use of the OmniStock Inventory Tracker system across environments (development, staging, production).

**6.2.1 Server Requirements (CPU)**

OmniStock will be deployed in a cloud-hosted environment using **Amazon Web Services (AWS)**. The minimum server specifications are as follows:

**Development Environment:**

* **Instance Type:** AWS EC2 t3.micro or t3.small
* **vCPUs:** 2 virtual CPUs
* **Use Case:** Local testing, limited developer access, single-user load

**Production Environment:**

* **Instance Type:** AWS EC2 t3.medium or m5.large
* **vCPUs:** Minimum 2 (scalable to 4 or 8 for high-traffic situations)
* **Load Balancing:** AWS Elastic Load Balancer (ELB) to scale across multiple EC2 instances in future versions
* **Database Hosting:** AWS RDS (PostgreSQL) with Multi-AZ failover enabled

All servers must support 64-bit architecture and virtualization. System updates and patching will be managed via automation scripts and AWS Systems Manager.

**6.2.2 RAM and Storage Requirements**

RAM and storage capacity directly affect the performance and scalability of the application.

**Development:**

* **RAM:** Minimum 2 GB
* **Storage:** 10–20 GB SSD
* **Disk IOPS:** 100–300 minimum (suitable for low-volume testing)

**Production:**

* **RAM:** 4–8 GB (scalable to 16 GB for high-concurrency support)
* **Storage:**
  + 50–100 GB EBS for server OS and application files
  + RDS storage: minimum 20 GB allocated storage with auto-scaling
* **Backups:** Nightly snapshots to AWS S3 (7-day retention)

**File Storage & Backups:**

* **AWS S3** used for:
  + Barcode images
  + CSV import/export files
  + Report archives
* **Redundancy:** All S3 buckets are versioned and set with lifecycle policies

**6.2.3 Platform Compatibility (OS, Browser, Mobile Support)**

OmniStock is a **web-first**, cross-platform application designed for both desktop and mobile usage with full responsiveness.

**Client Device Compatibility:**

* **Operating Systems (Web):**
  + Windows 10 or later
  + macOS 11+ (Big Sur and later)
  + ChromeOS (latest)
* **Browsers (Frontend UI):**
  + Google Chrome (v90+)
  + Mozilla Firefox (v85+)
  + Microsoft Edge (v95+)
  + Safari (v14+)
* **Mobile Devices:**
  + Android 10+ (via mobile browser)
  + iOS 13+ (Safari recommended)

**Barcode Scanner Support:**

* **Supported Devices:**
  + USB HID-based handheld barcode scanners (keyboard wedge mode)
  + Mobile camera scanning via HTML5 + JavaScript library
* **Future Support (planned):**
  + Native integration with Zebra SDK and other enterprise-grade scanners

**6.2.4 Summary of Hardware Architecture**

OmniStock has been designed with flexibility in mind—supporting modern browsers, scalable server infrastructure, and cross-device usability. The deployment infrastructure uses AWS services to deliver fault-tolerant, secure, and easily maintainable operations. The system can be scaled horizontally by increasing EC2 instances or vertically by upgrading existing resources. Compatibility with commodity barcode scanners ensures ease of adoption without the need for expensive hardware.

Additionally, the system is designed to handle **1,000 concurrent users** or up to **5,000 scan events per minute**, while maintaining **sub-300ms response times** under load-balanced EC2 environments.

**Real-time monitoring** is handled using **AWS CloudWatch** and integrated into a **Slack notification system** for alerting on CPU/RAM spikes, failed backups, or network anomalies.

A **disaster recovery plan** includes nightly infrastructure backups. In the event of EC2 failure, a pre-configured **AMI** is deployed within 5 minutes, with **Route53** switching DNS traffic to a standby instance.

# 6.3 User Experience (UX)

The User Experience (UX) of OmniStock has been meticulously designed to prioritize clarity, efficiency, and simplicity. The application supports a streamlined inventory workflow accessible from both desktop and mobile devices. This section defines user flows, navigation logic, feedback/error handling, and interface sketches to align with accessibility and usability best practices.

**6.3.1 User Flow Mapping (Login → Scan → Report)**

The following core flow represents the primary navigation path for most users:

**Navigation Flow:**

1. **Login/Register**
   * Users authenticate via OAuth 2.0 with email/password (and optional 2FA)
   * First-time users create accounts or verify via email
2. **Dashboard**
   * Landing page post-login
   * Lists current inventory sorted by stock status
   * Displays most recent scan events and critical alerts
3. **Scan Item**
   * Opens camera or supports USB scanner
   * Scanned item triggers a database search
   * If item exists: data is shown with update options
   * If item is new: prompts user to add it
4. **Inventory Detail**
   * Drill-down view into a specific item
   * Allows editing name, quantity, threshold, or deletion
   * Displays scan history and status over time
5. **Alerts**
   * Real-time notifications when items fall below threshold
   * Badge count shown on navigation bar
6. **Reports**
   * Users can generate CSV/PDF reports of inventory trends
   * Filters include date range, user, and status
7. **Logout**
   * Session invalidation and redirect to login

*This flow ensures rapid access to key features while maintaining a minimal and intuitive UI.*

**6.3.2 Feedback & Error Handling**

Feedback is designed to be proactive, informative, and non-disruptive. Error handling ensures users always understand what went wrong and how to correct it.

**Success Feedback:**

* Green toasts/popups for:
  + Successfully scanning an item
  + Saving or updating item data
  + Report exports/downloads

**Error Feedback:**

* Red toasts/modal alerts for:
  + “Item not found” on scan
  + Validation errors (e.g., missing required fields)
  + Authorization failures (expired JWTs)

**Warnings:**

* Yellow alerts shown for:
  + “Item quantity below threshold”
  + “Item has not been updated in 30+ days”
  + “Scan failed, try again”

**Input Validation:**

* Real-time checks on forms (e.g., required fields, max lengths)
* Inventory threshold must be ≥ 0
* Barcode must be 8–20 characters alphanumeric

**Accessibility Features:**

* All interactive elements are tab-navigable
* Color contrast meets WCAG 2.1 AA
* Screen reader compatibility in progress for v2.5

**6.3.3 Wireframes (Text-Based Sketches)**

**Login Page**

A screen shot of a login page

AI-generated content may be incorrect.

**Dashboard**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**Inventory Detail Page**

A screenshot of a computer

AI-generated content may be incorrect.

**6.3.4 UX Summary**

The UX design of OmniStock focuses on speed, clarity, and functionality. Core user actions are never more than two clicks away from the dashboard. The application is responsive, accessible, and visually segmented with clear feedback systems. Users are empowered to manage inventory quickly while receiving intuitive cues to guide their actions. The use of accessible labels, color-coded statuses, and mobile optimization make it ideal for warehouse, storefront, or home-office use.

All actions from scan to confirmation are targeted to complete in under 3 seconds. The UI enforces frontend caching and optimistic UI updates to ensure perceived speed. All interactive elements will include ARIA attributes and semantic roles, with full keyboard navigation available by v2.5. A dark mode/light mode toggle and touch gestures are also planned for future mobile UX improvements.

# 6.4 System Architecture

This section outlines the architectural design of OmniStock, focusing on the technical stack, data layer interactions, and responsibilities of each tier. The system is designed using modern web technologies with scalability, modularity, and maintainability in mind.

**6.4.1 Client-Server Architecture**

OmniStock follows a **3-tier client-server architecture** with clear separation of concerns between the **presentation**, **application**, and **data layers**.

**Layers Overview:**

1. **Frontend (Presentation Tier):**
   * Framework: React.js
   * Purpose: Handles UI rendering, form validation, scanner access, and routing
   * Responsibilities:
     + Send/receive data via REST API
     + Authenticate users (OAuth 2.0 frontend flow)
     + Manage session tokens (JWT stored in HttpOnly cookies or memory)
     + Handle routing and responsive layout
2. **Backend (Application Tier):**
   * Runtime: Node.js with Express
   * Purpose: Provides a secure API layer between frontend and database
   * Responsibilities:
     + Validate incoming requests
     + Handle business logic (e.g., inventory updates, scan events)
     + Interact with database using an ORM (e.g., Sequelize)
     + Enforce RBAC and JWT validation
     + Trigger alerts and log scan events
3. **Database (Data Tier):**
   * Engine: PostgreSQL hosted on AWS RDS
   * Purpose: Stores all persistent data
   * Responsibilities:
     + Maintain normalized entity structure
     + Enforce foreign keys, constraints, and indexes
     + Support relational queries and soft deletes
     + Store scan logs, alerts, and historical data

**6.4.2 Architecture Diagram (Described)**

User → [React Frontend] → [Node.js API Layer] → [PostgreSQL DB]

↑

Barcode Scanner Input

↓

[Alert Engine & Logs]

* The frontend communicates with the backend via HTTPS.
* The backend acts as the gatekeeper for all business logic and database access.
* All critical workflows (scanning, updating inventory, generating reports) pass through secure, token-protected API routes.
* Alert engine is a module that monitors inventory thresholds and pushes notifications to users via the dashboard and email (future integration).

**6.4.3 Layer Responsibilities**

| **Layer** | **Key Tech** | **Responsibility Summary** |
| --- | --- | --- |
| Frontend | React.js, HTML5 | UI, client-side routing, session handling, scanner integration, dynamic feedback |
| Application | Node.js, Express | RESTful APIs, user auth, business logic, data validation, alert system |
| Data Layer | PostgreSQL | Persistent storage, ERDs, normalized schema, data backups, inventory and scan records |

**6.4.4 Summary of System Architecture**

OmniStock’s architecture was chosen to balance simplicity and scalability. Each layer is replaceable or upgradable without affecting the others. The frontend could be migrated to a mobile framework (React Native) in the future, and the backend could be containerized and scaled with Docker/Kubernetes.

The frontend is deployed via GitHub Actions to Netlify with auto-preview branches for pull requests. The backend is deployed to Heroku using container-based release pipelines with rollback support. If the API becomes unavailable, the frontend falls back to displaying cached data stored in IndexedDB and alerts the user to reconnection status in real-time. Redis will be introduced in v2.5 to cache frequent queries.

# 6.5 Database

This section defines the database system used in OmniStock, its specifications, key entities, and their relationships.

**Requirement ID:** DB1  
**Description:** Inventory Tracking Database  
**Requirement:** The system shall use a PostgreSQL relational database hosted on AWS RDS to store user data, inventory items, scan history, and alerts with foreign key relationships and indexing for performance.

**6.5.1 Specifications**

* **DBMS:** PostgreSQL
* **Host Environment:** AWS RDS (Multi-AZ enabled)
* **Structure:** Fully normalized schema using foreign key constraints
* **Security:** AES-256 encryption for sensitive fields, parameterized queries to prevent SQL injection
* **Scalability:** Indexes on barcode, user\_id, and last\_updated to improve query performance
* **Backups:** Nightly snapshots with 7-day retention; stored in AWS S3
* **Retention Policy:** Scan logs and alerts older than 90 days are archived or purged unless pinned
* **Planned Features (v2.5):** Triggers for alert automation, Redis caching for frequent queries

**6.5.2 Key Entities and Attributes**

**User**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| user\_id | UUID | Unique identifier |
| email | String | Login credential (unique) |
| password\_hash | String | Encrypted password |
| role | Enum | Role: user, admin |
| created\_at | Timestamp | Account creation date |

**InventoryItem**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| item\_id | UUID | Unique identifier |
| user\_id | UUID | Foreign key to User |
| name | String | Item name |
| barcode | String | Barcode number |
| quantity | Integer | Quantity in stock |
| threshold | Integer | Minimum acceptable level |
| last\_updated | Timestamp | Auto-updated on changes |
| is\_deleted | Boolean | Soft delete flag |

**ScanEvent**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| scan\_id | UUID | Unique identifier |
| item\_id | UUID | Foreign key to InventoryItem |
| user\_id | UUID | Foreign key to User |
| source | String | Scanner or manual |
| timestamp | Timestamp | Scan time |

**Alert**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| alert\_id | UUID | Unique identifier |
| item\_id | UUID | Foreign key to InventoryItem |
| type | String | low\_stock, critical, etc. |
| message | String | Alert message |
| status | Enum | unread, read, dismissed |
| created\_at | Timestamp | Time of generation |

**6.5.3 Entity Relationship Diagram (ERD)** *A black screen with white text

AI-generated content may be incorrect.*

# 6.6. Top-Level Classes

This section describes the key classes used in the backend of OmniStock and their responsibilities.

**6.6.1 Specifications**

**Requirement ID:** CLASS1  
**Description:** Inventory Management Classes  
**Requirement:** The system shall implement four primary classes: User, InventoryItem, ScanEvent, and Alert. Each class encapsulates related logic and attributes to support the inventory tracking workflow.

OmniStock is organized around four major classes that form the core of its object-oriented logic. Each class represents a real-world concept in the inventory system and contains both data attributes and related behaviors (methods). These classes are built using a service-oriented approach to ensure modularity and testability.

#### User Class

This class represents an authenticated account holder in the system. It handles login credentials, permissions, and associations with owned inventory.

* Purpose: Authentication, authorization, and inventory ownership
* Key Logic:
  + Role-based permission checks
  + Secure password handling
  + Inventory retrieval for the user

#### InventoryItem Class

Encapsulates each trackable product. All CRUD operations on inventory are managed through this class.

* Purpose: Represent a single stock item and manage changes
* Key Logic:
  + Handles additions, edits, and soft deletes
  + Validates stock thresholds and triggers alerts
  + Supports search and barcode validation

#### Alert Class

Generated when stock levels fall below a predefined threshold. This class ensures that alert messages are linked to their respective inventory items.

* Purpose: Notify users of important stock conditions
* Key Logic:
  + Status transitions (unread → read → dismissed)
  + Alert generation is triggered by inventory logic
  + Alerts are viewable and dismissible from the UI

#### ScanEvent Class

This class logs scanning activity initiated by a user, either through manual entry or barcode devices.

* Purpose: Maintain a historical record of user interactions with inventory
* Key Logic:
  + Tracks source of the scan (camera, manual, etc.)
  + Links to both the user and inventory item
  + Used as a data source for reports and audit trails

**6.6.2 Class Method & Attribute Description**

* All class methods enforce access control and input validation via middleware.
* Business logic methods (e.g., check\_threshold()) trigger downstream processes such as alert generation.
* ORM: Classes are implemented as Sequelize models to map directly to the PostgreSQL schema.
* Testability: All classes are covered by unit tests using mock repositories (Jest/PyTest).
* Serialization: Classes include to\_dict() methods for API responses and logs.

**6.6.3 Class Diagram**

A black background with white text

AI-generated content may be incorrect.

# 6.7 Data Flow and States

**6.7.1. Specifications**

Requirement ID: DF1

**Description**: OmniStock must allow users to scan barcodes, update inventory, and trigger automated alerts based on thresholds.  
**Requirement**: The system must track how data flows between users, inventory items, scan events, and alerts, and reflect state transitions for inventory status and alerts.

OmniStock follows an event-driven model where user actions (such as scanning an item or updating inventory) initiate backend processes and state transitions.

**Data Inputs**

* *User Input:* Scans item, edits inventory, sets thresholds
* *Device Input:* Barcode scanner (USB or mobile camera)

**Processing Steps**

* System checks if the scanned item exists
* If found, it updates the quantity and logs a ScanEvent
* If the new quantity is ≤ threshold, it generates an Alert
* Inventory item status is recalculated (In Stock, Low Stock, Out of Stock)

**Outputs**

* Dashboard updates with new item state
* Alert notification (UI badge or pop-up)
* Option to generate a report with historical data

**6.7.2. State Transitions**

**InventoryItem States**

|  |  |
| --- | --- |
| **State** | **Trigger** |
| In Stock | Quantity > Threshold |
| Low Stock | Quantity ≤ Threshold but > 0 |
| Out of Stock | Quantity = 0 |
| Restocked | Quantity increased from 0 or low |

*\*Each state is recalculated automatically after a scan or manual update.*

**Alert States**

|  |  |
| --- | --- |
| **State** | **Trigger** |
| Unread | Alert created by system |
| Read | User views alert on dashboard |
| Dismissed | User manually archives or clears alert |

**Session States**

|  |  |
| --- | --- |
| **State** | **Description** |
| Unauthenticated | No token or expired session |
| Authenticated | Valid session with access token |
| Revalidated | Token recently re-verified for secure actions (e.g. delete item) |

**6.7.3. Data Flow Diagram**

A diagram of data flow

AI-generated content may be incorrect.

**Summary**

This section demonstrates how OmniStock handles event-driven inventory changes. Inventory states and alerts transition based on user actions, and real-time data updates are processed securely with frontend-backend communication and visual feedback.

# 6.8. Reports

**Requirement ID: REPORT1**

**Description**: Inventory Activity Report  
**Requirement**: The system shall generate an Inventory Activity Report that logs all inventory-related events (scans, updates, alerts, deletions) and supports CSV and PDF export formats. Reports shall be filterable by user, action type, and date range.

**6.8.1. Specification**

The Inventory Activity Report gives users visibility into all changes made to inventory items, including user actions and automated system events. Reports can be used for audit trails, operational tracking, or monthly summaries.

**Key Fields:**

* Item Name
* Barcode
* Action Type (scan, update, delete, restock, alert)
* Quantity Before / After
* User
* Timestamp
* Notes (optional)

**Report Features:**

* Filter by user, date, and action type
* Export to CSV (spreadsheet compatible)
* Export to PDF (printable for recordkeeping)
* Accessible via the dashboard
* Pagination (25 records/page)
* Admins receive enhanced filtering options in future builds

**Technical Notes:**

* PDF is generated using Puppeteer (HTML → PDF)
* CSV uses buffered streaming for large data sets
* Reports are retained for 30 days unless pinned

# 6.9. Internal Interfaces

OmniStock follows a modular architecture where each subsystem interacts through well-defined interfaces:

* **Frontend ↔ Backend:** All communication occurs via RESTful API calls secured with OAuth 2.0 tokens.
* **Backend ↔ Database:** Interactions are abstracted using an ORM (e.g., Sequelize) to perform queries and enforce data integrity.
* **Alert Engine ↔ Inventory Module:** The backend includes an alerting module that listens for inventory threshold events and triggers alerts accordingly.
* **Scan Module ↔ Inventory Service:** Scanned barcode data flows through a dedicated scanning service that either links to an existing item or routes it to the creation form.

# 6.10. External Interfaces

OmniStock integrates with several third-party services and plans for additional ones in the future:

* **Barcode Scanners:** Supports USB barcode readers using HID (keyboard emulation) mode, as well as mobile device cameras via JavaScript libraries.
* **Email Services (Planned):** Integration with AWS SES or SendGrid for sending low-stock or alert notifications.
* **Authentication Providers (Future):** While OAuth 2.0 is implemented internally, support for external identity providers (Google, Microsoft) is planned in v2.5.
* **CSV Import/Export:** Uses the File API and server-side parsers to handle user-uploaded CSVs for bulk inventory updates.

# 6.11. Other Outputs

Aside from inventory reports, OmniStock generates several other outputs as part of system use:

* **System Logs:** Logs include user activity (logins, scans, updates), system errors, and alert generation history.
* **Alerts:** Real-time low stock notifications are displayed on the dashboard and stored for later reference.
* **Audit Trails:** Every update to an inventory item is recorded with a timestamp, user ID, and before/after quantity values.
* **Data Backups:** Nightly automated database backups are stored in AWS S3 for disaster recovery purposes.

# 6.12. Configuration Data

OmniStock includes a dedicated configuration module where the following parameters can be adjusted:

* **Threshold Values:** Users can set custom low-stock thresholds per item.
* **Report Filters:** Default date ranges, output formats (CSV, PDF), and columns shown can be configured.
* **Security Settings:** 2FA can be enabled/disabled per user account. Session timeout durations are also configurable.
* **Roles and Permissions:** Admin users can configure new roles or modify existing ones for advanced access control.
* **Backup Preferences:** Admins can choose daily/weekly backup frequency and retention duration.

# 6.13. Training

OmniStock is designed to be intuitive, but a basic training and onboarding plan is available:

* **Built-In Tooltips:** Most pages include context-sensitive help (hover tooltips) and sample input formatting.
* **User Guide PDF:** A downloadable quick-start guide is available in the Reports section.
* **In-App Walkthrough (Planned):** A guided tour feature will walk new users through scanning, updating inventory, and exporting reports.
* **Admin Training:** A short onboarding session can be provided for team admins covering user management, permissions, and report customization.