**Inventory Management System - Technical Documentation**

The system follows a microservices architecture where seven specialized services work together through an API Gateway. Here is how requests flow through the system:

Nginx

User (Browser)

RABBIT MQ

NOTIFICATION DB

ROUTE DB

PRODUCT DB

APPROVAL DB

IDENTITY DB

NOTIFICATION MS

ROUTE MS

PRODUCT MS

APPROVAL MS

IDENTITY MS

Routes to services, validates JWT

API GATEWAY

SSL termination, rate limiting, static files

WEB APP

http or https request, it will redirect to https always

## Identity Service (Port 5003)

This service acts as the security guardian for the entire system. When users log in, this service verifies their credentials and issues JWT tokens that contain their identity and permissions. Think of it as an airport security checkpoint that issues boarding passes with your clearance level written on them.

The service manages user accounts, roles like Admin or User, and granular permissions such as "product.create" or "route.delete". Every other service trusts the tokens issued by Identity Service without needing to call back and verify each request.

**Key responsibilities:** User authentication, JWT token generation, role and permission management, user account CRUD operations.

**Database schema:** Users, Roles, Permissions, UserRoles, RolePermissions, UserPermissions, RefreshTokens.

### Product Service (Port 5001)

The Product Service maintains the master catalog of all inventory items in your organization. Each product has an inventory code, belongs to a category, resides in a department, and may be assigned to a specific worker. The service stores product images and tracks whether items are working or new.

When someone creates or modifies a product, this service can trigger approval workflows if the user lacks direct permissions. After approval, it publishes events to RabbitMQ so other services know about the change.

**Key responsibilities:** Product CRUD with approval workflows, category management, department management, image storage, permission-based access control.

**Database schema:** Products, Categories, Departments.

**Events published:** product.created, product.updated, product.deleted.

### Route Service (Port 5002)

Every time a product moves or changes, the Route Service records it. This creates an audit trail showing the complete lifecycle of each item. When a product transfers from Engineering to Sales, a route entry captures the before and after state, including photos and notes about the transfer.

The service listens for product events from RabbitMQ and automatically creates route entries when products are created, updated, or deleted. This ensures you never lose track of inventory history.

**Key responsibilities:** Transfer tracking, product movement history, automatic route creation from product events, transfer approval workflows, route completion tracking.

**Database schema:** InventoryRoutes with embedded product snapshots.

**Events consumed:** product.created, product.updated, product.deleted. **Events published:** route.created, route.completed, product.transferred.

### Approval Service (Port 5004)

When users without direct permissions attempt sensitive operations, the Approval Service steps in. It captures their request, stores all the details needed to execute it later, and notifies administrators who can approve or reject.

Upon approval, this service uses a special token to execute the original action on behalf of the requester. The ActionExecutor component contains the logic to recreate HTTP requests with proper authentication to Product or Route services.

**Key responsibilities:** Approval request creation, admin approval/rejection workflows, automatic action execution after approval, request cancellation, approval history tracking.

**Database schema:** ApprovalRequests with JSON action data.

**Events published:** approval.request.created, approval.request.processed, approval.request.cancelled.

### Notification Service (Port 5005)

This service ensures everyone stays informed about important events. It consumes messages from RabbitMQ, stores notifications in the database, and delivers them through multiple channels. Users receive real-time browser notifications via SignalR, while team WhatsApp groups get formatted messages with images.

The service maintains websocket connections with all active users, organized into groups by user ID and role. When an approval request arrives, it instantly notifies all admins without any polling or refresh needed.

**Key responsibilities:** Real-time SignalR notifications, WhatsApp group messaging, notification storage and retrieval, user connection management, multi-channel message delivery.

**Database schema:** Notifications.

**Events consumed:** approval.request.created, approval.request.processed, product.created, product.deleted, route.created, route.completed.

### API Gateway (Port 5000)

The Gateway provides a single entry point for all client requests. It validates JWT tokens, routes requests to appropriate services, and handles CORS for web browsers. The gateway uses Ocelot to configure routes declaratively and includes retry policies plus circuit breakers for resilience.

Rate limiting protects the login endpoint from brute force attacks, while detailed logging helps trace requests across services using correlation IDs.

**Key responsibilities:** Request routing, JWT validation, CORS handling, rate limiting, retry logic, circuit breaker patterns, request/response logging.

**Configuration:** Routes defined in ocelot.json for each environment.

### Web Application (Port 5051 development)

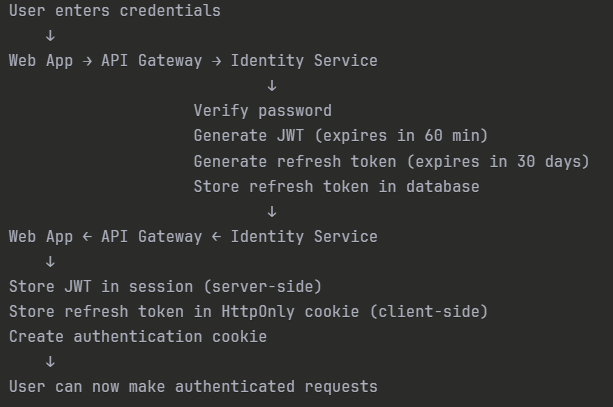
The web application provides the user interface for the entire system. Built with ASP.NET Core MVC, it renders server-side pages and uses JavaScript for interactive features. The app maintains user sessions, manages JWT token refresh, and establishes SignalR connections for real-time updates.

Users interact with products, routes, approvals, and notifications through intuitive forms and dashboards. The application handles both standard HTTP requests and multipart form uploads for images.

**Key responsibilities:** User interface rendering, session management, JWT token refresh, SignalR client connections, form handling, image uploads, dashboard visualizations.

### User Authentication Flow

When users log in, a carefully orchestrated sequence ensures security while maintaining a smooth experience:

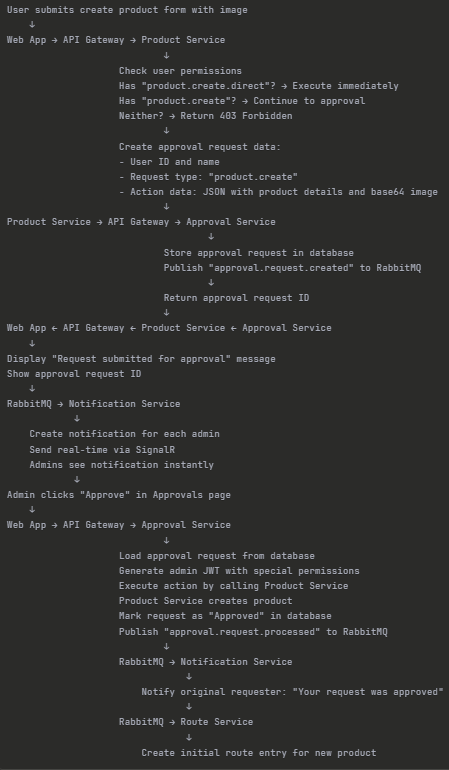


The JWT contains user ID, username, email, roles, and permissions as claims. Services verify this token locally without calling Identity Service, making the system fast and reducing coupling.

When the JWT expires after sixty minutes, the web application automatically uses the refresh token to obtain a new JWT without forcing the user to log in again. This refresh happens transparently in the background.

### Product Creation Flow (With Approval)

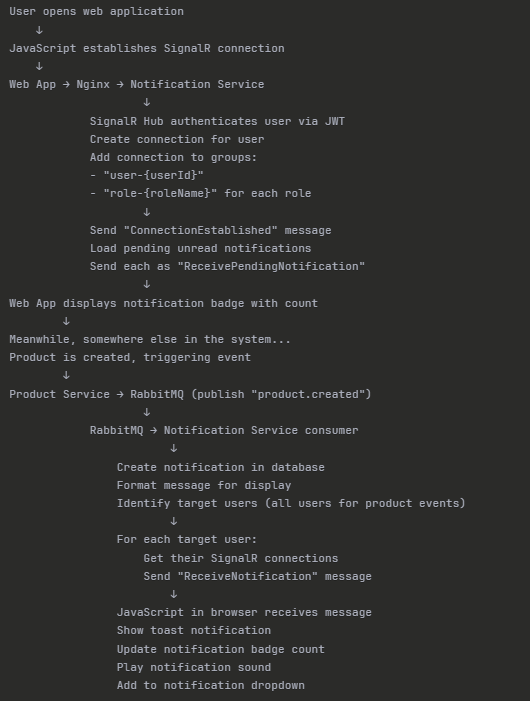
Creating a product demonstrates how the approval workflow integrates seamlessly with business logic:



This pattern allows organizations to enforce approval policies without modifying the core business logic. The same product creation code path works for both direct execution and approval-based execution.

### Real-Time Notification Flow

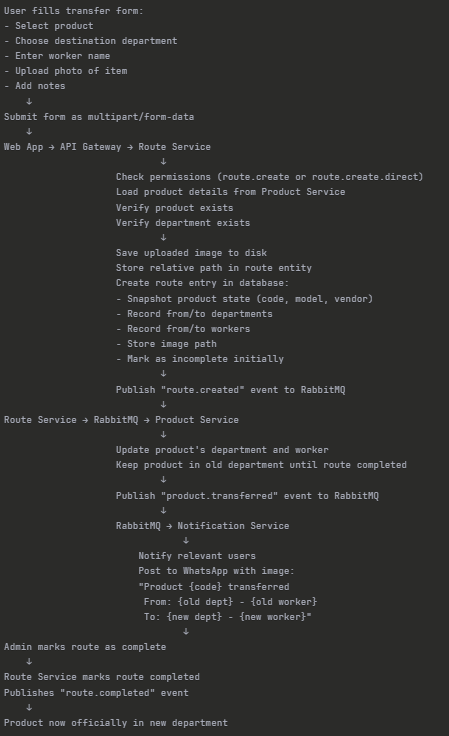
The notification system keeps users informed without requiring page refreshes. Here is how it works from start to finish:



The SignalR connection stays open as long as the browser tab remains active. If the connection drops, automatic reconnection attempts restore it within seconds. The groups pattern ensures admins receive approval notifications while regular users only see general updates.

### Product Transfer with Images

Transferring products between departments showcases how the system handles multipart form data and coordinates multiple services:



The system maintains referential integrity by taking snapshots of product data at transfer time. Even if someone later changes the product model name, historical routes still show what the model was called when transferred.

## Message Queue Events

RabbitMQ enables asynchronous communication between services. Each service publishes events when important things happen and subscribes to events it cares about. The exchange uses topic routing, allowing flexible subscription patterns.

### Published Events

**Product Service publishes:**

* product.created - When a new product is added (includes image data as base64 if available)
* product.updated - When product details change (includes list of changed fields)
* product.deleted - When a product is removed (includes last known state)
* product.transferred - When ownership changes (includes from/to department and worker)

**Route Service publishes:**

* route.created - When a new route entry is recorded (includes product snapshot)
* route.completed - When a transfer finishes (includes completion timestamp and image)

**Approval Service publishes:**

* approval.request.created - When someone requests approval (includes requester info and request type)
* approval.request.processed - When admin approves or rejects (includes decision and reason)
* approval.request.cancelled - When requester cancels their pending request

### Event Consumers

**Route Service consumes:**

* product.created - Creates initial route entry showing product arrival
* product.updated - Records the update as a route entry with changes documented
* product.deleted - Creates removal route entry

**Product Service consumes:**

* product.transferred - Updates product location after route service confirms transfer

**Notification Service consumes:**

* All events from all services - Creates notifications and sends them via SignalR and WhatsApp

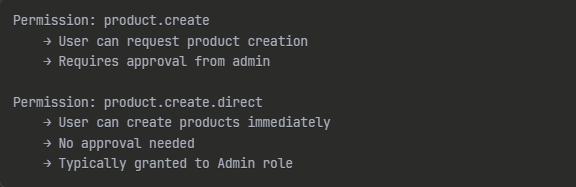
This event-driven architecture means services remain loosely coupled. The Product Service does not know or care which other services listen to its events. You can add new services that react to these events without modifying existing code.

## Security and Permissions

The security model protects sensitive operations while allowing flexibility for different user roles. Every user has one or more roles, and each role grants a set of permissions.

### Permission Hierarchy

The system distinguishes between regular permissions and "direct" permissions. A user with product.create can create products, but their request goes through approval. A user with product.create.direct creates products immediately without approval. Admins typically have all "direct" permissions.



## Technology Stack Summary

Here is the complete technology stack that powers the system:

**Backend Framework:** ASP.NET Core 8.0 provides the foundation for all services with built-in dependency injection, middleware pipeline, and excellent performance.

**Database:** PostgreSQL 15 stores all persistent data with strong ACID guarantees and excellent JSON support for flexible schemas.

**Message Broker:** RabbitMQ handles asynchronous communication between services with guaranteed delivery and flexible routing patterns.

**Authentication:** JWT tokens with refresh token rotation provide secure, stateless authentication that scales horizontally.

**API Gateway:** Ocelot routes requests with retry policies, circuit breakers, and request aggregation capabilities.

**Real-Time Communication:** SignalR enables websocket connections for instant notifications without polling overhead.

**Logging:** Serilog sends structured logs to Seq for centralized monitoring and querying across all services.

**Reverse Proxy:** Nginx terminates SSL, serves static files, implements rate limiting, and load balances traffic.

**Containerization:** Docker packages each service with its dependencies for consistent deployment across environments.

**External Integration:** Green API connects the system to WhatsApp for team notifications with images.

This stack balances proven technologies with modern practices. PostgreSQL and Nginx provide rock-solid reliability, while SignalR and RabbitMQ enable responsive user experiences and loose coupling between services.