Cargo Shipping Data Integration Portal

Project Documentation

Generated: 07 November 2025

Table of Contents

1. Project Overview

The Cargo Shipping Data Integration Portal is a comprehensive web application designed to integrate and manage cargo shipping data from CargoWise APIs. The portal provides a centralised platform for syncing, viewing, and managing shipment and order data from multiple API configurations.

Key Objectives

• Centralise cargo shipping data from multiple CargoWise API endpoints

• Provide a user-friendly interface for viewing and managing shipments and orders

• Enable API key management for multiple company configurations

• Offer data export capabilities for integration with external systems

• Implement robust user authentication and role-based access control

• Maintain comprehensive activity logging for audit purposes

2. System Architecture

The application follows a modern full-stack architecture with a React frontend and Node.js backend, utilising PostgreSQL for data persistence. The system is built using Vite for fast development and Hono for the backend API framework.

Architecture Components

Component	Technology	Purpose
Frontend	React 19 + TypeScript	User interface and interaction
Backend	Hono + Node.js	API server and business logic
Database	PostgreSQL + Kysely	Data persistence and type-safe queries
Authentication	JWT + bcrypt	Secure user authentication
UI Components	Radix UI	Accessible, customisable components
Data Visualisation	Recharts	Charts and analytics
Build Tool	Vite	Fast development and optimised builds

3. Technology Stack

Frontend Technologies

• React 19.0.0 - Core UI framework

• TypeScript - Type-safe development

• React Router DOM 6.26.2 - Client-side routing

• React Hook Form 7.54.2 - Form management

• Zod 3.25.76 - Schema validation

• TanStack Query 5.76.1 - Data fetching and caching

• Radix UI - Accessible component library

• Lucide React - Icon library

• Recharts 2.15.1 - Data visualisation

• React Big Calendar 1.19.4 - Calendar components

• date-fns 4.1.0 - Date manipulation

• Sonner 2.0.1 - Toast notifications

Backend Technologies

• Hono 4.7.5 - Web framework

• @hono/node-server 1.13.8 - Node.js adapter

• PostgreSQL - Database via postgres 3.4.7

• Kysely 0.26.3 - Type-safe SQL query builder

• kysely-postgres-js 3.0.0 - PostgreSQL dialect

• jose 6.1.0 - JWT authentication

• bcryptjs 3.0.3 - Password hashing

• nanoid 5.1.6 - Unique ID generation

• superjson 2.2.2 - JSON serialisation

Development Tools

• Vite 6.2.0 - Build tool and dev server

• @vitejs/plugin-react-swc 3.8.0 - Fast refresh

• tsx 4.20.3 - TypeScript execution

• vite-plugin-node-polyfills 0.24.0 - Node.js polyfills

4. Database Schema

The application uses PostgreSQL with Kysely for type-safe database queries. The schema includes nine main tables covering user management, API configurations, cargo data, and system administration.

4.1 Users Table

Stores user account information with role-based access control.

Field	Type	Description
id	Serial	Primary key, auto-generated
email	String	Unique user email address
displayName	String	User's display name
avatarUrl	String (nullable)	Profile picture URL
role	Enum	User role: 'admin' or 'user'
isActive	Boolean	Account status (default: true)
createdAt	Timestamp	Account creation date
updatedAt	Timestamp	Last update timestamp

4.2 API Configurations Table

Stores CargoWise API endpoint configurations for multiple companies.

Field	Type	Description
id	String (UUID)	Primary key
companyName	String	Company name
companyCode	String	Company identifier code
apiType	Enum	API type: 'REST' or 'SOAP'
endpointUrl	String	API endpoint URL
username	String	API authentication username
password	String	API authentication password
apiKey	String (nullable)	Optional API key
isActive	Boolean	Configuration status
createdAt	Timestamp	Creation timestamp
updatedAt	Timestamp	Last update timestamp

4.3 Shipments Table

Stores shipment data synced from CargoWise APIs.

Field	Type	Description
id	String (UUID)	Primary key
shipmentNumber	String	Unique shipment identifier
apiConfigId	String (FK)	Reference to API configuration
origin	String	Origin location
destination	String	Destination location
carrier	String	Shipping carrier name
status	String	Current shipment status
etd	Timestamp	Estimated time of departure
eta	Timestamp	Estimated time of arrival
rawData	JSON	Complete raw API response
createdAt	Timestamp	Record creation timestamp
updatedAt	Timestamp	Last update timestamp

4.4 Orders Table

Stores order data synced from CargoWise APIs.

Field	Type	Description
id	String (UUID)	Primary key
orderNumber	String	Unique order identifier
apiConfigId	String (FK)	Reference to API configuration
customerName	String	Customer name
orderDate	Timestamp	Order placement date
status	String	Current order status
rawData	JSON	Complete raw API response
createdAt	Timestamp	Record creation timestamp
updatedAt	Timestamp	Last update timestamp

4.5 Additional Tables

Table	Purpose
userPasswords	Stores bcrypt password hashes separately from user data
sessions	Manages user session tokens and expiry
loginAttempts	Tracks login attempts for security monitoring
activityLogs	Comprehensive audit log of all admin actions
systemSettings	Stores application configuration settings

5. Core Features

5.1 API Configuration Management

• Create and manage multiple CargoWise API configurations

• Support for both REST and SOAP API types

• Secure storage of API credentials

• Enable/disable configurations without deletion

• Edit existing configurations

• Delete unused configurations

5.2 Data Synchronisation

• Manual sync of shipment data from configured APIs

• Manual sync of order data from configured APIs

• Automatic handling of data updates and duplicates

• Raw data preservation for audit purposes

• Error handling and logging for failed syncs

5.3 Data Viewing & Export

• Tabbed interface for viewing shipments and orders

• Real-time data display with sorting and filtering

• Detailed view of individual shipments and orders

• Export data to CSV, JSON, or Excel formats

• Customisable export fields and filters

5.4 Admin Dashboard

- System statistics and overview
- User management (create, edit, delete users)
- Activity log viewing and export
- System settings management
- User role assignment
- Account activation/deactivation

6. API Endpoints

6.1 Authentication Endpoints

Method	Endpoint	Description
POST	/api/auth/login_with_password	User login with email and password
POST	/api/auth/register_with_password	Register new user account
POST	/api/auth/logout	End user session
GET	/api/auth/session	Get current user session

6.2 API Configuration Endpoints

Method	Endpoint	Description
GET	/api/api-config/list	Get all API configurations
POST	/api/api-config/create	Create new API configuration
POST	/api/api-config/update	Update existing configuration
POST	/api/api-config/delete	Delete API configuration

6.3 Shipments Endpoints

Method	Endpoint	Description
GET	/api/shipments/list	Get all shipments with filtering
POST	/api/shipments/sync	Sync shipments from CargoWise API

6.4 Orders Endpoints

Method	Endpoint	Description
GET	/api/orders/list	Get all orders with filtering
POST	/api/orders/sync	Sync orders from CargoWise API

6.5 Data Export Endpoint

Method	Endpoint	Description
POST	/api/data/export	Export shipments/orders data in various formats

6.6 Admin Endpoints

Method	Endpoint	Description
GET	/api/admin/stats	Get system statistics
GET	/api/admin/users/list	List all users
POST	/api/admin/users/create	Create new user
POST	/api/admin/users/update	Update user details
POST	/api/admin/users/delete	Delete user account
GET	/api/admin/activity-logs/list	List activity logs
GET	/api/admin/activity-logs/export	Export activity logs
GET	/api/admin/settings/list	List system settings
POST	/api/admin/settings/update	Update system setting
POST	/api/admin/settings/delete	Delete system setting

7. User Roles & Authentication

The application implements a robust authentication system using JWT tokens and bcrypt password hashing. Two user roles are supported with distinct permissions.

User Roles

Role	Permissions
User	• View API configurations • View shipments and orders • Export data • Sync data from APIs
Admin	• All user permissions • Create/edit/delete API configurations • Manage users • View activity logs • Manage system settings • Access admin dashboard

Authentication Flow

1. User submits email and password via login form

2. Server validates credentials against bcrypt hash

3. JWT token is generated with user ID and role

4. Token is stored in HTTP-only cookie

5. Subsequent requests include token for authentication

6. Protected routes verify token and check user permissions

7. Session expires after configured timeout period

8. Installation & Setup

8.1 Prerequisites

• Node.js (version 18 or higher recommended)

• PostgreSQL database (version 13 or higher)

• pnpm package manager

• CargoWise API credentials

8.2 Environment Configuration

The application requires an env.json file with the following structure. Generate JWT secrets using the provided command and configure your PostgreSQL connection string.

Generate JWT Secret:

```
node -e "console.log(require('crypto').randomBytes(32).toString('hex'))"
```

Required Environment Variables:

• DATABASE_URL - PostgreSQL connection string

• JWT_SECRET - Secret key for JWT token signing

• SESSION_EXPIRY - Session timeout duration (e.g., '7d', '24h')

• PORT - Server port (optional, default: 3000)

8.3 Installation Steps

1. Install pnpm globally: npm install -g pnpm

2. Install dependencies: pnpm install

3. Configure env.json with your credentials

4. Set up PostgreSQL database and run migrations

5. Build the application: pnpm vite build

6. Start the server: pnpm tsx server.ts

7. Access the application at http://localhost:3000

8.4 Database Setup

The database schema is managed using Kysely migrations. Ensure your PostgreSQL database is running and accessible via the connection string in env.json. The schema includes all necessary tables, indices, and constraints.

9. Security Features

The application implements multiple layers of security to protect sensitive cargo shipping data and user information.

Feature	Implementation
Password Security	Bcrypt hashing with salt rounds
Authentication	JWT tokens with expiry
Session Management	Secure HTTP-only cookies
Role-Based Access	Admin and user roles with permission checks
API Credentials	Encrypted storage of API keys and passwords
Activity Logging	Comprehensive audit trail of admin actions
Login Tracking	Failed login attempt monitoring
Input Validation	Zod schema validation on all inputs
SQL Injection Prevention	Kysely parameterised queries
CORS Protection	Configured CORS policies
Rate Limiting	Protected against brute force attacks

Security Best Practices

• Regularly rotate JWT secrets

• Use strong, unique passwords for database connections

• Keep dependencies updated to patch security vulnerabilities

• Enable HTTPS in production environments

• Regularly review activity logs for suspicious behaviour

• Implement IP whitelisting for admin access if possible

• Back up the database regularly

• Use environment-specific configurations

10. Project Structure

The project follows a modular structure with clear separation between frontend components, backend endpoints, and shared utilities.

Directory	Contents
components/	React components for UI elements and pages
pages/	Route-level page components
endpoints/	Backend API endpoint handlers
endpoints/auth/	Authentication endpoints
endpoints/api-config/	API configuration management
endpoints/shipments/	Shipment data endpoints
endpoints/orders/	Order data endpoints
endpoints/admin/	Admin panel endpoints
endpoints/data/	Data export endpoints
helpers/	Shared utility functions and hooks
helpers/schema.tsx	Database schema types
helpers/db.tsx	Database connection
helpers/apiHooks.tsx	React Query hooks for API calls

Key Components

• SharedLayout.tsx - Main application layout with navigation

• ApiConfigCard.tsx - Display API configuration cards

• CreateApiConfigForm.tsx - Form for creating API configs

• ShipmentsView.tsx - Shipments data table and viewer

• OrdersView.tsx - Orders data table and viewer

• AdminLayout.tsx - Admin panel layout

• UsersTable.tsx - User management table

• ProtectedRoute.tsx - Route protection wrapper

• PasswordLoginForm.tsx - Login form component

Configuration Files

• package.json - Dependencies and scripts

• vite.config.ts - Vite build configuration

• server.ts - Backend server setup

• env.json - Environment variables (not in repo)

• index.html - HTML entry point

• index.tsx - React application entry

Summary

The Cargo Shipping Data Integration Portal provides a comprehensive solution for managing cargo shipping data from multiple CargoWise API sources. With its robust authentication system, role-based access control, and extensive administrative features, it serves as a centralised hub for shipping operations. The application's modular architecture makes it easy to extend and maintain, whilst the TypeScript implementation ensures type safety throughout the codebase. The PostgreSQL database with Kysely provides reliable data storage with compile-time query validation. Key strengths include secure API credential management, flexible data export options, comprehensive activity logging, and an intuitive user interface built with React and Radix UI components.

Future Enhancement Possibilities

- Automated scheduled synchronisation
- Real-time notifications for shipment updates
- Advanced analytics and reporting dashboard
- Mobile application for on-the-go access
- Integration with additional shipping APIs
- Customs documentation management
- Multi-language support
- Advanced search and filtering capabilities