888Cargo - MERN Stack

Documentación de Arquitectura del Proyecto

Generado el: 22/8/2025
Rama: feature/refactoring-pdf-qr-system

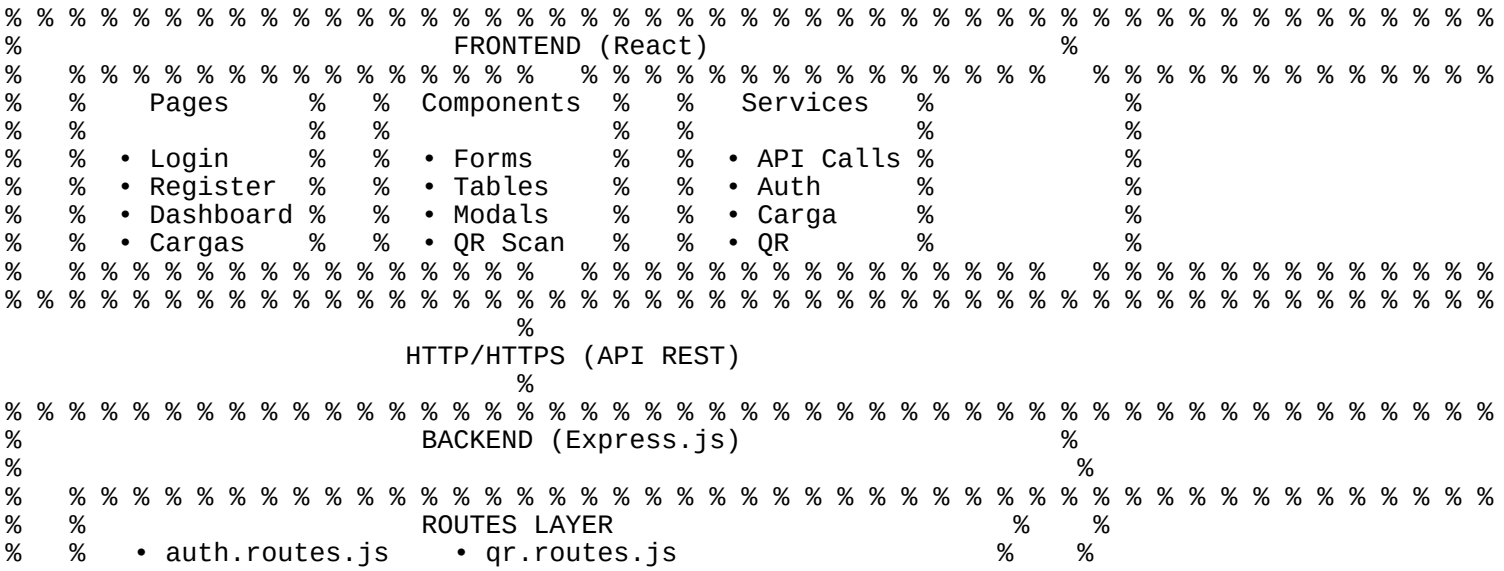
1. Descripción General del Proyecto

888Cargo es una aplicación web MERN (MongoDB/SQLite, Express.js, React, Node.js) diseñada para la gestión de cargas, packing lists y generación de códigos QR.

Características principales:
• Sistema de autenticación con JWT
• Gestión de cargas y packing lists
• Generación automática de códigos QR
• Exportación a PDF con códigos QR visuales
• Carga masiva de datos desde archivos Excel
• Sistema de recuperación por WhatsApp
• Arquitectura modular con patrón MVC

El proyecto implementa una arquitectura refactorizada que separa claramente las responsabilidades en capas: Controladores, Servicios, Repositorios y Validadores.

2. Arquitectura del Sistema



```
%   %   • carga.routes.js   • task.routes.js                  %    %
%   %   • recuperacion.routes.js                              %    %
%   % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
%                                %                                 %
%   % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
%   %                CONTROLLERS LAYER                      %    %
%   %   • auth.controller.js     • qr.controller.js           %    %
%   %   • carga.controller.js    • task.controller.js         %    %
%   %   • recuperacion.controller.js                          %    %
%   % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
%                                %                                 %
%   % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
%   %                 SERVICES LAYER                        %    %
%   %   • auth.service.js        • qr.service.js              %    %
%   %   • pdf.service.js         • whatsapp.service.js        %    %
%   %   • notification.service.js • audit.service.js          %    %
%   % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
%                                %                                 %
%   % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
%   %               REPOSITORIES LAYER                      %    %
%   %   • base.repository.js     • user.repository.js         %    %
%   %   • qr.repository.js       • articles.repository.js     %    %
%   %   • transaction.manager.js                              %    %
%   % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
%                                %                                 %
%   % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
%   %                MIDDLEWARE LAYER                       %    %
%   %   • validateToken.js       • fileValidation.middleware  %    %
%   %   • dataSanitization.middleware.js                      %    %
%   % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
                               %
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
%                         DATABASE (SQLite)                       %
%                                                                  %
%   Tables:                                                        %
%   • users                    • qr                               %
%   • carga                    • caja                             %
%   • articulo_packing_list    • audit_log                        %
%                                                                  %
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
```

3. Estructura de Directorios

```
888Cris-MERN/
% % %  backend/                         # Servidor Express.js
%    % % %  config/                     # Configuraciones
%    %    % % %  environments.js         # Variables de entorno
%    %    % % %  middleware.config.js    # Configuración de middlewares
%    %    % % %  swagger.config.js       # Documentación API
%    % % %  controllers/                # Controladores MVC
%    %    % % %  auth.controller.js      # Autenticación
%    %    % % %  carga.controller.js     # Gestión de cargas
%    %    % % %  qr.controller.js        # Códigos QR
%    %    % % %  recuperacion.controller.js # Recuperación passwords
%    %    % % %  task.controller.js      # Tareas generales
%    % % %  services/                   # Lógica de negocio
%    %    % % %  auth.service.js         # Servicios de autenticación
%    %    % % %  qr.service.js           # Servicios de QR y PDF
%    %    % % %  pdf.service.js          # Generación de PDFs
%    %    % % %  whatsapp.service.js     # Integración WhatsApp
%    %    % % %  notification.service.js # Notificaciones
%    %    % % %  audit.service.js        # Auditoría
%    % % %  repositories/               # Acceso a datos
%    %    % % %  base.repository.js      # Repositorio base
```

```
%    %    % % %  qr.repository.js          # Repositorio QR
%    %    % % %  user.repository.js        # Repositorio usuarios
%    %    % % %  articles.repository.js    # Repositorio artículos
%    %    % % %  transaction.manager.js    # Gestor transacciones
%    % % %  routes/                        # Definición de rutas
%    %    % % %  auth.routes.js            # Rutas de autenticación
%    %    % % %  carga.routes.js           # Rutas de cargas
%    %    % % %  qr.routes.js              # Rutas de QR
%    %    % % %  recuperacion.routes.js    # Rutas de recuperación
%    %    % % %  task.routes.js            # Rutas de tareas
%    % % %  middlewares/                   # Middlewares personalizados
%    %    % % %  validateToken.js          # Validación JWT
%    %    % % %  fileValidation.middleware.js # Validación archivos
%    %    % % %  dataSanitization.middleware.js # Sanitización datos
%    % % %  models/                        # Modelos de datos
%    %    % % %  user.model.js            # Modelo usuario
%    %    % % %  carga.model.js           # Modelo carga
%    %    % % %  qr.model.js              # Modelo QR
%    %    % % %  articulosPL.model.js     # Modelo artículos
%    % % %  validators/                    # Validadores de entrada
%    %    % % %  auth.validator.js        # Validación autenticación
%    %    % % %  qr.validator.js          # Validación QR
%    %    % % %  base.validator.js        # Validador base
%    % % %  utils/                         # Utilidades
%    %    % % %  auth.utils.js            # Utilidades autenticación
%    % % %  libs/                          # Librerías personalizadas
%    %    % % %  jwt.js                   # Manejo JWT
%    % % %  uploads/                       # Archivos subidos
%        % % %  qr-codes/                  # Códigos QR generados
% % %  client/                           # Aplicación React
%    % % %  src/                          # Código fuente
%    %    % % %  components/              # Componentes reutilizables
%    %    % % %  pages/                   # Páginas principales
%    %    % % %  services/                # Servicios API
%    %    % % %  assets/                  # Recursos estáticos
%    % % %  public/                       # Archivos públicos
%    % % %  dist/                         # Build de producción
% % %  uploads/                          # Archivos globales
%    % % %  images/                       # Imágenes subidas
% % %  db/                               # Base de datos SQLite
    % % %  packing_list.db               # Archivo de base de datos
```

4. Stack Tecnológico

FRONTEND:
• React 18.x - Librería de interfaces de usuario
• Vite - Herramienta de build y desarrollo
• React Router DOM - Enrutamiento del lado del cliente
• Axios - Cliente HTTP para API calls
• React QR Scanner - Escaneo de códigos QR
• ESLint - Linter de código JavaScript

BACKEND:
• Node.js 18+ - Runtime de JavaScript
• Express.js 4.x - Framework web para Node.js
• SQLite 3 - Base de datos ligera
• JWT (jsonwebtoken) - Autenticación basada en tokens
• PDFKit - Generación de documentos PDF
• QRCode - Generación de códigos QR
• Multer - Manejo de archivos multipart
• XLSX - Procesamiento de archivos Excel
• Bcryptjs - Encriptación de passwords
• Dotenv - Variables de entorno

• CORS - Cross-Origin Resource Sharing
• Express Rate Limit - Rate limiting para APIs
• Swagger/OpenAPI - Documentación de API

DESARROLLO:
• Nodemon - Auto-restart del servidor
• Concurrently - Ejecución paralela de scripts
• Git - Control de versiones
• GitHub - Repositorio remoto
• VS Code - Editor de código
• JavaScript Obfuscator - Ofuscación de código

PATRONES DE DISEÑO:
• MVC (Model-View-Controller)
• Repository Pattern
• Service Layer Pattern
• Dependency Injection
• Factory Pattern
• Singleton Pattern

ARQUITECTURA:
• RESTful API
• Layered Architecture
• Microservices Pattern (preparado)
• Event-Driven Architecture (parcial)

5. Esquema de Base de Datos

```
TABLA: users
% % %  id (INTEGER PRIMARY KEY AUTOINCREMENT)
% % %  username (VARCHAR(100) UNIQUE NOT NULL)
% % %  email (VARCHAR(255) UNIQUE NOT NULL)
% % %  password (VARCHAR(255) NOT NULL)
% % %  role (VARCHAR(50) DEFAULT 'user')
% % %  created_at (DATETIME DEFAULT CURRENT_TIMESTAMP)
% % %  updated_at (DATETIME DEFAULT CURRENT_TIMESTAMP)

TABLA: carga
% % %  id_carga (INTEGER PRIMARY KEY AUTOINCREMENT)
% % %  codigo_carga (VARCHAR(100) UNIQUE NOT NULL)
% % %  nombre_carga (VARCHAR(255))
% % %  descripcion (TEXT)
% % %  fecha_creacion (DATETIME DEFAULT CURRENT_TIMESTAMP)
% % %  fecha_actualizacion (DATETIME DEFAULT CURRENT_TIMESTAMP)
% % %  estado (VARCHAR(50) DEFAULT 'activa')
% % %  usuario_id (INTEGER REFERENCES users(id))
% % %  metadata (TEXT) -- JSON data

TABLA: articulo_packing_list
% % %  id_articulo (INTEGER PRIMARY KEY AUTOINCREMENT)
% % %  id_carga (INTEGER REFERENCES carga(id_carga))
% % %  codigo_articulo (VARCHAR(100))
% % %  descripcion_espanol (TEXT)
% % %  descripcion_ingles (TEXT)
% % %  cantidad_cajas (INTEGER)
% % %  peso_total (DECIMAL(10,2))
% % %  volumen_total (DECIMAL(10,2))
% % %  valor_unitario (DECIMAL(10,2))
% % %  valor_total (DECIMAL(12,2))
% % %  foto (VARCHAR(500)) -- path to image
```

```
% % %  observaciones (TEXT)
% % %  fecha_creacion (DATETIME DEFAULT CURRENT_TIMESTAMP)
% % %  fecha_actualizacion (DATETIME DEFAULT CURRENT_TIMESTAMP)

TABLA: caja
% % %  id_caja (INTEGER PRIMARY KEY AUTOINCREMENT)
% % %  id_articulo (INTEGER REFERENCES articulo_packing_list(id_articulo))
% % %  numero_caja (INTEGER NOT NULL)
% % %  total_cajas (INTEGER NOT NULL)
% % %  peso_caja (DECIMAL(8,2))
% % %  dimensiones (VARCHAR(100))
% % %  contenido (TEXT)
% % %  fecha_creacion (DATETIME DEFAULT CURRENT_TIMESTAMP)
% % %  fecha_actualizacion (DATETIME DEFAULT CURRENT_TIMESTAMP)

TABLA: qr
% % %  id_qr (INTEGER PRIMARY KEY AUTOINCREMENT)
% % %  id_caja (INTEGER REFERENCES caja(id_caja))
% % %  codigo_qr (TEXT UNIQUE NOT NULL)
% % %  codigo_unico (VARCHAR(100) UNIQUE NOT NULL)
% % %  imagen_qr_path (VARCHAR(500))
% % %  fecha_generacion (DATETIME DEFAULT CURRENT_TIMESTAMP)
% % %  fecha_escaneado (DATETIME)
% % %  veces_escaneado (INTEGER DEFAULT 0)
% % %  ultimo_escaneado_por (INTEGER REFERENCES users(id))
% % %  metadata (TEXT) -- JSON data
% % %  activo (BOOLEAN DEFAULT 1)

TABLA: audit_log
% % %  id (INTEGER PRIMARY KEY AUTOINCREMENT)
% % %  tabla_afectada (VARCHAR(100))
% % %  registro_id (INTEGER)
% % %  accion (VARCHAR(50)) -- INSERT, UPDATE, DELETE
% % %  datos_anteriores (TEXT) -- JSON
% % %  datos_nuevos (TEXT) -- JSON
% % %  usuario_id (INTEGER REFERENCES users(id))
% % %  ip_address (VARCHAR(45))
% % %  user_agent (TEXT)
% % %  fecha_creacion (DATETIME DEFAULT CURRENT_TIMESTAMP)

RELACIONES:
• users 1:N carga
• carga 1:N articulo_packing_list
• articulo_packing_list 1:N caja
• caja 1:N qr
• users 1:N audit_log (usuario que realiza la acción)
• users 1:N qr (último usuario que escaneó)
```

6. Endpoints de la API

AUTENTICACIÓN (/api/auth):
POST /register - Registro de usuario
POST /login - Inicio de sesión
POST /logout - Cierre de sesión
GET /profile - Obtener perfil del usuario
PUT /profile - Actualizar perfil

CARGAS (/api/carga):
GET / - Listar todas las cargas
POST / - Crear nueva carga
GET /:id - Obtener carga específica
PUT /:id - Actualizar carga
DELETE /:id - Eliminar carga
POST /procesar-excel - Procesar archivo Excel
POST /guardar-con-qr - Guardar carga y generar QRs

CÓDIGOS QR (/api/qr):
GET /carga/:idCarga - Obtener QRs de una carga
POST /articles/:articuloId/generate - Generar QRs para artículo
GET /pdf-carga/:idCarga - Descargar PDF con QRs
GET /pdf-test/:idCarga - PDF de prueba (sin auth)
POST /scan - Escanear código QR
PUT /:qrId/regenerate - Regenerar código QR específico
DELETE /:qrId - Eliminar código QR

TAREAS (/api/tasks):
GET / - Listar tareas
POST / - Crear nueva tarea
GET /:id - Obtener tarea específica
PUT /:id - Actualizar tarea
DELETE /:id - Eliminar tarea

RECUPERACIÓN (/api/recuperacion):
POST /enviar-codigo - Enviar código de recuperación
POST /verificar-codigo - Verificar código recibido
POST /cambiar-password - Cambiar password con código

MIDDLEWARES:
• authRequired - Verificación de JWT
• fileValidation - Validación de archivos
• dataSanitization - Sanitización de datos
• rateLimit - Limitación de requests

CÓDIGOS DE RESPUESTA:
200 - OK - Operación exitosa
201 - Created - Recurso creado
400 - Bad Request - Datos inválidos
401 - Unauthorized - Token inválido/expirado
403 - Forbidden - Sin permisos
404 - Not Found - Recurso no encontrado
422 - Unprocessable Entity - Error de validación
500 - Internal Server Error - Error del servidor

7. Información de Despliegue

ESTRUCTURA DE ARCHIVOS DE CONFIGURACIÓN:

% % % .env # Variables de entorno principales
% % % .env.development # Variables de desarrollo (client)
% % % .env.production # Variables de producción (client)
% % % package.json # Dependencias y scripts principales
% % % client/package.json # Dependencias del frontend
% % % backend/config/environments.js # Configuración por ambiente

VARIABLES DE ENTORNO PRINCIPALES:
• NODE_ENV # Ambiente (development/production)
• PORT # Puerto del servidor (4000)
• JWT_SECRET # Secreto para tokens JWT
• DB_PATH # Ruta de la base de datos SQLite
• UPLOAD_PATH # Directorio de archivos subidos
• CORS_ORIGIN # Origen permitido para CORS

• RATE_LIMIT_WINDOW_MS # Ventana de rate limiting
• RATE_LIMIT_MAX_REQUESTS # Máximo de requests por ventana

SCRIPTS DISPONIBLES:

Backend:
npm start # Iniciar servidor de producción
npm run dev # Desarrollo con nodemon
npm run dev:server # Solo servidor en desarrollo
npm test # Ejecutar pruebas

Cliente:
npm run dev # Servidor de desarrollo
npm run build # Build de producción
npm run preview # Preview del build
npm run lint # Verificar código con ESLint

Proyecto completo:
npm run dev # Ambos servidores concurrentemente
npm run build:client # Build solo del cliente
npm run start:full # Producción completa

PUERTOS:
• Backend: 4000 (configurable)
• Frontend Dev: 5173 (Vite)
• Frontend Prod: Servido por backend estático

CARACTERÍSTICAS DE SEGURIDAD:
• Autenticación JWT con expiración
• Rate limiting por IP
• Sanitización de datos de entrada
• Validación de archivos subidos
• Headers de seguridad HTTP
• CORS configurado específicamente
• Encriptación de passwords con bcrypt
• Validación de tokens en rutas protegidas

LOGS Y MONITOREO:
• Logs estructurados por módulo
• Audit log de todas las operaciones
• Tracking de errores por servicio
• Métricas de uso de API
• Monitoreo de archivos subidos

BACKUP Y RECUPERACIÓN:
• Base de datos SQLite (fácil backup)
• Archivos de uploads organizados por fecha
• Scripts de respaldo automatizables
• Versionado con Git para código
• Documentación completa de APIs

Versión del proyecto: feature/refactoring-pdf-qr-system
Fecha de generación: 22/8/2025, 12:43:42