Warehouse Frontend - Project Documentation

Author: Technical Documentation Team **Date:** October 13, 2025 **Version:** 1.0 **Project Type:** Depot Safety & Performance Station

---

Executive Summary

This document provides comprehensive technical documentation for the Warehouse Frontend application, a modern React-based interactive touchscreen information display system designed for depot operations, staff management, and safety compliance. The application serves as a centralized hub for warehouse personnel to access critical information, manage check-ins, view safety alerts, and monitor performance metrics.

Table of Contents

---

1. Project Overview

1.1 Purpose

The Warehouse Frontend application is designed to serve as an interactive touchscreen station for Rhomberg Sersa Rail Group (RSRG) depot operations. It provides real-time information access, staff check-in/out functionality, safety management, document control, and performance monitoring.

1.2 Key Objectives

- **Staff Management:** Enable efficient staff check-in and check-out processes
- **Safety Compliance:** Display safety alerts and provide access to safety videos
- **Information Access:** Centralized access to documents, technical libraries, and induction materials
- **Performance Monitoring:** Real-time visualization of depot performance metrics
- **News & Updates:** Display relevant news and announcements for depot personnel

1.3 Target Users

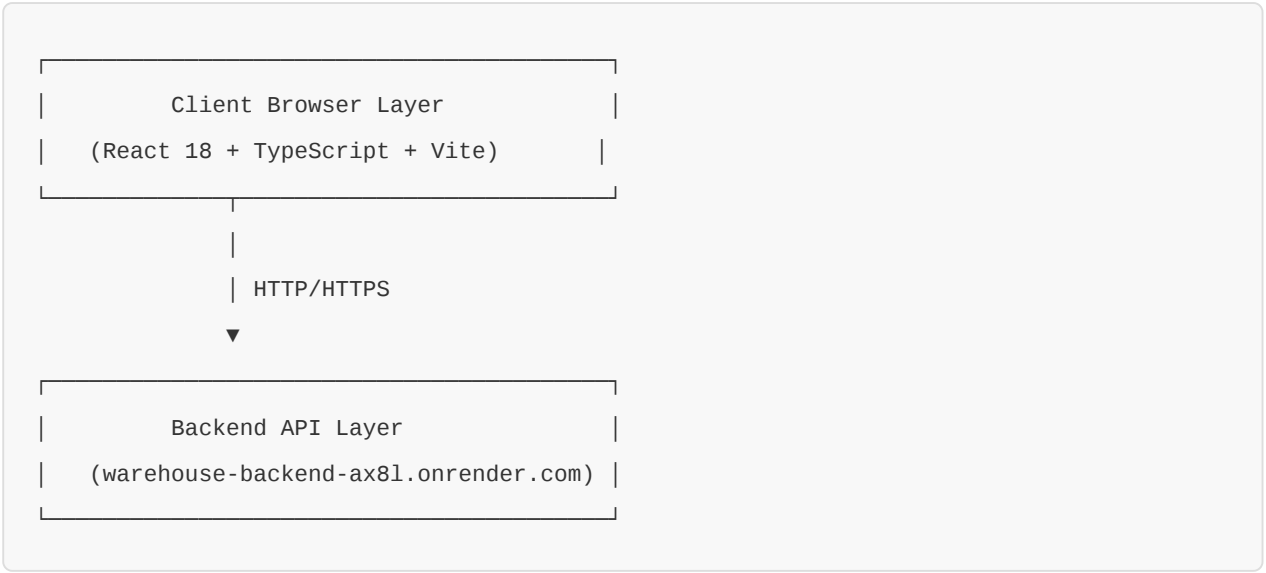
- Depot staff and contractors
- Safety officers
- Operations managers
- Administrative personnel

---

2. Technical Architecture

2.1 Architecture Overview

The application follows a modern React-based Single Page Application (SPA) architecture:



2.2 Technology Stack

Layer	Technology	Purpose
Framework	React 18.3.1	UI component framework
Language	TypeScript 5.8.3	Type-safe development
Build Tool	Vite 5.4.19	Fast development and optimized builds
Styling	Tailwind CSS 3.4.17	Utility-first CSS framework
UI Components	Radix UI + shadcn/ui	Accessible component primitives
State Management	TanStack Query 5.83.0	Server state management
Routing	React Router DOM 6.30.1	Client-side routing
Forms	React Hook Form 7.61.1 + Zod 3.25.76	Form validation
Charts	Recharts 2.15.4	Data visualization

2.3 Design Patterns

- **Component-Based Architecture:** Modular, reusable components

- **Hooks Pattern:** Custom hooks for shared logic
- **API Layer Abstraction:** Centralized API configuration and request handling
- **Server State Management:** React Query for caching and synchronization
- **Form State Management:** React Hook Form for performant form handling

---

3. Project Structure

3.1 Directory Layout

```
warehouse-frontend/
├── public/                    # Static assets
│   ├── RSRG.png              # Company logo
│   └── ...                   # Other static files
├── src/                      # Source code
│   ├── api/                  # API integration layer
│   │   ├── dashboard-api.ts  # YouTube video API
│   │   └── checkin-api.ts    # Staff check-in API
│   ├── components/           # React components
│   │   ├── admin/           # Admin panel components
│   │   ├── ui/              # Base UI components (shadcn/ui)
│   │   ├── DepotHeader.tsx  # Header component
│   │   ├── DepotNavigation.tsx
│   │   ├── DocumentControl.tsx
│   │   ├── DocumentViewer.tsx
│   │   ├── InDepotCard.tsx
│   │   ├── NewsCard.tsx
│   │   ├── PerformanceDashboard.tsx
│   │   ├── REDSafetyVideoCard.tsx
│   │   ├── SafetyAlerts.tsx
│   │   ├── StaffCheckIn.tsx
│   │   ├── VideoSection.tsx
│   │   ├── WeatherCard.tsx
│   │   └── WebViewer.tsx
│   ├── hooks/               # Custom React hooks
│   ├── lib/                 # Utility libraries
│   │   ├── api-config.ts   # API configuration
│   │   ├── auth-api.ts     # Authentication utilities
│   │   └── utils.ts        # General utilities
│   ├── pages/              # Page components
│   │   ├── Dashboard.tsx   # Main dashboard page
│   │   ├── AdminPanel.tsx  # Admin interface (commented out)
│   │   └── NotFound.tsx    # 404 page
│   ├── App.tsx             # Root application component
│   ├── main.tsx            # Application entry point
```



```
│   └── index.css           # Global styles
├── .env                    # Environment variables
├── index.html             # HTML entry point
├── package.json           # Dependencies and scripts
├── tailwind.config.ts     # Tailwind configuration
├── tsconfig.json          # TypeScript configuration
└── vite.config.ts         # Vite configuration
```

3.2 Key Directories

3.2.1 `/src/api/`

Contains all API integration code with type-safe interfaces and React Query hooks.

3.2.2 `/src/components/`

Houses all React components, organized by functionality:

- `/admin/` - Administrative interface components
- `/ui/` - Base UI components from shadcn/ui library

3.2.3 `/src/pages/`

Top-level page components that compose the application views.

3.2.4 `/src/lib/`

Shared utilities, configurations, and helper functions.

---

4. Core Technologies & Dependencies

4.1 Production Dependencies

4.1.1 Core Framework

```
{
  "react": "^18.3.1",
  "react-dom": "^18.3.1",
  "react-router-dom": "^6.30.1"
}
```

4.1.2 State Management & Data Fetching

```
{
  "@tanstack/react-query": "^5.83.0"
}
```

- **Purpose:** Server state management, caching, and synchronization
- **Key Features:** Automatic background refetching, cache invalidation, optimistic updates

4.1.3 Form Management

```
{
  "react-hook-form": "^7.61.1",
  "@hookform/resolvers": "^3.10.0",
  "zod": "^3.25.76"
}
```

- **Purpose:** Performant form handling with schema validation
- **Benefits:** Type-safe validation, minimal re-renders

4.1.4 UI Component Library

```
{
  "@radix-ui/react-*": "Multiple packages",
  "lucide-react": "^0.462.0"
}
```

- **Purpose:** Accessible, unstyled component primitives
- **Components:** Dialogs, dropdowns, tooltips, tabs, alerts, etc.

4.1.5 Styling

```
{
  "tailwindcss": "^3.4.17",
  "tailwindcss-animate": "^1.0.7",
  "tailwind-merge": "^2.6.0",
  "class-variance-authority": "^0.7.1"
}
```

4.1.6 Data Visualization

```
{
  "recharts": "^2.15.4"
}
```

- **Purpose:** Pie charts for performance metrics visualization

4.1.7 Additional Features

```
{
  "date-fns": "^3.6.0",          // Date formatting
  "react-youtube": "^10.1.0",     // YouTube video embedding
  "next-themes": "^0.3.0",        // Theme management
  "sonner": "^1.7.4"              // Toast notifications
}
```


4.2 Development Dependencies

```
{
  "@vitejs/plugin-react-swc": "^3.11.0",
  "typescript": "^5.8.3",
  "typescript-eslint": "^8.38.0",
  "eslint": "^9.32.0",
  "@tailwindcss/typography": "^0.5.16",
  "autoprefixer": "^10.4.21"
}
```

---

5. Configuration & Environment Setup

5.1 Environment Variables

The application uses environment variables for configuration:

File: `.env`

```
VITE_API_URL=https://warehouse-backend-ax8l.onrender.com
VITE_YOUTUBE_API_KEY=AIzaSyCHvl-a7ARUJ-c2xvBqCwZ2vFnXTYd9xhQ
```

Variable	Description	Required
`VITE_API_URL`	Backend API base URL	Yes
`VITE_YOUTUBE_API_KEY`	YouTube Data API key for video fetching	Yes

Important: Environment variables must be prefixed with `VITE_` to be exposed to the client.

5.2 Vite Configuration

File: `vite.config.ts`


```
export default defineConfig({
  server: {
    host: "::",      // Listen on all network interfaces
    port: 8080,      // Development server port
  },
  resolve: {
    alias: {
      "@": path.resolve(__dirname, "./src"),  // Path alias for imports
    },
  },
});
```

5.3 TypeScript Configuration

The project uses three TypeScript configuration files:

- `tsconfig.json` - Base configuration
- `tsconfig.app.json` - Application-specific settings
- `tsconfig.node.json` - Node.js environment settings

5.4 Tailwind Configuration

File: `tailwind.config.ts`

Custom theme configuration includes:

- **Custom Colors:** Depot-specific color palette
  - `depot-header`, `depot-primary`, `depot-accent`
  - `depot-warning`, `depot-success`
  - `depot-surface`, `depot-surface-elevated`
- **Responsive Breakpoints:** Default Tailwind breakpoints + custom
- **Animations:** Accordion, fade, slide animations
- **Dark Mode:** Class-based dark mode support

---

6. Application Features

6.1 Dashboard Overview

The main dashboard displays a comprehensive view with three columns:

6.1.1 Left Column

- **News Card** - Latest news and announcements
- **In Depot Card** - Real-time staff check-in status

6.1.2 Right Column (2 columns wide)

- **Performance Dashboard** - KPI metrics with charts
  - Spotlight reporting (YTD/MTD)
  - Safety tour statistics
  - Possession utilization metrics
  - Visual pie charts for status indicators
- **Weather Card** - Current weather information
- **RED Safety Video Card** - Safety training videos from YouTube

6.2 Staff Check-In System

Component: `StaffCheckIn.tsx`

Features:

- Staff check-in with name, company, and reason
- Visual display of currently checked-in staff
- Check-out functionality
- Re-check-in support for returning staff
- Real-time updates using React Query

6.3 Document Management

6.3.1 Document Viewer

- Browse and view warehouse documents

- PDF viewing capabilities
- Document categorization

6.3.2 Document Control

- Access to document control systems
- Version tracking
- Compliance documentation

6.4 Safety Features

6.4.1 Safety Alerts

Component: `SafetyAlerts.tsx`

- Display critical safety notifications
- Color-coded alert levels
- Dismissible notifications

6.4.2 Depot Induction

- Video-based induction training
- Progress tracking
- Compliance verification

6.4.3 RED Safety Videos

- Integration with YouTube channel
- Curated safety training content
- Video playback interface

6.5 Technical Library

Integration: SharePoint-based document repository

Access to:

- Technical specifications
- Operating procedures
- Maintenance documentation

- Engineering references

6.6 Spotlight Reporting

Integration: Microsoft Forms

- Incident reporting system
- Safety observations
- Continuous improvement submissions

---

7. Component Architecture

7.1 Page Components

7.1.1 Dashboard (`Dashboard.tsx`)

Location: `src/pages/Dashboard.tsx`

The main application page that orchestrates all dashboard components.

Key Features:

- Section-based routing (without URL changes)
- State management for active section
- Responsive layout with grid system
- Component composition

Props & State:

```
interface DashboardState {
  activeSection: string;        // Current view
  checkedInStaff: CheckInRecord[];  // Staff records
}
```

Sections:

- `dashboard` - Main overview
- `checkin` - Staff check-in interface

- `documents` - Document viewer
- `videos` / `induction` - Video section
- `safety-alerts` - Safety notifications
- `document-control` - Document control system
- `technical` - Technical library (SharePoint)
- `spotlight` - Spotlight reporting form

7.2 Core Components

7.2.1 DepotHeader

Location: `src/components/DepotHeader.tsx`

Purpose: Application header with branding and user info

Features:

- RSRG logo display
- Configurable title
- User profile dropdown (optional)
- Responsive sizing

Props:

```
interface DepotHeaderProps {
  title?: string;
  userName?: string;
  userEmail?: string;
  userAvatar?: string;
  onProfileClick?: () => void;
  onLogoutClick?: () => void;
}
```

7.2.2 DepotNavigation

Location: `src/components/DepotNavigation.tsx`

Purpose: Main navigation bar for section switching

Features:

- 7 navigation buttons
- Active state indication
- Responsive grid layout
- Color-coded buttons

Navigation Items:

1. Dashboard (Accent)
2. Staff Check In/Out (Default)
3. Depot Induction (Default)
4. Safety Alerts (Default)
5. Document Control (Default)
6. Technical Library (Default)
7. Spotlight Report (Accent)

7.2.3 PerformanceDashboard

Location: `src/components/PerformanceDashboard.tsx`

Purpose: Display KPI metrics and performance indicators

Features:

- Spotlight reporting statistics (YTD/MTD)
- Safety tour metrics
- Possession utilization data
- Three pie charts for visual representation:
  - In Process Status (Red/Yellow)
  - Possession Status (Blue/Gray)
  - Preparation Status (Purple/Orange)
- Responsive layout with background image
- Uses Recharts for data visualization

7.2.4 InDepotCard

Location: `src/components/InDepotCard.tsx`

Purpose: Display currently checked-in staff

Features:

- Real-time staff list
- Check-out functionality
- Time tracking
- Visual status indicators

7.2.5 StaffCheckIn

Location: `src/components/StaffCheckIn.tsx`

Purpose: Comprehensive staff check-in/out management

Features:

- Form-based check-in
- Staff listing with filters
- Check-out and re-check-in actions
- Form validation with Zod schema
- Integration with React Query mutations

Form Fields:

- Name (required, min 2 characters)
- Company (required, min 2 characters)
- Reason for visit (required, min 3 characters)

7.2.6 NewsCard

Location: `src/components/NewsCard.tsx`

Purpose: Display news and announcements

Features:

- Scrolling news feed
- Date/time stamps
- Priority indicators

7.2.7 WeatherCard

Location: `src/components/WeatherCard.tsx`

Purpose: Current weather information display

Features:

- Temperature display
- Weather conditions
- Location-based data

7.2.8 REDSafetyVideoCard

Location: `src/components/REDSafetyVideoCard.tsx`

Purpose: Display and play safety training videos

Features:

- YouTube video integration
- Video carousel
- Loading states
- Error handling
- Video metadata display

Props:

```
interface Props {
  videos: YouTubeVideo[];
  loading: boolean;
  error: string | null;
}
```

7.2.9 WebViewer

Location: `src/components/WebViewer.tsx`

Purpose: Embed external web content

Features:

- Iframe-based content display
- Back navigation
- Full-screen support

- Loading indicators

Use Cases:

- SharePoint Technical Library
- Microsoft Forms (Spotlight Reports)

---

8. API Integration

8.1 API Configuration

File: `src/lib/api-config.ts`

8.1.1 Base Configuration

```
const API_URL = import.meta.env.VITE_API_URL;

export const apiConfig = {
  baseURL: API_URL,
  headers: {
    "Content-Type": "application/json",
  },
};
```

8.1.2 Request Handler

The `apiRequest` function provides:

- Type-safe API calls
- Automatic token injection
- Token refresh on 401 errors
- Error handling
- Session management

Authentication Flow:

1. Check for access token in localStorage
2. Add Authorization header if token exists
3. On 401 error, attempt token refresh

4. Retry original request with new token
5. Redirect to signin on refresh failure

8.2 API Modules

8.2.1 Dashboard API

File: `src/api/dashboard-api.ts`

Purpose: YouTube video data fetching

Endpoints:

- `GET /api/youtube/list/` - Fetch safety videos

Types:

```
interface YouTubeVideo {
  id: number;
  video_id: string;
  title: string;
  description: string;
  thumbnail_url: string;
  video_url: string;
  duration: string;
  published_at: string;
  view_count: number;
  like_count: number;
  channel_title: string;
  fetched_at: string;
  updated_at: string;
  is_active: boolean;
}

interface YouTubeListResponse {
  success: boolean;
  message: string;
  videos: YouTubeVideo[];
  from_cache: boolean;
  fetched_today: boolean;
  total_videos: number;
  videos_updated?: number;
}
```

React Query Hook:


```
useYouTubeVideos()
```

- Cache time: 10 minutes
- Stale time: 5 minutes
- No refetch on window focus

8.2.2 Check-In API

File: `src/api/checkin-api.ts`

Purpose: Staff check-in/out management

Endpoints:

- `GET /api/depot/checkin/` - List all check-in records
- `POST /api/depot/checkin/` - Create new check-in
- `POST /api/depot/checkin/{id}/checkout/` - Check out staff
- `POST /api/depot/checkin/{id}/checkin/` - Re-check in staff

Types:

```
interface CheckInRecord {
  id: number;
  company: string;
  name: string;
  reason: string;
  check_in_time: string;
  check_out_time: string | null;
  status: "checked-in" | "checked-out";
  created_at: string;
  updated_at: string;
}
```

React Query Hooks:

- `useCheckInRecords()` - Fetch records (2 min stale time)
- `useCreateCheckIn()` - Create check-in with cache invalidation
- `useCheckOutUser()` - Check out with cache invalidation
- `useReCheckInUser()` - Re-check in with cache invalidation

8.3 Authentication

File: `src/lib/auth-api.ts`

While authentication is implemented, it's currently not actively used in the main dashboard flow. The system includes:

- Token storage (localStorage)
- Automatic token refresh
- Session management

---

9. Routing & Navigation

9.1 Route Structure

File: `src/App.tsx`

The application uses React Router DOM with the following routes:

```
<Routes>
  <Route path="/" element={<Dashboard />} />
  <Route path="/dashboard" element={<Dashboard />} />
  <Route path="*" element={<NotFound />} />
</Routes>
```

Note: Admin panel route is commented out:

```
{/* <Route path="/admin" element={<AdminPanel />} /> */}
```

9.2 Navigation Flow

The application uses a hybrid navigation approach:

9.2.1 URL-Based Routing

- `/` or `/dashboard` - Main dashboard
- `*` - 404 Not Found page

9.2.2 Section-Based Navigation (No URL change)

Within the dashboard, navigation occurs via state changes without URL updates:

- Dashboard overview
- Staff Check-In
- Documents
- Videos/Induction
- Safety Alerts
- Document Control
- Technical Library (iframe)
- Spotlight Reports (iframe)

Implementation:

```
const [activeSection, setActiveSection] = useState<string>("dashboard");
```

Navigation is triggered by the `DepotNavigation` component buttons.

---

10. UI/UX Design System

10.1 Design Tokens

10.1.1 Color Palette

Custom Depot Colors:

- **Header:** `hsl(var(--depot-header))` - Header background
- **Primary:** `hsl(var(--depot-primary))` - Primary actions
- **Accent:** `hsl(var(--depot-accent))` - Accent buttons, highlights
- **Warning:** `hsl(var(--depot-warning))` - Warning states
- **Success:** `hsl(var(--depot-success))` - Success states
- **Surface:** `hsl(var(--depot-surface))` - Card backgrounds
- **Surface Elevated:** `hsl(var(--depot-surface-elevated))` - Elevated elements

10.1.2 Typography

The application uses responsive typography with viewport-based units:

- Headers: `text-[3vw]` to `text-[5vw]`
- Navigation: `text-[1.5vw]`
- Body text: Standard Tailwind scales
- Small text: `text-[1vw]`

10.1.3 Spacing

Consistent spacing scale:

- Micro: `0.5` (2px)
- Small: `1` (4px)
- Medium: `2` (8px)
- Large: `3` (12px), `4` (16px)
- Extra Large: `6` (24px), `8` (32px)

10.2 Component Variants

10.2.1 Button Variants

- **Default:** Standard depot button
- **Secondary:** Secondary actions
- **Accent:** Primary/highlighted actions
- **Warning:** Caution actions
- **Success:** Positive actions
- **Surface:** Subtle actions

10.2.2 Card Variants

- Standard card with shadow
- Elevated card with higher shadow
- Flat card without shadow

10.3 Responsive Design

The application uses Tailwind's responsive utilities:

Breakpoints:

- `xs` : Extra small screens
- `sm` : 640px and up
- `md` : 768px and up
- `lg` : 1024px and up
- `xl` : 1280px and up
- `2xl` : 1536px and up
- Custom: `min-[1920px]` for large displays

Layout Approach:

- Mobile-first responsive design
- Grid-based layouts
- Flexible spacing
- Viewport-relative units for large screens

10.4 Accessibility

The application leverages Radix UI primitives which provide:

- ARIA attributes
- Keyboard navigation
- Focus management
- Screen reader support

---

11. Development Workflow

11.1 Getting Started

11.1.1 Prerequisites

- Node.js 18+ or Bun

- npm, yarn, or bun package manager

11.1.2 Installation

```
# Clone the repository
cd warehouse-frontend

# Install dependencies
npm install
# or
bun install

# Copy environment variables
cp .env.example .env  # Update with actual values

# Start development server
npm run dev
# or
bun dev
```

The application will be available at `http://localhost:8080`

11.2 Available Scripts

File: `package.json`

```
{
  "scripts": {
    "dev": "vite",                          // Start dev server
    "build": "vite build",                  // Production build
    "build:dev": "vite build --mode development",  // Dev build
    "lint": "eslint .",                     // Run linter
    "preview": "vite preview"               // Preview production build
  }
}
```

11.3 Development Guidelines

11.3.1 Code Organization

- Keep components small and focused
- Use TypeScript for type safety

- Extract shared logic into custom hooks
- Maintain consistent file naming (PascalCase for components)

11.3.2 Styling Conventions

- Use Tailwind utility classes
- Follow mobile-first approach
- Use custom depot color tokens
- Maintain consistent spacing

11.3.3 State Management

- Use React Query for server state
- Use local state (`useState`) for UI state
- Avoid prop drilling with composition

11.3.4 API Integration

- Define TypeScript interfaces for all API responses
- Use React Query hooks for data fetching
- Handle loading and error states
- Implement optimistic updates for mutations

11.4 Git Workflow

Current Branch: `main`

Recent Commits:

```
651974a Added new feature
97e3ec5 updated navigation bar
b55b7aa update: some features
1d2740f fix: some issue
22bec28 fix: excel
```

Recommended Workflow:

1. Create feature branches from `main`
2. Make focused commits with clear messages

3. Test thoroughly before merging
4. Use pull requests for code review

---

12. Build & Deployment

12.1 Production Build

```
# Build for production
npm run build

# Output directory: dist/
```

Build Process:

1. TypeScript compilation
2. Asset optimization
3. Code splitting
4. Minification
5. Tree shaking

12.2 Build Modes

12.2.1 Production Build

```
npm run build
```

- Optimized and minified
- Source maps disabled
- Production environment variables

12.2.2 Development Build

```
npm run build:dev
```

- Includes source maps
- Development environment variables
- Faster build time

12.3 Preview Build

```
npm run preview
```

Preview the production build locally before deployment.

12.4 Deployment Configuration

File: `vercel.json`

```
{
  "rewrites": [
    { "source": "/(.*)", "destination": "/" }
  ]
}
```

This configuration ensures client-side routing works correctly by redirecting all routes to the index page.

12.5 Environment-Specific Configuration

Ensure environment variables are properly set for each environment:

Development:

- Local API URL (if applicable)
- Development API keys

Production:

- Production API URL: `https://warehouse-backend-ax8l.onrender.com`
- Production API keys

12.6 Hosting Recommendations

The application is configured for static hosting platforms:

- **Vercel** (configured with `vercel.json`)
- **Netlify**
- **AWS S3 + CloudFront**
- **GitHub Pages**

Requirements:

- SPA routing support (URL rewrites)
- HTTPS enabled
- Environment variable configuration

---

13. Best Practices

13.1 Performance Optimization

13.1.1 Code Splitting

- Lazy load route components
- Dynamic imports for large components
- Split vendor bundles

13.1.2 React Query Configuration

- Appropriate stale times (2-5 minutes)
- Cache times (5-10 minutes)
- Disable unnecessary refetching
- Implement optimistic updates

13.1.3 Image Optimization

- Use appropriate image formats (WebP, AVIF)

- Lazy load images
- Provide responsive images

13.1.4 Bundle Size

- Tree shake unused dependencies
- Monitor bundle size
- Use production builds

13.2 Security Best Practices

13.2.1 Environment Variables

- Never commit `.env` files
- Use different keys for dev/prod
- Rotate API keys regularly

13.2.2 API Security

- Always use HTTPS
- Implement proper CORS policies
- Validate all inputs
- Sanitize user data

13.2.3 Authentication

- Secure token storage
- Implement token refresh
- Handle session expiration
- Clear tokens on logout

13.3 Code Quality

13.3.1 TypeScript

- Enable strict mode
- Define interfaces for all data structures
- Avoid `any` type

- Use proper type guards

13.3.2 ESLint

```
npm run lint
```

- Fix linting errors before committing
- Follow ESLint recommendations
- Use TypeScript ESLint rules

13.3.3 Component Design

- Single Responsibility Principle
- Prop validation with TypeScript
- Error boundaries for error handling
- Consistent naming conventions

13.4 Testing Recommendations

While tests are not currently implemented, consider:

- **Unit Tests:** Component logic, utilities
- **Integration Tests:** API interactions, form submissions
- **E2E Tests:** Critical user flows
- **Tools:** Vitest, React Testing Library, Playwright

---

14. Troubleshooting

14.1 Common Issues

14.1.1 Build Failures

Issue: TypeScript errors during build

Solution:


```
# Check TypeScript errors
npx tsc --noEmit

# Fix type errors
# Review tsconfig.json settings
```

14.1.2 API Connection Issues

Issue: Cannot connect to backend API

Checklist:

- Verify `VITE_API_URL` in `.env`
- Check API server status
- Verify CORS configuration
- Check network/firewall settings
- Inspect browser console for errors

14.1.3 Environment Variables Not Loading

Issue: Environment variables return `undefined`

Solution:

- Ensure variables are prefixed with `VITE_`
- Restart development server after changing `.env`
- Check `.env` file is in project root
- Verify `.env` is not in `.gitignore` locally

14.1.4 Module Resolution Errors

Issue: Cannot resolve '@/...' imports

Solution:

- Verify `vite.config.ts` alias configuration
- Check `tsconfig.json` paths configuration
- Restart TypeScript server in IDE

14.2 Performance Issues

14.2.1 Slow Initial Load

Potential Causes:

- Large bundle size
- Unoptimized images
- Slow API responses

Solutions:

- Analyze bundle with `vite build --mode production`
- Implement code splitting
- Optimize images
- Add loading states
- Implement skeleton screens

14.2.2 React Query Cache Issues

Issue: Stale data or unnecessary refetches

Solution:

```
// Adjust query configuration
useQuery({
  queryKey: ['key'],
  queryFn: fetchFn,
  staleTime: 5 * 60 * 1000,      // Data considered fresh for 5 minutes
  gcTime: 10 * 60 * 1000,        // Cache kept for 10 minutes
  refetchOnWindowFocus: false,   // Disable automatic refetch
});
```

14.3 Debugging

14.3.1 React Query DevTools

Enable during development:


```
import { ReactQueryDevtools } from '@tanstack/react-query-devtools';

// In App.tsx
<ReactQueryDevtools initialIsOpen={false} />
```

14.3.2 Console Logging

Use structured logging:

```
console.log('[ComponentName]', 'Message', data);
```

14.3.3 Browser DevTools

- **Network Tab:** Monitor API calls
- **Console Tab:** View logs and errors
- **React DevTools:** Inspect component tree
- **Performance Tab:** Profile render performance

14.4 Getting Help

14.4.1 Documentation Resources

- React: https://react.dev
- Vite: https://vitejs.dev
- TanStack Query: https://tanstack.com/query
- Tailwind CSS: https://tailwindcss.com
- Radix UI: https://www.radix-ui.com

14.4.2 Common Error Messages

"Failed to fetch"

- API server is down or unreachable
- CORS issue
- Network connectivity problem

"401 Unauthorized"

- Invalid or expired token
- Token refresh failed
- Authentication required

“Module not found”

- Incorrect import path
- Missing dependency
- Path alias misconfiguration

Appendix A: API Endpoints Reference

Base URL

```
https://warehouse-backend-ax8l.onrender.com
```

Endpoints

Method	Endpoint	Description	Response
GET	`/api/youtube/list/`	Fetch YouTube videos	`YouTubeListResponse`
GET	`/api/depot/checkin/`	List check-in records	`CheckInListResponse`
POST	`/api/depot/checkin/`	Create check-in	`CheckInResponse`
POST	`/api/depot/checkin/{id}/checkout/`	Check out staff	`CheckOutResponse`
POST	`/api/depot/checkin/{id}/checkin/`	Re-check in staff	`CheckInResponse`

Appendix B: File Structure Reference

```
warehouse-frontend/
├── .git/                          # Git repository
├── node_modules/                  # Dependencies
├── public/                        # Static assets
│   └── RSRG.png                  # Company logo
├── src/
│   ├── api/
│   │   ├── checkin-api.ts        # 120 lines - Check-in API
│   │   └── dashboard-api.ts      # 45 lines - YouTube API
│   ├── components/
│   │   ├── admin/                # Admin components (4 files)
│   │   │   ├── AnalyticsDashboard.tsx
│   │   │   ├── ContentManagement.tsx
│   │   │   ├── SystemSettings.tsx
│   │   │   ├── UserManagement.tsx
│   │   │   └── VideoManagement.tsx
│   │   ├── ui/                   # Base UI components (40+ files)
│   │   ├── AutoCarousel.tsx
│   │   ├── DepotHeader.tsx       # 74 lines
│   │   ├── DepotNavigation.tsx   # 121 lines
│   │   ├── DocumentControl.tsx
│   │   ├── DocumentViewer.tsx
│   │   ├── InDepotCard.tsx
│   │   ├── NewsCard.tsx
│   │   ├── PerformanceDashboard.tsx  # 176 lines
│   │   ├── REDSafetyVideoCard.tsx
│   │   ├── SafetyAlerts.tsx
│   │   ├── StaffCheckIn.tsx      # Complex form component
│   │   ├── VideoSection.tsx
│   │   ├── WeatherCard.tsx
│   │   └── WebViewer.tsx
│   ├── hooks/                    # Custom hooks directory
│   ├── lib/
│   │   ├── api-config.ts         # 90 lines - API configuration
│   │   ├── auth-api.ts           # Authentication utilities
│   │   └── utils.ts              # General utilities
│   ├── pages/
```



```
│   │   ├── AdminPanel.tsx       # Admin interface
│   │   ├── Dashboard.tsx        # 126 lines - Main dashboard
│   │   └── NotFound.tsx         # 404 page
│   ├── App.tsx                  # 30 lines - Root component
│   ├── index.css                # Global styles
│   ├── main.tsx                 # 6 lines - Entry point
│   └── vite-env.d.ts           # Vite type declarations
├── .env                         # Environment variables
├── .gitignore                   # Git ignore rules
├── bun.lockb                    # Bun lock file
├── components.json              # shadcn/ui config
├── eslint.config.js             # ESLint configuration
├── index.html                   # HTML entry point
├── package-lock.json            # npm lock file
├── package.json                 # 86 lines - Project metadata
├── postcss.config.js            # PostCSS configuration
├── tailwind.config.ts           # 101 lines - Tailwind config
├── tsconfig.app.json            # TypeScript app config
├── tsconfig.json                # TypeScript base config
├── tsconfig.node.json           # TypeScript Node config
├── vercel.json                  # Vercel deployment config
└── vite.config.ts               # 16 lines - Vite configuration
```

Appendix C: Color Reference

Depot Color Palette

```
/* Custom Depot Colors (defined in CSS variables) */
--depot-header: /* Header background */
--depot-primary: /* Primary actions */
--depot-accent: /* Accent highlights */
--depot-warning: /* Warning states */
--depot-success: /* Success states */
--depot-surface: /* Card backgrounds */
--depot-surface-elevated: /* Elevated elements */
```


Chart Colors

Performance Dashboard:

- Critical (Red): `#ef4444`
- Warning (Yellow): `#facc15`
- Active (Blue): `#3b82f6`
- Inactive (Gray): `#e5e7eb`
- Preparation (Purple): `#8b5cf6`
- Breakdown (Orange): `#f97316`

Appendix D: Component Props Reference

DepotHeader Props

```
interface DepotHeaderProps {
  title?: string;              // Default: "Safety & Performance Station"
  userName?: string;           // Optional user name
  userEmail?: string;          // Optional user email
  userAvatar?: string;         // Optional avatar URL
  onProfileClick?: () => void; // Profile click handler
  onLogoutClick?: () => void;  // Logout click handler
}
```

DepotNavigation Props

```
interface DepotNavigationProps {
  onItemClick: (itemId: string) => void;  // Navigation handler
  activeItem?: string;                     // Current active section
}
```


REDSafetyVideoCard Props

```
interface REDSafetyVideoCardProps {
  videos: YouTubeVideo[];    // Array of video objects
  loading: boolean;          // Loading state
  error: string | null;      // Error message
}
```

WebViewer Props

```
interface WebViewerProps {
  url: string;                        // URL to embed
  title: string;                      // Viewer title
  onBack: () => void;                // Back button handler
}
```

Conclusion

This documentation provides a comprehensive overview of the Warehouse Frontend application. It covers the technical architecture, project structure, key features, and development guidelines necessary for understanding, maintaining, and extending the application.

For additional support or questions, please refer to the official documentation links provided in the Troubleshooting section or contact the development team.

Document Version: 1.0 **Last Updated:** October 13, 2025 **Maintained By:** Technical Documentation Team

