Frontend Architecture

React Frontend Application

> Auto-generated documentation for the client module

Last Updated: 2026-05-06T07:42:11.908Z

Table of Contents

Overview

React Frontend Application

File Structure

```
client/src/utils/machineUtils.js
client/src/services/qrPaymentService.js
client/src/services/pricingService.js
client/src/services/machineService.js
client/src/pages/index.js
client/src/hooks/useWifiManager.js
client/src/hooks/useTheme.js
```

```
client/src/hooks/useReconfigureSocket.js
client/src/hooks/usePaymentTracking.js
client/src/hooks/useNetworkStatus.js
client/src/hooks/useMaintenanceSocket.js
client/src/hooks/useMachineSocket.js
client/src/hooks/useKioskSecurity.js
client/src/hooks/useIpAddress.js
client/src/hooks/useIdleTimeout.js
client/src/hooks/useCoinAcceptorStatus.js
client/src/context/createThemeContext.js
client/src/config/api.js
client/src/components/index.js
client/src/main.jsx
client/src/App.jsx
client/src/pages/WelcomePage.jsx
client/src/pages/PaymentTimeoutPage.jsx
client/src/pages/PaymentSuccessPage.jsx
client/src/pages/MaintenancePage.jsx
client/src/pages/LaundryMachinesPricingPage.jsx
client/src/pages/LaundryMachinesPage.jsx
client/src/pages/ConfigurePage.jsx
client/src/context/ThemeContext.jsx
client/src/components/Header.jsx
client/src/components/Footer.jsx
client/src/components/ui/PricingOptions.jsx
client/src/components/ui/PaymentSummary.jsx
client/src/components/ui/NumericKeyboard.jsx
client/src/components/ui/MachineCard.jsx
client/src/components/ui/LoadingSpinner.jsx
client/src/components/ui/LoadingPage.jsx
client/src/components/ui/InternalKeyboard.jsx
client/src/components/ui/ErrorPage.jsx
client/src/components/ui/DuitNowBorder.jsx
```

Components/Modules

client/src

Component: main

Component: App

Hooks Used: useEffect, useNavigate, useLocation, useKioskSecurity, useIdleTimeout, useMaintenanceSocket, useReconfigureSocket, useTheme

client/src/components

client/src/components/Header.jsx

Header Component Global header for the kiosk application. Displays branding, network status, and coin acceptor status. Features: - Branded logo and title - Real-time network status indicator - Real-time coin acceptor status indicator - Optional configure button (maintenance mode) - Theme support (laundro/cuci) - Responsive design

Parameters:

- `props` (`Object`): - Component props
- `props.showConfigureButton` (`boolean`): - Show configure button for maintenance
- `props.theme` (`string`): - Theme override (laundro/cuci/green)

Example:

```
// Standard usage: <Header />
```

```
// With configure button and custom theme: <Header showConfigureButton={true} theme="green" />
```

Component: Header

Hooks Used: useNavigate, useNetworkStatus, useCoinAcceptorStatus, useTheme

client/src/components/Footer.jsx

Footer Component Global footer for the kiosk application. Displays copyright, branding, and support contact information. Features: - Dynamic copyright year - Support phone number from configuration - Theme support (laundro/cuci) - Responsive design

Parameters:

- `props` (`Object`): - Component props
- `props.theme` (`string`): - Theme override (laundro/cuci/green)

Example:

```
// Standard usage: <Footer />
```

```
// With theme override: <Footer theme="green" />
```

Fetch support number from configuration

Component: Footer

Hooks Used: useState, useEffect, useTheme

client/src/components/ui

Component: PricingOptions

Hooks Used: useTheme

Component: PaymentSummary

Hooks Used: useState, useEffect, useRef, useTheme

Component: NumericKeyboard

Component: MachineCard

Hooks Used: useTheme

client/src/components/ui/LoadingSpinner.jsx

LoadingSpinner Component A reusable loading spinner with customizable message

Parameters:

- `props` (`Object`): - Component props
- `props.message` (`string`): - Loading message to display
- `props.size` (`string`): - Size classes for the spinner
- `props.textSize` (`string`): - Text size class for the message
- `props.showMessage` (`boolean`): - Whether to show the loading message

Component: LoadingSpinner

Hooks Used: useTheme

Component: LoadingPage

Component: InternalKeyboard

Hooks Used: useState

Component: ErrorPage

Hooks Used: useNavigate, useTheme

Component: DuitNowBorder

client/src/config

client/src/config/api.js

API Configuration Centralized API endpoint configuration for the Offline Kiosk frontend. Handles environment-specific base URLs and provides typed endpoint constants.

Example:

```
// Using in a component: import API_ENDPOINTS from '../config/api'; const fetchMachines = async () => { const response = await fetch(API_ENDPOINTS.MACHINES); const data = await response.json(); return data; };
```

Base URL for all API requests Can be overridden via VITE_API_BASE_URL environment variable

API Endpoint Constants Provides a centralized location for all API endpoints used in the application. All endpoints are relative to the API_BASE_URL.

client/src/context

Component: ThemeContext

Hooks Used: useState, useEffect

client/src/hooks

client/src/hooks/useWifiManager.js

useWifiManager Hook Manages WiFi connectivity operations for the kiosk. Interfaces with WiFi manager API endpoints to scan, connect, and manage networks. Features: - Fetch available WiFi networks - Get currently connected network - Connect to WiFi with credentials - Forget/disconnect from networks - Loading and error states

Returns:

- `Object`: management state and functions
- `Array`: - Array of available WiFi networks
- `string|null`: - Currently connected network SSID
- `boolean`: - Loading state for operations
- `string|null`: - Error message if operation failed
- `Function`: - Fetch available networks
- `Function`: - Connect to a network
- `Function`: - Forget a saved network

Example:

```
const { wifiList, connectedSsid, loading, connectWifi, fetchWifiList } = useWifiManager();
useEffect(() => { fetchWifiList(); }, []); const handleConnect = async (ssid, password) => {
const result = await connectWifi(ssid, password); if (result.success) {
console.log('Connected!'); } };
```

Fetch available WiFi networks and current connection status

Returns:

- `Promise<void>`:

Connect to a WiFi network

Parameters:

- `ssid` (`string`): - Network SSID
- `password` (`string`): - Network password

Returns:

- `Promise<Object>`: object with success boolean

Forget (remove) a saved WiFi network

Parameters:

- `ssid` (`string`): - Network SSID to forget

Returns:

- `Promise<Object>`: object with success boolean

client/src/hooks/usePaymentTracking.js

usePaymentTracking Hook Tracks real-time payment progress via WebSocket. Monitors coin and e-payment amounts, calculates remaining balance and change. Features: - Real-time payment updates via Socket.IO - Automatic payment completion detection - Change amount calculation - Connection status tracking

Parameters:

- `targetAmount` (`number`): - Target payment amount

Returns:

- `Object`: tracking state
- `Object`: - Payment details object
- `number`: - Current coin payment amount
- `number`: - Current e-payment amount
- `number`: - Total amount paid (coin + epay)
- `number`: - Remaining amount to complete payment
- `boolean`: - Whether payment is complete
- `number`: - Change amount (overpayment)
- `boolean`: - WebSocket connection status
- `Object`: - Socket.IO client instance

Example:

```
const { paymentData, isConnected } = usePaymentTracking(5.00); useEffect(() => { if (paymentData.isPaymentComplete) { console.log('Payment complete! Change:', paymentData.changeAmount); } }, [paymentData]);
```

client/src/hooks/useNetworkStatus.js

useNetworkStatus Hook Monitors internet connectivity by pinging Google's generate_204 endpoint. Sends heartbeat status updates to backend server for kiosk monitoring. Features: - Periodic internet connectivity checks (every 5 seconds) - IP address detection - Heartbeat status reporting to backend (every minute) - Automatic retry on failure

Returns:

- `Object`: status
- `boolean`: - Whether kiosk has internet connectivity

Example:

```
const { isOnline } = useNetworkStatus(); return ( <div> Status: {isOnline ? 'Connected' : 'Offline'} </div> );
```

client/src/hooks/useMachineSocket.js

useMachineSocket Hook Establishes WebSocket connection for real-time machine status updates. Subscribes to 'machines-update' events from the server. Features: - Auto-connects to Socket.IO server on mount - Receives real-time machine status updates - Tracks connection status - Auto-cleanup on unmount

Returns:

- `Object`: state and data
- `Array`: - Array of machine objects with current status
- `boolean`: - WebSocket connection status
- `Object`: - Socket.IO client instance

Example:

```
const { machines, isConnected } = useMachineSocket(); useEffect(() => { if (machines.length > 0) { console.log('Machines updated:', machines); } }, [machines]);
```

client/src/hooks/useKioskSecurity.js

Custom hook for implementing kiosk security measures Prevents common user interactions that could compromise kiosk mode

client/src/hooks/useIpAddress.js

useIpAddress Hook Fetches and tracks the kiosk's local IP address. IP is detected by the ip-detector service running with host network. Features: - Fetch IP address on mount - Loading and error states - Retry capability via refetchIpAddress

Returns:

- `Object`: address state
- `string|null`: - Local IP address (e.g., '192.168.1.100')
- `boolean`: - Loading state
- `string|null`: - Error message if fetch failed
- `Function`: - Manual IP setter
- `Function`: - Refetch IP address

Example:

```
const { ipAddress, loading, error } = useIpAddress(); if (loading) return <Spinner />; if (error) return <div>Error: {error}</div>; return <div>Kiosk IP: {ipAddress}</div>;
```

client/src/hooks/useIdleTimeout.js

Custom hook for handling idle timeout in kiosk applications Automatically triggers a callback after a specified period of user inactivity

Parameters:

- `onIdle` (`Function`): - Callback function to execute when user becomes idle
- `timeout` (`number`): - Timeout duration in milliseconds (default: 60000ms = 1 minute)
- `enabled` (`boolean`): - Whether the idle timeout is enabled (default: true)
- `events` (`Array`): - Array of event types to listen for (default: common user interaction events)

Returns:

- `Object`: Object containing reset function and idle state

client/src/hooks/useCoinAcceptorStatus.js

useCoinAcceptorStatus Hook Monitors coin acceptor status in real-time via WebSocket. Used to determine if kiosk can accept coin payments. Features: - Real-time coin acceptor status updates - Connection status tracking - Auto-reconnect on disconnect

Returns:

- `Object`: acceptor state
- `boolean`: - Whether coin acceptor is available
- `boolean`: - WebSocket connection status

Example:

```
const { coinAcceptorStatus, isConnected } = useCoinAcceptorStatus(); if (!coinAcceptorStatus) { console.warn('Coin acceptor not available'); }
```

client/src/pages

client/src/pages/WelcomePage.jsx

WelcomePage Component The landing page of the kiosk application. Displays a welcome screen with: - Branding and welcome message - Configuration status check - Coin acceptor status validation - Navigation to machine selection Features: - Automatic redirect to configuration if not configured - Disabled state when coin acceptor is offline - Themed design with gradient backgrounds - Responsive layout for different screen sizes

Example:

```
// Used in App.jsx routing: <Route path="/" element={<WelcomePage />} />
```

Check configuration status on component mount Redirects to configuration page if kiosk is not configured Maintenance mode is handled globally in App.jsx

Handle start button click Only navigates to machines page if coin acceptor is online

Component: WelcomePage

Hooks Used: useEffect, useNavigate, useTheme, useCoinAcceptorStatus

Component: PaymentTimeoutPage

Hooks Used: useEffect, useState, useNavigate, useLocation

client/src/pages/PaymentSuccessPage.jsx

PaymentSuccessPage Component Displays payment success confirmation after a successful transaction. Shows transaction details, machine information, and next steps. Features: - Success animation - Transaction summary - Change/balance display (if applicable) - Phone number points notification (if provided) - Auto-redirect to home after 10 seconds - Manual navigation options

Example:

```
// Rendered via React Router with transaction state: navigate('/payment-success', { state: { machine, pricing, transaction, machineResponse, duration, currentPrice, isDryer, change, phoneNumber } })
```

Auto-redirect to home after 10 seconds

Navigate to home page

Navigate to machines page

Redirect if required state is missing

Component: PaymentSuccessPage

Hooks Used: useEffect, useNavigate, useLocation

Component: MaintenancePage

Hooks Used: useEffect, useNavigate, useMaintenanceSocket, useTheme

client/src/pages/LaundryMachinesPricingPage.jsx

LaundryMachinesPricingPage Component Handles pricing selection and payment for a chosen machine. Supports both washer (fixed pricing) and dryer (time-based pricing) workflows. Features: - Dynamic pricing calculation based on machine type - Dryer duration selection (initial time + incremental additions) - Coin acceptor integration for payments - QR payment support (if enabled) - Real-time payment tracking via WebSocket - Network status monitoring - Change tracking (coin balance)

Example:

```
// Rendered via React Router with machine state: <Route path="/pricing"
element={<LaundryMachinesPricingPage />} /> // Navigate with machine data:
navigate('/pricing', { state: { machine } })
```

Load pricing data for the selected machine Fetches pricing configurations from API based on machine type and weight. Initializes default duration for dryers.

Handle dryer duration change Updates both selected and custom duration states.

Parameters:

- `d` (`number`): - New duration in minutes

Increment dryer duration Adds runtime increment based on step value (from outlet settings). Respects maximum price/duration limits.

Component: LaundryMachinesPricingPage

Hooks Used: useState, useEffect, useCallback, useNavigate, useLocation, usePaymentTracking, useNetworkStatus, useTheme

client/src/pages/LaundryMachinesPage.jsx

LaundryMachinesPage Component Displays all available laundry machines (washers and dryers) in the outlet. Shows machine status, availability, and allows users to select a machine for payment. Features: - Real-time machine status updates via WebSocket - Visual indicators for machine state (online/offline, running/idle, maintenance) - Machine selection for payment flow - WiFi signal strength display - Automatic status refresh

Example:

```
// Rendered via React Router: <Route path="/machines" element={<LaundryMachinesPage />} />
```

Get status badge CSS class based on machine state

Parameters:

- `machine` (`Object`): - Machine object

Returns:

- `string`: CSS classes for status badge

Load machines from API Fetches initial machine data from the server. Called on component mount.

Update machines when socket receives new data Real-time synchronization of machine status via WebSocket.

Handle machine selection Navigates to pricing page with selected machine data.

Parameters:

- `machine` (`Object`): - Selected machine object

Navigate back to welcome page

Component: LaundryMachinesPage

Hooks Used: useState, useEffect, useNavigate, useMachineSocket, useTheme

Component: ConfigurePage

Hooks Used: useState, useEffect, useRef, useNavigate, useIpAddress, useWifiManager, useNetworkStatus

client/src/services

client/src/services/machineService.js

Machine Service Service layer for interacting with laundry machine-related API endpoints. Handles fetching machine data, status updates, and error handling.

Fetch all available laundry machines from the backend Retrieves a list of all laundry machines configured in the outlet, including their status, type, and availability.

Returns:

- `Promise<Object>`: data object
- `Promise<Object>`: - Operation success status

- `Promise<Array>`: - Array of machine objects
- `Promise<string>`: - Machine ID
- `Promise<string>`: - Machine display name
- `Promise<string>`: - Machine type (washer/dryer)
- `Promise<boolean>`: - Machine active status
- `Promise<boolean>`: - Machine availability

Example:

```
// Fetch machines in a component: import { fetchMachines } from '../services/machineService'; const loadMachines = async () => { try { const data = await fetchMachines(); if (data.success) { console.log('Machines:', data.machines); } } catch (error) { console.error('Failed to load machines:', error.message); } };
```

Machine Service Object Exported service object containing all machine-related API functions. Can be imported as a default export for easier mocking in tests.

client/src/utils

Revision #1
Created 6 May 2026 07:42:19 by Jagjit
Updated 6 May 2026 07:42:19 by Jagjit