Backend Architecture

Node.js Backend Server

> Auto-generated documentation for the server module

Last Updated: 2026-05-06T07:42:11.952Z

Table of Contents

Overview

Node.js Backend Server

File Structure

```
server/server.js
server/utils/winston.js
server/utils/wifiSignal.js
server/utils/logCollection.js
server/utils/dockerRestart.js
server/utils/deployment.js
server/utils/clearDatabase.js
```

```
server/services/transactionSyncScheduler.js
server/services/syncScheduler.js
server/services/sendKioskTransaction.js
server/routes/index.js
server/models/paymentCredentials.js
server/models/Value.js
server/models/Transaction.js
server/models/Pricing.js
server/models/Outlet.js
server/models/Device.js
server/models/Config.js
server/middlewares/auth.js
server/configs/socket.js
server/configs/sequelize.js
server/configs/mqtt.js
server/configs/database.js
server/routes/api/wifimanager.routes.js
server/routes/api/value.routes.js
server/routes/api/sync.routes.js
server/routes/api/qrPayment.routes.js
server/routes/api/pricing.routes.js
server/routes/api/payment.routes.js
server/routes/api/external.routes.js
server/routes/api/device.routes.js
server/routes/api/configure.routes.js
server/routes/api/auth.routes.js
server/controller/api/wifimanager.controller.js
server/controller/api/value.controller.js
server/controller/api/sync.controller.js
server/controller/api/qrPayment.controller.js
server/controller/api/pricing.controller.js
server/controller/api/payment.controller.js
server/controller/api/external.controller.js
server/controller/api/device.controller.js
server/controller/api/configure.controller.js
server/controller/api/auth.controller.js
```

Components/Modules

server

server/server.js

Offline Kiosk Backend Server Main server application for the Offline Kiosk system. Handles: - RESTful API endpoints for kiosk operations - Real-time WebSocket communication with clients - MQTT integration for machine communication - Payment processing and transaction management - Device status monitoring and updates - WiFi and network management - Automatic synchronization with external services

Socket.IO server instance for real-time communication Configured with CORS to allow connections from any origin

File paths for shared data between containers These files are mounted as volumes from host network detection services

Middleware Configuration

API Routes All API endpoints are prefixed with /api

WebSocket Connection Handler Handles new client connections and sends initial state data: - Current IP address from host network - WiFi connection status - Coin acceptor device status

Parameters:

- `socket` (`Socket`): - Socket.IO socket instance

server/configs

server/controller/api

server/controller/api/wifimanager.controller.js

WiFi Manager Controller Manages WiFi connectivity for the kiosk system. Interfaces with the wifi-manager Docker container to: - Scan for available networks - Connect to WiFi networks - Disconnect from networks - Monitor connection status

Path to WiFi list file (shared volume with wifi-manager container)

Path to current WiFi connection file

Get list of available WiFi networks Retrieves scanned WiFi networks from the wifi-manager service. Networks are sorted by signal strength (strongest first).

Parameters:

- `req` (`Object`): - Express request object
- `res` (`Object`): - Express response object

Returns:

- `Array`: of WiFi network objects
- `Object`: - Network name
- `number`: - Signal strength (0-100)
- `string`: - Security type (WPA2, Open, etc.)

Example:

```
// Response: [ { "ssid": "MyNetwork", "signal": 85, "security": "WPA2" }, { "ssid": "GuestWiFi", "signal": 60, "security": "Open" } ]
```

Get currently connected WiFi network Returns the SSID of the currently connected WiFi network. Returns null if not connected.

Parameters:

- `req` (`Object`): - Express request object
- `res` (`Object`): - Express response object

Returns:

- `Object`: WiFi status
- `string|null`: - Connected network name or null

Example:

```
// Connected: { "ssid": "MyNetwork" } // Not connected: { "ssid": null }
```

Connect to a WiFi network Attempts to connect to the specified WiFi network with provided credentials. Executes wifi-connect.sh script in the wifi-manager container.

Parameters:

- `req` (`Object`): - Express request object
- `req.body` (`Object`): - Request body
- `req.body.ssid` (`string`): - Network SSID to connect to
- `req.body.password` (`string`): - Network password

- `res` (`Object`): - Express response object

Returns:

- `Object`: result

Example:

```
// Request: POST /api/wifi-manager/connect { "ssid": "MyNetwork", "password": "mypassword123" } // Success: { "success": true, "message": "Connected successfully" } // Error: { "success": false, "message": "Incorrect password" }
```

Forget (disconnect from) a WiFi network Removes the specified WiFi network from saved connections.

Parameters:

- `req` (`Object`): - Express request object
- `req.body` (`Object`): - Request body
- `req.body.ssid` (`string`): - Network SSID to forget
- `res` (`Object`): - Express response object

Returns:

- `Object`: result

Example:

```
// Request: POST /api/wifi-manager/disconnect { "ssid": "MyNetwork" } // Response: { "success": true, "message": "Network forgotten" }
```

server/controller/api/sync.controller.js

Sync Controller Handles synchronization of data between the kiosk and external core systems. Responsible for: - Syncing outlet information from core system - Syncing pricing data - Syncing device configurations - Syncing payment credentials - Ensuring data consistency across systems

Core system base URL for synchronization

Synchronize outlet data from core system Fetches outlet configuration from the core system and updates local database. Handles both new outlets and updates to existing outlets.

Parameters:

- `req` (`Object`): - Express request object

- `req.body` (`Object`): - Request body
- `req.body.outletId` (`string`): - Unique identifier of the outlet to sync
- `res` (`Object`): - Express response object

Returns:

- `Promise<Object>`: result with statistics

Example:

```
// Request: POST /api/sync/outlet { "outletId": "outlet_123" } // Response: { "success": true, "stats": { "updated": 1, "added": 0, "errors": 0 } }
```

server/controller/api/qrPayment.controller.js

QR Payment Controller Handles QR code generation and payment processing via external payment gateway. Supports e-payment methods like DuitNow QR for cashless transactions. Features: - QR code generation for specific amounts - Payment status tracking - Integration with RHB DuitNow payment gateway - Transaction ID generation and management

External payment API URL

Generate QR code for payment Creates a QR code for the specified amount and machine. Integrates with external payment gateway to generate payment QR codes.

Parameters:

- `req` (`Object`): - Express request object
- `req.body` (`Object`): - Request body
- `req.body.machineId` (`string`): - Machine identifier
- `req.body.amount` (`number`): - Payment amount in smallest currency unit (cents)
- `res` (`Object`): - Express response object

Returns:

- `Promise<Object>`: code data (base64 image)

Example:

```
// Request: POST /api/qr-payment/create { "machineId": "W001", "amount": 500 } // Response: { "success": true, "qrCode": "...", "transactionId": "tx-1234567890-12345" }
```

server/controller/api/pricing.controller.js

Pricing Controller Manages pricing information for laundry machines. Retrieves pricing configurations for different machine types and cycles.

Get all pricing configurations Retrieves all pricing data including machine types, cycle types, and associated outlet information.

Parameters:

- `req` (`Object`): - Express request object
- `res` (`Object`): - Express response object

Returns:

- `Promise<Array>`: of pricing objects with outlet information

Example:

```
// Response: [ { "id": 1, "machine_type": "washer", "cycle_type": "normal", "price": 500, "duration": 30, "outlet": { "id": "outlet_123", "name": "Downtown Laundry" } } ]
```

server/controller/api/payment.controller.js

Payment Controller Handles all payment-related operations for the kiosk system including: - Processing cash and e-payment transactions - Validating payment amounts - Communicating with laundry machines via MQTT - Recording transactions in the database - Sending transaction data to external systems

Process a payment transaction Validates payment amount, creates transaction record, and triggers machine operation. Supports both coin and e-payment methods.

Parameters:

- `req` (`Object`): - Express request object
- `req.body` (`Object`): - Request body
- `req.body.machineId` (`string`): - Unique identifier of the laundry machine
- `req.body.machineType` (`string`): - Type of machine (washer/dryer)
- `req.body.pricingId` (`string`): - Pricing plan identifier
- `req.body.amount` (`number`): - Required payment amount in cents/smallest currency unit
- `req.body.outletName` (`string`): - Name of the outlet/location
- `req.body.outletId` (`string`): - Unique identifier of the outlet
- `req.body.number_phone` (`string`): - Optional customer phone number
- `res` (`Object`): - Express response object

Returns:

- `Promise<Object>`: result with status and details

Example:

```
// Request body example: { "machineId": "W001", "machineType": "washer", "pricingId":
"price_123", "amount": 500, "outletName": "Downtown Laundry", "outletId": "outlet_001",
"number_phone": "+1234567890" } // Success response: { "success": true, "transactionId":
"txn_456", "machineId": "W001", "amount": 500, "change": 0 }
```

server/controller/api/device.controller.js

Device Controller Manages device-related operations including: - Retrieving local IP address from host network detector - Fetching all devices (machines) with outlet information - Getting machine-specific data for UI display - Updating kiosk online/offline status - Heartbeat monitoring

Get local IP address from host network detector Reads the IP address detected by the ip-detector service running with host network. This provides the actual host machine's IP address for kiosk identification.

Parameters:

- `req` (`Object`): - Express request object
- `res` (`Object`): - Express response object

Returns:

- `Promise<Object>`: address object

Example:

```
// Response: { "ipAddress": "192.168.1.100" }
```

Get all devices with outlet information Retrieves complete list of all devices (machines) including associated outlet data.

Parameters:

- `req` (`Object`): - Express request object
- `res` (`Object`): - Express response object

Returns:

- `Promise<Array>`: of device objects with outlet information

Get all machines for UI display Retrieves machines with only essential attributes needed for kiosk UI. Returns machines sorted by machine code for consistent display order.

Parameters:

- `req` (`Object`): - Express request object
- `res` (`Object`): - Express response object

Returns:

- `Promise<Array>`: of machine objects with selected attributes
- `Promise<Object>`: - Machine ID
- `Promise<string>`: - Machine code (W001, D001, etc.)
- `Promise<string>`: - Type (washer/dryer)
- `Promise<boolean>`: - Online status
- `Promise<string>`: - Status (running/idle)
- `Promise<boolean>`: - Locked status
- `Promise<string>`: - Display name

Update kiosk online/offline status Receives heartbeat signals from kiosk to track online status. Can optionally forward status to external monitoring systems.

Parameters:

- `req` (`Object`): - Express request object
- `req.body` (`Object`): - Request body
- `req.body.ipAddress` (`string`): - Kiosk IP address
- `req.body.status` (`string`): - Status (online/offline)
- `res` (`Object`): - Express response object

Returns:

- `Promise<Object>`: update confirmation

Example:

```
// Request: POST /api/device/update-kiosk-status { "ipAddress": "192.168.1.100", "status": "online" }
```

server/controller/api/configure.controller.js

Configure Controller Manages kiosk configuration including: - Initial kiosk setup and configuration - Retrieving configuration data - Maintenance mode management - MQTT reconfiguration and clustering - Database clearing for re-configuration - WebSocket broadcasting for real-time updates

Socket.IO instance for broadcasting configuration changes

Set Socket.IO instance Allows the main server to inject the Socket.IO instance for real-time communication.

Parameters:

- `socketIO` (`Object`): - Socket.IO server instance

Get current kiosk configuration Retrieves the current kiosk configuration including outlet ID, kiosk ID, maintenance status, and branding information.

Parameters:

- `req` (`Object`): - Express request object
- `res` (`Object`): - Express response object

Returns:

- `Promise<Object>`: data or unconfigured status

Example:

```
// Response when configured: { "success": true, "configured": true, "data": { "outletId": "outlet_123", "kioskId": "kiosk_001", "outletName": "Downtown Laundry", "maintenance": false, "brand": "laundro" } }
```

Configure kiosk Performs initial configuration or reconfiguration of the kiosk. Triggers synchronization of outlet data, devices, and pricing. Optionally handles EMQX clustering for secondary nodes.

Parameters:

- `req` (`Object`): - Express request object
- `req.body` (`Object`): - Request body
- `req.body.outletId` (`string`): - Unique outlet identifier (required)
- `req.body.kioskId` (`string`): - Unique kiosk identifier (required)
- `req.body.outletName` (`string`): - Outlet display name (required)
- `req.body.ipAddress` (`string`): - Kiosk IP address (required)
- `req.body.isSecondary` (`boolean`): - Whether this is a secondary MQTT node
- `req.body.primaryIp` (`string`): - Primary node IP (required if isSecondary is true)
- `req.body.supportNumber` (`string`): - Support contact number
- `req.body.brand` (`string`): - Brand theme (laundro/cuci)
- `res` (`Object`): - Express response object

Returns:

- `Promise<Object>`: result

Example:

```
// Request: POST /api/configure { "outletId": "outlet_123", "kioskId": "kiosk_001",
"outletName": "Downtown Laundry", "ipAddress": "192.168.1.100", "brand": "laundro" }
```

server/middlewares

server/models

server/models/Transaction.js

Transaction Model Records all payment transactions made at the kiosk. Stores both coin and e-payment transaction data for audit and reconciliation.

Example:

```
// Create a transaction record: await Transaction.create({ coin_inserted: 5.00,
epayment_amount: 0, machine_id: 'W001', transactionId: 'tx-1234567890', outletId: 'outlet_123'
});
```

```
// Find transactions for a specific machine: const txns = await Transaction.findAll({ where: {
machine_id: 'W001' }, order: [['timestamp', 'DESC']] });
```

server/models/Pricing.js

Pricing Model Stores pricing configurations for different machine types and capacities. Supports both fixed pricing and dynamic pricing models.

Example:

```
// Create washer pricing (fixed): await Pricing.create({ weight: 10, machine_type: 'washer',
amount: 5.00, outletId: 'outlet_123', mode: 'weight-based' });
```

```
// Create dryer pricing (time-based): await Pricing.create({ weight: 10, machine_type:
'dryer', minprice: 1.00, maxprice: 10.00, initialtime: 15, runtime: 5, outletId: 'outlet_123',
mode: 'time-based' });
```

server/models/Device.js

Device Model Represents a laundry machine (washer or dryer) in the kiosk system. Stores machine configuration, status, and operational data.

Example:

```
// Create a new device: await Device.create({ machine_code: 'W001', machine_type: 'washer', active: true, outletId: 'outlet_123' });
```

```
// Find all active machines: const machines = await Device.findAll({ where: { active: true } });
```

server/models/Config.js

Config Model Stores kiosk system configuration including: - Outlet and kiosk identifiers - Network settings - MQTT clustering configuration - Maintenance mode status - Branding/theme settings

Example:

```
// Create initial configuration: await Config.create({ outletId: 'outlet_123', kioskId: 'kiosk_001', outletName: 'Downtown Laundry', ipAddress: '192.168.1.100', brand: 'laundro' });
```

```
// Check if configured: const config = await Config.findOne(); if (config) { console.log('Kiosk is configured'); }
```

server/routes

server/routes/api

API Endpoints:

- `GET` /list
- `GET` /current
- `POST` /connect
- `POST` /forget

API Endpoints:

- `GET` /current
- `PUT` /update

- `POST` /reset

API Endpoints:

- `POST` /outlet
- `POST` /device
- `POST` /pricing
- `POST` /credentials
- `POST` /all

API Endpoints:

- `POST` /generate

API Endpoints:

- `POST` /getPricing

API Endpoints:

- `POST` /process
- `GET` /status/:transactionId
- `POST` /payment
- `POST` /acknowledge

API Endpoints:

- `POST` /getAllMachines
- `GET` /local-ip
- `POST` /update-kiosk-status

API Endpoints:

- `GET` /
- `POST` /
- `POST` /getOutletId
- `POST` /machinevalid
- `POST` /reset

API Endpoints:

- `POST` /login

server/services

server/services/transactionSyncScheduler.js

Transaction Sync Scheduler Service Manages periodic synchronization of transaction data to core system (Laundro). Uploads pending transactions at regular intervals and removes successfully synced ones. Features: - Scheduled sync every 30 minutes - Manual sync trigger - Batch transaction upload - Automatic transaction cleanup after successful sync - Error handling and retry logic - Detailed logging

Sync all unsent transactions to Laundro core system Fetches all pending transactions from local database and uploads them to core system. Deletes successfully synced transactions from local database.

Returns:

- `Promise<Object>`: summary with statistics
- `boolean`: - Overall operation success
- `number`: - Total transactions processed
- `number`: - Successfully synced count
- `number`: - Failed sync count
- `Array`: - Detailed results per transaction

Example:

```
const result = await syncUnsentTransactions(); console.log(`Synced ${result.successCount} of ${result.totalTransactions} transactions`);
```

Schedule transaction sync to run every 2 hours

Run transaction sync on server startup

server/services/syncScheduler.js

Sync Scheduler Service Manages scheduled synchronization of data from core system to kiosk. Runs periodic sync operations via node-cron to keep local data up-to-date. Features: - Scheduled sync every 2 hours - Startup sync on server initialization - Manual sync trigger capability - Syncs outlets, devices, pricing, payment credentials - Error handling and logging

Run all synchronization operations Executes complete data sync from core system including: - Outlet configuration - Device list - Pricing data - Payment credentials

Returns:

- `Promise<Object>`: sync results

Example:

```
// Manual trigger: await runAllSyncOperations();
```

server/utils

server/utils/wifiSignal.js

Get WiFi signal strength and raw dBm Returns { percentage, dbm } or { percentage: 0, dbm: null } if not available Reads from shared volume written by wifi-detector service

server/utils/logCollection.js

Collect logs from all Docker containers and zip them

Returns:

- `Promise<string>`: to the zip file

Create a zip file from a directory

Parameters:

- `sourceDir` (`string`): - Directory to zip
- `outputPath` (`string`): - Output zip file path

Returns:

- `Promise<void>`:

Send logs to Laundro API

Parameters:

- `zipPath` (`string`): - Path to the zip file

Returns:

- `Promise<Object>`: response

Clean up old log zip files

Parameters:

- `zipPath` (`string`): - Path to the zip file to delete

Main function to collect logs and send to Laundro

Returns:

- `Promise<Object>`: of the operation

server/utils/dockerRestart.js

Call Docker API via Unix socket

Parameters:

- `path` (`string`): - API path
- `method` (`string`): - HTTP method

Restart EMQX container using Docker API

Execute command in container using Docker API

Parameters:

- `containerName` (`string`): - Container name
- `cmd` (`array`): - Command array

Join EMQX cluster by connecting to primary node

Parameters:

- `primaryIp` (`string`): - IP address of the primary EMQX node

Perform the actual cluster join operation

server/utils/deployment.js

Execute shell command with Docker socket access

Parameters:

- `command` (`string`): - Command to execute

Returns:

- `Promise<{stdout: string, stderr: string}>`:

Call Docker API via Unix socket

Parameters:

- `path` (`string`): - API path
- `method` (`string`): - HTTP method
- `body` (`object`): - Request body (optional)

Returns:

- `Promise<object>`:

Get Docker Compose project name from environment or default

Returns:

- `string`:

Pull images for all services in docker-compose

Returns:

- `Promise<object>`:

Build EMQX image using docker exec

Returns:

- `Promise<object>`:

Pulls latest image first (or builds for local images), then starts services one by one Order: postgres -> ip-detector -> emqx -> client -> offline-collector Note: server is excluded because this script runs inside the server container

Returns:

- `Promise<object>`:

Restart all containers in the compose project

Returns:

- `Promise<object>`:

Handle deployment automation via MQTT Executes docker compose pull, build emqx, and up -d

Returns:

- `Promise<object>`: result

API Documentation

Base URL

`http://localhost:5000/api`

Authentication

Some endpoints require authentication. Include the auth token in the request headers.

Revision #1
Created 6 May 2026 07:42:21 by Jagjit
Updated 6 May 2026 07:42:21 by Jagjit