Bukidnon State University
 Malaybalay City

College of Technologies
Information Technology Department
Bachelor of Science in Information Technology
$1^{\text {st }}$ Semester SY: 2025 - 2026
IT137 - Integrative Programming and Technologies 2

LaundryPOS Omnichannel Management System API Documentation

I. Overview

1.1 Project Name

LaundryPOS Omnichannel Management System

1.2 Client / Respondents of the System

LaundryPOS Franchise Operations Board

1.3 Description

The LaundryPOS Omnichannel Management System is a comprehensive, web-based platform designed to help franchise operators manage daily laundry orders and branch payments. This all-inclusive solution strives to reduce human error while offering a user-friendly interface for handling laundry services, pricing, and branch-level transactions. The system guarantees accurate recording and arrangement of operational data pertaining to every branch by providing an intuitive interface. The platform's capacity to generate digital receipts and invoices is a crucial feature that improves transparency by giving branch managers instant proof of successful payments.

The system incorporates distinct user roles with specific permissions to maintain data integrity and security. Admins are authorized to configure branches, services, employees, and backups, while staff are granted access to create orders, submit expenses, and update customer information. In addition, the franchise owner monitors customers in the system while the platform administrator has the responsibility of maintaining audit logs and backups. This role-based structure ensures that each user can perform designated tasks effectively while maintaining

appropriate access controls. Automated notifications further streamline the workflow, reducing manual administrative tasks and providing customers with instant proof of order status. It also generates comprehensive reports on service performance and cashflow for the organization.

The LaundryPOS Management System is designed to improve the overall efficiency of laundry business activities. It offers a comprehensive solution for handling customer orders, classifying various services, keeping an up-to-date database of employees, and communicating payment status clearly via automated receipts. By automating several areas of branch operations, the system considerably minimizes the possibility of human error, which improves record accuracy and increases customer satisfaction. This specialized approach allows branch teams to focus on better servicing clients while ensuring a dependable, user-friendly financial tracking system that keeps every member informed and up to date.

1.4 Key Features

- **Secure Login**: Utilizes email/username + password with JWT session tokens so only authorized users can log in.
- **Error Handling**: Implements consistent error objects for validation issues, missing permissions, and expired tokens to enhance troubleshooting.
- **Order Management**: Allows branches to create drafts, finalize orders, apply discounts, and collect partial payments directly from the Admin or Staff apps.
- **Payment Reporting**: Generates per-branch reports, sales dashboards, and exportable summaries for finance reviews.
- **Receipt Management**: Facilitates viewing and emailing of invoices or receipts, with optional PDF attachments for transparency.
- **Calendar & Notifications**: Integrates pickup schedules, branch announcements, and alert banners so staff never miss deadlines.
- **Email Dispatch**: Sends automated notifications to customers and franchise owners covering invoices, payment reminders, and approvals.
- **Cloud Backups**: Provides scheduled and on-demand backups (files and databases) with retention policies managed in the backend.
- **Role-Based Access**: Enforces RBAC so admins, staff, and auditors only see features appropriate to their roles.
- **Archive, Unarchive Records, and Manage Accounts**: Archives orders, customers, and employees instead of deleting them to preserve history.
- **Secure Logout**: Ensures stateless JWT sessions are closed and activity logged after each logout.

1.5 Version

1.6

1.6 Base URL

http://localhost:5000/api

1.7 Authentication

The LaundryPOS Omnichannel Management System uses JWT (JSON Web Token) for secure user authentication. Users supply either an email or username plus password to `/auth/login`, the credentials are validated against the MongoDB user store, and a signed token is returned. Optional safeguards—such as email OTP verification and rate limiting—can be toggled per environment. Password-reset OTPs are delivered through the `/auth/forgot-password` and `/auth/reset-password` endpoints, and all sensitive changes are written to the audit log.

II. Endpoints

The following are the detailed endpoints of LaundryPOS Omnichannel Management System API. These include HTTP methods used in the API call, parameters, configuration, request, and response of the API.

Authentication

Authentication Method

All protected endpoints require JWT authentication via the `Authorization` header:

```
Authorization: Bearer <jwt_token>
```

Token Expiration

- Default: 7 days (configurable via `JWT_EXPIRE` environment variable)

Getting a Token

1. Register a new account: `POST /api/auth/register`
2. Login: `POST /api/auth/login`
3. Receive token in response

API Base URL

- **Development HTTP**: `http://localhost:5000`
- **Development HTTPS**: `https://localhost:5443`
- **Production**: Configured via environment variables (`ALLOWED_ORIGINS`)

Response Format

Success Response

```
{
  "success": true,
  "data": { ... },
  "count": 10  // Optional, for list endpoints
}
```

Error Response

```
{
  "success": false,
  "message": "Error description",
  "error": "Detailed error (development only)"
}
```

III. Authentication Requirements

1. Every protected endpoint must include the `Authorization: Bearer <token>` header.

2. JWT tokens expire after seven days by default; enabling `rememberMe` extends the session to thirty days.
3. Accounts are locked for two hours after five failed login attempts.
4. OTP codes expire after ten minutes and are limited to three requests per hour.
5. `POST /auth/logout` records the logout event; disabling the user account is required to fully revoke stateless tokens.

IV. Error Handling

Status	Meaning	Typical Cause
200	OK	Request processed successfully
201	Created	Resource generated (user, OTP, reset entry)
400	Bad Request	Missing required fields or malformed JSON
401	Unauthorized	Missing/invalid token or incorrect credentials
403	Forbidden	Role lacks permission to access the resource
404	Not Found	Resource not available (profile/email mismatch)
409	Conflict	Duplicate user, locked account, or reused OTP
422	Unprocessable Entity	OTP expired or password policy violation
429	Too Many Requests	Rate limit triggered for login/OTP
500	Internal Server Error	Unexpected error (database outage, service failure)

Modules & Endpoints

2.1 AUTHENTICATION AND GENERAL MODULE API

2.1.1 Module Description

The LaundryPOS authentication module backs both the Admin and Staff apps, covering login, session inspection, password changes, OTP-based resets, and logout. Every sensitive action flows through the JWT middleware (server/middleware/auth.js) and RBAC guard (server/middleware/rbac.js). The API employs comprehensive response codes: 200 (OK) for successful operations, 201 (Created) for resource creation, 400 (Bad Request) for invalid input, 401 (Unauthorized) for missing or invalid tokens, 403 (Forbidden) for insufficient permissions, 404 (Not Found) for unavailable resources, and 500 (Internal Server Error) for unexpected server issues, ensuring secure and clear communication of request outcomes.

The API have the following endpoints:
http://localhost:5000/api/auth/login
http://localhost:5000/api/auth/register
http://localhost:5000/api/auth/me
http://localhost:5000/api/auth/forgot-password
http://localhost:5000/api/auth/reset-password
http://localhost:5000/api/auth/logout

Method	Path	Description
POST	/auth/register	Bootstrap a user account; typically used by admins only.
POST	/auth/login	Issue JWT after verifying email/username + password.
GET	/auth/me	Retrieve the currently authenticated user profile.
PUT	/auth/me	Update profile details such as email, username, or location.
PUT	/auth/change-password	Change password while logged in.
POST	/auth/forgot-password	Send OTP to email for password reset.
POST	/auth/verify-reset-code	Validate OTP prior to resetting password.

POST	/auth/reset-password	Replace password using a valid OTP.
POST	/auth/send-verification-code	Email verification OTP for newly registered users.
POST	/auth/verify-email-code	Confirm the verification OTP.
POST	/auth/logout	Record logout event (JWT remains stateless).
GET	/auth/users	List users with optional role/status filters (admin only).
GET	/auth/profile/:userId	Fetch arbitrary user profile (admin only).
PUT	/auth/deactivate/:userId	Disable a user account.
PUT	/auth/activate/:userId	Reactivate a previously disabled account.

Response Codes of This API

Code	Message	Description
500	Internal Server Error	The server encountered an unexpected condition that prevented it from fulfilling the request.
429	Too Many Requests	The client sends too many requests within a certain period.
404	Not Found	The requested resource could not be found.
403	Forbidden	Invalid token or token expired.
401	Unauthorized	The request was not successful because it lacks valid authentication credentials.
400	Bad Request	The request was invalid.

201	Created	The request was successful, and a new resource has been created.
200	OK	The request succeeded, and the resource is in the message body.

2.1.1.1 Authenticate User and Create Session

Version: 1.6

Date: November 26, 2025

Description: This API endpoint enables the creation of login sessions for secure access to LaundryPOS. Error responses are generated for invalid login attempts, including incorrect credentials or missing parameters. This ensures that only registered users with valid login details can access the system.

Endpoint: http://localhost:5000/api/auth/login

Method: POST

Configurations:

- The API request requires the Content-Type: application/json header.
- When reCAPTCHA is enabled, the request must include a valid token in the body.
- Successful logins store device and IP metadata in the audit log.

Parameters:

{{content}}gt;$ _email - required if _username is not provided; specifies the user's email.

{{content}}gt;$ _username - required if _email is not provided; specifies the user's username.

{{content}}gt;$ _password - required; user's password.

{{content}}gt;$ _recaptchaToken - optional; token submitted by the user when reCAPTCHA is enabled.

{{content}}gt;$ _rememberMe - optional; indicates whether the session should remain active longer.

{{content}}gt;$ _errorMessage - optional; returned in the response body to provide details about any login issues, such as "Invalid username or password" or "reCAPTCHA verification failed."

Requests:

Valid Request

```
{
    "email": "user@example.com",
    "password": "SecurePassword123!",
    "recaptchaToken": "recaptcha_token_here"
}
```

Not Valid Request

```
{
    "email": "user@example.com",
    "password": "wrongpassword"
}
```

Response Format: JSON

Responses:

Success Response

```
{
    "success": true,
    "data": {
        "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
        "user": {
            "_id": "user_id",
            "email": "user@example.com",
            "username": "johndoe",
            "role": "staff",
            "fullName": "John Doe",
            "isActive": true,
            "stationId": "station_id"
        }
    }
}
```


Error Response


```
{
    "success": false,
    "message": "Invalid credentials"
}
```


The API returns a 200 OK status for successful login attempts, confirming access to the LaundryPOS system. The response contains the user's username, email, role, and ID within the organization. Failed reCAPTCHA responses return a 400 Bad Request status code with the message "reCAPTCHA verification failed." Additionally, unauthorized access due to missing or invalid authentication credentials triggers a 401 Unauthorized status code with the message "Invalid email or password," ensuring that secure access policies are enforced.

2.1.1.2 Register User

Version: 1.6

Date: November 26, 2025

Description: Bootstrap a user account; typically used by admins only. This endpoint allows administrators to create new user accounts in the system.

Endpoint: http://localhost:5000/api/auth/register

Method: POST

Configurations:

- The API request requires the Content-Type: application/json header.
- The request must include valid user information in the body.

Parameters:

{{content}}gt;$ _email - required; specifies the user's email address.

{{content}}gt;$ _username - required; specifies the user's username.

{{content}}gt;$ _password - required; user's password (must meet security requirements).

{{content}}gt;$ _role - optional; defines the user's role within the system (defaults to "staff").

Requests:

Valid Request

```
{
    "email": "user@example.com",
    "username": "johndoe",
    "password": "SecurePassword123!",
    "role": "staff"
}
```

Not Valid Request

```
{
    "email": "invalid-email",
    "username": "",
    "password": "123"
}
```

Response Format: JSON

Responses:

Success Response

```
{
    "success": true,
    "data": {
        "token": "jwt_token_here",
        "user": {
            "_id": "user_id",
            "email": "user@example.com",
            "username": "johndoe",
            "role": "staff"
        }
    }
}
```

Error Response

```
{
    "success": false,
    "message": "User already exists"
}
```

The API returns a 201 Created status for successful user registration, confirming the creation of a new user account in the LaundryPOS system. The response contains the user's information and a JWT token for immediate authentication. A 400 Bad Request status indicates invalid input, such as missing required fields or invalid email format. A 409 Conflict status is returned if the user already exists in the system.

2.1.1.3 Forgot Password

Version: 1.6

Date: November 26, 2025

Description: Send OTP to email for password reset. This endpoint initiates the password recovery process by sending a verification code to the user's registered email address.

Endpoint: http://localhost:5000/api/auth/forgot-password

Method: POST

Configurations:

- The API request requires the Content-Type: application/json header.
- Rate limiting is applied to prevent abuse (maximum 3 requests per hour per email).

Parameters:

{{content}}gt;$ _email - required; specifies the user's email address for password reset.

Requests:

Valid Request

```
{
    "email": "user@example.com"
}
```

Not Valid Request

```
{
    "email": "nonexistent@example.com"
}
```

Response Format: JSON

Responses:

Success Response

```
{
    "success": true,
    "message": "Password reset code sent to email"
}
```

Error Response

```
{
    "success": false,
    "message": "User not found"
}
```

The API returns a 200 OK status for successful password reset code delivery, confirming that the OTP has been sent to the user's email. A 404 Not Found status indicates that the email address is not registered in the system. A 429 Too Many Requests status is returned if the rate limit has been exceeded.

2.1.1.4 Reset Password

Version: 1.6

Date: November 26, 2025

Description: Replace password using a valid OTP. This endpoint allows users to reset their password after verifying the OTP code sent to their email.

Endpoint: http://localhost:5000/api/auth/reset-password

Method: POST

Configurations:

- The API request requires the Content-Type: application/json header.
- OTP codes expire after 10 minutes from generation.

Parameters:

{{content}}gt;$ _email - required; specifies the user's email address.

{{content}}gt;$ _code - required; the OTP verification code sent to the user's email.

{{content}}gt;$ _newPassword - required; the new password that meets security requirements.

Requests:

Valid Request

```
{
    "email": "user@example.com",
    "code": "123456",
    "newPassword": "NewSecurePassword123!"
}
```

Not Valid Request

```
{
    "email": "user@example.com",
    "code": "000000",
    "newPassword": "123"
}
```

Response Format: JSON

Responses:

Success Response

```
{
    "success": true,
    "message": "Password reset successfully"
}
```

Error Response

```
{
    "success": false,
    "message": "Invalid or expired OTP code"
}
```

The API returns a 200 OK status for successful password reset, confirming that the user's password has been updated. A 400 Bad Request status indicates invalid input, such as an expired or incorrect OTP code. A 422 Unprocessable Entity status is returned if the new password does not meet security requirements.

- **Request Body**:

```
{
  "email": "user@example.com",
  "username": "johndoe",
  "password": "SecurePassword123!",
  "role": "staff"  // Optional, defaults to "staff"
}
```

- **Response** (201):

```
{
  "success": true,
  "data": {
    "token": "jwt_token_here",
    "user": {
      "_id": "user_id",
      "email": "user@example.com",
      "username": "johndoe",
      "role": "staff"
    }
  }
}
```

Login

- **Endpoint**: `POST /api/auth/login`
- **Description**: Authenticate user and receive JWT token
- **Request Body**:

```
{
  "email": "user@example.com",  // or "username": "johndoe"
  "password": "SecurePassword123!",
  "recaptchaToken": "recaptcha_token"  // Optional, for reCAPTCHA v3
}
```

- **Response** (200):

```
{
  "success": true,
  "data": {
    "token": "jwt_token_here",
    "user": {
      "_id": "user_id",
      "email": "user@example.com",
      "username": "johndoe",
      "role": "staff",
      "fullName": "John Doe"
    }
  }
}
```

Forgot Password

- **Endpoint**: `POST /api/auth/forgot-password`
- **Description**: Request password reset code via email
- **Request Body**:

```
{
  "email": "user@example.com"
}
```

- **Response** (200):

```
{
  "success": true,
  "message": "Password reset code sent to email"
}
```

Verify Reset Code

- **Endpoint**: `POST /api/auth/verify-reset-code`
- **Description**: Verify password reset code
- **Request Body**:

```
{
  "email": "user@example.com",
  "code": "123456"
}
```

- **Response** (200):

```
{
  "success": true,
  "message": "Code verified"
}
```

Reset Password

- **Endpoint**: `POST /api/auth/reset-password`
- **Description**: Reset password with verified code
- **Request Body**:

```
{
  "email": "user@example.com",
  "code": "123456",
  "newPassword": "NewSecurePassword123!"
}
```

- **Response** (200):

```
{
  "success": true,
  "message": "Password reset successfully"
}
```

Protected Endpoints (Authentication Required)

Logout

- **Endpoint**: `POST /api/auth/logout`
- **Description**: Logout current user (token invalidation)
- **Headers**: `Authorization: Bearer <token>`
- **Response** (200):

```
{
  "success": true,
  "message": "Logged out successfully"
}
```

Get Current User

- **Endpoint**: `GET /api/auth/me`
- **Description**: Get current authenticated user profile
- **Headers**: `Authorization: Bearer <token>`
- **Response** (200):

```
{
  "success": true,
  "data": {
    "_id": "user_id",
    "email": "user@example.com",
    "username": "johndoe",
    "role": "staff",
    "fullName": "John Doe",
    "isActive": true,
    "stationId": "station_id"
  }
}
```

Update Profile

- **Endpoint**: `PUT /api/auth/me`
- **Description**: Update current user profile
- **Headers**: `Authorization: Bearer <token>`
- **Request Body**:

```
{
  "email": "newemail@example.com",  // Optional
  "username": "newusername",  // Optional
  "location": {  // Optional, for mobile apps
    "latitude": 14.5995,
    "longitude": 120.9842
  }
}
```

- **Response** (200):

```
{
  "success": true,
  "data": {
    "_id": "user_id",
    "email": "newemail@example.com",
    "username": "newusername"
  }
}
```

Change Password

- **Endpoint**: `PUT /api/auth/change-password`
- **Description**: Change user password
- **Headers**: `Authorization: Bearer <token>`
- **Request Body**:

```
{
  "currentPassword": "OldPassword123!",
  "newPassword": "NewPassword123!"
}
```

- **Response** (200):

```
{
  "success": true,
  "message": "Password changed successfully"
}
```

Send Email Verification Code

- **Endpoint**: `POST /api/auth/send-verification-code`
- **Description**: Send email verification code
- **Headers**: `Authorization: Bearer <token>`
- **Request Body**:

```
{
  "email": "user@example.com"
}
```

- **Response** (200):

```
{
  "success": true,
  "message": "Verification code sent to email"
}
```

Verify Email Code

- **Endpoint**: `POST /api/auth/verify-email-code`
- **Description**: Verify email with code
- **Headers**: `Authorization: Bearer <token>`
- **Request Body**:

```
{
  "email": "user@example.com",
  "code": "123456"
}
```

- **Response** (200):

```
{
  "success": true,
  "message": "Email verified successfully"
}
```

Get User Profile by ID

- **Endpoint**: `GET /api/auth/profile/:userId`
- **Description**: Get user profile by user ID
- **Headers**: `Authorization: Bearer <token>`
- **URL Parameters**: `userId` - User ID

- **Response** (200):

```
{
  "success": true,
  "data": {
    "_id": "user_id",
    "email": "user@example.com",
    "username": "johndoe",
    "role": "staff",
    "fullName": "John Doe"
  }
}
```

Admin Only Endpoints

Get All Users

- **Endpoint**: `GET /api/auth/users`
- **Description**: Get all users (admin only)
- **Headers**: `Authorization: Bearer <token>`
- **Required Role**: `admin`
- **Response** (200):

```
{
  "success": true,
  "data": [
    {
      "_id": "user_id",
      "email": "user@example.com",
      "username": "johndoe",
      "role": "staff",
      "isActive": true
    }
  ],
  "count": 10
}
```

Deactivate User

- **Endpoint**: `PUT /api/auth/deactivate/:userId`
- **Description**: Deactivate user account (admin only)
- **Headers**: `Authorization: Bearer <token>`
- **Required Role**: `admin`
- **URL Parameters**: `userId` - User ID to deactivate
- **Response** (200):

```
{
  "success": true,
  "message": "User deactivated successfully"
}
```

Activate User

- **Endpoint**: `PUT /api/auth/activate/:userId`
- **Description**: Activate user account (admin only)
- **Headers**: `Authorization: Bearer <token>`
- **Required Role**: `admin`
- **URL Parameters**: `userId` - User ID to activate
- **Response** (200):

```
{
  "success": true,
  "message": "User activated successfully"
}
```

Required Permissions:

- `auth:read` - View user profiles
- `auth:update` - Update profiles
- `auth:admin` - Admin operations

2.2 ORDERS MODULE API

2.2.1 Module Description

The Orders module manages all order-related operations including creation, updates, archiving, draft management, and invoice generation. All endpoints require authentication and appropriate RBAC permissions. Staff users are automatically restricted to orders from their assigned station. The API employs comprehensive response codes: 200 (OK) for successful operations, 201 (Created) for resource creation, 400 (Bad Request) for invalid input, 401 (Unauthorized) for missing or invalid tokens, 403 (Forbidden) for insufficient permissions, 404 (Not Found) for unavailable resources, and 500 (Internal Server Error) for unexpected server issues, ensuring secure and clear communication of request outcomes.

The API have the following endpoints:
http://localhost:5000/api/orders
http://localhost:5000/api/orders/:id
http://localhost:5000/api/orders/draft
http://localhost:5000/api/orders/:id/archive
http://localhost:5000/api/orders/:id/unarchive
http://localhost:5000/api/orders/:id/send-email

2.2.1.1 Get All Orders

Version: 1.6

Date: November 26, 2025

Description: Get all orders with filtering and search capabilities. Staff users can only see orders from their assigned station. This endpoint enables retrieval of order lists with various filtering options to help manage and track laundry orders efficiently.

Endpoint: http://localhost:5000/api/orders

Method: GET

Configurations:

- The API request requires the Authorization: Bearer <token> header.
- Staff users are automatically filtered to show only orders from their assigned station.
- Results are sorted by date in descending order (newest first).

Parameters:

{{content}}gt;$ _token - required; must be included in the request header for user authentication.

{{content}}gt;$ _search - optional; search term to filter orders by order ID or customer name.

{{content}}gt;$ _payment - optional; filter by payment status: "Paid", "Unpaid", "Partial", or "All" (default: "All").

{{content}}gt;$ _showArchived - optional; boolean flag to include archived orders (default: false).

{{content}}gt;$ _showDrafts - optional; boolean flag to include draft orders (default: false).

Requests:

Valid Request

```
GET http://localhost:5000/api/orders?search=ORD-
001&payment=Paid&showArchived=false
Headers: Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Not Valid Request

```
GET http://localhost:5000/api/orders
(No Authorization header)
```

Response Format: JSON

Responses:

Success Response

```
{
  "success": true,
  "data": [
    {
      "_id": "order_id",
      "id": "ORD-001",
      "date": "2024-01-15T10:30:00.000Z",
      "customer": "John Doe",
      "customerPhone": "+1234567890",
      "customerId": "customer_id",
      "payment": "Paid",
      "total": "₱1,500.00",
      "paid": 1500,
      "balance": "₱0.00",
      "items": [
        {
          "service": "Wash & Fold",
          "quantity": "5kg",
          "status": "Completed",
          "amount": 500
        }
      ],
      "isDraft": false,
      "isArchived": false,
      "createdBy": {
        "_id": "user_id",
        "username": "staff1",
        "email": "staff1@example.com"
      }
    }
  ],
  "count": 1
}
```


Get Single Order

- **Endpoint**: `GET /api/orders/:id`

- **Description**: Get single order by order ID
- **Headers**: `Authorization: Bearer <token>`
- **URL Parameters**: `id` - Order ID (e.g., "ORD-001")
- **Response** (200):

```
{
  "success": true,
  "data": {
    "_id": "order_id",
    "id": "ORD-001",
    "date": "2024-01-15T10:30:00.000Z",
    "customer": "John Doe",
    "customerPhone": "+1234567890",
    "customerId": {
      "_id": "customer_id",
      "name": "John Doe",
      "email": "john@example.com",
      "phone": "+1234567890"
    },
    "payment": "Paid",
    "total": "₱1,500.00",
    "paid": 1500,
    "balance": "₱0.00",
    "items": [...],
    "discount": "10%",
    "discountId": "discount_id",
    "notes": "Handle with care",
    "pickupDate": "2024-01-16T10:00:00.000Z",
    "deliveryDate": null,
    "createdBy": {...},
    "lastEditedBy": {...}
  }
}
```

Create Order

- **Endpoint**: `POST /api/orders`
- **Description**: Create a new order

- **Headers**: `Authorization: Bearer <token>`
- **Required Permission**: `orders:create`
- **Request Body**:

```
{
  "customer": "John Doe",
  "customerPhone": "+1234567890",
  "customerId": "customer_id",  // Optional, if customer
exists
  "payment": "Unpaid",  // "Paid", "Unpaid", or "Partial"
  "items": [
    {
      "service": "Wash & Fold",
      "quantity": "5kg",
      "status": "Pending",
      "amount": 500
    }
  ],
  "total": "₱1,500.00",
  "paid": 0,  // Required if payment is "Partial"
  "balance": "₱1,500.00",
  "discount": "0%",
  "discountId": null,  // Optional
  "notes": "Handle with care",  // Optional
  "pickupDate": "2024-01-16T10:00:00.000Z",  // Optional
  "deliveryDate": null,  // Optional
  "stationId": "station_id"  // Optional, defaults to user's
station
}
```

- **Response** (201):

```
{
  "success": true,
  "data": {
    "_id": "order_id",
    "id": "ORD-001",
    "date": "2024-01-15T10:30:00.000Z",
    ...
  }
}
```

Save Draft Order

- **Endpoint**: `POST /api/orders/draft`
- **Description**: Save order as draft (can be completed later)
- **Headers**: `Authorization: Bearer <token>`
- **Required Permission**: `orders:create`
- **Request Body**: Same as Create Order
- **Response** (201): Same as Create Order, but with `isDraft: true`

Update Order

- **Endpoint**: `PUT /api/orders/:id`
- **Description**: Update existing order
- **Headers**: `Authorization: Bearer <token>`
- **Required Permission**: `orders:update`
- **URL Parameters**: `id` - Order ID
- **Request Body**: Same fields as Create Order (all optional)
- **Response** (200): Updated order object

Archive Order

- **Endpoint**: `PUT /api/orders/:id/archive`
- **Description**: Archive an order (soft delete)
- **Headers**: `Authorization: Bearer <token>`
- **Required Permission**: `orders:archive`
- **URL Parameters**: `id` - Order ID
- **Response** (200):

```
{
  "success": true,
  "message": "Order archived successfully"
}
```

Unarchive Order

- **Endpoint**: `PUT /api/orders/:id/unarchive`
- **Description**: Unarchive an order
- **Headers**: `Authorization: Bearer <token>`
- **Required Permission**: `orders:unarchive`
- **URL Parameters**: `id` - Order ID
- **Response** (200):

```
{
  "success": true,
  "message": "Order unarchived successfully"
}
```

Mark Draft as Completed

- **Endpoint**: `PUT /api/orders/:id/mark-completed`
- **Description**: Convert draft order to completed order
- **Headers**: `Authorization: Bearer <token>`
- **Required Permission**: `orders:update`
- **URL Parameters**: `id` - Order ID
- **Response** (200):

```
{
  "success": true,
  "message": "Draft marked as completed",
  "data": { ... }  // Updated order
}
```

Schedule Draft Deletion

- **Endpoint**: `PUT /api/orders/:id/schedule-deletion`
- **Description**: Schedule draft for automatic deletion after 30 days
- **Headers**: `Authorization: Bearer <token>`
- **Required Permission**: `orders:update`
- **URL Parameters**: `id` - Order ID
- **Response** (200):

```
{
  "success": true,
  "message": "Draft scheduled for deletion"
}
```

Delete Order

- **Endpoint**: `DELETE /api/orders/:id`
- **Description**: Permanently delete an order
- **Headers**: `Authorization: Bearer <token>`
- **Required Permission**: `orders:delete`
- **URL Parameters**: `id` - Order ID
- **Response** (200):

```
{
  "success": true,
  "message": "Order deleted successfully"
}
```

Send Invoice Email

- **Endpoint**: `POST /api/orders/:id/send-email`
- **Description**: Send invoice via email to customer
- **Headers**: `Authorization: Bearer <token>`
- **Required Permission**: `orders:read`
- **URL Parameters**: `id` - Order ID
- **Request Body** (optional):

```
{
  "email": "customer@example.com"  // Optional, uses
customer email if not provided
}
```

- **Response** (200):

```
{
  "success": true,
  "message": "Invoice sent successfully"
}
```

Acquire Edit Lock

- **Endpoint**: `POST /api/orders/:id/lock`
- **Description**: Acquire edit lock to prevent concurrent editing
- **Headers**: `Authorization: Bearer <token>`
- **Required Permission**: `orders:update`
- **URL Parameters**: `id` - Order ID
- **Response** (200):

```
{
  "success": true,
  "message": "Edit lock acquired",
  "data": {
    "isLocked": true,
    "lockedBy": {
      "name": "John Doe",
      "email": "john@example.com"
    },
    "lockedAt": "2024-01-15T10:30:00.000Z"
  }
}
```

Release Edit Lock

- **Endpoint**: `DELETE /api/orders/:id/lock`
- **Description**: Release edit lock on order
- **Headers**: `Authorization: Bearer <token>`
- **Required Permission**: `orders:update`
- **URL Parameters**: `id` - Order ID
- **Response** (200):

```
{
  "success": true,
  "message": "Edit lock released"
}
```

Check Edit Lock Status

- **Endpoint**: `GET /api/orders/:id/lock`
- **Description**: Check if order is locked and by whom
- **Headers**: `Authorization: Bearer <token>`
- **Required Permission**: `orders:read`
- **URL Parameters**: `id` - Order ID
- **Response** (200):

```
{
  "success": true,
  "data": {
    "isLocked": false,
    "lockedBy": null,
    "lockedAt": null
  }
}
```

Required Permissions:

- `orders:read` - View orders
- `orders:create` - Create orders
- `orders:update` - Update orders
- `orders:delete` - Delete orders
- `orders:archive` - Archive orders

- `orders:unarchive` - Unarchive orders

2.3 CUSTOMERS MODULE API

2.3.1 Module Description

The Customers module manages all customer-related operations including creation, updates, archiving, and retrieval. All endpoints require authentication and appropriate RBAC permissions. Staff users are automatically restricted to customers from their assigned station. The API employs comprehensive response codes: 200 (OK) for successful operations, 201 (Created) for resource creation, 400 (Bad Request) for invalid input, 401 (Unauthorized) for missing or invalid tokens, 403 (Forbidden) for insufficient permissions, 404 (Not Found) for unavailable resources, and 500 (Internal Server Error) for unexpected server issues, ensuring secure and clear communication of request outcomes.

The API have the following endpoints:
http://localhost:5000/api/customers
http://localhost:5000/api/customers/:id
http://localhost:5000/api/customers/:id/archive
http://localhost:5000/api/customers/:id/unarchive

Get All Customers

- **Endpoint**: `GET /api/customers`
- **Description**: Get all customers with filtering and sorting
- **Headers**: `Authorization: Bearer <token>`
- **Query Parameters**:
    - `search` (string, optional) - Search by name, email, or phone
    - `sortBy` (string, optional) - Sort option: `name-asc`, `name-desc`, `orders-asc`, `orders-desc`, `spent-asc`, `spent-desc` (default: `name-asc`)
    - `showArchived` (boolean, optional) - Show archived customers (default: `false`)
- **Response** (200):

```
{
  "success": true,
  "data": [
    {
      "_id": "customer_id",
      "name": "John Doe",
      "email": "john@example.com",
      "phone": "+1234567890",
      "totalOrders": 5,
      "totalSpent": 7500,
      "lastOrder": "2024-01-15T10:30:00.000Z",
      "isArchived": false,
      "notes": "VIP customer",
      "stationId": "station_id"
    }
  ],
  "count": 1
}
```


Get Single Customer

- **Endpoint**: `GET /api/customers/:id`
- **Description**: Get single customer by ID
- **Headers**: `Authorization: Bearer <token>`
- **URL Parameters**: `id` - Customer ID
- **Response** (200): Customer object

Create Customer

- **Endpoint**: `POST /api/customers`
- **Description**: Create a new customer
- **Headers**: `Authorization: Bearer <token>`
- **Required Permission**: `customers:create`
- **Request Body**:

```
{
  "name": "John Doe",  // Required
  "email": "john@example.com",  // Optional
  "phone": "+1234567890",  // Required
  "notes": "VIP customer",  // Optional
  "stationId": "station_id"  // Optional, defaults to user's station
}
```

- **Response** (201): Created customer object

Update Customer

- **Endpoint**: `PUT /api/customers/:id`
- **Description**: Update customer information
- **Headers**: `Authorization: Bearer <token>`
- **Required Permission**: `customers:update`
- **URL Parameters**: `id` - Customer ID
- **Request Body**: Same as Create Customer (all fields optional)
- **Response** (200): Updated customer object

Archive Customer

- **Endpoint**: `PUT /api/customers/:id/archive`
- **Description**: Archive a customer (soft delete)
- **Headers**: `Authorization: Bearer <token>`
- **Required Permission**: `customers:archive`
- **URL Parameters**: `id` - Customer ID
- **Response** (200):

```
{
  "success": true,
  "message": "Customer archived successfully"
}
```

Unarchive Customer

- **Endpoint**: `PUT /api/customers/:id/unarchive`
- **Description**: Unarchive a customer
- **Headers**: `Authorization: Bearer <token>`
- **Required Permission**: `customers:unarchive`
- **URL Parameters**: `id` - Customer ID
- **Response** (200):

```
{
  "success": true,
  "message": "Customer unarchived successfully"
}
```

Delete Customer

- **Endpoint**: `DELETE /api/customers/:id`
- **Description**: Permanently delete a customer
- **Headers**: `Authorization: Bearer <token>`
- **Required Permission**: `customers:delete`
- **URL Parameters**: `id` - Customer ID
- **Response** (200):

```
{
  "success": true,
  "message": "Customer deleted successfully"
}
```

Required Permissions:

- `customers:read` - View customers
- `customers:create` - Create customers
- `customers:update` - Update customers
- `customers:delete` - Delete customers
- `customers:archive` - Archive customers
- `customers:unarchive` - Unarchive customers

2.4 SERVICES MODULE API

2.4.1 Module Description

The Services module manages all service-related operations including creation, updates, archiving, and retrieval. All endpoints require authentication and appropriate RBAC permissions. Services define the types of laundry services offered by the system. The API employs comprehensive response codes: 200 (OK) for successful operations, 201 (Created) for resource creation, 400 (Bad Request) for invalid input, 401 (Unauthorized) for missing or invalid tokens, 403 (Forbidden) for insufficient permissions, 404 (Not Found) for unavailable resources, and 500 (Internal Server Error) for unexpected server issues, ensuring secure and clear communication of request outcomes.

The API have the following endpoints:
http://localhost:5000/api/services
http://localhost:5000/api/services/:id
http://localhost:5000/api/services/:id/archive
http://localhost:5000/api/services/:id/unarchive

2.5 EXPENSES MODULE API

2.5.1 Module Description

The Expenses module manages all expense-related operations including creation, approval workflow, updates, archiving, and retrieval. All endpoints require authentication and appropriate RBAC permissions. Staff users can create expense requests, while admins can approve or reject them. The API employs comprehensive response codes: 200 (OK) for successful operations, 201 (Created) for resource creation, 400 (Bad Request) for invalid input, 401 (Unauthorized) for missing or invalid tokens, 403 (Forbidden) for insufficient permissions, 404 (Not Found) for unavailable resources, and 500 (Internal Server Error) for unexpected server issues, ensuring secure and clear communication of request outcomes.

The API have the following endpoints:
http://localhost:5000/api/expenses
http://localhost:5000/api/expenses/:id
http://localhost:5000/api/expenses/:id/approve
http://localhost:5000/api/expenses/:id/reject
http://localhost:5000/api/expenses/:id/archive
http://localhost:5000/api/expenses/:id/unarchive

2.6 EMPLOYEES MODULE API

2.6.1 Module Description

The Employees module manages all employee-related operations including creation, updates, performance tracking, archiving, and retrieval. All endpoints require authentication and appropriate RBAC permissions. Admins can manage all employees, while staff can view their own information. The API employs comprehensive response codes: 200 (OK) for successful operations, 201 (Created) for resource creation, 400 (Bad Request) for invalid input, 401 (Unauthorized) for missing or invalid tokens, 403 (Forbidden) for insufficient permissions, 404 (Not Found) for unavailable resources, and 500 (Internal Server Error) for unexpected server issues, ensuring secure and clear communication of request outcomes.

The API have the following endpoints:
http://localhost:5000/api/employees
http://localhost:5000/api/employees/:id
http://localhost:5000/api/employees/:id/performance
http://localhost:5000/api/employees/:id/toggle-account
http://localhost:5000/api/employees/:id/archive
http://localhost:5000/api/employees/:id/unarchive

2.7 DISCOUNTS MODULE API

2.7.1 Module Description

The Discounts module manages all discount-related operations including creation, updates, usage tracking, archiving, and retrieval. All endpoints require authentication and appropriate RBAC permissions. Discounts can be applied to orders to provide promotional pricing. The API employs comprehensive response codes: 200 (OK) for successful operations, 201 (Created) for resource creation, 400 (Bad Request) for invalid input, 401 (Unauthorized) for missing or invalid tokens, 403 (Forbidden) for insufficient permissions, 404 (Not Found) for unavailable resources, and 500 (Internal Server Error) for unexpected server issues, ensuring secure and clear communication of request outcomes.

The API have the following endpoints:
http://localhost:5000/api/discounts
http://localhost:5000/api/discounts/:id
http://localhost:5000/api/discounts/:id/reset-usage
http://localhost:5000/api/discounts/:id/archive
http://localhost:5000/api/discounts/:id/unarchive

2.8 DASHBOARD MODULE API

2.8.1 Module Description

The Dashboard module provides aggregated statistics and key performance indicators for the LaundryPOS system. All endpoints require authentication and appropriate RBAC permissions. The dashboard displays revenue trends, order statistics, customer metrics, and recent activity. The API employs comprehensive response codes: 200 (OK) for successful operations, 400 (Bad Request) for invalid input, 401 (Unauthorized) for missing or invalid tokens, 403 (Forbidden) for insufficient permissions, and 500 (Internal Server Error) for unexpected server issues, ensuring secure and clear communication of request outcomes.

The API have the following endpoints:
http://localhost:5000/api/dashboard/stats

2.8.1.1 Get Dashboard Statistics

Version: 1.6

Date: November 26, 2025

Description: Get comprehensive dashboard statistics and metrics including revenue, orders, customers, and trends. This endpoint provides aggregated data for the admin dashboard view.

Endpoint: http://localhost:5000/api/dashboard/stats

Method: GET

Configurations:

- The API request requires the Authorization: Bearer <token> header.
- Staff users see statistics only for their assigned station.
- Results are calculated based on the specified time range.

Parameters:

{{content}}gt;$ _token - required; must be included in the request header for user authentication.

{{content}}gt;$ _timeRange - optional; time range for statistics: "today", "week", "month", or "year" (default: "today").

Requests:

Valid Request

```
GET http://localhost:5000/api/dashboard/stats?
timeRange=month
Headers: Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Not Valid Request

```
GET http://localhost:5000/api/dashboard/stats?
timeRange=invalid
(No Authorization header)
```

Response Format: JSON

Responses:

Success Response

```
{
  "success": true,
  "data": {
    "stats": {
      "orders": 25,
      "revenue": 37500,
      "pending": 5,
      "customers": 150
    },
    "trends": {
      "orders": 15,  // Percentage change from previous
period
      "revenue": 20,
      "pending": -10,
```

```
      "customers": 5
    },
    "orderStatus": [
      { "name": "Pending", "value": 5 },
      { "name": "In Progress", "value": 10 },
      { "name": "Completed", "value": 10 }
    ],
    "revenueTrend": [
      {
        "name": "Mon",
        "value": 5000,
        "target": 3000
      },
      {
        "name": "Tue",
        "value": 6000,
        "target": 3000
      }
      // ... last 7 days
    ],
    "recentActivity": [
      {
        "id": "order_id",
        "type": "order",
        "message": "New order ORD-001 from John Doe",
        "time": "2 hours ago",
        "orderId": "ORD-001"
      }
    ],
    "pendingExpenses": 3  // Admin only
  }
}
```

Required Permissions:

- `dashboard:read` - View dashboard

2.9 REPORTS MODULE API

2.9.1 Module Description

The Reports module generates comprehensive reports for orders, revenue, customers, expenses, services, employees, and branch performance. All endpoints require authentication and appropriate RBAC permissions. Staff users can only generate reports for their own data, while admins can generate system-wide reports. The API employs comprehensive response codes: 200 (OK) for successful operations, 400 (Bad Request) for invalid input, 401 (Unauthorized) for missing or invalid tokens, 403 (Forbidden) for insufficient permissions, and 500 (Internal Server Error) for unexpected server issues, ensuring secure and clear communication of request outcomes.

The API have the following endpoints:
http://localhost:5000/api/reports/orders
http://localhost:5000/api/reports/revenue
http://localhost:5000/api/reports/customers
http://localhost:5000/api/reports/expenses
http://localhost:5000/api/reports/services
http://localhost:5000/api/reports/employee
http://localhost:5000/api/reports/sales-per-branch
http://localhost:5000/api/reports/cashflow-per-branch

Generate Orders Report

- **Endpoint**: `POST /api/reports/orders`
- **Description**: Generate orders report for a date range
- **Headers**: `Authorization: Bearer <token>`
- **Request Body**:

```
{
  "dateFrom": "2024-01-01",
  "dateTo": "2024-01-31"
}
```

- **Response** (200):

```
{
  "success": true,
  "data": {
    "reportType": "orders",
    "dateRange": {
      "from": "2024-01-01",
      "to": "2024-01-31"
    },
    "summary": {
      "totalOrders": 100,
      "totalRevenue": "150000.00",
      "paidOrders": 80,
      "unpaidOrders": 15,
      "partialOrders": 5
    },
    "orders": [
      {
        "id": "ORD-001",
        "date": "2024-01-15",
        "customer": "John Doe",
        "customerPhone": "+1234567890",
        "payment": "Paid",
        "total": "₱1,500.00",
        "paid": 1500,
        "balance": "₱0.00",
        "items": [...],
        "pickupDate": "2024-01-16",
        "deliveryDate": null,
        "notes": "",
        "createdBy": "staff1"
      }
    ]
  }
}
```


Generate Revenue Report

- **Endpoint**: `POST /api/reports/revenue`
- **Description**: Generate revenue report with profit/loss analysis
- **Headers**: `Authorization: Bearer <token>`
- **Request Body**:

```
{
  "dateFrom": "2024-01-01",
  "dateTo": "2024-01-31"
}
```

- **Response** (200):

```
{
  "success": true,
  "data": {
    "reportType": "revenue",
    "dateRange": {...},
    "summary": {
      "totalRevenue": 150000,
      "totalExpenses": 30000,
      "profit": 120000,
      "profitMargin": 80
    },
    "dailyBreakdown": {
      "2024-01-01": {
        "revenue": 5000,
        "expenses": 1000,
        "profit": 4000
      }
    }
  }
}
```

Generate Customers Report

- **Endpoint**: `POST /api/reports/customers`
- **Description**: Generate customers report with statistics

- **Headers**: `Authorization: Bearer <token>`
- **Request Body**: Same date range format
- **Response** (200): Customer statistics and list

Generate Expenses Report

- **Endpoint**: `POST /api/reports/expenses`
- **Description**: Generate expenses report
- **Headers**: `Authorization: Bearer <token>`
- **Request Body**: Same date range format
- **Response** (200): Expense statistics and list

Generate Services Report

- **Endpoint**: `POST /api/reports/services`
- **Description**: Generate services usage report
- **Headers**: `Authorization: Bearer <token>`
- **Request Body**: Same date range format
- **Response** (200): Service statistics

Generate Employee Report

- **Endpoint**: `POST /api/reports/employee`
- **Description**: Generate employee performance report
- **Headers**: `Authorization: Bearer <token>`
- **Request Body**: Same date range format
- **Response** (200): Employee performance metrics

Generate Sales Per Branch Report

- **Endpoint**: `POST /api/reports/sales-per-branch`
- **Description**: Generate sales breakdown by branch/station
- **Headers**: `Authorization: Bearer <token>`
- **Request Body**: Same date range format
- **Response** (200): Sales data grouped by station

Generate Cashflow Per Branch Report

- **Endpoint**: `POST /api/reports/cashflow-per-branch`

- **Description**: Generate cashflow analysis per branch
- **Headers**: `Authorization: Bearer <token>`
- **Request Body**: Same date range format
- **Response** (200): Cashflow data per station

Required Permissions:

- `reports:read` - Generate and view reports

2.10 STATIONS MODULE API

2.10.1 Module Description

The Stations module manages all branch/station-related operations including creation, updates, archiving, and retrieval. All endpoints require authentication and appropriate RBAC permissions except for the public endpoint. Stations represent physical branch locations in the franchise network. The API employs comprehensive response codes: 200 (OK) for successful operations, 201 (Created) for resource creation, 400 (Bad Request) for invalid input, 401 (Unauthorized) for missing or invalid tokens, 403 (Forbidden) for insufficient permissions, 404 (Not Found) for unavailable resources, and 500 (Internal Server Error) for unexpected server issues, ensuring secure and clear communication of request outcomes.

The API have the following endpoints:
http://localhost:5000/api/stations/public
http://localhost:5000/api/stations
http://localhost:5000/api/stations/:id
http://localhost:5000/api/stations/:id/archive
http://localhost:5000/api/stations/:id/unarchive

2.11 BACKUPS MODULE API

2.11.1 Module Description

The Backups module manages database and file backup operations including creation, restoration, listing, and cleanup. All endpoints require authentication and appropriate RBAC permissions. Backups ensure data safety and enable disaster recovery. The API employs comprehensive response codes: 200 (OK) for successful operations, 201 (Created) for resource creation, 400 (Bad

Request) for invalid input, 401 (Unauthorized) for missing or invalid tokens, 403 (Forbidden) for insufficient permissions, 404 (Not Found) for unavailable resources, and 500 (Internal Server Error) for unexpected server issues, ensuring secure and clear communication of request outcomes.

The API have the following endpoints:
http://localhost:5000/api/backups
http://localhost:5000/api/backups/stats
http://localhost:5000/api/backups/:backupName/restore
http://localhost:5000/api/backups/cleanup

2.12 RBAC (ROLE-BASED ACCESS CONTROL) MODULE API

2.12.1 Module Description

The RBAC module manages role-based access control permissions for the system. Most endpoints require authentication and admin role, except for the emergency recovery endpoint. This module allows administrators to configure and manage permissions for different user roles. The API employs comprehensive response codes: 200 (OK) for successful operations, 201 (Created) for resource creation, 400 (Bad Request) for invalid input, 401 (Unauthorized) for missing or invalid tokens, 403 (Forbidden) for insufficient permissions, 404 (Not Found) for unavailable resources, and 500 (Internal Server Error) for unexpected server issues, ensuring secure and clear communication of request outcomes.

The API have the following endpoints:
http://localhost:5000/api/rbac/emergency-recover-admin
http://localhost:5000/api/rbac/resources
http://localhost:5000/api/rbac/me
http://localhost:5000/api/rbac/:role
http://localhost:5000/api/rbac/initialize

2.13 SYSTEM SETTINGS MODULE API

2.13.1 Module Description

The System Settings module manages system-wide configuration settings including inactivity timeouts and other system parameters. All endpoints require authentication and typically admin role. The API employs comprehensive response codes: 200 (OK) for successful operations, 400

(Bad Request) for invalid input, 401 (Unauthorized) for missing or invalid tokens, 403 (Forbidden) for insufficient permissions, and 500 (Internal Server Error) for unexpected server issues, ensuring secure and clear communication of request outcomes.

The API have the following endpoints:
http://localhost:5000/api/system-settings/inactivity

2.14 UPLOAD MODULE API

2.14.1 Module Description

The Upload module handles file upload operations for images used in receipts, profiles, and other system features. All endpoints require authentication. The API employs comprehensive response codes: 200 (OK) for successful operations, 400 (Bad Request) for invalid input, 401 (Unauthorized) for missing or invalid tokens, 413 (Payload Too Large) for files exceeding size limits, and 500 (Internal Server Error) for unexpected server issues, ensuring secure and clear communication of request outcomes.

The API have the following endpoints:
http://localhost:5000/api/upload/image
http://localhost:5000/api/upload/images

2.15 SUPPORT MODULE API

2.15.1 Module Description

The Support module handles user feedback and support ticket submissions. All endpoints require authentication. The API employs comprehensive response codes: 200 (OK) for successful operations, 400 (Bad Request) for invalid input, 401 (Unauthorized) for missing or invalid tokens, and 500 (Internal Server Error) for unexpected server issues, ensuring secure and clear communication of request outcomes.

The API have the following endpoints:
http://localhost:5000/api/support/feedback

2.16 NOTIFICATIONS MODULE API

2.16.1 Module Description

The Notifications module manages real-time notifications for users including Server-Sent Events (SSE) streaming, notification retrieval, and read status management. All endpoints require authentication. The API employs comprehensive response codes: 200 (OK) for successful operations, 400 (Bad Request) for invalid input, 401 (Unauthorized) for missing or invalid tokens, and 500 (Internal Server Error) for unexpected server issues, ensuring secure and clear communication of request outcomes.

The API have the following endpoints:
http://localhost:5000/api/notifications/stream
http://localhost:5000/api/notifications
http://localhost:5000/api/notifications/:notificationId/read
http://localhost:5000/api/notifications/read-all

2.17 AUDIT LOGS MODULE API

2.17.1 Module Description

The Audit Logs module provides access to system audit logs for tracking user actions and system events. All endpoints require authentication and appropriate RBAC permissions. Audit logs capture user activities, IP addresses, timestamps, and action details for security and compliance. The API employs comprehensive response codes: 200 (OK) for successful operations, 400 (Bad Request) for invalid input, 401 (Unauthorized) for missing or invalid tokens, 403 (Forbidden) for insufficient permissions, 404 (Not Found) for unavailable resources, and 500 (Internal Server Error) for unexpected server issues, ensuring secure and clear communication of request outcomes.

The API have the following endpoints:
http://localhost:5000/api/audit-logs
http://localhost:5000/api/audit-logs/stats
http://localhost:5000/api/audit-logs/:id

System Utility Endpoints

Health Check

- **Endpoint**: `GET /api/health`
- **Description**: Health check endpoint for monitoring system status
- **Authentication**: Not required

- **Response** (200):

```
{
  "status": "healthy",
  "timestamp": "2024-01-15T10:30:00.000Z",
  "uptime": 3600,
  "memory": {
    "used": "150 MB",
    "total": "200 MB",
    "rss": "180 MB"
  },
  "database": "connected",
  "version": "1.0.0",
  "environment": "production"
}
```

SMS Test (Development/Debug)

- **Endpoint**: `GET /api/test-sms`
- **Description**: Test SMS functionality (for debugging)
- **Authentication**: Not required
- **Query Parameters**:
  - `phone` (string, required) - Phone number (e.g., `+1234567890`)
  - `message` (string, optional) - Test message (default: "Test SMS from Laundry POS")
- **Response** (200):

```
{
  "success": true,
  "message": "SMS sent successfully",
  "data": {
    "messageId": "sms_id",
    "status": "sent"
  }
}
```

V. Sample Requests and Responses

1. **Login**: Use the payload shown in Section 2.1.1.1.
2. **Send Reset OTP**: Follow the `/auth/forgot-password` request in Section 2.1.1.3.
3. **Reset Password**: Submit the payload in Section 2.1.1.4 to `/auth/reset-password`.
4. **Draft Order**: Use the Staff autosave payload from Section 2.2.1.2.
5. **Email Invoice**: Call `/orders/:id/send-email` as detailed in Section 2.2.1.3.

VI. Security Notes

- All passwords are hashed with bcrypt (12 salt rounds).
- Environment variables required: `JWT_SECRET`, `MONGODB_URI`, SMTP credentials, reCAPTCHA keys, and `ALLOWED_ORIGINS`.
- HTTPS should be enforced in production using certificates stored in `server/certs/`.
- Audit logs capture user, IP, user-agent, and payload metadata for every sensitive endpoint.
- Rate limiting is enforced via `middleware/rateLimiter.js` to protect authentication workflows.

VII. Changelog

Date	Changes
26 Nov 2025	Reformatted documentation to mirror the BuKSU template and embedded comprehensive endpoint details.
15 Nov 2025	Added OTP verification, password reset, and logout coverage.

VIII. References

- LaundryPOS backend source (`server/routes`, `server/controllers`)
- Internal API Reference (`docs/API_REFERENCE.md`)
- Thunder Client captures stored under `docs/images/`
- SBO Fee Collection Management System API Documentation (Sample PDF)
- API Documentation Rubric.docx

Last Updated: November 26, 2025

Documentation Version: 1.6

API Version: 1.6