

# RPOS - React Native Point of Sale System

## Project Overview Document

**Generated:** January 7, 2026 **Project:** RPOS (React Native POS) **Version:** 0.0.1

### Executive Summary

This is a **full-stack POS application** built with:

- Frontend:** React Native (iPad-optimized, landscape mode)
- Backend:** Node.js/Express with MongoDB
- State Management:** Redux with Redux-Persist for offline capability

## 1. Architecture Overview

### Project Structure

```
rpos-envato/
├── app/                                # React Native Mobile App
│   ├── src/app/
│   │   ├── actions/                  # Redux action creators
│   │   ├── reducers/                 # Redux reducers
│   │   ├── containers/              # Screen components
│   │   ├── services/                # API service layer
│   │   ├── common/                  # Utilities, themes, config
│   │   └── main/                    # App entry, navigation
│   └── server/                      # Node.js Backend
│       ├── src/
│       │   ├── controllers/          # Route handlers
│       │   ├── models/               # Mongoose schemas
│       │   ├── routes/               # Express routes
│       │   ├── services/             # Business logic services
│       │   └── middlewares/          # Auth, validation
│       └── document/                 # Documentation
```

### Technology Stack

Layer	Technology
Mobile App	React Native 0.66.4
State Management	Redux + Redux-Thunk + Redux-Persist
Navigation	React Navigation 4.x
Backend	Node.js + Express 4.16
Database	MongoDB + Mongoose 5.5
Authentication	Passport.js + JWT

## 2. Key Business Logic Implementations

### 2.1 Point of Sale (POS) Screen

File: `app/src/app/containers/POS/index.js`

#### Core Features:

- **Product Grid Display:** FlatList with 4-column layout, pagination (lazy loading)
- **Category Filtering:** Dropdown to filter products by category
- **Text Search:** Real-time search across product name, SKU, description
- **Cart Management:**
  - Add products to cart (auto-increment quantity if exists)
  - Swipe-to-delete items
  - Quantity adjustment per item
- **Coupon Application:** Supports fixed amount and percentage discounts
- **Customer Selection:** Link customer to order
- **Tax/VAT Calculation:** Configurable VAT percentage from settings

#### Payment Calculation Logic:

```
subTotal = sum(item.sellingPrice * item.quantity)
discount = coupon.type === 'percentage'
  ? (subTotal * coupon.amount / 100)
  : coupon.amount
vat = VAT > 0 ? (VAT * subTotal / 100) : 0
total = subTotal - discount + vat
```

**Offline Mode Support:** Creates local orders with `local-` prefixed IDs when offline.

### 2.2 Offline-First Data Synchronization

File: `app/src/app/main/App.js`

#### Critical Implementation:

- Uses `NetInfo` to monitor connectivity
- When online, syncs:
  1. **Local additions** (items with `local-` ID prefix)
  2. **Local edits** (items with `status: 'editing'`)
  3. **Local deletions** (stored in `deletedXxx` arrays)
- Syncs: Categories, Products, Orders, Customers, Coupons, Staff

#### Sync Logic Flow:

```
Network Online → syncUpdatingData() → syncDeletingData() → refreshData()
```

#### Key Sync Functions:

- `syncAddCategory()` - Sync locally created categories

- `syncAddProducts()` - Sync locally created products
  - `syncAddOrders()` - Sync locally created orders
  - `syncEditingCategories()` - Sync edited categories
  - `syncEditingProducts()` - Sync edited products
  - `syncDeletingCategories()` - Sync deleted categories
  - `syncDeletingProducts()` - Sync deleted products
- 

## 2.3 Order Processing

**Backend:** `server/src/controllers/orders.js` **Service:** `server/src/services/order.js`

### Order Creation Flow:

1. Validate order items (product IDs, quantities)
2. Calculate payment (subTotal, discount, total)
3. Update product `sellingInfo` (quantity sold, amount, profit)
4. Create order document with references to products, customer, coupon
5. Generate order number
6. Create activity log entry

### Order Data Structure:

```
{
  number: Number,           // Order number
  items: [{
    product: ObjectId,      // Reference to Product
    quantity: Number
  }],
  coupon: ObjectId,         // Reference to Coupon (optional)
  customer: ObjectId,       // Reference to Customer (optional)
  payment: {
    subTotal: Number,
    discount: Number,
    total: Number
  },
  createdBy: ObjectId,      // Reference to User
  createdAt: Date
}
```

---

## 2.4 Product Management

**Files:** `app/src/app/containers/Products/` , `server/src/controllers/products.js`

### Features:

- CRUD operations for products
- Category linkage with automatic count tracking
- Image upload and compression (resized to 500x500 JPEG)
- Product activity logs (tracks creation, edits, orders)
- Profit tracking

### Product Fields:

Field	Type	Description
name	String	Product name
sku	String	Stock Keeping Unit
category	ObjectId	Reference to Category
quantity	Number	Available stock
sellingPrice	Number	Retail price
purchasePrice	Number	Cost price
images	[String]	Array of image URLs
desc	String	Product description
sellingInfo	Object	Sales tracking data

#### Selling Info Structure:

```
sellingInfo: {
  quantity: Number, // Total units sold
  amount: Number,   // Total revenue
  profit: Number     // Total profit (sellingPrice - purchasePrice) * quantity
}
```

## 2.5 Customer Management

#### Features:

- Full CRUD for customers
- Customer order history tracking
- Customer selection during checkout

#### Customer Fields:

Field	Type	Description
name	String	Customer name
email	String	Email address
phone	String	Phone number
address	String	Physical address
avatar	String	Profile image URL

## 2.6 Coupon System

**Model:** `server/src/models/coupon.js`

#### Coupon Types:

Type	Behavior
fixed	Flat amount discount (e.g., \$10 off)
percentage	Percentage of subtotal (e.g., 15% off)

#### Coupon Fields:

Field	Type	Description
code	String	Unique coupon code
name	String	Display name
type	String	'fixed' or 'percentage'
amount	Number	Discount value
expiredAt	Date	Expiration date

**Validation Logic:** Filters expired coupons before display using:

```
allCoupons.filter(item =>
  !item.expiredAt || moment(item.expiredAt).isAfter(moment().toISOString(),
    'minutes')
)
```

## 2.7 Staff Management

#### Role-based Access Control:

Role	Permissions
admin	Full system access
manager	Business management, staff management
staff	POS operations only

#### Features:

- Add/edit/delete staff members
- Role assignment
- Staff profile management

## 3. State Management (Redux)

### 3.1 Reducer Structure

```
{
  userReducers: {
    login: { status, result, error, requesting },
```

```

    updateProfile: { status, result, error, requesting },
    staffs: { status, result: [], error, requesting },
    deletedStaffs: []
  },
  settingReducers: {
    isDarkMode: Boolean,
    isOfflineMode: Boolean,
    config: {
      language: Object,
      currency: Object,
      tax: String
    }
  },
  categoryReducers: {
    categories: { status, result: [], error, requesting },
    deletedCategories: []
  },
  productReducers: {
    products: { status, result: [], error, requesting },
    deletedProducts: []
  },
  orderReducers: {
    orders: { status, result: [], error, requesting }
  },
  couponReducers: {
    coupons: { status, result: [], error, requesting },
    deletedCoupons: []
  },
  customerReducers: {
    customers: { status, result: [], error, requesting },
    deletedCustomers: []
  }
}

```

### 3.2 Redux Persist Configuration

File: `app/src/app/reducers/index.js`

```

const config = {
  key: 'root',
  storage: AsyncStorage,
  transforms: [
    createWhitelistFilter('userReducers', ['login.result', 'staffs.result',
'deletedStaffs']),
    createWhitelistFilter('settingReducers'),
    createWhitelistFilter('categoryReducers', ['categories.result',
'deletedCategories']),
    createWhitelistFilter('productReducers', ['products.result',
'deletedProducts']),
    createWhitelistFilter('orderReducers', ['orders.result']),
    createWhitelistFilter('couponReducers', ['coupons.result', 'deletedCoupons']),

```

```

    createWhitelistFilter('customerReducers', ['customers.result',
'deletedCustomers']),
  ],
}

```

### 3.3 Action Pattern (Request/Success/Fail)

All async operations follow the pattern:

ACTION\_REQUEST → API call → ACTION\_SUCCESS or ACTION\_FAIL

**Example - Add Product:**

```

ADD_PRODUCT_REQUEST // Set requesting: true
↓
API Call to /products
↓
ADD_PRODUCT_SUCCESS // Add to products array, requesting: false
or
ADD_PRODUCT_FAIL    // Set error message, requesting: false

```

## 4. Backend API Architecture

### 4.1 Authentication

File: server/src/controllers/auth.js

- JWT-based authentication with Passport.js
- Bearer token scheme
- Password hashing with bcrypt (10 salt rounds)
- Forgot password with email verification code (6-digit)

**JWT Token Generation:**

```

const token = jwt.sign(
  { id: user._id, email: user.email },
  Constants.JWTSecret
)

```

### 4.2 API Endpoints

**Authentication**

Endpoint	Method	Description
/users/login	POST	User authentication
/users/register	POST	New business registration
/auth/forgotPassword	POST	Initiate password reset
/auth/resetPassword	POST	Complete password reset

**Products**

Endpoint	Method	Description
/products/sync	GET	Sync all products
/products	GET	Get products (paginated)
/products	POST	Create product
/products/:id	PUT	Update product
/products/:id	DELETE	Delete product
/products/:id/logs	GET	Get product activity logs

**Orders**

Endpoint	Method	Description
/orders/sync	GET	Sync all orders
/orders	GET	Get orders (paginated)
/orders	POST	Create order
/orders/:id	PUT	Update order
/orders/:id	DELETE	Delete order

**Categories**

Endpoint	Method	Description
/categories/sync	GET	Sync all categories
/categories	POST	Create category
/categories/:id	PUT	Update category
/categories/:id	DELETE	Delete category

**Customers**

Endpoint	Method	Description
/customers/sync	GET	Sync all customers
/customers	POST	Create customer
/customers/:id	PUT	Update customer
/customers/:id	DELETE	Delete customer

**Coupons**

Endpoint	Method	Description
----------	--------	-------------

/coupons/sync	GET	Sync all coupons
/coupons	POST	Create coupon
/coupons/:id	PUT	Update coupon
/coupons/:id	DELETE	Delete coupon

**Reports**

Endpoint	Method	Description
/reports/daily	GET	Daily sales report
/reports/weekly	GET	Weekly sales report
/reports/monthly	GET	Monthly sales report
/reports/top_products	GET	Best selling products

**4.3 Database Models (MongoDB/Mongoose)**

**User Model**

```
{
  firstName: String,
  lastName: String,
  avatar: String,
  email: String,
  role: String,           // 'admin', 'manager', 'staff'
  authType: String,      // 'email'
  business: ObjectId,    // Reference to Business
  resetCode: String,
  hash: String,          // Bcrypt password hash
  enable: Boolean,
  createdAt: Date
}
```

**Product Model**

```
{
  name: String,
  sku: String,
  category: ObjectId,
  desc: String,
  quantity: Number,
  sellingPrice: Number,
  purchasePrice: Number,
  images: [String],
  business: ObjectId,
  sellingInfo: {
    quantity: Number,
```

```
    amount: Number,  
    profit: Number  
  },  
  enable: Boolean,  
  createdAt: Date  
}
```

### Order Model

```
{  
  number: Number,  
  items: [{  
    product: ObjectId,  
    quantity: Number  
  }],  
  coupon: ObjectId,  
  customer: ObjectId,  
  guest: {  
    name: String,  
    email: String,  
    phone: String,  
    address: String  
  },  
  business: ObjectId,  
  payment: {  
    subTotal: Number,  
    discount: Number,  
    total: Number  
  },  
  createdBy: ObjectId,  
  createdAt: Date  
}
```

### Category Model

```
{  
  name: String,  
  image: String,  
  count: Number,           // Number of products  
  parent: ObjectId,        // Reference to parent category  
  business: ObjectId,  
  enable: Boolean,  
  createdAt: Date  
}
```

### Customer Model

```
{  
  name: String,
```

```
avatar: String,  
email: String,  
phone: String,  
address: String,  
business: ObjectId,  
enable: Boolean,  
createdAt: Date  
}
```

### Coupon Model

```
{  
  code: String,  
  name: String,  
  type: String,           // 'fixed' or 'percentage'  
  amount: Number,  
  business: ObjectId,  
  expiredAt: Date,  
  enable: Boolean,  
  createdAt: Date  
}
```

### Business Model (Multi-tenancy)

```
{  
  name: String,  
  createdAt: Date  
}
```

### Log Model (Activity Tracking)

```
{  
  type: String,           // 'new_product', 'edit_product', 'new_order'  
  payload: {  
    product: ObjectId,  
    order: ObjectId  
  },  
  user: {  
    id: ObjectId,  
    name: String  
  },  
  business: ObjectId,  
  createdAt: Date  
}
```

---

## 5. Reporting & Analytics

## Dashboard Features

File: `app/src/app/containers/Dashboard/index.js`

- **Monthly Sales Chart:** Line chart showing revenue by month
- **Top Products Chart:** Bar chart showing best sellers by quantity
- **Quick Links:** Navigation shortcuts to all major screens

## Report Types

### Daily Report

- All orders for a specific day
- Full order details with products, customer, coupon
- Date filter support

### Weekly Report

- Sales aggregated by day of week
- Returns object: `{ mon, tue, wed, thu, fri, sat, sun }`

### Monthly Report

- Sales aggregated by month
- Returns object with 12 months: `{ 1: amount, 2: amount, ... 12: amount }`

### Top Products Report

- Products sorted by `sellingInfo.quantity` (units sold)
- Configurable limit (default 20)
- Returns: `{ _id, name, quantity, amount, profit }`

---

## 6. Hardware Integration

### Bluetooth Thermal Printer

File: `app/src/app/common/printer/BlePrinter.js`

#### Features:

- ESC/POS compatible printing
- Auto-reconnect on connection loss
- Device scanning and pairing
- Cross-platform support (iOS & Android)

**Receipt Layout:** `` [Business Name]

**Time** [Current DateTime] **Order** [Order Number]

**Items Qty Amount [Product 1] [Qty] [Price] [Product 2] [Qty] [Price]**

Subtotal [Amount] Discount -[Amount] Total [Amount]

```
**Printer Events Handled**:
- `EVENT_CONNECTED` - Device connected
- `EVENT_CONNECTION_LOST` - Auto-reconnect trigger
```

```

- `EVENT_DEVICE_FOUND` - New device discovered
- `EVENT_DEVICE_ALREADY_PAIRED` - Paired device list

---

## 7. Internationalization (i18n)

### Language Support
- English (en)
- Vietnamese (vi)

### Implementation
**Files**:
- `app/src/app/common/I18n.js`
- `app/src/app/common/locales/en.js`
- `app/src/app/common/locales/vi.js`

**Device Locale Detection**:
```javascript
const locales = RNLocalize.getLocales();
const deviceLang = languages.filter(item => item.value === locales[0].languageTag);

```

## Currency Support

File: `app/src/app/common/data/Currencies.js`

Currency	Symbol	Code	Position
US Dollar	\$	USD	Prefix
Euro	€	EUR	Prefix
Japanese Yen	¥	JPY	Prefix
Vietnamese Dong	₫	VND	Suffix

### Currency Formatting:

```

Utils.formatCurrency(value, currency, config) => I18n.toCurrency(value, {
  precision: 1,
  format: currency.isSuffix ? '%n%u' : '%u%n',
  unit: currency.symbol,
  ...config
})

```

## 8. UI/UX Features

### Dark Mode

- Full dark mode support via `react-native-dark-mode`
- Dynamic stylesheets with `useDynamicStyleSheet`

- Theme colors defined in `Colors.js`

## Responsive Design

- Optimized for iPad landscape orientation
- Uses `ResponsiveUtils.normalize()` for scaling
- Orientation locked to landscape

## Navigation Structure

```
Root
├── LoginScreen
├── RegisterScreen
├── ForgotPasswordScreen
└── MainStack (Authenticated)
    ├── DashboardScreen
    ├── POSScreen
    ├── OrderScreen
    ├── Categories Stack
    │   ├── CategoriesScreen
    │   ├── AddCategoryScreen
    │   └── EditCategoryScreen
    ├── Products Stack
    │   ├── ProductsScreen
    │   ├── AddProductScreen
    │   ├── ProductDetailScreen
    │   └── EditProductScreen
    ├── Customers Stack
    │   ├── CustomersScreen
    │   ├── AddCustomerScreen
    │   ├── CustomerDetailScreen
    │   └── EditCustomerScreen
    ├── Coupon Stack
    │   ├── CouponsScreen
    │   ├── AddCouponScreen
    │   └── EditCouponScreen
    ├── Setting Stack
    │   ├── SettingScreen
    │   ├── AddStaffScreen
    │   ├── EditStaffScreen
    │   └── PrintersScreen
    ├── EditProfileScreen
    ├── ChangePasswordScreen
    └── ReportsScreen
```

## UI Components

- **DropdownAlert:** Success/error notifications
  - **SwipeListView:** Swipe-to-delete on list items
  - **Modal:** Coupon selection, customer selection
  - **Charts:** LineChart, BarChart from react-native-chart-kit
-

## 9. Configuration & Environment

### App Configuration

File: `app/src/app/common/Config.js`

```
{
  DomainApi: env.DOMAIN_API,
  LogoHeader: require('@assets/logo/logo_white.png'),
  Logo: require('@assets/logo/logo.png'),
  EnableRegister: true,
  HelpLink: 'https://mposs.io/support'
}
```

### Constants

File: `app/src/app/common/Constants.js`

- `NUMBER_ITEMS_PER_PAGE` : Pagination size
- `NUMBER_OF_ORDER_LENGTH` : Order number digit count
- `NUMBER_CHART_COLUMNS` : Chart data points
- `MAX_LENGTH_CHART_LABEL` : Chart label truncation

### Server Environment

File: `server/.env`

Required environment variables:

- `DOMAIN_API` : API base URL
- `MONGODB_URI` : MongoDB connection string
- `JWT_SECRET` : JWT signing secret
- `SMTP_*` : Email configuration

---

## 10. Dependencies

### Mobile App (Key Dependencies)

```
{
  "react": "17.0.2",
  "react-native": "0.66.4",
  "redux": "^4.0.5",
  "react-redux": "^7.1.3",
  "redux-persist": "^6.0.0",
  "redux-thunk": "^2.3.0",
  "react-navigation": "^4.0.10",
  "axios": "^0.19.0",
  "react-native-bluetooth-escpos-printer": "custom",
  "react-native-chart-kit": "^5.2.0",
  "react-native-dark-mode": "^0.2.1",
  "moment": "^2.24.0",
}
```

```
"lodash": "^4.17.15",  
"i18n-js": "^3.5.1"  
}
```

## Server (Key Dependencies)

```
{  
  "express": "~4.16.0",  
  "mongoose": "^5.5.12",  
  "passport": "^0.4.0",  
  "passport-jwt": "^4.0.0",  
  "jsonwebtoken": "^8.5.1",  
  "bcrypt": "3.0.7",  
  "stripe": "^8.24.0",  
  "nodemailer": "^6.6.3",  
  "socket.io": "^2.3.0",  
  "firebase-admin": "^9.5.0"  
}
```

## 11. Summary

This POS system is a **production-ready, offline-capable** solution for retail businesses.

### Key Strengths

1. **Offline-First Architecture:** Full functionality without internet connection
2. **Multi-tenant Design:** Business isolation via MongoDB references
3. **Complete POS Flow:** Product selection → Cart → Coupons → Checkout → Receipt
4. **Hardware Integration:** Bluetooth thermal printer support
5. **Comprehensive Reporting:** Sales analytics and top products
6. **Role-based Access:** Admin/Manager/Staff permissions
7. **Multi-language/Currency:** International support
8. **Dark Mode:** Full theme support

### Code Quality

- Clean separation of concerns (MVC pattern)
- Consistent Redux action patterns
- Reusable service layer
- Comprehensive data models

### Scalability Considerations

- Pagination support for all list endpoints
- Image compression for uploads
- Efficient sync mechanism for offline data
- MongoDB indexing on business field

---

### Document End

*This document was auto-generated from codebase analysis.*