← Back to API Documentation

Restaurant POS System - Frontend Development Guide

📋 Table of Contents

🎯 Project Overview

You are tasked with building a **React-based Frontend** for a Restaurant Point of Sale (POS) System. This system allows restaurant staff (Managers, Waiters, Chefs, and Cashiers) to manage orders, tables, menu items, and restaurant operations.

System Features

- **Multi-role Support**: Manager, Waiter, Chef, Cashier
- **Restaurant Isolation**: Each user can only access their own restaurant's data
- **Real-time Order Management**: Create, update, and track orders
- **Table Management**: Monitor table status (free/occupied)
- **Menu Management**: Organize menu items by categories
- **Automatic Calculations**: SubTotal and TotalAmount calculated automatically
- **JWT Authentication**: Secure token-based authentication

Technology Stack

- **Backend**: Node.js, Express.js, MySQL, JWT
- **Frontend**: React (you will build this)
- **API Documentation**: Swagger UI (available at `/api-docs`)

🚀 Getting Started

Prerequisites

- Node.js (v16 or higher)
- npm or yarn
- Git
- A code editor (VS Code recommended)
- Postman or similar API testing tool

Frontend Setup (Your Task)

1. **Create React App**

```
npx create-react-app restaurant-pos-frontend
cd restaurant-pos-frontend
```

2. **Install Additional Dependencies**

```
npm install axios react-router-dom
# Optional: UI libraries
npm install @mui/material @emotion/react @emotion/styled
# or
npm install react-bootstrap bootstrap
```

3. **Configure API Base URL**

   Create `src/config/api.js`:

```
export const API_BASE_URL = 'http://localhost:3000';
```

📚 API Documentation

Base URL

```
http://localhost:3000
```

Authentication

All protected endpoints require a JWT token in the Authorization header:

```
Authorization: Bearer <your-jwt-token>
```

API Endpoints Summary

1. Authentication Endpoints

Method	Endpoint	Description	Access
POST	`/users/login`	User login	Public
POST	`/users/signup`	User registration	Public

Login Example:

```
POST /users/login
Content-Type: application/json

{
  "UserName": "john_waiter",
  "Password": "password123"
}

Response:
{
  "error": false,
  "message": "Operation Successful",
  "data": {
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
}
```

2. Users Endpoints

Method	Endpoint	Description	Access
GET	`/users`	List all users	Manager
GET	`/users/:id`	Get user by ID	Manager
POST	`/users`	Create user	Manager
PATCH	`/users/:id`	Update user	Manager

DELETE	`/users/:id`	Delete user	Manager
POST	`/users/updatePassword/:id`	Update password	Manager

Note: All user operations are filtered by restaurant - managers can only manage users from their own restaurant.

3. Restaurants Endpoints

Method	Endpoint	Description	Access
GET	`/restaurants`	Get manager's restaurant	Manager
GET	`/restaurants/:id`	Get restaurant by ID	Manager
PATCH	`/restaurants/:id`	Update restaurant	Manager

Note: Managers can only access their own restaurant.

4. Tables Endpoints

Method	Endpoint	Description	Access
GET	`/tables`	List all tables	Manager, Waiter, Chef, Cashier
GET	`/tables/:id`	Get table by ID	Manager, Waiter, Chef, Cashier
GET	`/tables/restaurant/:restaurantId`	Get tables by restaurant	Manager, Waiter, Chef, Cashier
POST	`/tables`	Create table	Manager
PATCH	`/tables/:id`	Update table	Manager, Cashier
DELETE	`/tables/:id`	Delete table	Manager

Table Status: `free` or `occupied`

5. Menu Categories Endpoints

Method	Endpoint	Description	Access
GET	`/menu-categories`	List all categories	Manager, Waiter, Chef, Cashier
GET	`/menu-categories/:id`	Get category by ID	Manager, Waiter, Chef, Cashier
GET	`/menu-categories/restaurant/:restaurantId`	Get categories by restaurant	Manager, Waiter, Chef, Cashier
POST	`/menu-categories`	Create category	Manager
PATCH	`/menu-categories/:id`	Update category	Manager
DELETE	`/menu-categories/:id`	Delete category	Manager

6. Menu Items Endpoints

Method	Endpoint	Description	Access
GET	`/menu-items`	List all items	Manager, Waiter, Chef, Cashier
GET	`/menu-items/:id`	Get item by ID	Manager, Waiter, Chef, Cashier
GET	`/menu-items/category/:categoryId`	Get items by category	Manager, Waiter, Chef, Cashier
POST	`/menu-items`	Create item	Manager
PATCH	`/menu-items/:id`	Update item	Manager
DELETE	`/menu-items/:id`	Delete item	Manager

7. Orders Endpoints

Method	Endpoint	Description	Access
GET	`/orders`	List all orders	Manager, Waiter, Chef, Cashier
GET	`/orders/:id`	Get order by ID	Manager, Waiter, Chef, Cashier
GET	`/orders/table/:tableId`	Get orders by table	Manager, Waiter, Chef, Cashier
POST	`/orders`	Create order	Manager, Waiter
PATCH	`/orders/:id`	Update order	Manager, Chef, Cashier
DELETE	`/orders/:id`	Delete order	Manager

Order Status: `pending` → `preparing` → `served` → `paid`

Role Restrictions:

- **Chef**: Can only update status to `preparing` or `served`
- **Cashier**: Can only update status to `paid` (automatically frees table)
- **Manager**: Can update all fields
- **Waiter**: Read-only (cannot update orders)

8. Order Items Endpoints

Method	Endpoint	Description	Access
GET	`/order-items`	List all order items	Manager, Waiter, Chef, Cashier
GET	`/order-items/:id`	Get order item by ID	Manager, Waiter, Chef, Cashier
GET	`/order-items/order/:orderId`	Get items by order	Manager, Waiter, Chef, Cashier
GET	`/order-items/menu-item/:menuItemId`	Get items by menu item	Manager, Waiter, Chef, Cashier
POST	`/order-items`	Create order item	Manager, Waiter
PATCH	`/order-items/:id`	Update order item	Manager, Waiter
DELETE	`/order-items/:id`	Delete order item	Manager, Waiter

Note: SubTotal is automatically calculated as `Quantity × MenuItemPrice`

🔐 Authentication & Authorization

Authentication Flow

1. **Login**

```
// POST /users/login
const response = await axios.post('http://localhost:3000/users/login', {
  UserName: 'john_waiter',
  Password: 'password123'
});

const token = response.data.data.token;
localStorage.setItem('token', token);
```

2. **Store Token**
   - Store JWT token in `localStorage` or `sessionStorage`
   - Include token in all subsequent API requests

3. **API Request with Token**

```
const token = localStorage.getItem('token');
const response = await axios.get('http://localhost:3000/tables', {
```

```
    headers: {
      'Authorization': `Bearer ${token}`
    }
  });
```

4. **Token Expiration**
   - Tokens expire after 1 hour
   - Handle 401 responses by redirecting to login

Axios Interceptor Setup

Create `src/utils/axios.js`:

```
import axios from 'axios';

const api = axios.create({
  baseURL: 'http://localhost:3000',
});

// Request interceptor - add token to all requests
api.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('token');
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);

// Response interceptor - handle 401 errors
api.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      localStorage.removeItem('token');
      window.location.href = '/login';
    }
    return Promise.reject(error);
  }
);

export default api;
```

👥 Role-Based Access Control

User Roles

Role	Permissions
Manager	Full access to all endpoints within their restaurant
Waiter	View tables/orders, create orders and order items, cannot edit/delete orders
Chef	View tables/orders, update order status to `preparing` or `served`
Cashier	View tables/orders, update order status to `paid`, update table status to `free`

Restaurant Isolation

Important: All operations are automatically filtered by restaurant. Users can only access resources belonging to their restaurant (determined by `RestaurantID` in JWT token).

Role-Based UI

Implement role-based navigation and feature visibility:

```
// src/utils/roles.js
export const getUserRole = () => {
  const token = localStorage.getItem('token');
  if (!token) return null;

  try {
```

```
    const payload = JSON.parse(atob(token.split('.')[1]));
    return payload.data.UserRole;
  } catch (error) {
    return null;
  }
};

export const hasRole = (allowedRoles) => {
  const userRole = getUserRole();
  return allowedRoles.includes(userRole);
};

// Usage in components
import { hasRole } from '../utils/roles';

{hasRole(['manager']) && (
  <button>Manage Users</button>
)}
```

⚛️ React Implementation Guide

1. Project Structure

```
restaurant-pos-frontend/
├── public/
├── src/
│   ├── components/
│   │   ├── Layout/
│   │   │   ├── Sidebar.jsx
│   │   │   ├── Header.jsx
│   │   │   └── Layout.jsx
│   │   ├── ProtectedRoute.jsx
│   │   ├── Loading.jsx
│   │   └── ErrorBoundary.jsx
│   ├── pages/
│   │   ├── Login.jsx
│   │   ├── Dashboard.jsx
│   │   ├── Tables/
│   │   │   ├── TableList.jsx
│   │   │   └── TableForm.jsx
│   │   ├── Orders/
│   │   │   ├── OrderList.jsx
│   │   │   ├── OrderForm.jsx
│   │   │   └── OrderDetails.jsx
│   │   ├── Menu/
│   │   │   ├── MenuCategoryList.jsx
│   │   │   ├── MenuItemList.jsx
│   │   │   └── MenuItemForm.jsx
│   │   └── Users/
│   │       ├── UserList.jsx
│   │       └── UserForm.jsx
│   ├── services/
│   │   ├── api.js
│   │   ├── authService.js
│   │   ├── tableService.js
│   │   ├── orderService.js
│   │   ├── menuService.js
│   │   └── userService.js
│   ├── context/
│   │   └── AuthContext.jsx
│   ├── utils/
│   │   ├── axios.js
│   │   ├── roles.js
│   │   └── constants.js
│   ├── App.js
│   └── index.js
```

2. Authentication Context

Create `src/context/AuthContext.jsx`:

```
import React, { createContext, useState, useContext, useEffect } from 'react';
import api from '../utils/axios';

const AuthContext = createContext();

export const useAuth = () => {
  const context = useContext(AuthContext);
  if (!context) {
    throw new Error('useAuth must be used within AuthProvider');
  }
  return context;
};
```

```
export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const token = localStorage.getItem('token');
    if (token) {
      // Decode token to get user info
      try {
        const payload = JSON.parse(atob(token.split('.')[1]));
        setUser(payload.data);
      } catch (error) {
        localStorage.removeItem('token');
      }
    }
    setLoading(false);
  }, []);

  const login = async (username, password) => {
    try {
      const response = await api.post('/users/login', {
        UserName: username,
        Password: password,
      });

      if (!response.data.error) {
        const token = response.data.data.token;
        localStorage.setItem('token', token);

        // Decode token to get user info
        const payload = JSON.parse(atob(token.split('.')[1]));
        setUser(payload.data);

        return { success: true };
      } else {
        return { success: false, message: response.data.message };
      }
    } catch (error) {
      return {
        success: false,
        message: error.response?.data?.message || 'Login failed'
      };
    }
  };

  const logout = () => {
    localStorage.removeItem('token');
    setUser(null);
  };

  return (
    <AuthContext.Provider value={{ user, login, logout, loading }}>
      {children}
    </AuthContext.Provider>
  );
};
```

3. Protected Route Component

Create `src/components/ProtectedRoute.jsx`:

```
import React from 'react';
import { Navigate } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';

const ProtectedRoute = ({ children, allowedRoles = [] }) => {
  const { user, loading } = useAuth();

  if (loading) {
    return <div>Loading...</div>;
  }

  if (!user) {
    return <Navigate to="/login" replace />;
  }

  if (allowedRoles.length > 0 && !allowedRoles.includes(user.UserRole)) {
    return <Navigate to="/dashboard" replace />;
  }

  return children;
};

export default ProtectedRoute;
```

4. Service Layer Example

Create `src/services/orderService.js`:

Create `src/services/orderService.js`:

```
import api from '../utils/axios';

export const orderService = {
  getAll: async () => {
    const response = await api.get('/orders');
    return response.data;
  },

  getById: async (id) => {
    const response = await api.get(`/orders/${id}`);
    return response.data;
  },

  getByTable: async (tableId) => {
    const response = await api.get(`/orders/table/${tableId}`);
    return response.data;
  },

  create: async (orderData) => {
    const response = await api.post('/orders', orderData);
    return response.data;
  },

  update: async (id, orderData) => {
    const response = await api.patch(`/orders/${id}`, orderData);
    return response.data;
  },

  updateStatus: async (id, status) => {
    const response = await api.patch(`/orders/${id}`, { OrderStatus: status });
    return response.data;
  },

  delete: async (id) => {
    const response = await api.delete(`/orders/${id}`);
    return response.data;
  },
};
```

5. Example Component - Order List

Create `src/pages/Orders/OrderList.jsx`:

```
import React, { useState, useEffect } from 'react';
import { orderService } from '../../services/orderService';
import { useAuth } from '../../context/AuthContext';

const OrderList = () => {
  const [orders, setOrders] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const { user } = useAuth();

  useEffect(() => {
    fetchOrders();
  }, []);

  const fetchOrders = async () => {
    try {
      setLoading(true);
      const response = await orderService.getAll();
      if (!response.error) {
        setOrders(response.data);
      } else {
        setError(response.message);
      }
    } catch (err) {
      setError('Failed to fetch orders');
    } finally {
      setLoading(false);
    }
  };

  const handleStatusUpdate = async (orderId, newStatus) => {
    try {
      const response = await orderService.updateStatus(orderId, newStatus);
      if (!response.error) {
        fetchOrders(); // Refresh list
      } else {
        alert(response.message);
      }
    } catch (err) {
      alert('Failed to update order status');
    }
  };

  if (loading) return <div>Loading orders...</div>;
  if (error) return <div>Error: {error}</div>;
```

```
  return (
    <div>
      <h2>Orders</h2>
      <table>
        <thead>
          <tr>
            <th>Order ID</th>
            <th>Table</th>
            <th>Status</th>
            <th>Total Amount</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          {orders.map((order) => (
            <tr key={order.OrderID}>
              <td>{order.OrderID}</td>
              <td>Table {order.TableNumber}</td>
              <td>{order.OrderStatus}</td>
              <td>${order.TotalAmount}</td>
              <td>
                {user.UserRole === 'chef' && (
                  <>
                    <button onClick={() => handleStatusUpdate(order.OrderID, 'preparing')}>
                      Start Preparing
                    </button>
                    <button onClick={() => handleStatusUpdate(order.OrderID, 'served')}>
                      Mark Served
                    </button>
                  </>
                )}
                {user.UserRole === 'cashier' && (
                  <button onClick={() => handleStatusUpdate(order.OrderID, 'paid')}>
                    Mark Paid
                  </button>
                )}
              </td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
};

export default OrderList;
```

📁 Project Structure Suggestions

Recommended Folder Structure

```
src/
├── components/          # Reusable components
│   ├── common/         # Common UI components (Button, Input, etc.)
│   ├── layout/         # Layout components (Header, Sidebar, Footer)
│   └── features/       # Feature-specific components
├── pages/              # Page components
├── services/           # API service functions
├── context/            # React Context providers
├── hooks/              # Custom React hooks
├── utils/              # Utility functions
├── constants/          # Constants and configuration
└── styles/             # Global styles
```

🎨 Key Features to Implement

1. Authentication System

- [ ] Login page
- [ ] Logout functionality
- [ ] Token storage and management
- [ ] Protected routes
- [ ] Auto-redirect on token expiration

2. Dashboard

- [ ] Role-based dashboard
- [ ] Statistics overview (orders, tables, revenue)
- [ ] Quick actions based on role

3. Table Management

- [ ] View all tables with status
- [ ] Create/Edit/Delete tables (Manager only)
- [ ] Visual table status indicators
- [ ] Filter tables by status

4. Order Management

- [ ] View all orders
- [ ] Create new order
- [ ] View order details
- [ ] Update order status (role-based)
- [ ] Order history

5. POS Interface (Waiter)

- [ ] Table selection
- [ ] Menu item selection
- [ ] Shopping cart
- [ ] Add items to order
- [ ] Calculate totals
- [ ] Submit order

6. Kitchen Display (Chef)

- [ ] View pending orders
- [ ] Update order status
- [ ] Filter by status
- [ ] Order preparation queue

7. Menu Management (Manager)

- [ ] Manage menu categories
- [ ] Manage menu items
- [ ] Upload images
- [ ] Price management

8. User Management (Manager)

- [ ] View all users
- [ ] Create/Edit/Delete users
- [ ] Change user roles
- [ ] Reset passwords

📦 API Response Format

All API responses follow this format:

```
{
  "error": false,           // boolean: true if error occurred
  "message": "Operation Successful",  // string: status message
  "data": { ... }          // object/array: response data
}
```

Success Response Example

```
{
  "error": false,
```

```
  "message": "Operation Successful",
  "data": [
    {
      "OrderID": 1,
      "TableID": 5,
      "TableNumber": 5,
      "TotalAmount": 45.97,
      "OrderStatus": "pending",
      "CreatedAt": "2024-01-15T10:30:00Z"
    }
  ]
}
```

Error Response Example

```
{
  "error": true,
  "message": "Access denied: This order does not belong to your restaurant",
  "data": null
}
```

HTTP Status Codes

- `200` : Success
- `400` : Bad Request (validation error)
- `401` : Unauthorized (invalid/missing token)
- `403` : Forbidden (insufficient permissions)
- `404` : Not Found
- `500` : Internal Server Error

💡 Common Patterns & Best Practices

1. Error Handling

```
const handleApiCall = async () => {
  try {
    const response = await api.get('/endpoint');
    if (response.data.error) {
      // Handle API-level error
      console.error(response.data.message);
      return;
    }
    // Success
    return response.data.data;
  } catch (error) {
    // Handle network/HTTP errors
    if (error.response) {
      // Server responded with error status
      console.error('Error:', error.response.data.message);
    } else {
      // Network error
      console.error('Network error:', error.message);
    }
  }
};
```

2. Loading States

```
const [loading, setLoading] = useState(false);

const fetchData = async () => {
  setLoading(true);
  try {
    // API call
  } finally {
    setLoading(false);
  }
};
```

3. Form Handling

```
const [formData, setFormData] = useState({
  TableNumber: '',
```

```
  TableCapacity: '',
  TableStatus: 'free'
});

const handleChange = (e) => {
  setFormData({
    ...formData,
    [e.target.name]: e.target.value
  });
};

const handleSubmit = async (e) => {
  e.preventDefault();
  // Submit form
};
```

4. Conditional Rendering Based on Role

```
import { useAuth } from '../context/AuthContext';

const MyComponent = () => {
  const { user } = useAuth();

  return (
    <div>
      {user?.UserRole === 'manager' && (
        <button>Manage Users</button>
      )}
      {['manager', 'waiter'].includes(user?.UserRole) && (
        <button>Create Order</button>
      )}
    </div>
  );
};
```

🧪 Testing Guidelines

1. Test Different User Roles

- Create test accounts for each role (Manager, Waiter, Chef, Cashier)
- Verify role-based access restrictions
- Test restaurant isolation

2. Test API Integration

- Test all CRUD operations
- Test error handling (401, 403, 404)
- Test form validations
- Test loading states

3. Test Edge Cases

- Empty data lists
- Network failures
- Token expiration
- Invalid inputs
- Cross-restaurant access attempts

4. UI/UX Testing

- Responsive design
- Loading indicators
- Error messages
- Success notifications
- Form validations

🔧 Troubleshooting

Common Issues

1. **401 Unauthorized**
   - Check if token is included in request headers
   - Verify token hasn't expired
   - Ensure token format: `Bearer <token>`
2. **403 Forbidden**
   - Check user role permissions
   - Verify restaurant ownership
   - Ensure correct endpoint access
3. **CORS Errors**
   - Backend should handle CORS (already configured)
   - Check if API base URL is correct
4. **Empty Data**
   - Verify restaurant filtering is working
   - Check if user has RestaurantID in token
   - Ensure database has data for user's restaurant
5. **Token Not Persisting**
   - Check localStorage/sessionStorage usage
   - Verify token is being saved after login
   - Check browser storage permissions

📝 Additional Resources

API Documentation

- Swagger UI: `http://localhost:3000/api-docs`
- Interactive API testing interface

Recommended Libraries

- **Routing**: `react-router-dom`
- **HTTP Client**: `axios`
- **State Management**: React Context API or Redux
- **UI Framework**: Material-UI, React Bootstrap, or Tailwind CSS
- **Form Handling**: React Hook Form
- **Date Handling**: date-fns or moment.js

Project Deliverables

Your project should include:

1. ✅ Complete React application
2. ✅ Authentication system
3. ✅ Role-based access control
4. ✅ All CRUD operations for main entities
5. ✅ Responsive design
6. ✅ Error handling
7. ✅ Loading states
8. ✅ Clean, readable code
9. ✅ Code comments where necessary
10. ✅ README with setup instructions

🎓 Learning Objectives

By completing this project, you will:

By completing this project, you will:

- Understand RESTful API integration
- Implement JWT authentication
- Work with role-based access control
- Handle state management in React
- Create responsive UI components
- Implement error handling and validation
- Practice clean code principles
- Understand restaurant POS system workflows

📞 Support

If you encounter issues:

1. Check the Swagger documentation at `/api-docs`
2. Review error messages in browser console
3. Test API endpoints directly with Postman
4. Verify database connectivity
5. Check environment variables

Good luck with your project! 🚀

Remember: Focus on understanding the API structure first, then build your React components step by step. Start with authentication, then move to simpler features before tackling complex workflows.