🎯 DELIVEREI - Priority Action Plan

Date: October 13, 2025
Status: Backend deployment issues RESOLVED ✅
Focus: Feature implementation and system enhancement

📸 Dashboard Analysis

Based on the uploaded dashboard screenshot (`dashboard-sem-filtro.png`), the current system shows:

✅ Working Features

- Sales statistics display (R$ 0,00 today)
- Open orders counter (3 orders)
- Average ticket calculation (R$ 65,70)
- Low stock warnings (1 item)
- Store URL sharing (“Vitrine da Loja”)
- Recent orders section

🚧 Features Needing Implementation

- **Sales graph** showing “Erro ao carregar dados de vendas” (Error loading sales data)
- **Date filtering** - Dropdown shows “Últimos 7 dias” (Last 7 days) but no data
- **Graph visualization** - Empty chart area
- **Recent orders details** - Section shows “Pedido #0000” but incomplete

---

🎯 Priority 1: Fix Dashboard Sales Graph (CRITICAL)

Issue

The sales graph shows “Erro ao carregar dados de vendas - Tente novamente mais tarde” (Error loading sales data - Try again later)

Action Items

1. **Verify Backend Endpoint**
   `bash`
   ```
      # Test the sales endpoint
      GET /api/dashboard/vendas?periodo=semana
   ```
   - Check if endpoint returns proper data structure
   - Verify date filtering logic (dia/semana/mes)
   - Ensure empresaId filtering works correctly

2. **Check Frontend Integration**
   - Verify API call in dashboard component
   - Check error handling and display logic
   - Ensure date range parameters are passed correctly

3. **Add Sample Data (if needed)**
   - Create test orders with varied dates
   - Populate product sales data
   - Test with different date ranges

Expected Outcome

- Sales graph displays properly with bars/lines showing sales over time
- Date filter dropdown works (dia/semana/mes)
- Smooth data loading with proper loading states

🎯 Priority 2: Implement Dashboard Filtering

Current State

The dashboard has a filter dropdown (“Últimos 7 dias”) but filtering functionality appears incomplete.

Action Items

1. **Add Date Range Filtering**
   - Implement date picker or range selector
   - Add preset ranges (Today, Last 7 days, Last 30 days, This month)
   - Custom date range selection

2. **Real-time Data Updates**
   - Refresh statistics when date range changes
   - Update all dashboard widgets simultaneously
   - Add loading states during data refresh

3. **Persist Filter Preferences**
   - Save user’s selected date range
   - Restore on page reload
   - Store in localStorage or user preferences

Implementation Files

```
frontend/src/pages/Dashboard.tsx
frontend/src/components/DashboardFilter.tsx
frontend/src/services/dashboard.service.ts
```

🎯 Priority 3: Complete Recent Orders Section

Current State

Shows “Pedidos recentes” section with “Pedido #0000” placeholder

Action Items

1. **Fetch Recent Orders**
   ```typescript
   ```

// API endpoint
GET /api/dashboard/pedidos-recentes?limit=10

// Expected response
{
pedidos: [
{
id: number,
numero: string,
cliente: { nome: string, telefone: string },
status: string,
total: number,
createdAt: string
}
]
}

1. **Display Order Cards**
   - Order number and timestamp
   - Customer name
   - Order status badge (pendente, em preparo, saiu para entrega, entregue)
   - Order total
   - Quick action buttons (view details, update status)

2. **Add Order Status Updates**
   - Allow quick status changes from dashboard
   - Real-time notifications when status changes
   - Status history tracking

🎯 Priority 4: Low Stock Alerts Implementation

Current State

Dashboard shows “Baixo estoque: 1” (Low stock: 1 item)

Action Items

1. **Complete Low Stock Logic**
   ```typescript
   // Backend endpoint
   GET /api/dashboard/produtos-baixo-estoque?threshold=5

// Returns products with quantity <= threshold
```

1. **Low Stock Alert Panel**
   - List products below minimum stock level
   - Show current quantity vs. minimum required
   - Link to product management page
   - Option to mark as “ordered” or “restocked”

2. **Stock Notifications**
   - Real-time alerts when stock goes below threshold
   - Email/WhatsApp notifications to admin
   - Daily stock report summary

🎯 Priority 5: Multi-Tenant Store Frontend (Vitrine)

Current State

Dashboard shows store URL: `https://deliverei.netlify.app/loja/pizza-express`

Action Items

1. **Public Store Page**
   - Customer-facing product catalog
   - Shopping cart functionality
   - Checkout and order placement
   - Store branding (logo, colors, description)

2. **URL Structure**
   `https://deliverei.netlify.app/loja/{empresa-slug}`

3. **Store Features**
   - Product listing with images and prices
   - Category filtering
   - Search functionality
   - Product details modal
   - Add to cart with quantity selection
   - Order tracking for customers

🎯 Priority 6: WhatsApp Integration

Business Impact

Essential for Brazilian market - customers expect WhatsApp communication

Action Items

1. **Order Notifications via WhatsApp**
   - Order confirmation message
   - Status update messages
   - Delivery tracking link
   - Customer support quick replies

2. **WhatsApp Business API Integration**
   `typescript`
   ```
     // backend/src/modules/whatsapp/whatsapp.service.ts
     sendOrderConfirmation(pedidoId: number)
     sendStatusUpdate(pedidoId: number, newStatus: string)
     sendDeliveryNotification(pedidoId: number, entregadorInfo: object)
   ```

3. **Template Messages**
   - Order confirmation template
   - Status update templates
   - Delivery ETA template
   - Feedback request template

🎯 Priority 7: Authentication & User Management

Security Enhancements

1. **JWT Token Management**
   - Refresh token implementation
   - Token expiration handling
   - Secure token storage

2. **Role-Based Access Control (RBAC)**
   - Super Admin: Full system access
   - Admin Empresa: Tenant-specific access
   - Entregador: Delivery-specific features
   - Cliente: Customer features only

3. **User Registration Flow**
   - Email verification
   - Password reset functionality
   - Two-factor authentication (optional)

🎯 Priority 8: Order Management System

Current Workflow Gaps

1. **Order Lifecycle**
   `Novo → Em Preparação → Pronto → Saiu para Entrega → Entregue`

2. **Kitchen Display System (KDS)**
   - Real-time order queue for kitchen
   - Time tracking per order
   - Preparation time estimates
   - Order priority management

3. **Delivery Management**
   - Assign orders to delivery personnel
   - Route optimization
   - Real-time location tracking
   - Delivery completion confirmation

📊 Implementation Roadmap

Week 1: Dashboard & Data Visualization

- [ ] Fix sales graph data loading
- [ ] Implement date filtering
- [ ] Complete recent orders section
- [ ] Test with real data

Week 2: Store Frontend & Multi-tenant

- [ ] Build public store page
- [ ] Implement shopping cart
- [ ] Add checkout flow
- [ ] Test with multiple tenants

Week 3: Communication & Notifications

- [ ] WhatsApp integration setup
- [ ] Order notification templates
- [ ] Email notifications (backup)
- [ ] Test message delivery

Week 4: Order Management & Delivery

- [ ] Kitchen display system
- [ ] Delivery assignment logic
- [ ] Order status workflow
- [ ] Real-time updates

🛠️ Technical Setup Required

1. Development Environment

```
# Clone repository (if not done)
git clone https://github.com/nerdrico2025/deliverei-v1.git
cd deliverei-v1

# Backend setup
cd backend
npm install
npx prisma generate
npm run start:dev

# Frontend setup (new terminal)
cd ../frontend
npm install
npm run dev
```

2. Environment Variables

```
# Backend (.env)
DATABASE_URL="postgresql://..."
JWT_SECRET="your-secret-key"
WHATSAPP_API_KEY="..."
WHATSAPP_PHONE_NUMBER="+55..."

# Frontend (.env)
VITE_API_URL="http://localhost:3000"
```

3. Database Seeding

```
# Create seed script for test data
npx prisma db seed

# Seed data includes:
# - Test empresa (Pizza Express)
# - Sample products
# - Sample orders
# - Test users
```

🧪 Testing Strategy

1. Unit Tests

```
# Backend
npm run test

# Frontend
npm run test:unit
```

2. Integration Tests

- API endpoint testing
- Database transaction testing
- Authentication flow testing

3. End-to-End Tests

```
# Using Cypress or Playwright
npm run test:e2e
```

Test Coverage Goals

- Backend: > 80%
- Frontend: > 70%
- Critical paths: 100%

📈 Success Metrics

User Experience

- Dashboard loads in < 2 seconds
- Order placement in < 30 seconds
- WhatsApp notification within 1 minute
- 99.5% uptime

Business Metrics

- Zero data loss
- < 1% error rate
- Average order processing time < 5 minutes
- Customer satisfaction > 4.5/5

🚨 Immediate Actions (Next 48 Hours)

1. **Fix Sales Graph** ⚠️ URGENT
   - This is the most visible error on the dashboard
   - Investigate backend endpoint `/api/dashboard/vendas`
   - Check frontend data fetching logic
   - Test with sample data

2. **Create Test Data**
   - Create seed script for development
   - Add sample empresa (Pizza Express)
   - Add 10-20 sample products
   - Create sample orders with different dates

3. **Test Multi-tenant Logic**
   - Verify empresa isolation
   - Test subdomain routing
   - Validate data filtering by empresaId

4. **Documentation**
   - API endpoint documentation
   - Frontend component structure
   - Database schema relationships
   - Deployment procedures

🔗 Related Resources

Documentation

- Main Deployment Summary (./DEPLOYMENT_SUMMARY.md)
- Render Migration Setup (./backend/render-migration-setup.md) (if exists)
- API Documentation (./backend/docs/API.md) (to be created)

External Links

- Production Backend (https://deliverei-backend.onrender.com)
- Production Frontend (https://deliverei.netlify.app)
- GitHub Repository (https://github.com/nerdrico2025/deliverei-v1)
- Render Dashboard (https://dashboard.render.com/web/srv-d3lh0q0gjchc73cdeokg)

💬 Questions & Decisions Needed

Before proceeding with implementation, please clarify:

1. **Dashboard Graph Priority**
   - Should we fix the sales graph first or focus on other features?
   - What data granularity is needed (hourly, daily, weekly)?

2. **Multi-tenant Strategy**
   - Use subdomain routing (pizza-express.deliverei.com) or path-based (/loja/pizza-express)?
   - Custom domains for tenants?

3. **WhatsApp Integration**
   - Do you have WhatsApp Business API access?
   - Alternative: Use WhatsApp Web API or third-party service?

4. **Payment Integration**
   - Payment methods needed (PIX, credit card, cash)?
   - Use Stripe, Mercado Pago, or other payment gateway?

5. **Real-time Features**
   - Should we implement WebSockets for real-time updates?
   - Consider using Socket.io or native WebSocket?

📞 Need Help?

If you need assistance with any of these priorities:
1. Clarify specific requirements
2. Provide access credentials (if needed)
3. Share any existing design mockups
4. Discuss timeline and resource constraints

Document Created: October 13, 2025
Last Updated: October 13, 2025
Version: 1.0