# **Delivery Admin Dashboard & Routing Logic -**

# **Project Walkthrough**

## **Overview**

This full-stack delivery management and route optimization system is built for dispensary orders. It includes a modern admin dashboard, simulated route batching, and a modular codebase for easy extension.

### **Tech Stack**

* **Backend:** Node.js, Express, MongoDB (Mongoose)
* **Frontend:** Next.js (React), TypeScript, Tailwind CSS

## **Backend Walkthrough (backend/)**

### **1. Entry Point & Server Setup**

* **src/app.ts**: Initializes Express app, applies middleware (JSON, CORS), mounts order routes.
* **src/server.ts**: Connects to MongoDB, starts server using src/config/db.ts and environment variables.

### **2. Database Model**

* **src/models/Order.ts**: Mongoose schema with:
  + customer: String
  + dispensary: String
  + borough: String (e.g., Brooklyn)
  + status: Enum (PLACED, DISPATCHED, etc.)
  + createdAt: Date (auto-generated)

### **3. Controllers**

* **src/controllers/orderController.ts**:
  + getOrders(req, res): Fetch orders, optionally by borough, sorted by creation date.
  + updateOrderStatus(req, res): Update order status by ID.
  + createOrder(req, res): Create order with default status PLACED.
  + optimizeRoutes(req, res): Simulates batching by borough; creates batches if 5+ orders in borough.

### **4. Routes**

* **src/routes/orderRoutes.ts**:
  + GET /orders: List all orders
  + POST /orders: Create order
  + PATCH /orders/:id/status: Update status
  + GET /orders/optimize: Optimize routes

### **5. Utilities & Config**

* **src/utils/asyncHandler.ts**: Wraps async handlers to catch errors.
* **src/config/db.ts**: MongoDB connection setup.

### **6. Environment & Scripts**

* **.env**: Stores MONGODB\_URI and other secrets.
* **Scripts**:
  + npm run dev: Dev mode with nodemon
  + npm start: Production start

## **Frontend Walkthrough (frontend/)**

### **1. API Layer**

* **api/api.ts**:
  + getOrders(borough?)
  + createOrder(order)
  + updateOrderStatus(id, status)
  + optimizeRoutes()

### **2. TypeScript Types**

* **types/index.ts**: Type definitions for Order, OrderStatus, Borough, Batch, etc.

### **3. App Pages (Next.js App Router)**

* **app/admin/page.tsx**: Redirects or acts as dashboard entry.
* **app/admin/dashboard/page.tsx**: Main dashboard page with table and filter.
* **app/admin/orders-grouped/page.tsx**: (Optional) Displays borough-grouped orders.
* **app/page.tsx**: Landing page (optional).

### **4. Components**

* **Order Table & Display**:
  + OrderTable.tsx, OrderRow.tsx, OrderTableHeader.tsx
  + EmptyState.tsx, TableSkeleton.tsx
* **Filters & Grouping**:
  + BoroughFilter.tsx
  + OrdersByBoroughDisplay.tsx
* **Batching & Optimization**:
  + OptimizeButton.tsx
  + OptimizedBatches.tsx
* **Layout/UI**:
  + NavbarHeader.tsx, MuiProvider.tsx

### **5. Context & State Management**

* **context/OrderContext.tsx**: Global order state context and methods.

### **6. Utilities**

* **utils/groupByBorough.ts**: Utility for grouping orders.
* **utils/ui.tsx**: UI helpers.

### **7. Styling**

* **Tailwind CSS** setup with globals.css, tailwind.config.js

### **8. Environment & Scripts**

* **.env.local**: Set NEXT\_PUBLIC\_API\_URL
* **Scripts**:
  + npm run dev: Start dev server
  + npm run build && npm start: Build and run production

## **Feature Walkthrough (End-to-End)**

### **1. Viewing Open Delivery Orders**

* Path: /admin/dashboard
* Displays PLACED orders in a table
* Shows customer, dispensary, status, borough
* Uses loading and empty state components

### **2. Filtering by Borough**

* BoroughFilter.tsx allows borough selection
* Filters via API or utility grouping

### **3. Updating Order Status**

* Status update buttons/dropdowns per row
* Uses updateOrderStatus(id, status)
* Triggers backend update and state refresh

### **4. Simulated Routing Logic**

* Triggered by OptimizeButton.tsx
* Backend groups PLACED orders by borough
* Creates batches (5+ orders/borough)
* Assigns fleet (e.g., Fleet A, Fleet B)
* UI shows:
  + Borough
  + Order count
  + Fleet assignment
  + Order list
  + Unbatched order count

## **How to Run (Development)**

### **Backend**

cd backend  
npm install  
Add .env with MONGODB\_URI  
npm run dev

* Runs at <http://localhost:5000>

### **Frontend**

cd frontend  
npm install  
Add .env.local with NEXT\_PUBLIC\_API\_URL=http://localhost:5000/api  
npm run dev

* Runs at <http://localhost:3000>

## **Extending the Project**

* Add borough/fleet admin tools
* Custom batch sizes or real routing APIs
* Admin dashboard enhancements
* Role-based auth
* Real-time updates (WebSockets)
* Mobile-responsive design

## **Summary**

This delivery admin dashboard system offers route optimization, order status tracking, and borough filtering. The modular full-stack setup supports future enhancements and real-world deployment. Check code comments and modules for deeper insight.