Data Flow Models - Telecom Mission Control SaaS

This document outlines the frontend-to-backend data flow for each feature in the Telecom SaaS application. It ensures clarity on how data travels across components, Redux, API, and database layers.

# Generic Data Flow

User Interaction → React Component → Redux Action → Axios Request → Node.js Route → MongoDB → Response → Redux Reducer → React View Update

# 1. Authentication Flow

1. User submits login form  
2. `login()` Redux action sends POST to `/api/auth/login`  
3. Backend verifies user in MongoDB  
4. Returns accessToken, refreshToken, and role  
5. `authReducer` updates state  
6. UI redirects to `/dashboard`

# 2. Dashboard Summary Flow

1. Dashboard screen uses `useEffect` to fetch `/api/towers`, `/api/devices`, `/api/policies`, `/api/users`  
2. Each response updates component state via `setSummary`  
3. UI renders total counts of each resource

# 3. Towers Flow

1. `TowerScreen` fetches data using axios GET `/api/towers`  
2. Node.js controller fetches tower list from MongoDB  
3. Response is stored in local state (`towers[]`)  
4. UI filters and renders list  
5. Search input filters the displayed results

# 4. Devices Flow

1. `DeviceScreen` loads device list via GET `/api/devices`  
2. MongoDB returns array of devices  
3. `devices[]` rendered in grid with live filtering by deviceId  
4. Future update: Add/Edit/Delete device

# 5. Policies Flow

CREATE:  
1. User submits policy form  
2. POST `/api/policies` is called with appName, role, allowed/denied actions  
3. Server saves document in MongoDB and returns the new policy  
4. Local `policies[]` state is updated and re-rendered  
  
LIST:  
1. GET `/api/policies` on load  
2. Display policies in accessible list  
  
EDIT:  
1. Click edit icon → prefill form  
2. PUT `/api/policies/:id` with updated fields  
3. Update state and re-render  
  
DELETE:  
1. Click delete icon  
2. DELETE `/api/policies/:id`  
3. Filter removed item from `policies[]`

# 6. User Management Flow

LIST:  
1. Admin fetches user list via GET `/api/users`  
2. UI renders all usernames and roles  
  
ADD:  
1. Fill form → POST `/api/users`  
2. New user created and list refreshed

# 7. Redux Interaction Model

1. Component dispatches Redux action (e.g., fetchPolicies)  
2. Thunk middleware triggers Axios call  
3. On success, reducer updates state  
4. Components subscribed via `useSelector()` re-render automatically