Below is a **creative yet practical** plan for your **frontend** built with **React/Next.js**. The plan covers recommended **folder structure**, **pages**, **dashboards**, and **role-based UX** considerations. It also includes some **UI libraries** options, **real-time** data handling, and best practices for maintainable, scalable code.

# ****1. Technology and Libraries****

**Next.js**

* 1. Leverage server-side rendering (SSR) for performance and SEO where needed.
  2. Leverage dynamic routing for resources (e.g., /interventions/[id]).

**UI Libraries**

* 1. **Material UI (MUI)** or **Chakra UI** or **Ant Design** to accelerate development with a polished, responsive design system.
  2. Alternatively, a custom design system if you have unique brand guidelines.

**State Management & Data Fetching**

* 1. **React Query** (TanStack Query) for data fetching, caching, and real-time updates.
     1. Great synergy with Next.js, easy to integrate with RESTful or GraphQL endpoints.
     2. Automatically handles caching, refetching, stale data, etc.
  2. Or **Redux Toolkit** if you prefer a centralized store for more complex state across many components.

**Real-time Communication**

* 1. **Socket.IO** for real-time events (e.g., vehicle status changes, alerts).
  2. Alternatively, **SWR** subscriptions or **React Query** subscriptions if using a custom event stream.

**Authentication**

* 1. **NextAuth** or a custom JWT solution.
  2. **Role-based access control**: conditionally render pages/components based on user roles.

**Map & Geolocation**

* 1. **Mapbox** or **Google Maps** for displaying interventions, vehicle/user locations in real-time.

# ****2. Folder Structure****

A Next.js project typically follows:

my-frontend/

├── public/

│ └── assets, images, etc.

├── src/

│ ├── components/

│ │ ├── common/

│ │ │ └── Layout.tsx

│ │ ├── dashboard/

│ │ │ └── ...

│ │ └── ...

│ ├── hooks/

│ │ └── useVehicles.ts

│ ├── pages/

│ │ ├── \_app.tsx

│ │ ├── api/

│ │ ├── index.tsx

│ │ ├── login.tsx

│ │ ├── register.tsx

│ │ ├── dashboard/

│ │ │ ├── index.tsx

│ │ │ ├── interventions.tsx

│ │ │ └── vehicles.tsx

│ │ └── ...

│ ├── services/

│ │ └── apiClient.ts

│ ├── store/ or ├── queries/

│ └── utils/

└── package.json

* pages/: Next.js routing (e.g., pages/login.tsx -> /login).
* components/: Reusable UI components (e.g., <NavBar>, <Sidebar>, <Card>, etc.).
* services/: API utilities, wrappers for fetch or axios calls, authentication helpers.
* store/ or queries/: If using Redux or React Query.
* utils/: Helper functions (e.g., date formatting, error handling).

# ****3. User Roles & High-Level Pages****

You have multiple roles in your system (firefighter, coordinator, logistic\_officer, etc.). Each role should see a **role-specific dashboard** that highlights the data they need at a glance.

Below is a **role-based** overview of pages. We’ll group them under a single “dashboard” umbrella, but each user’s role determines which routes and components they see.

## ****3.1 Common (All Roles)****

**Login Page** (/login)

* + Simple form with email/password.
  + Integration with the backend’s /api/auth/login.
  + On success, store JWT or session in cookies for SSR.

**Register Page** (/register)

* + Usually for admin use or open sign-up if that’s the flow.
  + Could be restricted so that only certain roles (like “coordinator” or “admin”) can add new users.

**Profile Page** (/profile)

* + Allows user to update personal info (first name, last name, password, etc.).
  + Optionally update location or status (if relevant to the role).

**Home/Dashboard Landing** (/dashboard)

* + A unified landing page that detects role and redirects to the correct sub-dashboard.

## ****3.2 Firefighter (Sapeur-Pompier) Pages****

**My Interventions** (/dashboard/my-interventions)

* + List of interventions assigned to the user.
  + Action buttons to **Update Status**: “en\_route”, “on\_site”, “finished”.
  + A “Map View” to see the location of the intervention, plus quick access to local weather info.

**Real-Time Location Sharing**

* + Possibly integrated into **My Interventions** or a separate page:  
    /dashboard/location
  + A map or a toggle button that lets them share location with the command center.

**Health & Safety** (/dashboard/my-health) (Optional)

* + If tracking vitals (e.g., heart rate, temperature).
  + Possibly only visible if wearables or IoT integrations are used.

## ****3.3 Chef d’Agrès / Officier de Commandement Pages****

**Team Management** (/dashboard/team)

* + View assigned team members (firefighters).
  + Assign roles or tasks (e.g., driver, medic).
  + Track location & status of each team member.

**Active Interventions** (/dashboard/interventions)

* + A consolidated view of all ongoing interventions under their command.
  + Real-time status updates: which vehicles are on-site, how many firefighters deployed.
  + Possibly a Kanban or timeline view of each intervention stage.

**Map Overview** (/dashboard/map)

* + A real-time map showing all interventions, vehicles, and personnel.
  + Filter by region or resource type.

## ****3.4 Coordonnateur Régional Pages****

**Centralized Dashboard** (/dashboard/coordination)

* + View **all** interventions across multiple regions.
  + Resource availability by region (vehicles, staff).
  + Predictive resource alerts (AI module) for upcoming needs.

**Inter-Regional Assignments** (/dashboard/assignments)

* + Tools to shift resources between regions if short on vehicles or staff.
  + Graphical or tabular interface showing each region’s capacity/usage.

**Unified Dispatch Orders** (/dashboard/dispatch)

* + Form to create new dispatch orders that get shared with subordinate officers.

## ****3.5 Logistic Officer Pages****

**Resource Inventory** (/dashboard/resources)

* + Track vehicles, equipment, consumables (water, fuel).
  + Real-time or near real-time updates of resource status.

**Alerts Management** (/dashboard/alerts)

* + List of triggered resource alerts (e.g., fuel below threshold).
  + Ability to mark alerts as resolved, reorder supplies, or escalate.

**Maintenance Requests** (/dashboard/maintenance)

* + Manage the scheduling of vehicle/equipment maintenance.
  + Link maintenance tasks with resource availability.

## ****3.6 Transcription & AI Pages**** (Common or specialized)

**Transcription Console** (/dashboard/transcriptions)

* + Displays a list of recent or ongoing transcriptions from emergency calls.
  + Option to review transcripts, attach them to specific interventions.
  + Possibly restricted to officers or coordinators.

**Predictive Analytics** (/dashboard/ai-insights)

* + Summaries of resource forecasting models.
  + Charts showing predicted vs. actual usage over time.
  + Download or export data for offline analysis or reports.

# ****4. Key Pages in Detail****

## ****4.1 Login Page****

* **Route**: /login
* **Components**:
  + <LoginForm />
* **Flow**:
  1. User enters email/password.
  2. API call to /api/auth/login.
  3. Store token in an HTTP-only cookie or local storage (based on your security strategy).
  4. Redirect to /dashboard.

## ****4.2 Dashboard Layout****

* **Route**: /dashboard
* **Components**:
  + <Sidebar />, <Header />, <Footer />, <NotificationBell />
* **Logic**:
  + Checks user role from session or token.
  + Dynamically loads the relevant sub-pages or redirects.
  + Contains a main content area where nested routes are displayed.

## ****4.3 My Interventions (Firefighter)****

* **Route**: /dashboard/my-interventions
* **UI**:
  + Table or list of interventions with status.
  + **Action**: “Change status” (dropdown or button).
  + Map snippet to visualize location.
* **Real-time**:
  + Subscribes to WebSocket event intervention-updated.

## ****4.4 Interventions Overview (Officers)****

* **Route**: /dashboard/interventions
* **UI**:
  + Table/Calendar/Kanban for all active interventions.
  + Filter or search by incident type, region, status.
  + Real-time updates for changes in assigned vehicles/personnel.

## ****4.5 Resource Inventory (Logistics)****

* **Route**: /dashboard/resources
* **UI**:
  + Table with resource type, quantity/status, assigned region.
  + Ability to create new resources or mark resources as **maintenance**.
* **Real-time**:
  + Subscribes to events like vehicle-updated, alert-triggered.

## ****4.6 Coordination (Coordinator)****

* **Route**: /dashboard/coordination
* **UI**:
  + High-level region summary: total interventions, vehicles in use, staff on duty.
  + Possibly an integrated map or chart showing multi-region data.
* **Actions**:
  + Button to open **Inter-Regional Assignments** page to move resources.

# ****5. Integration & Data Flow****

**Data Fetching with React Query**

* + Create “hooks” like useInterventions(), useVehicles(), etc.:

import { useQuery } from '@tanstack/react-query';

import apiClient from '../services/apiClient';

function useVehicles() {

return useQuery(['vehicles'], () => apiClient.get('/vehicles').then(res => res.data));

}

* + In your component:

const { data: vehicles, isLoading } = useVehicles();

* + The data auto-refreshes if you configure refetch intervals or respond to socket events.

**Real-Time Updates**

* + In \_app.tsx or a custom provider, initialize **Socket.IO** client:

import { io } from 'socket.io-client';

import { createContext, useEffect, useState } from 'react';

const socket = io(process.env.NEXT\_PUBLIC\_SOCKET\_URL);

// Provide it via context to children, or use a custom hook

* + Listen for events (e.g., vehicle-updated) and trigger a React Query invalidation or local state update.

**Role-Based Rendering**

* + Next.js has **middleware** or simple checks in pages:

// pages/dashboard/index.tsx

import { useContext } from 'react';

import { AuthContext } from '../../contexts/AuthContext';

export default function Dashboard() {

const { user } = useContext(AuthContext);

if (!user) return <p>Loading...</p>;

if (user.role === 'firefighter') {

return <FirefighterDashboard />;

} else if (user.role === 'coordinator') {

return <CoordinatorDashboard />;

}

// etc.

}

* + Or use dynamic routes and separate pages, requiring checks in each.

# ****6. Advanced UI/UX Considerations****

**Offline Capabilities**

* + If needed for firefighters in remote areas, use local storage or service workers to cache data.
  + Show partial data from the last sync.

**Accessibility (A11y)**

* + High-contrast themes and ARIA labels for critical UI elements.
  + Keyboard navigation for forms and tables.

**Localization**

* + Next.js’ built-in i18n support or libraries like react-i18next.
  + If multi-language transcripts are used, consider toggling transcript language in the UI.

**Notifications**

* + Real-time toast or banner alerts for new interventions, resource alerts, etc.
  + Use a library like react-toastify or MUI’s Snackbar for pop-up notifications.

**Charts & Graphs**

* + For analytics dashboards or predictive data: **Recharts**, **Chart.js**, or **Victory**.
  + Display trending data about interventions, resource usage, or predictions.

# ****7. Testing****

**Unit Tests**

* + Use **Jest** + **React Testing Library** to test individual components and pages.
  + Example: check if the login page renders the login form.

**End-to-End Tests**

* + **Cypress** or **Playwright** to test flows: user logs in, sees dashboard, updates an intervention, etc.

**Integration with CI/CD**

* + Similar to the backend, run tests automatically on each pull request.

# ****8. Deployment****

**Hosting**

* + Next.js can be hosted on **Vercel** (native support), **Netlify**, or your own server with **Node.js**.
  + If using SSR, ensure your environment variables and Node server run properly in production.

**Environment Variables**

* + .env.local for dev, .env.production for production.
  + Keep secrets (API keys, etc.) in a secure place.

**CI/CD Pipeline**

* + Lint, test, and build steps run on push.
  + Automatic deployment to Vercel or a container-based environment (Docker + ECS/Kubernetes).

## ****Sample Sprint Breakdown for the Frontend****

**Sprint 1**

* + Project scaffolding, Next.js setup, folder structure.
  + Login/Logout flow with Auth.
  + Basic Role-based routing.

**Sprint 2**

* + Firefighter’s “My Interventions” page.
  + Coordinator’s overview dashboard.
  + Integrate real-time updates (Socket.io client).
  + Unit tests for these pages.

**Sprint 3**

* + Logistics dashboard (resource inventory, alerts).
  + Map integration for real-time location.
  + Transcription page + UI for calls.
  + Basic e2e tests with Cypress.

**Sprint 4**

* + Predictive analytics dashboards.
  + Advanced charts, AI insights page.
  + Offline caching / performance optimization.
  + Polish UI, finalize accessibility, i18n if needed.

# ****Conclusion****

With this **frontend plan**, you’ll have a clear roadmap for building a **role-based** user experience in Next.js. Each page or dashboard is tailored to a user’s responsibilities—whether they’re on the front lines as a **firefighter**, overseeing multiple teams as a **coordinator**, or managing supplies as a **logistic officer**. By leveraging Next.js’ SSR (where beneficial), React Query or Redux Toolkit for data fetching and caching, and Socket.io for real-time updates, you’ll ensure a responsive, robust, and maintainable frontend application.