Low Level Design Document

Introduction

This Low Level Design (LLD) document outlines the implementation details for **PulseCare - Real-Time Patient Monitoring Dashboard**. PulseCare is a futuristic healthcare dashboard for real-time patient monitoring, featuring dynamic data visualization, responsive UI, advanced form handling, and live updates via REST APIs. The stack includes React, TypeScript, Tailwind CSS, and Redux Toolkit.

---

1. System Components

Component	Description / Responsibility
PatientList	Displays list of patients and status
PatientDetail	Shows real-time vitals and history for selected patient
VitalsChart	Visualizes live vital signs (charts/graphs)
AlertBanner	Displays critical alerts
PatientForm	Handles patient data entry/editing
Redux Store	Centralized state management
API Service Layer	Handles REST API calls for data fetching/updating
AuthGuard	Protects routes, manages authentication
Layout/ThemeProvider	Manages layout and futuristic theming

---

2. Class/Interface Overview

Name	Type	Key Methods/Attributes	Relationships
`Patient`	Interface	id, name, age, vitals, status	Used in Redux state, API responses
`Vitals`	Interface	heartRate, bp, spo2, timestamp	Nested in Patient
`PatientList`	Component	fetchPatients(), selectPatient()	Uses Redux, API Service
`PatientDetail`	Component	fetchVitals(), renderCharts()	Uses Redux, VitalsChart
`VitalsChart`	Component	render(), updateData()	Receives vitals as props
`APIService`	Class	getPatients(), getVitals(), updatePatient()	Used by Redux thunks

Key Redux Slices:

- `patientsSlice`: manages patient list, selected patient
- `vitalsSlice`: manages real-time vitals data

3. Data Structure Overview

Model	Fields
Patient	id: string, name: string, age: number, status: ‘stable’
Vitals	heartRate: number, bp: string, spo2: number, timestamp: string
Alert	id: string, patientId: string, type: string, message: string, timestamp: string

4. Algorithms / Logic

Real-Time Data Fetching & Update Flow:

```
// Redux thunk for polling vitals
async function pollVitals(patientId: string) {
  while (dashboardActive) {
    const vitals = await APIService.getVitals(patientId);
    dispatch(updateVitals(vitals));
    await sleep(POLL_INTERVAL);
  }
}
```

Alert Generation Logic:

```
if (vitals.heartRate > HR_THRESHOLD || vitals.spo2 < SPO2_THRESHOLD) {
  dispatch(showAlert({ patientId, type: 'critical', ... }));
}
```

5. Error Handling

Scenario	Handling Approach
API failure (fetch/update)	Show error banner, retry with exponential backoff
Invalid form input	Inline validation, disable submit, show messages
Unauthorized access	Redirect to login, clear sensitive state
Data parsing error	Log error, show fallback UI
WebSocket/connection loss	Show offline indicator, attempt reconnection

End of Document