**Minimal Frontend Changes Needed**

Your **frontend structure** (UI components in the components folder) mostly remains the same, but **some small updates** are needed to replace static data with **dynamic API calls**.

**1️⃣ Convert Static Data to API Calls**

In your **list components (StaffList.tsx, BookingList.tsx, etc.)**, replace hardcoded data arrays with API calls.

**Example: Updating StaffList.tsx**

**Before (Static Data)**

javascript

CopyEdit

const staff = [

{ id: 'STF-1001', name: 'Alice Johnson', role: 'Manager', status: 'Active' },

{ id: 'STF-1002', name: 'Bob Smith', role: 'Receptionist', status: 'On Leave' },

];

export function StaffList() {

return (

<div className="table-container">

<table className="table">

<tbody>

{staff.map((member) => (

<tr key={member.id}>

<td>{member.name}</td>

<td>{member.role}</td>

<td>{member.status}</td>

</tr>

))}

</tbody>

</table>

</div>

);

}

**After (Fetching from API)**

javascript

CopyEdit

import { useEffect, useState } from 'react';

export function StaffList() {

const [staff, setStaff] = useState([]);

const [loading, setLoading] = useState(true);

const [error, setError] = useState(null);

useEffect(() => {

fetch('/api/staff') // Adjust the API route based on your backend

.then((res) => res.json())

.then((data) => {

setStaff(data);

setLoading(false);

})

.catch((err) => {

console.error('Error fetching staff:', err);

setError(err);

setLoading(false);

});

}, []);

if (loading) return <p>Loading staff...</p>;

if (error) return <p>Error loading staff</p>;

return (

<div className="table-container">

<table className="table">

<tbody>

{staff.map((member) => (

<tr key={member.id}>

<td>{member.name}</td>

<td>{member.role}</td>

<td>{member.status}</td>

</tr>

))}

</tbody>

</table>

</div>

);

}

✅ **Similar updates should be applied to all other List components (BookingsList.tsx, BillingList.tsx, etc.).**

**2️⃣ Fetch Filter Options from API**

Your filter components (StaffFilters.tsx, RoomFilters.tsx, etc.) need **dynamic data** instead of hardcoded options.

**Example: Updating StaffFilters.tsx**

**Before (Static Options)**

javascript

CopyEdit

<select className="form-input">

<option value="all">All Roles</option>

<option value="Manager">Manager</option>

<option value="Receptionist">Receptionist</option>

</select>

**After (Fetching Roles from API)**

javascript

CopyEdit

import { useEffect, useState } from 'react';

export function StaffFilters() {

const [roles, setRoles] = useState([]);

useEffect(() => {

fetch('/api/staff/roles')

.then((res) => res.json())

.then((data) => setRoles(data))

.catch((err) => console.error('Error fetching roles:', err));

}, []);

return (

<select className="form-input">

<option value="all">All Roles</option>

{roles.map((role) => (

<option key={role} value={role}>{role}</option>

))}

</select>

);

}

✅ **Similar updates should be applied to all Filter components (RoomFilters.tsx, BillingFilters.tsx, etc.).**

**3️⃣ Adjust API Routes in Next.js**

If your **API routes (app/api/...)** were using **mock data**, update them to fetch from **PostgreSQL**.

**Example: Updating app/api/staff/route.js**

**Before (Mock Data)**

javascript

CopyEdit

export async function GET() {

return Response.json([

{ id: 'STF-1001', name: 'Alice Johnson', role: 'Manager', status: 'Active' }

]);

}

**After (Fetching from PostgreSQL)**

javascript

CopyEdit

import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

export async function GET() {

const staff = await prisma.staff.findMany();

return Response.json(staff);

}

✅ **Ensure all API routes (/api/guests, /api/rooms, /api/bookings) connect to PostgreSQL.**

**4️⃣ Minor UI Fixes**

* **Show loading indicators** (useState for loading).
* **Show error messages** (catch() handling).
* **Format data correctly** (e.g., format amount values as currency).

**Final Answer: Are Frontend Changes Needed?**

✅ **Yes, but only for API integration** (replacing static data with API calls).  
❌ **No major structural changes** (your components and UI remain the same).

**Next Steps**

1. **Update list components** (StaffList.tsx, BookingsList.tsx, etc.) to fetch data from the API.
2. **Update filter components** (StaffFilters.tsx, RoomFilters.tsx, etc.) to fetch options dynamically.
3. **Ensure API routes in Next.js (app/api/...) connect to PostgreSQL.**
4. **Test the frontend to ensure data is loading correctly.**

Would you like me to generate **all the necessary API route files** for Next.js? 🚀