Cleanroom HVAC Calculator Application

Overview

Create a web application that replicates the HVAC calculation logic from the Excel file BOD-PB-C-Pilot.xlsx. The system will calculate cleanroom HVAC parameters (CFM, cooling loads, TR, etc.) based on room dimensions and requirements.

Project Structure

Backend Repository (`backend/`)

- **FastAPI** application for REST API
- **SQLAlchemy** ORM for Aurora RDS PostgreSQL
- **Pydantic** models for validation
- Core calculation engine based on Excel formulas

Frontend Repository (`frontend/`)

- **React** with TypeScript
- **Material-UI** or **Tailwind CSS** for styling
- Form handling for room input data
- Results visualization

Docker Setup

- `docker-compose.yml` orchestrating all services
- PostgreSQL for local dev (Aurora RDS for production)
- Hot-reload for development

Implementation Plan

Phase 1: Backend Foundation

1. Project Setup

- Initialize FastAPI project structure
- Set up SQLAlchemy with Aurora RDS connection
- Configure Docker for backend service

- Create database migration system (Alembic)

2. Database Schema

- **rooms** table: Store room calculation inputs
  -
  - id, ahu_no, room_name, length_m, width_m, height_ft
  - air_changes, occupancy, equipment_load_kw
  - class_type, temp_c, humidity_pct
- **motors** table: Motor pricing static data (from Motors sheet)
- **blowers** table: Blower pricing static data (from Blowers sheet)
- **constants** table: Configurable calculation constants

3. HVAC Calculation Engine - Core Logic from BOD-PB-C-Pilot.xlsx

Key calculations to implement:

- **Area** = Length × Width
- **Volume (CFT)** = Area × Height × 10.76
- **Room CFM** = Volume × Air_Changes / 60
- **Fresh Air CFM** = Room_CFM × 0.1
- **Exhaust Air CFM** = Room_CFM × 0 or 1 (based on room type)
- **Resultant CFM** = MAX(Room_CFM, Dehumidification_CFM) × 1.1
- **AC Load (TR)** calculations:
  -
  - Wall/ceiling heat load: (Length+Width) × 2 × 10.76 × Height × 0.4 × ΔT
  - Equipment load, occupancy load, fresh air load
  - Total cooling load in Tons of Refrigeration (TR)
- **Chilled Water Flow** = TR × 2.4 (GPM), × 0.0630902 (L/s)
- **Pipe Sizing** = 0.26 × √(Flow)
- **Temperature conversions**: F = (9×C/5) + 32

4. REST API Endpoints

```
POST /api/rooms/calculate
  - Input: Room parameters (dimensions, class, temp, etc.)
  - Output: All calculated values (CFM, TR, pipe sizes, etc.)
```

```
GET /api/rooms/{id}
```

```
POST /api/rooms
```

```
PUT /api/rooms/{id}
```

```
DELETE /api/rooms/{id}
```

```
GET /api/static-data/motors
```

```
GET /api/static-data/blowers
```

Phase 2: Frontend Development

1. Project Setup

- Initialize React + TypeScript project (Vite or Create React App)
- Set up UI component library (Material-UI recommended)
- Configure Docker for frontend service
- Set up API client (Axios or Fetch)

2. Core Components

Room Input Form:

- Room identification (AHU number, room name)
- Dimensions (length, width, height with unit labels)
- HVAC parameters (air changes, occupancy, equipment load)
- Classification (dropdown: 100K, 10K, NC)
- Temperature and humidity inputs
- Submit button → triggers calculation

Results Display:

- Organized sections:
  -
  - **Basic Calculations**: Area, Volume, CFM values
  - **Cooling Load**: Room load, CFM load, resultant TR
  - **Water System**: Flow rates, pipe sizing
  - **Air System**: Supply CFM, exhaust CFM, fresh air
- Color-coded results (green for within spec, yellow for warnings)
- Export results option (future: PDF/Excel)

3. State Management

- Use React Context or Zustand for state management
- Store current room data, calculation results
- Handle loading/error states

Phase 3: Integration & Docker

1. Docker Configuration

Create three containers:

- **backend**: Python FastAPI app (port 8000)
- **frontend**: React dev server (port 3000) / Nginx (production)
- **database**: PostgreSQL (port 5432)

2. Environment Configuration

- `.env` files for database connection strings
- Separate configs for dev/production (Aurora RDS in prod)

3. Database Seeding

- Create seed script to populate Motors and Blowers tables from Excel
- Use `process_excel.py` as reference to extract data

Phase 4: Testing & Validation

1. Backend Tests

- Unit tests for calculation engine (compare with Excel results)
- API endpoint tests
- Test data: Use rows 41, 47, 53 from BOD sheet as test cases

2. Frontend Tests

- Component tests
- Integration tests for form submission and results display

3. Validation

- Compare app calculations against Excel file for 5-10 sample rooms
- Ensure formula accuracy (±0.1% tolerance)

Technology Stack

Backend

- **FastAPI** (Python 3.10+)
- **SQLAlchemy** 2.0 (ORM)
- **Alembic** (migrations)
- **Pydantic** (validation)
- **psycopg2** (PostgreSQL driver)
- **pytest** (testing)

Frontend

- **React** 18 with TypeScript
- **Material-UI** v5 (components)
- **Axios** (HTTP client)
- **React Hook Form** (form handling)
- **Vite** (build tool)

Infrastructure

- **Docker** & Docker Compose
- **PostgreSQL** 15 (local) / Aurora RDS (production)
- **Nginx** (production frontend serving)

Key Files to Create

Backend

- `backend/app/main.py` - FastAPI app entry point
- `backend/app/models/` - SQLAlchemy models
- `backend/app/schemas/` - Pydantic schemas
- `backend/app/services/hvac_calculator.py` - Core calculation engine
- `backend/app/api/routes/` - API route handlers
- `backend/app/database.py` - Database connection
- `backend/Dockerfile`
- `backend/requirements.txt`

Frontend

- `frontend/src/App.tsx`
- `frontend/src/components/RoomInputForm.tsx`
- `frontend/src/components/CalculationResults.tsx`
- `frontend/src/services/api.ts` - API client
- `frontend/src/types/` - TypeScript interfaces
- `frontend/Dockerfile`
- `frontend/package.json`

Root

- `docker-compose.yml`
- `docker-compose.prod.yml`
- `.env.example`
- `README.md`

Success Criteria

1. User can input room parameters via React UI
2. Backend calculates all HVAC parameters matching Excel logic
3. Results display accurately in frontend
4. Calculations match Excel file within 0.1% tolerance for test cases
5. Application runs in Docker containers
6. Database stores static data (motors, blowers) accessible via API

Future Enhancements (Not in Initial Scope)

- Complete offer generation with pricing
- Admin UI for static data configuration
- Multi-project management
- Excel import/export
- PDF report generation