Pharmaceutical Facility Design Copilot - Development Specification

Project Overview

Create a web-based application called "Design Copilot" that assists pharmaceutical facility designers in creating adjacency diagrams by leveraging a Neo4j knowledge graph containing industry-specific spatial relationships and compliance requirements.

Technical Stack Requirements

- **Frontend**: React with TypeScript
- **UI Framework**: Material-UI or Ant Design
- **Canvas Library**: React Flow or JointJS for diagram creation
- **Backend**: Node.js with Express or NestJS
- **Database**: Neo4j (primary) with neo4j-driver
- **State Management**: Redux Toolkit or Zustand
- **Real-time Updates**: WebSocket (Socket.io)

Core Functionality Requirements

1. Neo4j Knowledge Graph Integration

- Connect to Neo4j database containing:
  - Node types: FunctionalArea, Equipment, Utility, ComplianceRule
  - Relationship types: ADJACENT_TO, REQUIRES_ACCESS, PROHIBITED_NEAR, SHARES_UTILITY
  - Properties: cleanroom_class, material_flow, personnel_flow, hazard_level
- Implement efficient Cypher queries for:
  - Fetching available node types
  - Getting adjacency recommendations
  - Validating placement rules
  - Checking compliance constraints

2. Interactive Canvas Features

- Implement drag-and-drop functionality for placing functional areas
- Create a responsive grid/snap system for precise alignment
- Display nodes as customizable rectangles with:
  - Name labels
  - Color coding by function type
  - Size indicators (square footage)
  - Cleanroom classification badges
- Enable pan, zoom, and viewport controls

3. Intelligent Suggestion System

- When a node is placed or selected, query Neo4j for:

  ```
  MATCH (selected:FunctionalArea {id: $nodeId})MATCH (selected)-
  [r:ADJACENT_TO|REQUIRES_ACCESS]->(suggested:FunctionalArea)WHERE NOT
  exists((placed:FunctionalArea)-[:PLACED_IN_DIAGRAM]->(suggested))RETURN
  suggested, r.priority, r.reasonORDER BY r.priority DESC
  ```

- Display suggestions as semi-transparent "ghost" nodes
- Show connection lines with different styles for relationship types
- Provide tooltips explaining why connections are recommended

4. Validation and Compliance Engine

- Real-time validation against:
  - GMP requirements (material/personnel flow separation)
  - Cleanroom classification transitions
  - Hazardous material handling rules
  - Cross-contamination prevention
- Display violations as red highlights with explanatory messages
- Prevent invalid placements with clear user feedback

5. User Interface Components

- **Left Panel**: Searchable node palette organized by categories
  - Production Areas (Weighing, Granulation, Compression, Coating)
  - Quality Control (Analytical Labs, Microbiology, Stability Chambers)
  - Warehouse (Raw Materials, Finished Goods, Quarantine)
  - Utilities (HVAC, Purified Water, Compressed Air)
  - Personnel (Gowning, Break Rooms, Offices)
- **Top Toolbar**: Save, Load, Export (PDF/DWG), Validate, Templates
- **Right Panel**: Properties editor for selected nodes
- **Bottom Panel**: Compliance warnings and suggestions

6. Advanced Features

- **Auto-layout**: Implement force-directed or hierarchical layout algorithms
- **Flow Visualization**: Overlay material/personnel flow paths
- **Version Control**: Track diagram changes with Neo4j temporal features
- **Collaboration**: Multi-user editing with conflict resolution

Data Model Examples

Neo4j Node Structure

```
// FunctionalArea Node
{
  id: "gowning-area-01",
  type: "FunctionalArea",
  name: "Gowning Area",
  category: "Personnel",
```

```
  cleanroom_class: "D",
  min_size_sqm: 20,
  max_size_sqm: 50,
  requires_utilities: ["hvac", "compressed_air"]
}

// Relationship Example
(:FunctionalArea {name: "Gowning Area"})-[
  :ADJACENT_TO {
    priority: 10,
    reason: "Personnel flow control",
    door_type: "airlock"
  }
]->(:FunctionalArea {name: "Production Corridor"})
```

API Endpoints Design

```
// Core endpoints
GET /api/nodes/categories - Get all available node categories
GET /api/nodes/:category - Get nodes by category
POST /api/diagram/validate - Validate current diagram
GET /api/suggestions/:nodeId - Get suggestions for a specific node
POST /api/diagram/save - Save diagram to database
GET /api/templates - Get available templates
```

Performance Considerations

- Implement query result caching for frequently accessed nodes
- Use GraphQL for efficient data fetching
- Implement virtual scrolling for large diagrams
- Optimize Canvas rendering with React.memo and useMemo
- Batch Neo4j queries where possible

Testing Requirements

- Unit tests for graph queries and validation logic
- Integration tests for Neo4j connection
- E2E tests for critical user workflows
- Performance tests for large diagrams (100+ nodes)

Deliverables

1. Fully functional React application
2. Neo4j database schema and sample data
3. API documentation
4. User guide with pharma-specific examples
5. Deployment instructions for cloud platforms