Service Catalog Development Assignment

Full Stack Developer (1 Year Experience)

Project Overview

Develop a service catalog application based on the Backstage system model that enables organizations to manage and discover their software components, APIs, resources, and organizational structure.

System Architecture Requirements

Backend Stack

- **FastAPI** - REST API framework
- **SQLite** - Database for development/testing
- **Alembic** - Database migration tool
- **SQLAlchemy** - ORM for database operations

Frontend Stack

- **ShadCN/UI** - Component library (https://ui.shadcn.com/docs/installation/vite)
- **Vite** - Build tool and development server
- **Tailwind CSS** - Utility-first CSS framework

Core Entities Implementation

Based on the provided Backstage model, implement the following entities:

1. **Domain (Orange)**

- Represents business domains, models, metrics, KPIs
- **Fields**: `id`, `name`, `description`, `owner_id`, `parent_domain_id`
- **Relationships**: Hierarchical structure (parent-child)
- **Types**: Business domain categories

2. **System (Yellow)**

- Collection of entities that cooperate to perform a function
- **Fields**: `id`, `name`, `description`, `domain_id`, `owner_id`
- **Relationships**: Belongs to Domain, contains Components

3. **Component (Green)**

- Individual software components (backend services, data pipelines, websites, libraries)
- **Fields**: `id`, `name`, `description`, `system_id`, `component_type`, `owner_id`
- **Types**: `service`, `website`, `library`
- **Relationships**: Belongs to System, consumes/provides APIs

4. **API (Green)**

- Interfaces between components
- **Fields**: `id`, `name`, `description`, `api_type`, `provider_component_id`
- **Types**: `openapi`, `asyncapi`, `graphql`, `grpc`
- **Relationships**: Provided by Components, consumed by Components

5. **Resource (Green)**

- Infrastructure and data resources
- **Fields**: `id`, `name`, `description`, `resource_type`, `component_id`
- **Types**: `database`, `s3-bucket`, `cluster`
- **Relationships**: Depends on Components

6. **Group (Blue)**

- Organizational structure
- **Fields**: `id`, `name`, `description`, `group_type`, `parent_group_id`
- **Types**: `team`, `business-unit`, `product-area`, `root`
- **Relationships**: Hierarchical structure

7. **User (Blue)**

- Individual users in the system
- **Fields**: `id`, `name`, `email`, `groups[]`
- **Relationships**: Member of Groups

8. **Location & Template**

- **Location**: References to external catalog data
- **Template**: Parameters for scaffolding processes

Functional Requirements

1. **Hierarchical Group Filtering**

- Header dropdown showing hierarchical group structure

- Filter all entities based on selected group ownership
- Cascade filtering from parent to child groups

2. **Entity Management Interface**

- Left navigation with all system entities
- Entity listing pages with search and filter capabilities
- Detailed entity pages with relationships visualization
- CRUD operations for all entity types

3. **Actions & Operations**

- Create new entities from detail pages
- Edit existing entities
- Delete entities (with dependency checking)
- View entity relationships and dependencies

4. **Discovery Features**

- Search across all entities
- Filter by entity type, owner, tags
- Relationship browsing (depends on, part of, owned by)

Technical Implementation Tasks

Phase 1: Backend Development (Weeks 1-3)

Database Schema Design

```sql
-- Implement tables for all entities with proper relationships
-- Foreign keys for hierarchical structures
-- Junction tables for many-to-many relationships
-- Indexes for performance optimization
```

FastAPI Application Structure

```
app/
 ├── models/        # SQLAlchemy models
 ├── schemas/       # Pydantic schemas
 ├── api/           # API endpoints
│   ├── domains.py
│   ├── systems.py
│   ├── components.py
│   ├── groups.py
│   └── users.py
 ├── crud/          # Database operations
 ├── core/          # Configuration
 └── main.py        # FastAPI app
```

API Endpoints Required

- **Groups**: `GET /groups` (hierarchical), `POST /groups`, `PUT /groups/{id}`, `DELETE /groups/{id}`
- **Domains**: Full CRUD with hierarchy support
- **Systems**: CRUD with domain relationships
- **Components**: CRUD with system relationships and filtering
- **APIs**: CRUD with component relationships
- **Resources**: CRUD with dependency management
- **Users**: User management and group membership

Phase 2: Frontend Development (Weeks 4-6)

Component Structure

```
src/
 ├── components/
│   ├── ui/          # ShadCN components
│   ├── layout/      # Header, Navigation, Layout
│   ├── entities/    # Entity-specific components
│   └── common/      # Shared components
 ├── pages/          # Route components
 ├── hooks/          # Custom React hooks
 ├── lib/            # Utilities and API client
 └── types/          # TypeScript interfaces
```

Required Pages

1. **Dashboard** - Overview of all entities
2. **Entity Listing Pages** - For each entity type

3. **Entity Detail Pages** - Individual entity views
4. **Entity Creation/Edit Forms** - CRUD operations
5. **Search/Discovery Page** - Global search interface

Key Features Implementation

- **Header Component**: Group dropdown with hierarchical display
- **Left Navigation**: Entity type navigation with counts
- **Entity Cards**: Consistent display across all entity types
- **Relationship Visualization**: Component dependencies and connections
- **Form Components**: Dynamic forms for entity creation/editing

Phase 3: Integration & Polish (Weeks 7-8)

Advanced Features

- Real-time filtering based on group selection
- Entity relationship graphs
- Bulk operations
- Export functionality
- Advanced search with filters

Performance Optimization

- Implement pagination for large datasets
- Add loading states and error handling
- Optimize database queries with proper joins
- Frontend caching strategies

Acceptance Criteria

Functional Requirements

- [ ] All entity types can be created, read, updated, and deleted
- [ ] Hierarchical group filtering works across all entities
- [ ] Entity relationships are properly maintained
- [ ] Search and discovery features are fully functional
- [ ] Responsive design works on desktop and tablet

Technical Requirements

- [ ] Backend API follows RESTful conventions
- [ ] Database migrations are version controlled

- [ ] Frontend components follow ShadCN design patterns
- [ ] TypeScript is used throughout the frontend
- [ ] Error handling is implemented across all operations

Code Quality

- [ ] Code is well-documented with comments
- [ ] Consistent naming conventions
- [ ] Proper error handling and validation
- [ ] Basic unit tests for critical functions

Deliverables

1. **Complete FastAPI backend** with all endpoints
2. **SQLite database** with sample data
3. **React frontend** with ShadCN components
4. **Database migration scripts**
5. **README** with setup and running instructions
6. **API documentation** (FastAPI auto-generated)

Additional Notes

- Focus on clean, maintainable code over complex features
- Ensure the UI closely matches ShadCN design patterns
- Implement proper TypeScript interfaces for all data structures
- Consider using React Query for API state management
- Add basic authentication if time permits