Overview

The goal of this project is to offer a solution for customers to manage their own internal developments in our catalog.

1. Database Structure

- **Customers Collection**

```
{
  "_id": ObjectId("customer_id"),
  "name": "Customer A",
  "email": "foo@example.com",
}
```

- **Fields Collection**

```
{
  "id": ObjectId("field_id"),
  "customerId": ObjectId("customer_id"),
  "name": "projectName",
  "type": "String",
  "required": true,
}
```

- **CatalogEntries Collection**

• Stores the actual entries for each customer's catalog.

```
{
  "_id": ObjectId("entry_id"),
  "customerId": ObjectId("customer_id"),
  "fields": [
    {field_id: ObjectId("field_id"), value: "foo"},
```

```
    {field_id: ObjectId("field_id"), value: "bar"},
    {field_id: ObjectId("field_id"), value: false},
    ...
  ]
}
```

2. Middleware & Microservices

a. Middleware

The middleware serves as a layer between the client and the backend microservices. It will handle data transformation, authentication, request validation, logging and error handling. The middleware is implemented at the API Gateway level.

b. Customer Microservice

• **Purpose**: Manages customers.

• **Key APIs:**

POST /customers/{customerId}: Creates a new customer.
GET /customers: Retrieve a list of all customers.
GET /customers/{customerId}: Fetch a specific customer.
PUT /customers/{customerId}: Update customer details.
DELETE /customers/{customerId}: Remove a field.

c. Field Microservice

• **Purpose:** Manages customer-specific dynamic fields.

• **Key APIs:**

POST /customers/{customerId}/fields: Creates a new field for a customer.
GET /customers/{customerId}/fields: Retrieve all fields for a customer.
GET /customers/{customerId}/fields?field=:fieldId: Fetch a specific field.
PUT /customers/{customerId}/fields?field=:fieldId: Update field details.
DELETE /customers/{customerId}/fields?field=:fieldId: Remove a field.

d. Catalog Microservice

• **Purpose:** Handles catalog entries.

• **Key APIs:**

POST /customers/{customerId}/catalog: Add a new catalog entry.
GET /customers/{customerId}/catalog: Fetch all catalog entries.
GET /customers/{customerId}/catalog?entryId=:entryId: Fetch a specific catalog entry.
PUT /customers/{customerId}/catalog?entryId=:entryId: Update a catalog entry.
DELETE /customers/{customerId}/catalog?entryId=:entryId: Delete a catalog entry.

3. UI Approach

a. Dynamic Form for Catalog Entries

• **Form Renderer:** The UI would dynamically render forms based on the field structure retrieved from the backend for the specific customer. Fields will be dynamically created using Angular's FormBuilder based on the fields.

b. Catalog Entry Listing

• A table or grid displaying catalog entries based on dynamic, customer-specific fields.

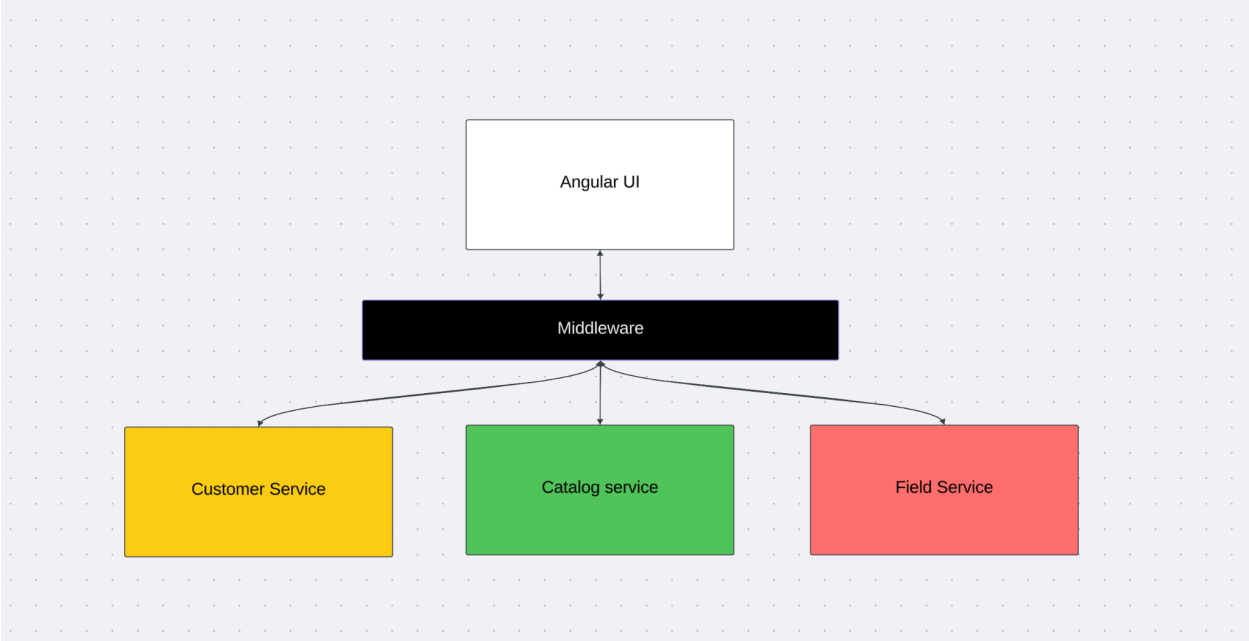
c. Add / Update Entry

• The form to add or update a catalog entry would dynamically create input fields as per the field list defined for that customer.

d. Field Management

• UI for customers to manage their fields. They can view, add, update, or remove fields from their dynamic form.

4. Relationships Between Components



• **Customer Service:** Provides customer data to middleware which will be used to find fields and catalog entries, in addition it will be used in the UI

• **Field Service:** Provides dynamic form structures to be used with the UI and catalog service.

• **Catalog Service:** Handles catalog data specific to the dynamic fields provided by the Field Service.