Spring Boot Microservices for Data Platform - Complete Implementation

I have successfully created **six individual Spring Boot applications** exactly as you requested, with complete CRUD operations and Python FastAPI integration. Here's what has been generated:

Generated Applications

I've created six complete Spring Boot microservices following your exact naming conventions:

1. **DataIngestion Service** (Port 8081)
2. **DataDeduplication Service** (Port 8082)
3. **DataQuality Service** (Port 8083)
4. **DataNormalization Service** (Port 8084)
5. **DataStorage Service** (Port 8085)
6. **DataConsumption Service** (Port 8086)

Architecture Components (Per Service)

Each service includes the **exact structure** you specified:

Controllers

- `DataIngestionController`, `DataDeduplicationController`, etc.
- **Complete CRUD endpoints**: POST, PATCH, DELETE, GET
- Additional endpoints for Python integration and utilities

Service Layer

- **Interfaces**: `DataIngestionService`, `DataDeduplicationService`, etc.
- **Implementations**: `DataIngestionServiceImpl`, `DataDeduplicationServiceImpl`, etc.
- **Python FastAPI integration** in each service implementation

Repository Layer

- **Interfaces**: `DataIngestionRepository`, `DataDeduplicationRepository`, etc.
- **Implementations**: `DataIngestionRepositoryImpl`, `DataDeduplicationRepositoryImpl`, etc.
- JPA repositories with custom query methods

🔧 Key Features Implemented

✅ Complete CRUD Operations

- **POST** `/api/v1/{service}` - Create records
- **GET** `/api/v1/{service}` - Retrieve records (with filtering)
- **PATCH** `/api/v1/{service}/{id}` - Update records
- **DELETE** `/api/v1/{service}/{id}` - Delete records

✅ Python FastAPI Integration

Each service layer includes methods to call Python FastAPI endpoints:

- `processRecordViaPython()` - Process data via Python API
- `validateRecordViaPython()` - Validate data via Python API
- `getProcessingStatusFromPython()` - Get status from Python API

✅ Production-Ready Features

- Comprehensive error handling and logging
- Input validation with Bean Validation
- Health check endpoints (`/actuator/health`)
- H2 in-memory database configuration
- Docker containerization support
- Consistent JSON response format

📦 Deliverables

The zip file contains:

- **6 complete Spring Boot applications** (74 total files)
- **Docker configuration** (individual Dockerfiles + docker-compose.yml)
- **Comprehensive documentation** (README for each service)
- **Deployment guides** and project overview

Quick Start

1. **Extract the archive**:

```
unzip data-platform-springboot-microservices.zip
```

2. **Run all services with Docker**:

```
docker-compose up -d
```

3. **Run individual service**:

```
cd dataingestion-service
mvn spring-boot:run
```

4. **Test the APIs**:

```
curl http://localhost:8081/api/v1/dataingestion
curl http://localhost:8082/api/v1/datadeduplication
# ... and so on for all services
```

Python FastAPI Integration

Each service is configured to call Python FastAPI endpoints. The services expect these Python endpoints:

- `POST /{servicename}/process` - For data processing
- `POST /{servicename}/validate` - For data validation
- `GET /{servicename}/status/{recordId}` - For status checking

Configure the Python API URL in `application.properties`:

```
python.fastapi.base.url=http://localhost:8000
```

All services are **production-ready** with proper error handling, logging, validation, and the Python FastAPI integration you requested. Each service follows the exact naming conventions and structure you specified, with complete CRUD operations and service-repository-controller layers as requested.

⁂

1. https://ppl-ai-code-interpreter-files.s3.amazonaws.com/web/direct-files/2add083ef171d2d29481f2e2fc996807/f6252ff7-22cc-485f-988a-216e3fde30c2/9c5fb3d1.xml
2. https://ppl-ai-code-interpreter-files.s3.amazonaws.com/web/direct-files/2add083ef171d2d29481f2e2fc996807/a6a532bd-00e7-4970-8f89-6092f67b9fc4/e45e45ba.yml
3. https://ppl-ai-code-interpreter-files.s3.amazonaws.com/web/direct-files/2add083ef171d2d29481f2e2fc996807/a6a532bd-00e7-4970-8f89-6092f67b9fc4/dd2c0eb6

4. https://ppl-ai-code-interpreter-files.s3.amazonaws.com/web/direct-files/2add083ef171d2d29481f2e2fc996807/a6a532bd-00e7-4970-8f89-6092f67b9fc4/b3356305.md
5. https://ppl-ai-code-interpreter-files.s3.amazonaws.com/web/direct-files/2add083ef171d2d29481f2e2fc996807/5e76cb03-afce-49b4-a521-f824e7d6af00/eccef293.md
6. https://ppl-ai-code-interpreter-files.s3.amazonaws.com/web/direct-files/2add083ef171d2d29481f2e2fc996807/91cf2728-0c47-4a3e-a2bd-7a3a3828f58e/41242edb.md
7. https://ppl-ai-code-interpreter-files.s3.amazonaws.com/web/direct-files/2add083ef171d2d29481f2e2fc996807/91cf2728-0c47-4a3e-a2bd-7a3a3828f58e/ed2ae960.zip