Java Development Test - Integration Service

In this development test you will be building an integration service that acts as a simple gateway between the DHIS2 web API and an unknown third-party system.

Functional Requirements

The integration service should fetch metadata from a DHIS2 instance and make it available through its own API endpoints. You may utilize the public DHIS2 demo at https://play.dhis2.org/2.34.1. Username/password is admin/district.

1. The DHIS2 instance for which to read metadata from should be configurable in a **application.properties** file (from *Spring Boot*) and default to:

```
dhis2.username = admin
dhis2.password = district
dhis2.url = https://play.dhis2.org/2.34.1
```

2. The service should fetch two types from DHIS2, data elements, and data element groups (feel free to add more properties):

```
/api/29/dataElements.json?paging=false&fields=id,displayName,dataElementGroups[id]
/api/29/dataElementGroups.json?paging=false&fields=id,displayName,dataElements[id]
```

3. You should define your own data format for internal storage, and add REST-style endpoints that expose this format (*JSON* support is required; *XML* support is encouraged).
4. Add security using *Spring Security* and a simple in-memory username/password table (usernames/passwords can be hardcoded); alternatively use Hibernate for persisting users.

Non-functional Requirements

1. Test the service with mocked payloads using Spring Test. Write at least one integration test which verifies that the integration service works correctly. For this test the upstream DHIS2 instance should be mocked, i.e. the component which retrieves metadata from the DHIS2 instance should return “fake” objects. You might use the *Mockito* library for mocking. Ensure that the integration service is written in a test-friendly way.

Technical Requirements

You should develop the service using the following technologies:

1. *Maven* as the build tool.
2. *Spring Boot* 2+ as the application framework.
3. *Jackson* to deserialize JSON payloads from DHIS2 and to serialize output through endpoints.
4. *Spring Security* for securing the endpoints.
5. Metadata from DHIS2 should be persisted in the service (in-memory is acceptable using *Caffeine* or similar caches/stores), and fetched with a fixed interval from the DHIS2 instance.
6. Expose two endpoints from your service:

    a) **/api/dataElements** with response payload:

```
{
  "id": "id",
  "name": "name",
  "groups": ["group1", "group2"]
}
```

    b) **/api/dataElementGroups** with response payload:

```
{
  "id": "id",
  "name": "name",
  "members": ["dataElement1", "dataElement2"]
}
```

7. Endpoint formats should be resolved by looking at *extension* first, then *Accept* header (a sensible default should be configured for Accept header "*/*").

Optional Technical Requirements

1. Document endpoints using *Spring Rest Docs* or similar frameworks (such as *Swagger*).

Delivery

The service source code should be made available in a GitHub repository. The repository should be shared with your contact person.