Feed API

Technology Stack

- Java RESTful API - Jersey Implementation
- MySql - Database
- Application Server - Apache Tomcat
- Build Tool - Maven
- API tester : Google chrome plugin “Postman” (I had used this. Feel free to use any other client)

How To Run -

1. Build the project using Maven
2. Database connectivity - Create the database (MySQL).
    a. Create schema/database named : “feed_api”
    b. Use the sql script placed in “sql” directory to create table, add foreign keys, and add few records.
    c. Change the configuration (url, username, password) in class “ConnectionManager” in “feed.util” package.

   P.S - I was trying to create MySQL database on Google cloud. I have $150 student credit. But there was some issue connecting Google cloud from my local machine. So I was not not able to test my application.
3. Start the application server and test the API’s.

Project Architecture

1. Controller layer : in “feed.controller” package - all endpoints are defined. One class per resource
2. Service layer : in “feed.service” package all service layer/business logic has been implemented.
3. DAO layer : in “feed.dao” all database query related classes.
4. Bean : All POJO are defined in “feed.beans” package
5. Util : common project related stuff e.g connection manager is defined in this package.

Service is injected in Controller and DAO is injected in Service. (As of now Spring and Hibernate are not used.)

Controller, DAO and Service are Singleton.

API’s

There are 4 endpoints created as per assignment description.

Base url : http://localhost:8090/FeedAPI/
Here -
8090 : Apache Tomcat Port
FeedAPI - Application name.

Endpoints

1. Add Articles to a Feed
    a. Method Type : POST
    b. URL - http://localhost:8090/FeedAPI/articles/
    c. POST data (application/json):

```
{
        "content": "test-6",
        "feedId": 1,
        "author": "av",
        "title": "Test-6"
}
```

    d. Sample response:

```
HTTP Status: 201 Created
{
   "success": "Article created.",
   "status": true
}
```

If feedId is not valid, the following response is returned:

```
HTTP Status: 500 Internal Server Error
{
   "error": "Something went wrong",
   "status": false
}
```

P.S.: This could also be implemented as
http://localhost:8090/FeedAPI/feeds/{feedId}/articles

2. Get Articles from the set of Feeds a Subscriber is following
    a. Method Type : GET
    b. URL - http://localhost:8090/FeedAPI/articles/user/{userId}

c. Sample GET request:

```
http://localhost:8090/FeedAPI/articles/user/1
```

d. Sample response:

```
HTTP Status: 200 OK
{
   "articles": {
      "Technology": [
         {
            "id": 1,
            "title": "Test",
            "content": "test",
            "author": "av"
         },
         {
            "id": 2,
            "title": "Test-2",
            "content": "test-2",
            "author": "av"
         },
         {
            "id": 7,
            "title": "Test-6",
            "content": "test-6",
            "author": "av"
         }
      ]
   },
   "status": true
}
```

If no content is available for an user, then the HTTP Status returned is: 204 No Content

As the number of feeds and articles can grow, it is better to use Async response. As of now it is using Synchronous response. But this can should be implemented using Async response.

3. Subscribe a User to a Feed
    a. Method Type : POST

b. URL - http://localhost:8090/FeedAPI/subscribe/
c. Sample POST request:

```
{
        "feedId": 1,
        "userId": 3
}
```

d. Sample response:

```
HTTP Status: 201 Created
{
   "success": "User is subscribed",
   "status": true
}
```

If feedId or userId is not valid, the following response is returned:

```
HTTP Status: 500 Internal Server Error
{
   "error": "Something went wrong",
   "status": false
}
```

4. Unsubscribe a User to a Feed
   a. Method Type : PUT
   b. URL - http://localhost:8090/FeedAPI/unsubscribe/
   c. Sample PUT request:

```
{
        "feedId": 3,
        "userId": 2
}
```

   d. Sample response:

```
HTTP Status: 200 OK
{
    "success": "User is unsubscribed",
    "status": true
}
```

If feedId or userId is not valid, following response is returned:

```
HTTP Status: 500 Internal Server Error
{
    "error": "Something went wrong",
    "status": false
}
```

5. Get all Feeds a Subscriber is following
    a. Method Type : GET
    b. URL - http://localhost:8090/FeedAPI/feeds/user/{userId}
    c. Sample GET request:

```
http://localhost:8090/FeedAPI/feeds/user/1
```

    d. Sample response:

```
HTTP Status: 200 OK
{
    "feeds": [
        {
            "id": 1,
            "name": "Technology"
        },
        {
            "id": 2,
            "name": "Arts"
        }
    ],
    "status": true
}
```

If userId is invalid, then the HTTP Status returned is: 204 No Content

Areas of Improvement

1. Exception Handling and HTTP status code : As of now when any error occurs - http error code 500 is sent. Also at Java layers no custom exceptions are defined. This can be implemented.
2. Code Style - as of now, error message or success messages are hard coded in Java classes wherever required. But this must be moved to properties or xml or other constant file where it can be re-used.
3. Handle duplicate requests.
4. Use of Audit fields like created by, modified by, created date, and modified date.

Future Scope :

1. UI - where user can login
2. Authorization, Authentication for each request
3. PUSH notification : Daily email feed to user.
4. Push events - when any new article is added, push notification can be sent to user.

PS : I haven't implemented this before. But during my internship at Salesforce I was supposed to work on streaming API's and Push Notifications.
https://developer.salesforce.com/docs/atlas.en-us.api_streaming.meta/api_streaming/intro_stream.htm

I had started reading stuff but due to other prioritize task, scope of this functionality was decommissioned.

5. Use Async rest API for larger request.
6. Use of Spring and Hibernate for dependency/beans management, connection/database management.