Newsservice REST API v1.0

This document shows the architecture of the Newsservice API REST – the Controller and HTTP response codes, configuration of payload marshalling and content negotiation.

Building a back-end API introduces a whole new layer of coordination between server and client code. While there are many aspects to this delicate dance of communication, one key ingredient to minimizing back-and-forth-confusion-about what-call-does-what, is consistently communicating about the API endpoints.

1. User Roles and Privileges

First, let's start with our entities. We have three main entities regarding the Security and Roles management in this API:

- the *User*
- the *Role* – this represents the high-level roles of the user in the system; each role will have a set of low-level privileges
- the *Privilege* – represents a low-level, granular privilege/authority in the system

Here's the user:

```
5   public class User {
6
7       private int id;
8       private String username;
9       private String firstName;
10      private String lastName;
11      private String email;
12      private Role role;
13
```

The user contains the roles, but also a few additional details that are necessary for a proper registration mechanism.

Next – here's the role:

```java
import java.util.List;

public class Role {

    private int id;
    private String name;
    private List<Privilege> privileges;
```

And finally the privilege:

```java
package news;

public class Privilege {

    private int id;

    private String name;

    public Privilege(int id, String name) {
        this.id = id;
        this.name = name;
    }
```

We are considering both the User <-> Role as well as the Role <-> Privilege relationships many-to-many bidirectional.

2. News data

Newsservice REST API v1 is not working with a database, instead it is working with Mocked Data. To simulate the database functionality it is using the following classes:

- UsersMockedData
- RolesMockedData
- PrivilegesMockedData
- PicturesMockedData
- NewsMockedData

These classes are located in the package called “data”. They contain a list of hard-coded values. The classes must be a singleton so that changes can be persisted across different HTTP requests. A singleton is a design pattern used in object-oriented programming to permit no more than one instance of a class.

In other words, if a class is a singleton, we can only have one instance of that class throughout the application lifecycle of the Newsservice.

A singleton is implemented in Java by creating a static method that returns an instance of that class.

The static method checks if the class is already instantiated and return the existing instance, otherwise it creates a new instance of that class and returns it.

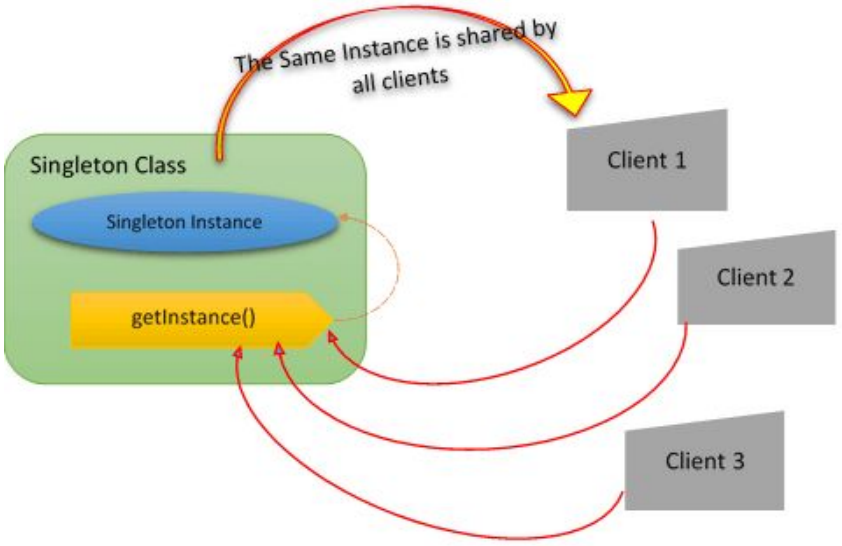


Figure 1: Singleton Design Pattern Overview

So, whenever we need an instance of NewsMockedData, we instantiate it like this:

```
NewsMockedData newsMockedData = NewsMockedData.getInstance();
```

3. NewsController & REST Endpoints

Controllers typically do the following things - they receive input, and generate output.
In Spring, a controller class, which is capable of serving REST API requests, is called rest controller. It should be annotated with @RestController annotation. **Newsservice REST API v1** has one Controller and it is the *NewsController* located in package *news.* The resource Uris are specified in @RequestMapping annotations.

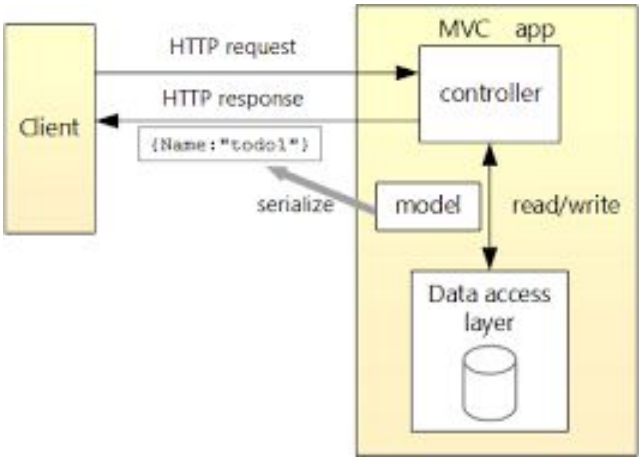


Figure 2: High level architecture of REST API

Newsservice API v1 has the following REST endpoints:

1. URL /actualNews
Gets all unread news articles, allowed for READER, ADMIN and PUBLISHER
Method type - **GET**
Note: If a user makes such request the news articles are not yet been marked as "Read" for him. Only if the user requests a particular news Article, then it is marked for him as "read" and it will not be shown when /actualNews is being requested.

2. URL /newsArticle/{id}
Method type - **GET**
Gets news article by id, allowed for READER, ADMIN and PUBLISHER
After user reads article it's read status is updated and it will not appear
in the bulk news section accessible through /newsArticle

3. URL /newsArticle/{id}
Method type - **POST**
Create/publish newsArticle, only ADMIN and PUBLISHER are allowed to

4. URL /newsArticle/{id}
Method type - **PUT**
Edit newsArticle, only ADMIN and PUBLISHER are allowed to

5. URL /newsArticle/{id}
Method type - **DELETE**
Delete newsArticle, only ADMIN and PUBLISHER are allowed to

6. URL /setReadStatus/{newsId}
Method type - **PUT**
Set Read Status of newsArticle, only ADMIN and PUBLISHER are allowed to

7. URL /picture/{pictureId}
Method type - **GET**
Returns a picture for a single news Article, READER, ADMIN and PUBLISHER are allowed

8. URL /permittedRoles/{roleId}
Method type - **POST**
Creates a new Role, only ADMIN is allowed to

4. Security

The security of **Newsservice REST API v1** is controlled by *WebSecurityConfig.java* using Basic Authentication. Basic authentication sends a Base64-encoded string that contains a

user name and password for the client. Base64 is not a form of encryption and should be considered the same as sending the username and password in clear text.

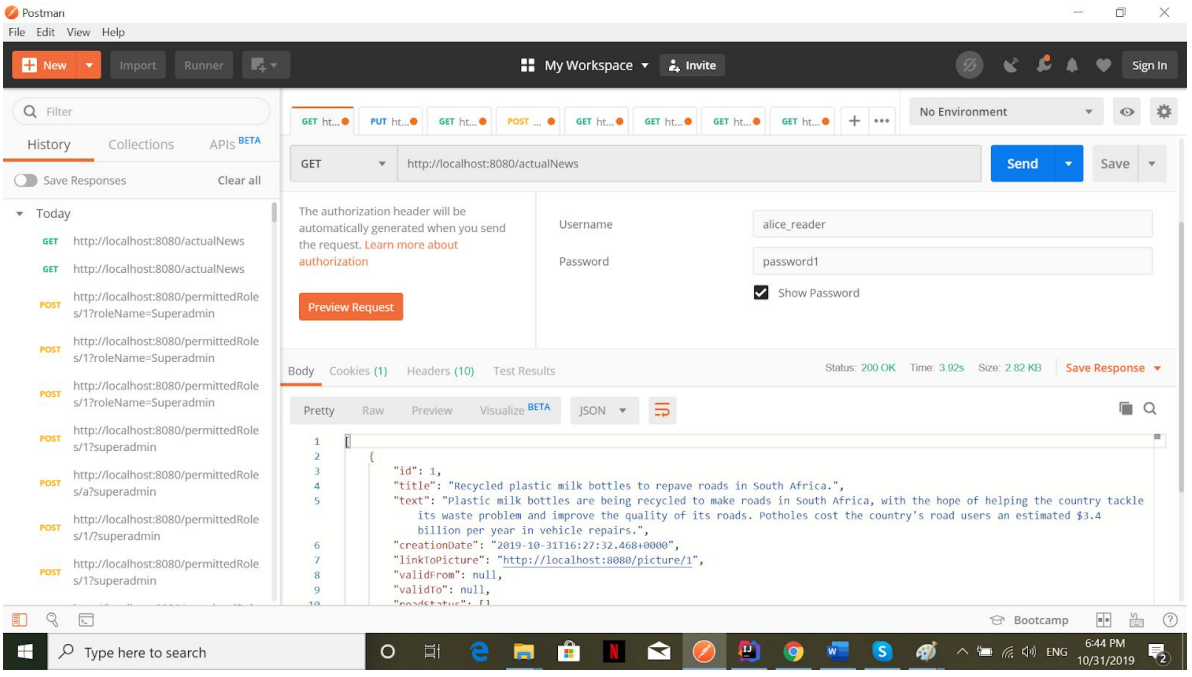
Below are the credentials for the existing 4 Users and their corresponding Roles:

Username	Password	Role
alice_reader	password1	READER
judy_pub	password2	PUBLISHER
kev_admin	password3	ADMIN
richmc	password4	READER

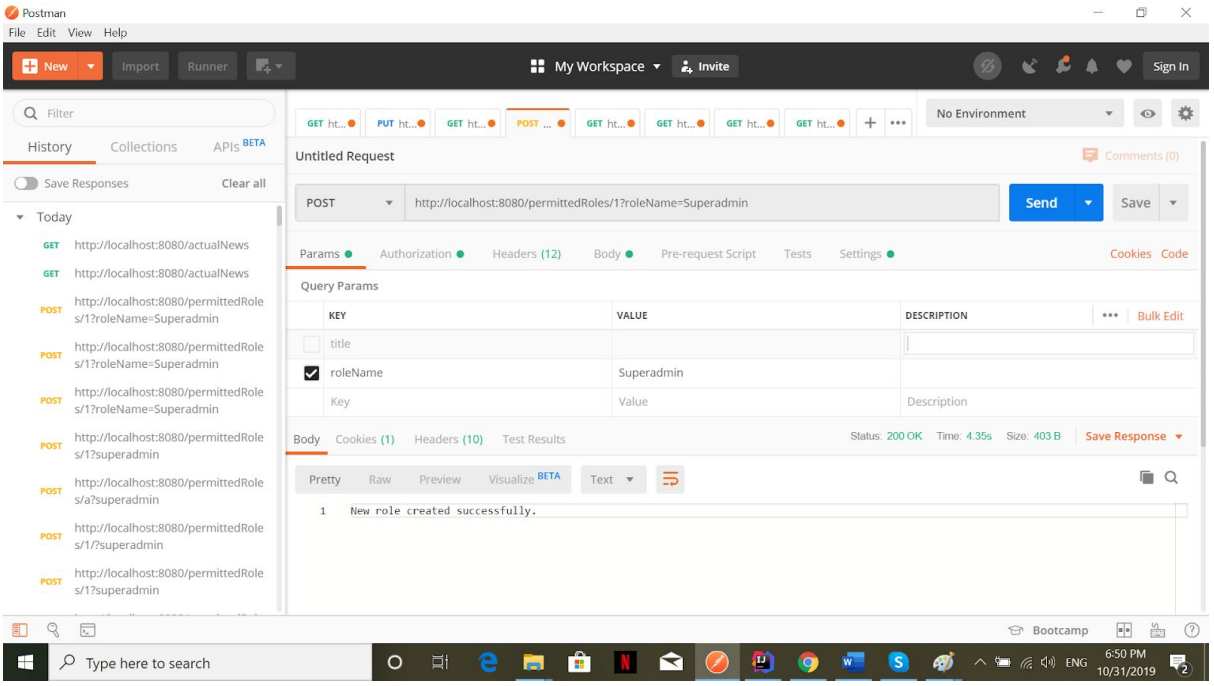
!!!Note: Make sure to send requests to the API with the correct parameters and the correct user credentials, otherwise you will get 401 Unauthorized response message.

Below are some examples of requests and responses sent using Postman:

GET http://localhost:8080/actualNews



POST http://localhost:8080/permittedRoles/1?roleName=Superadmin



GET http://localhost:8080/picture/1