German International University
Faculty of Engineering
Department of Media Engineering and Technology

Architecture of Massively Scalable Applications, Spring 2025
Task 3

Due by Tuesday 15.04.2025 @11:59 PM

1 Introduction

The objective of this task is to design and implement a database schema using NoSQL (MongoDB) and integrate it with a Spring Boot application. The schema will include two main entities: **User** and **Post**. You will establish the appropriate relationships between these entities and implement the necessary methods in the service layer. You will also load balance the application using NGINX and load test it using JMeter.

Please accept this assignment through this link: `https://classroom.github.com/a/S0H0PKv4`.

You should then clone the created repository on your device.

2 Pre-setup

Make sure you don't have any service running on port 27017 so that it does not introduce errors while connecting with MongoDB. A detailed pdf on the CMS contains detailed steps for Windows users.

3 Collections

3.1 Users

- String id (Primary Key)
- String username
- String email

3.2 Post

- String id (Primary Key)
- String title
- String content
- User author (using ***Document Referencing***)
- List of Comments (using ***Document Embedding***)

3.3 Comment

- String text
- String date

4 Repositories

4.1 User Repository

You are required to create a User Repository that extends `MongoRepository`.

4.2 Post Repository

You are required to create a Post Repository that extends `MongoRepository`.

5 Services

5.1 User Service

The User Service contains several methods that you can use for basic testing (basic CRUD operations). Additionally, you should implement the following method:

5.1.1 Update Username of a User

```
public User updateUserUsername(String id, String username) {}
```

This method should update the username of a specific user by taking as input the user ID and the new username. (*Hint: You can refer to the method in the lab 4 manual, or implement an alternative approach.*) You should handle if the user does not exist and throw a not found response with the message "`User not found`".

5.2 Post Service

The Post Service contains several methods that you can use for basic testing (basic CRUD operations). Additionally, you should implement the following methods:

5.2.1 Get Posts by Author ID

```
public List<Post> getPostsByAuthorID(String authorID) {}
```

This method should return a list of all posts of a specific user specified by taking as an input the user ID (*Hint: You may use Custom ORM by following the naming convention*). You should handle if the author does not exist and throw a not found response with the message "`User not found`".

5.2.2 Add Comment to a Post

```
public Post addCommentToPost(String postID, Comment newComment) {}
```

This method should add a new comment to a post by taking as input the post ID and the new comment (*Hint: You may use the getters and setters of the Post class*). You should handle if the post does not exist and throw a not found response with the message "`Post not found`".

6 Controllers

6.1 User Controller

The user controller handles the requests coming to the implemented user services. It already has the basic handlers for the already implemented services (basic CRUD operations). You are required to add the following handler:

6.1.1 Update User's Username

`PUT /updateUser`, which updates the username of a specific user. The username is passed down in the request body in an attribute called `username` in the `Map<String,String>` variable. Your method should ***return the updated user***.

6.2 Post Controller

The post controller handles the requests coming to the implemented post services. It already has basic handlers for the already implemented services (basic CRUD operations). You are required to add the following handlers:

6.2.1 Get the Posts of the Author

`GET /postsByUser/{userId}`, which gets all the posts of a specific author. This method should ***return a list of posts***.

6.2.2 Add Comment

`POST /{id}/comments`, which adds a new comment to a specific post. The method takes ***the post ID from the URL and the new comment from the request body as a comment object and not a Map***. This method should ***return the updated post***.

6.3 Main Controller

This is the main controller that has the handler for the database seed and also the method for testing the instance of the application seen in lab 4. You should uncomment this file.

7 application.properties

You should **not** modify the `application.properties` file. You should ensure that you set up the other services properly for your application to run outside docker. Additionally, any modifications for the application to run inside docker should be overridden using docker compose environment variables.

8 Docker

Dockerize your application by creating the necessary docker-compose and docker file for the application image as seen in Lab 4. Your docker compose file should include the following services:

8.1 Spring Boot App Instances

- You need to create 2 instances of your application named `webapp1` and `webapp2`.
- Each instance need to include the environment variable `instance`, whose value corresponds the name of the instance (i.g `webapp1` has a value of `1`). This variable is used in the `MainController`.
- You should introduce environment variables to ensure that the application is able to connect to mongodb both inside and outside docker.

8.2 Mongo Service

- Service must be named `mongodb`
- You need to port-bind it to port 27017 on the host machine.
- The database names and credentials should be the same as those in `application.properties`.

8.3 Nginx Service

- You need to port-bind it to port 80 on the host machine.
- You should use the `volumes/nginx/nginx.conf` file.

9 Endpoint Testing

To run the public API tests, run `mvn test` in terminal. Please note that the tests require a valid connection to mongoDB.

10 Load Balancer (Nginx)

Fill in the created `nginx.conf` file in the directory `volumes/nginx`. You can refer to the file in lab 4 for guidance.

a) The upstream should be named `webapp`.

b) You should implement weighted round robin load balancing with weights 1 and 2 for `app1` and `app2` respectively.

11 Load Testing (JMeter)

Modify the created `Task_3_Plan.jmx` file in the directory `jmeter`. You can refer to the file in lab 4 for guidance.

- You should create a thread group which creates **180 requests** with a **ramp up of 2 seconds**, and a **loop count of 1**.
- You should configure the the thread group to send its requests to the running nginx server. The tests should call the `MainController` endpoint which displays the instance number.
- You should add a "View Results Tree" listener to your test plan.

12 Submission Guidelines

Submission will be through GitHub Classroom. Please note that passing all tests is not an indicator of a full grade. Additional tests may be performed afterwards.