Build a Full Stack Reddit Clone with – Spring boot and Angular – Part 7

0 Comments

BY SAI UPADHYAYULA | February 24, 2020

Reddit clone with Spring boot and Angular
#7
Implement API to manage Votes

Share Tweet Pin

Welcome to Part 7 of Build a Full Stack Reddit Clone with Spring boot and Angular series. In Part 6, we saw how to create an API to create comments for posts in our application.

What are we building?

In this part, we will write API to submit a vote on the Posts created by the user. We will also refactor the post API to retrieve the vote information along with the Post Details. This functionality is similar to what we see in applications like Stackoverflow and of course Reddit 😊

If you are a visual learner like me you can check out the video version of this tutorial below:

Thank you for visiting. You can now buy me a coffee!

> The source code of this project, is hosted on Github –
> Backend – https://github.com/SaiUpadhyayula/spring-reddit-clone
> Frontend – https://github.com/SaiUpadhyayula/angular-reddit-clone

API Design for Votes

The API Design for Votes is pretty simple as we have only one endpoint which just submits the vote for a Post.

Mapping	HTTP Method	Method Name
/api/votes/	POST	vote

Let's create our **VoteController.java** class which is responsible to submit votes.

Implementation

VoteController.java

```
package com.example.springredditclone.controller;

import com.example.springredditclone.dto.VoteDto;
import com.example.springredditclone.service.VoteService;
import lombok.AllArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/votes/")
@AllArgsConstructor
public class VoteController {

    private final VoteService voteService;

    @PostMapping
    public ResponseEntity<Void> vote(@RequestBody VoteDto voteDto) {
        voteService.vote(voteDto);
        return new ResponseEntity<>(HttpStatus.OK);
    }
}
```

- For our controller, we have the Request Mapping as **"/api/votes/"**
- We have only one method in this controller, which has **@PostMapping** annotation on top of the method
- The method is taking RequestBody as **VoteDto.java** , which represents the request coming in from the client.
- Inside the method we are calling the **vote()** method from the **VoteService.java** class, and once this vote is saved to the database we are returning an **HttpStatus.OK** response (ie. HTTP Status 200)

VoteDto.java

```
package com.example.springredditclone.dto;

import com.example.springredditclone.model.VoteType;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class VoteDto {
    private VoteType voteType;
    private Long postId;
}
```

VoteService.java

```
package com.example.springredditclone.service;

import com.example.springredditclone.dto.VoteDto;
import com.example.springredditclone.exceptions.PostNotFoundException;
import com.example.springredditclone.exceptions.SpringRedditException;
import com.example.springredditclone.model.Post;
import com.example.springredditclone.model.Vote;
import com.example.springredditclone.repository.PostRepository;
import com.example.springredditclone.repository.VoteRepository;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.Optional;

import static com.example.springredditclone.model.VoteType.UPVOTE;

@Service
@AllArgsConstructor
public class VoteService {

    private final VoteRepository voteRepository;
    private final PostRepository postRepository;
    private final AuthService authService;

    @Transactional
    public void vote(VoteDto voteDto) {
        Post post = postRepository.findById(voteDto.getPostId())
                .orElseThrow(() -> new PostNotFoundException("Post Not Found with ID - " +
voteDto.getPostId()));
        Optional<Vote> voteByPostAndUser = voteRepository.findTopByPostAndUserOrderByVoteIdDesc(post,
authService.getCurrentUser());
        if (voteByPostAndUser.isPresent() &&
                voteByPostAndUser.get().getVoteType()
                        .equals(voteDto.getVoteType())) {
            throw new SpringRedditException("You have already "
                    + voteDto.getVoteType() + "'d for this post");
        }
        if (UPVOTE.equals(voteDto.getVoteType())) {
            post.setVoteCount(post.getVoteCount() + 1);
        } else {
                post.setVoteCount(post.getVoteCount() - 1);
            }
            voteRepository.save(mapToVote(voteDto, post));
            postRepository.save(post);
        }

        private Vote mapToVote(VoteDto voteDto, Post post) {
            return Vote.builder()
                    .voteType(voteDto.getVoteType())
                    .post(post)
                    .user(authService.getCurrentUser())
                    .build();
        }
    }
```

- Line 28-29: Inside the **vote()** method of **VoteService**, we are first retrieving the **Post** object from the database using the **PostRepository.findById()** method. If there is no **Post** with that id we are throwing a **PostNotFoundException**.
- Line 30: We are retrieving the recent **Vote** submitted by the currently logged-in user for the given **Post**. We are doing this using the method – **findTopByPostAndUserOrderByVoteIdDesc()**
- Line 31-36: We are first checking whether the user has already performed the same Vote action or not. ie. If the user has already upvoted a particular post, he/she is not allowed to upvote that particular post again.
- Line 37-43: We are setting the voteCount field according to the **VoteType** provided by the user, and then we are mapping the **VoteDto** object to **Vote** object and saving the **Vote** as well as **Post** objects to the database.
- We are using the **builder()** from **Lombok's @Builder** inside the **mapToVote** method. This builder method is an implementation of the Builder Design Pattern. If you want to learn more about Builder Pattern or any design pattern, I highly recommend you to buy the Head First Design Pattern Book

VoteRepository.java

```
package com.example.springredditclone.repository;

import com.example.springredditclone.model.Post;
import com.example.springredditclone.model.User;
import com.example.springredditclone.model.Vote;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface VoteRepository extends JpaRepository<Vote, Long> {
    Optional<Vote> findTopByPostAndUserOrderByVoteIdDesc(Post post, User currentUser);
}
```

With this, we have implemented the API to submit votes. Now its time to update the Post API to include the vote information.

Update Post API with Vote

When the user queries the Post API to get a single post or all posts. We have to also provide the vote information along with post details.

As a first step, let's update the **PostResponse.java** with some new fields.

```
package com.example.springredditclone.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class PostResponse {
    private Long id;
    private String postName;
    private String url;
    private String description;
    private String userName;
    private String subredditName;
    private Integer voteCount;
    private Integer commentCount;
    private String duration;
}
```

We added 3 new fields inside the **PostResponse.java** class – **voteCount**, **commentCount**, **duration**

Now we have to update our **PostMapper.java** class to map also these new additional fields. One advantage of creating this **PostMapper.java** is now our mapping logic is decoupled from the actual business logic.

PostMapper.java

```
package com.example.springredditclone.mapper;

import com.example.springredditclone.dto.PostRequest;
import com.example.springredditclone.dto.PostResponse;
import com.example.springredditclone.model.Post;
import com.example.springredditclone.model.Subreddit;
import com.example.springredditclone.model.User;
import com.example.springredditclone.repository.CommentRepository;
import com.example.springredditclone.repository.VoteRepository;
import com.example.springredditclone.service.AuthService;
import com.github.marlonlom.utilities.timeago.TimeAgo;
import org.mapstruct.Mapper;
import org.mapstruct.Mapping;
import org.springframework.beans.factory.annotation.Autowired;

@Mapper(componentModel = "spring")
public abstract class PostMapper {

    @Autowired
    private CommentRepository commentRepository;
    @Autowired
    private VoteRepository voteRepository;
    @Autowired
    private AuthService authService;


    @Mapping(target = "createdDate", expression = "java(java.time.Instant.now())")
    @Mapping(target = "description", source = "postRequest.description")
    @Mapping(target = "subreddit", source = "subreddit")
    @Mapping(target = "voteCount", constant = "0")
    public abstract Post map(PostRequest postRequest, Subreddit subreddit, User user);

    @Mapping(target = "id", source = "postId")
    @Mapping(target = "subredditName", source = "subreddit.name")
    @Mapping(target = "userName", source = "user.username")
    @Mapping(target = "commentCount", expression = "java(commentCount(post))")
    @Mapping(target = "duration", expression = "java(getDuration(post))")
    public abstract PostResponse mapToDto(Post post);

    Integer commentCount(Post post) {
        return commentRepository.findByPost(post).size();
    }

    String getDuration(Post post) {
```

Thank you for visiting. You can now buy me a coffee!

```
            return TimeAgo.using(post.getCreatedDate().toEpochMilli());
        }
    }
```

- I changed **PostMapper.java** from an interface to abstract class, this is because we need information to map the new fields (voteCount, commentCount and duration).
- As I said before, instead of updating **PostService.java** class with this logic, we can just change the interface to abstract class and inject the needed dependencies into our class to fill the new field's information.
- At line 30 – You can observe that the voteCount is set to a constant value – 0. This is because whenever we want to create a new **Post** object, we set the default vote count as 0.
- When mapping from Post to PostResponse, we explicitly need to map only the fields commentCount (handled through commentCount() method) and duration (handled through getDuration())
- The **getDuration()** is using a library called TimeAgo. This is a java library that shows us the dates in the relative Time Ago format.

Let's add this library to our **pom.xml** file. As this is a Kotlin Library we need to add some external dependencies like **kotlin-stdlib-jdk8** , **kotlin-test-junit** and a plugin – **kotlin-maven-plugin**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.8.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>spring-reddit-clone</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>spring-reddit-clone</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
        <org.mapstruct.version>1.3.1.Final</org.mapstruct.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-mail</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.8</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <!-- JWT related dependencies-->
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt-api</artifactId>
        <version>0.10.5</version>
    </dependency>
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt-impl</artifactId>
        <scope>runtime</scope>
        <version>0.10.5</version>
    </dependency>
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt-jackson</artifactId>
        <scope>runtime</scope>
        <version>0.10.5</version>
    </dependency>
    <dependency>
        <groupId>org.mapstruct</groupId>
        <artifactId>mapstruct</artifactId>
        <version>${org.mapstruct.version}</version>
        <scope>compile</scope>
    </dependency>
    <!-- For Displaying time as Relative Time Ago ("Posted 1 Day ago"),
     as this is a Kotlin library, we also need Kotlin runtime libraries-->
    <dependency>
        <groupId>com.github.marlonlom</groupId>
        <artifactId>timeago</artifactId>
        <version>4.0.1</version>
    </dependency>
    <dependency>
        <groupId>org.jetbrains.kotlin</groupId>
        <artifactId>kotlin-stdlib-jdk8</artifactId>
        <version>${kotlin.version}</version>
    </dependency>
    <dependency>
        <groupId>org.jetbrains.kotlin</groupId>
        <artifactId>kotlin-test-junit</artifactId>
        <version>${kotlin.version}</version>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.5.1</version> <!-- or newer version -->
                <configuration>
                    <source>1.8</source> <!-- depending on your project -->
                    <target>1.8</target> <!-- depending on your project -->
                    <annotationProcessorPaths>
                        <path>
                            <groupId>org.mapstruct</groupId>
                            <artifactId>mapstruct-processor</artifactId>
                            <version>${org.mapstruct.version}</version>
                        </path>
                        <path>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                            <version>1.18.8</version>
                        </path>
                    </annotationProcessorPaths>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.jetbrains.kotlin</groupId>
                <artifactId>kotlin-maven-plugin</artifactId>
                <version>${kotlin.version}</version>
                <executions>
                    <execution>
                        <id>compile</id>
                        <phase>process-sources</phase>
                        <goals>
                            <goal>compile</goal>
                        </goals>
                        <configuration>
                            <sourceDirs>
                                <source>src/main/java</source>
                                <source>target/generated-sources/annotations</source>
                            </sourceDirs>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

Testing

So its time for testing, we have implemented the Vote API and update the Post API to include the vote details. Let's start our application and test the API's using Postman.

Let's create a vote

Thank you for visiting. You can now buy me a coffee!

API Call to Submit Vote

Get Posts

- When we called the endpoint to create a vote – /api/votes/ , we got the response back as 200 OK. Here we have provided the VoteType as DOWNVOTE and id as 3
- Now we made a call to get All Posts in our application, and for the post with id – 3 we can see that the voteCount is

I hope you are finding this tutorial series helpful, please subscribe to this blog, to get the l

mailbox. I will see you in the next part, until then **Happy Coding Techies** 🙂

About the author

Sai Upadhyayula

Thank you for visiting. You can now buy me a coffee!