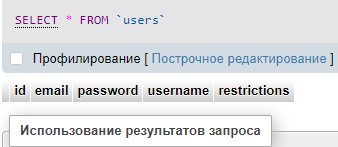
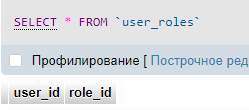
1. Spring web application
2. **User can register and authenticate**

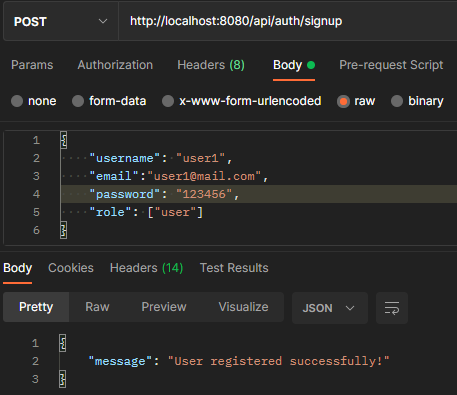
**URL to register:** <http://localhost:8080/api/auth/signup> Post method

in AuthController

Tables “users” and many-to-many joined “user\_roles” are empty

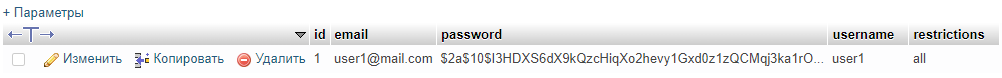
 

Request from postman with body where entered user’s data



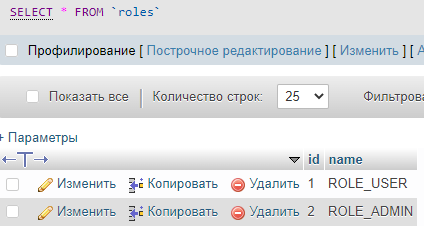
By default user profile restriction is “all”

“users” table

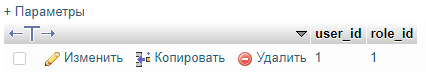


“roles” table

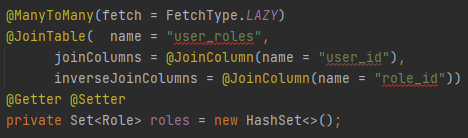
In initial point, we inserted 2 roles: ROLE\_USER and ROLE\_ADMIN



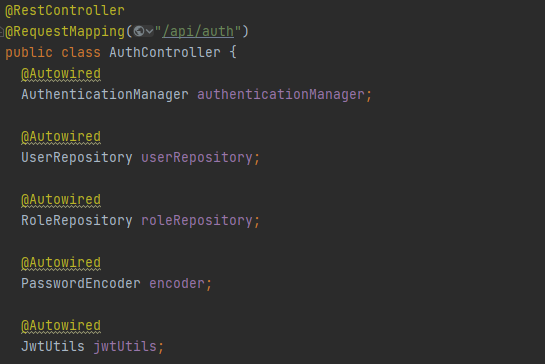
Many-to-many relationship users and roles in table “user\_roles”

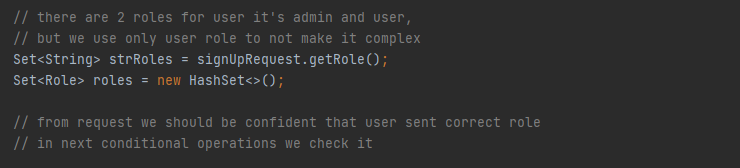
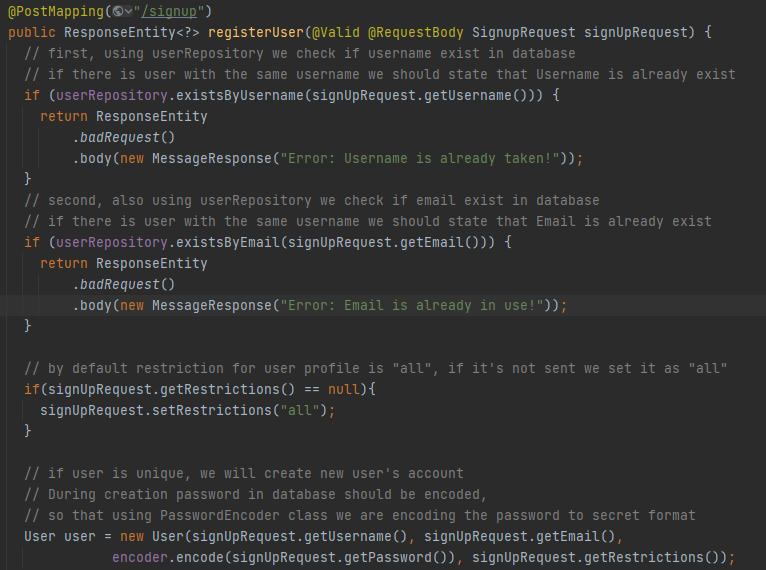
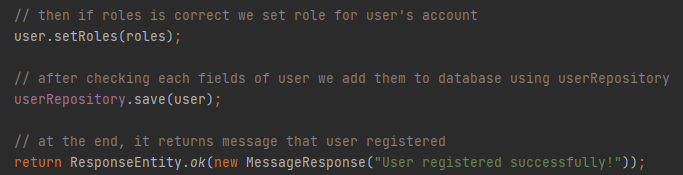
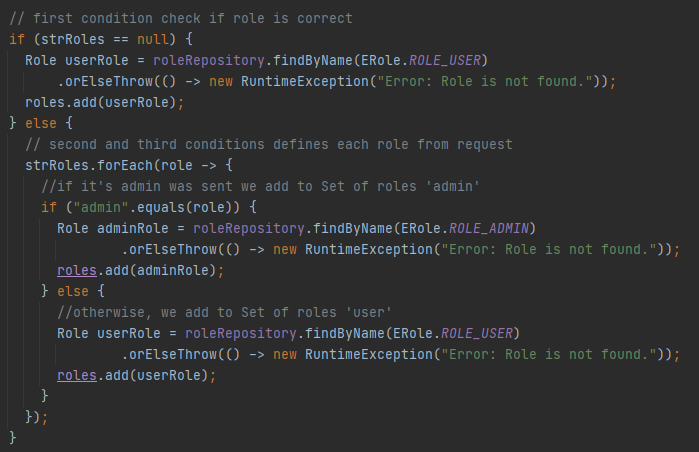


This table automatically creates, because we have used annotation many-to-many in User class.



**AuthController**

****

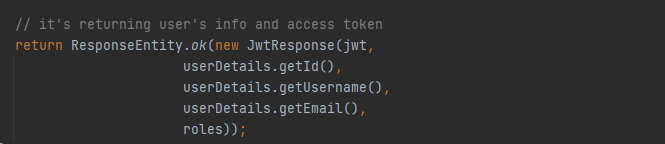
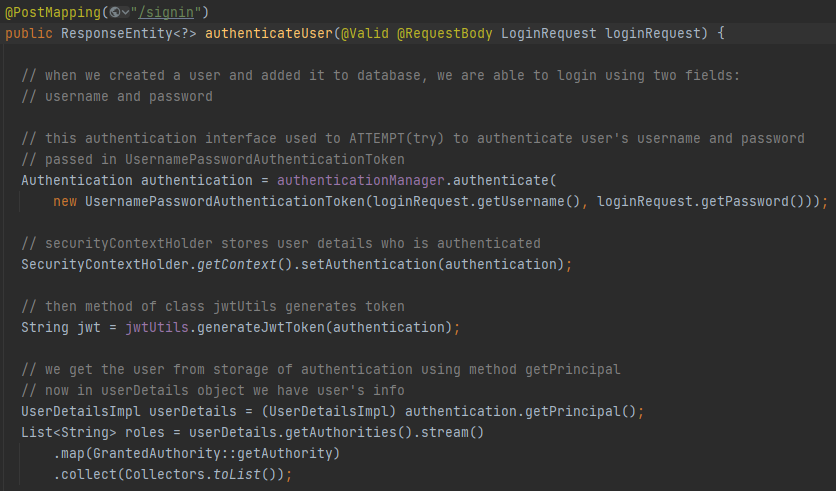
** **

**URL to authenticate:** <http://localhost:8080/api/auth/signin> Post method

in AuthController

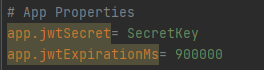


AuthController

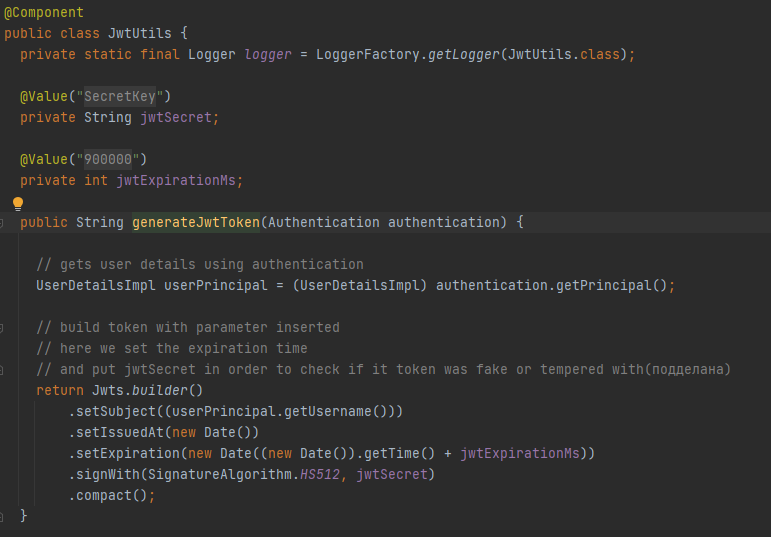


As you can see, to authenticate user we have used Bearer TOKEN.

In application properties we’ve set token’s expiration time in milliseconds (900.000ms=15min)

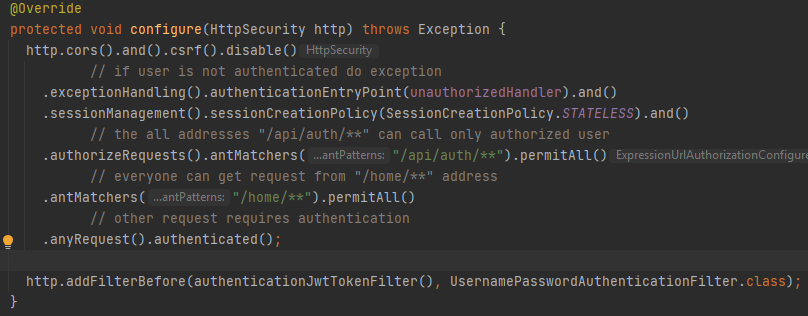


JwtUtils Class



After TOKEN generated, it’ll returned like access token in Jwts builder with user’s info.

In order to check TOKEN on each request, you can see that, we added .anyRequest().authenticated() options which requires to enter token in URL’s we permitted.



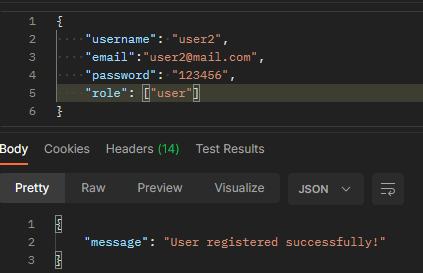
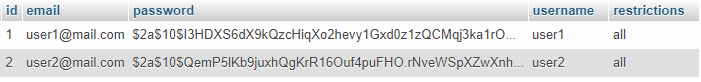
Also in the video Elnur demonstrates this checking in details.

1. **Users can add friends: send/accept friend requests**

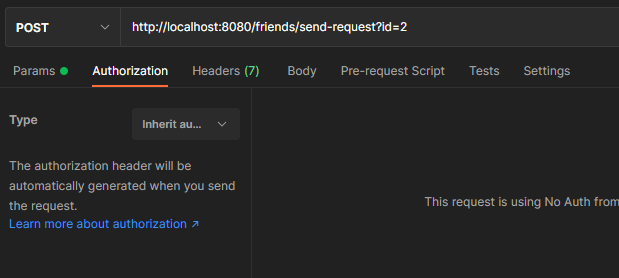
**URL to send request to friend:** <http://localhost:8080/friends/send-request>

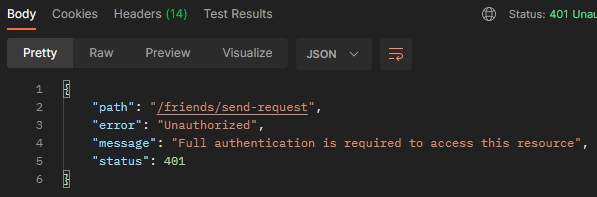
Post method FriendController

In order to add friend we need to send request to another user, so let’s create 2nd user.

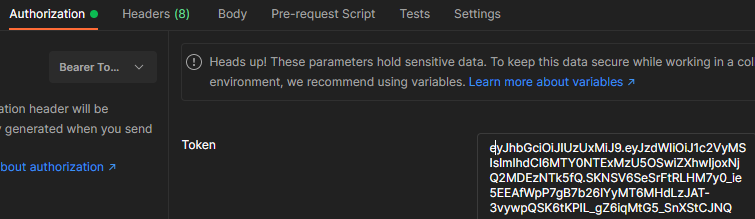
So, we need to add id of user whom send request. User1 sends to User2. In order to do that input in parameters key and value, and in Authorization header input TOKEN of user1 which is logged in. If we didn’t add it we’ll get message about “Authorize”

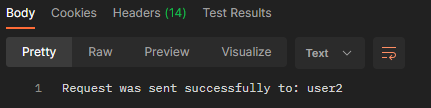




So that let’s put the user1’s TOKEN and check the message



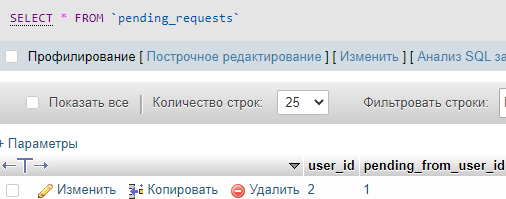
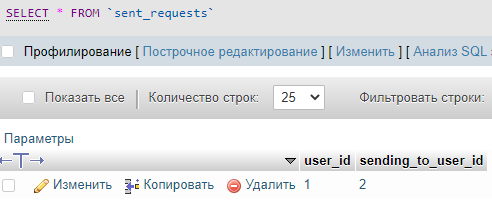




In model for User, we’ve automatically created many-to-many tables for send\_requests (отправленные), pending\_requests (входящие) and friends like it was for Roles (user\_roles)



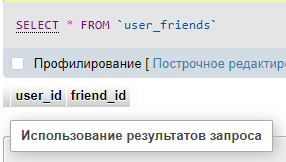
So let’s check in database tables’ sending\_requests and pending\_requests



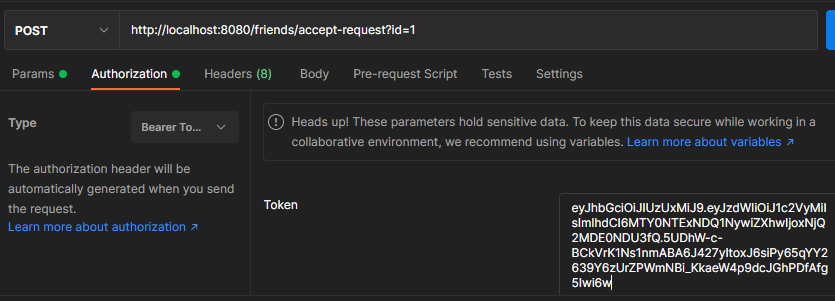
As you can see, user1 is sent request to user2, and in the table sent\_requests it’s shown.

Also user2 in pending\_requests has user1. They are not friends yet. Now let’s accept user1 being as a user2. We should get user2’s TOKEN and use another URL.

Before user2 will accept user1, we can see user\_friends table is empty.

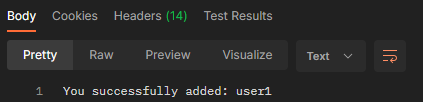
****

**URL to accept request to friend:** <http://localhost:8080/friends/accept-request>

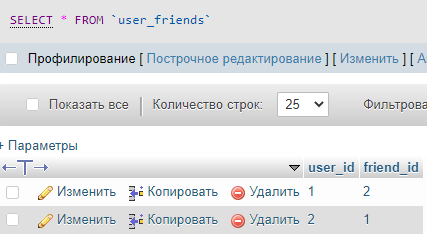
****

Here we should pass id of user in parameters and token of current user, in our case he is user2.

Send request and get the message:

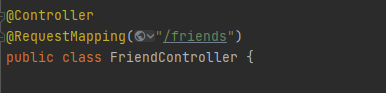
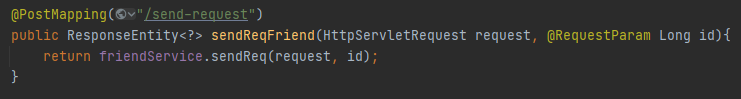


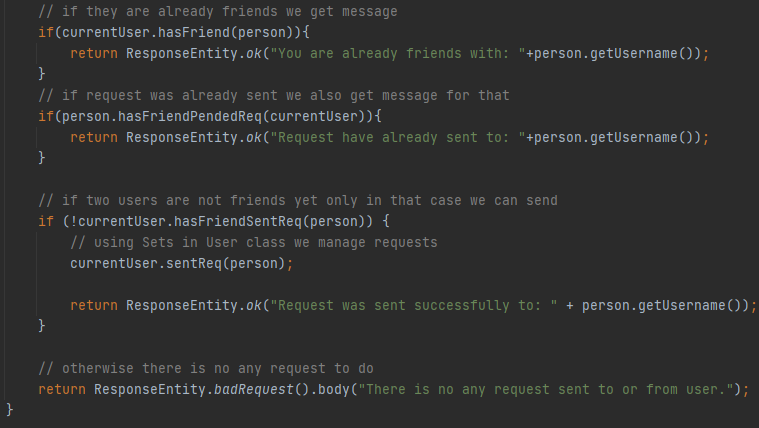
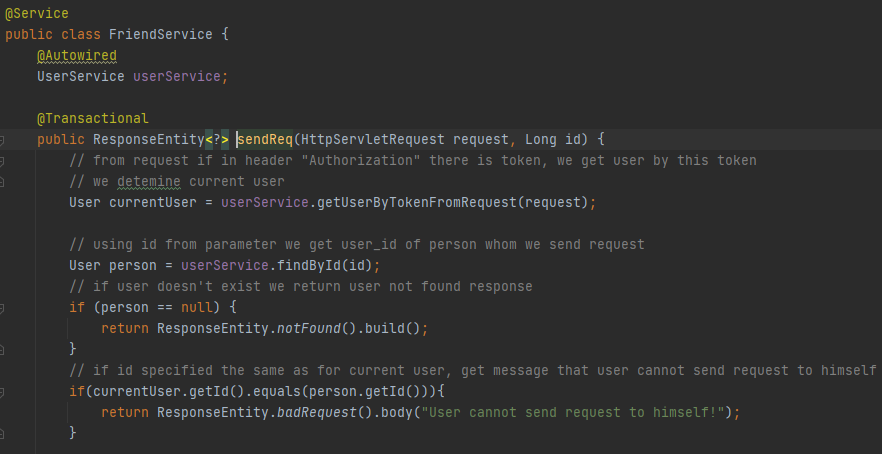
Also in users\_friends table there is data appeared.



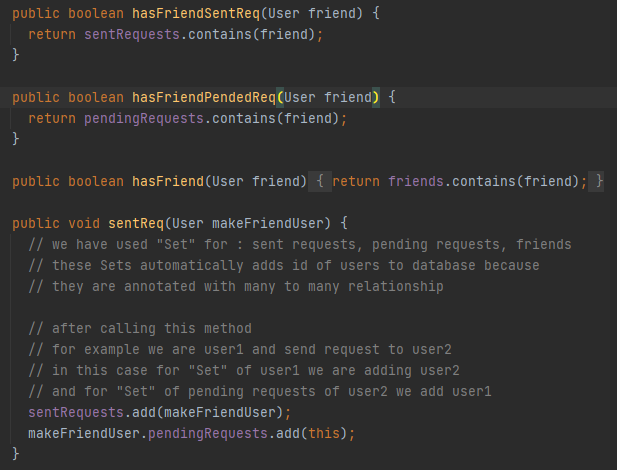
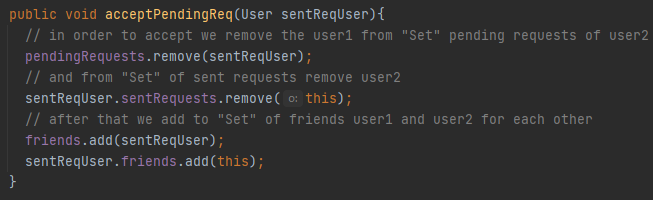
We’ve added two users to friends.

In code part you can see the logic:



In User class as we already mentioned there Sets of sent, pending request, they just add the users to appropriate Sets. And after that since hibernate in properties specified update it add the data to tables in database.

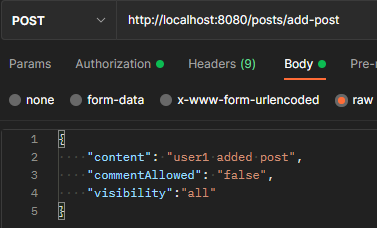
Nevertheless, it’s available to cancel the sent request and reject the sent request

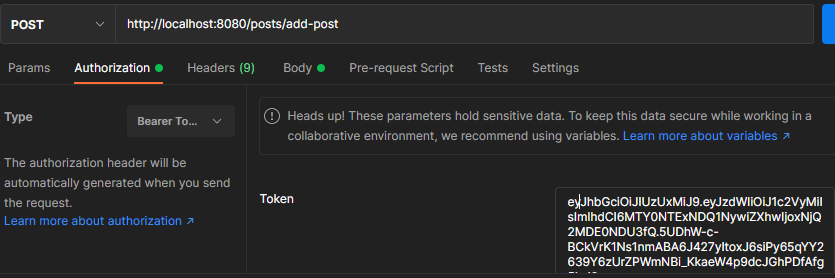
1. **User can create post which visible to**
2. Whole internet
3. Only authorized users
4. Only friends

**URL to create post:** <http://localhost:8080/posts/add-post> Post method

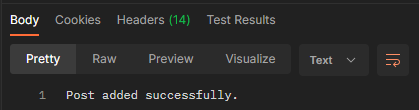
PostController

In order to add post we should input access TOKEN of user2 and fields of post in Body

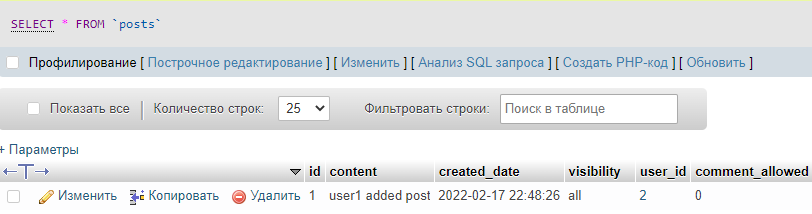




Sending the request and getting message:



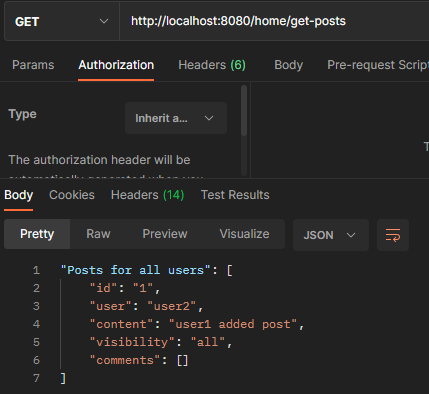
Also check in database:



Now, let’s see how works post visibility. Visibility of post1 is all and it means everyone able to see this post, we do request without TOKEN:

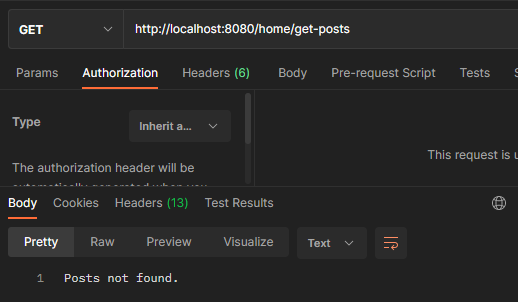
**URL to get post:** <http://localhost:8080/home/get-posts> GET method

HomeController

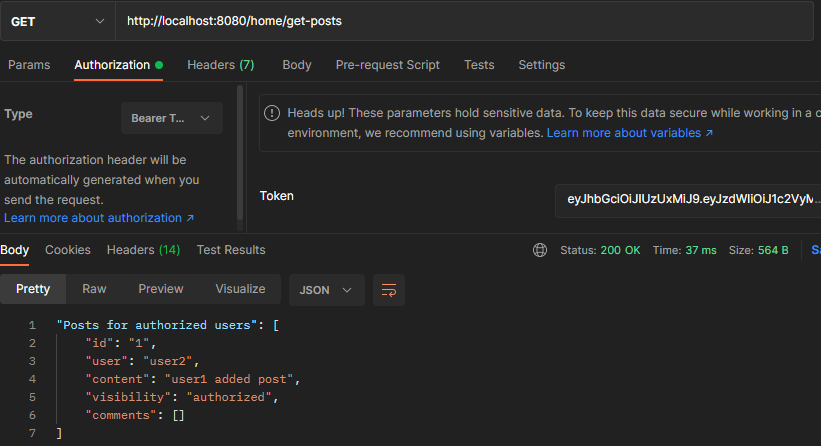


Let’s change the visibility to “authorized” and see the response:





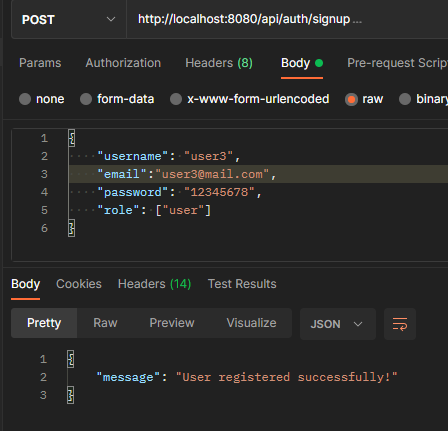
Authorized the 1st user and the response:



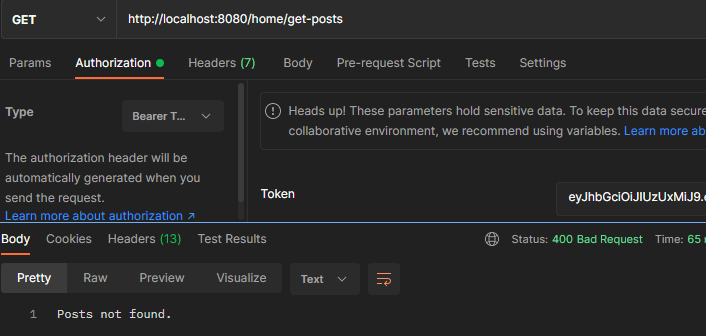
And change visibility to “friends”:



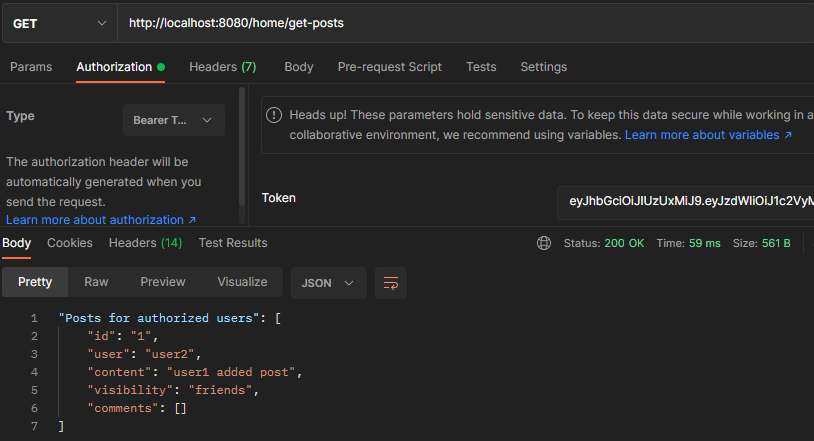
Add 3rd user and check access for not friend user



Put the access TOKEN of 3rd user and see result:

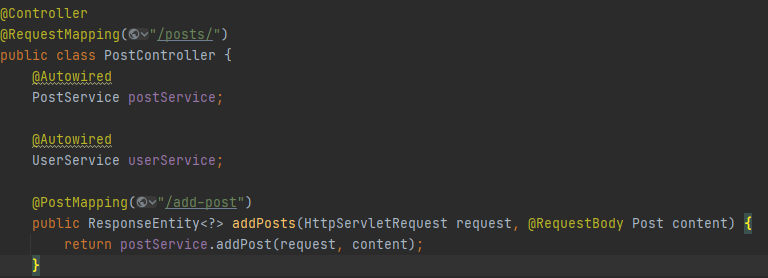


Since, post not found for user3, in ideal the post of user2 should be seen for user1:



Here you are. Post is visible for user1.

Logic in code with comments:

PostController

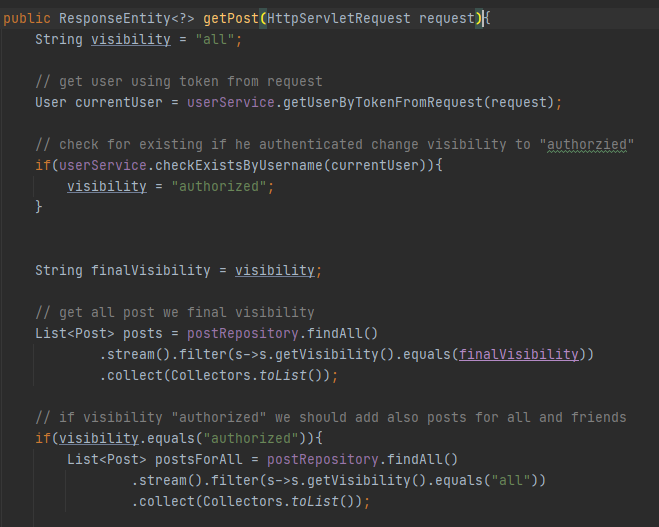
PostService



HomeController

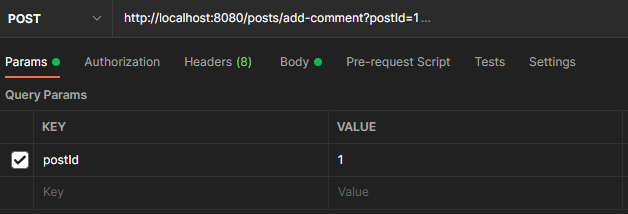


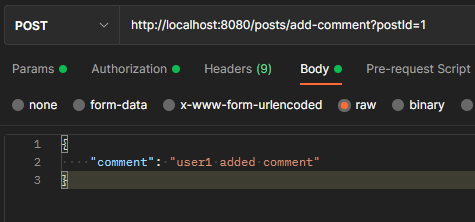
PostService



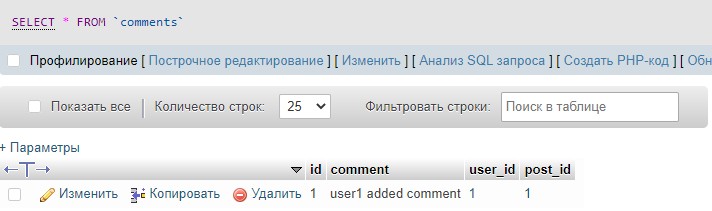
1. **User can comment post**

**URL to add comment to post:** <http://localhost:8080/posts/add-comment>

Obviously, in order comment post we should have post id, user and content of comment. For that reason, in next screenshot you can see the request

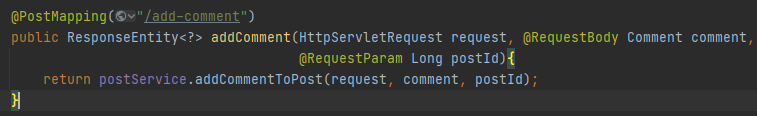


So, comment was added, we can check it in database:

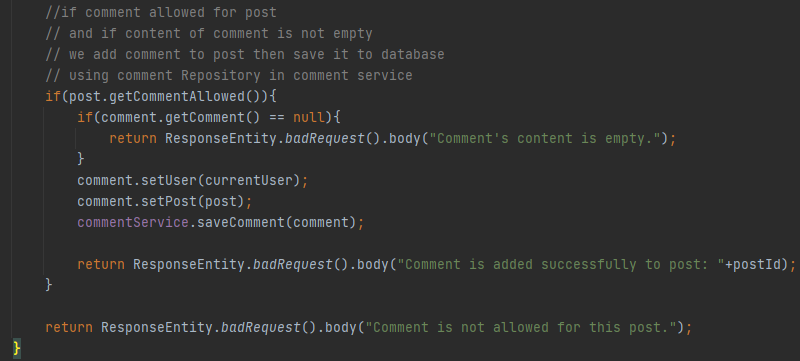
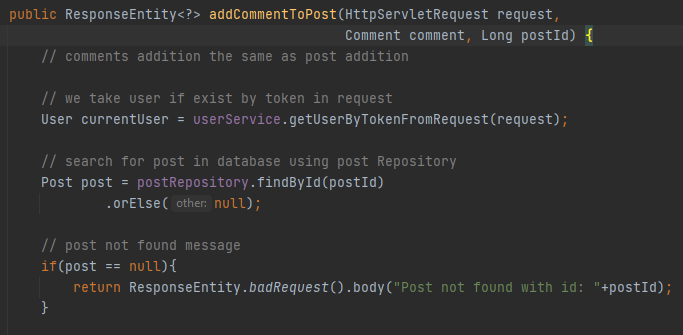


Code for adding comment:

PostController



PostService

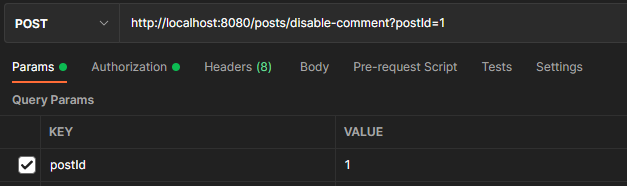


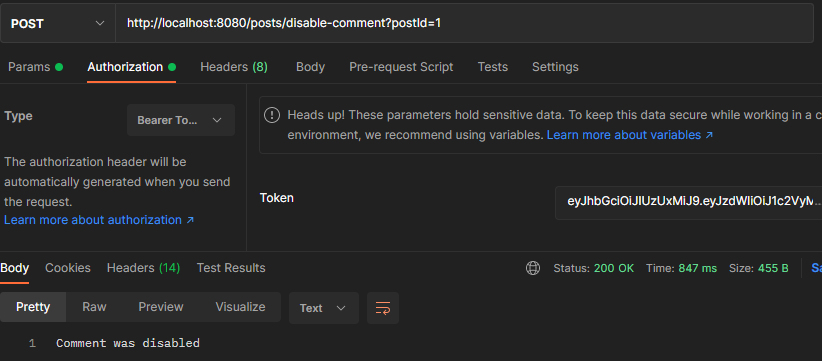
1. **User can disable comments on post**

By default, comments are allowed, but in case we need to disable comment we use the request

**URL to disable comment in post:** <http://localhost:8080/posts/disable-post>

As we know post1 was added by user2 let’s disable it, in order to do that we should have post id and current user using TOKEN:

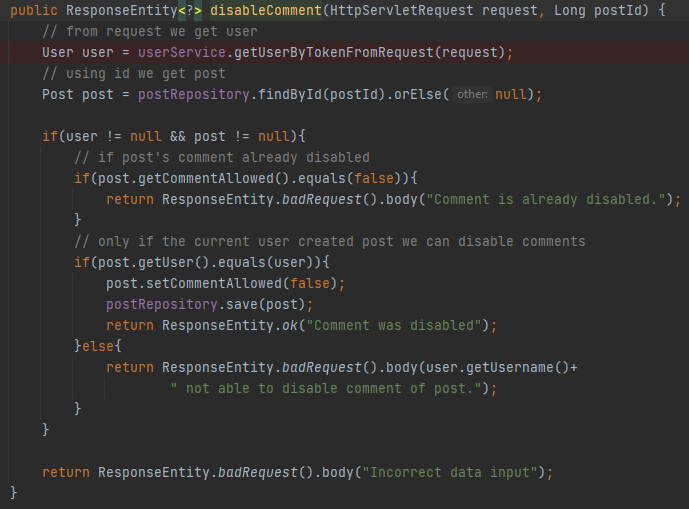




Check in database 

Code:

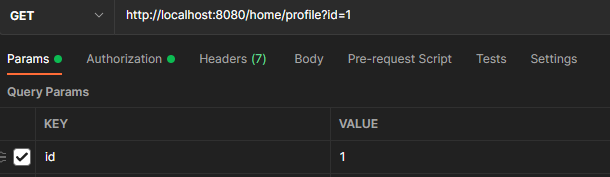
PostController

PostService

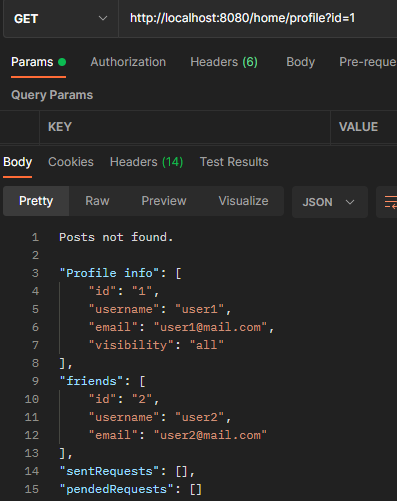
1. **User profile with only user’s posts which are visible:**
2. Whole internet
3. Only authorized users
4. Only friends

**URL to get user profile:** <http://localhost:8080/home/profile>

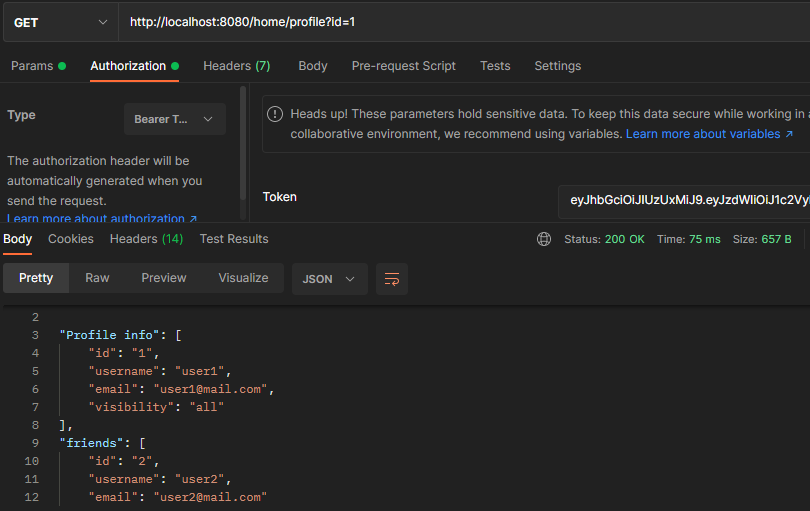
In order to see profile of user we need to pass id and for guest we don’t pass the token:



See results where user id is 1:

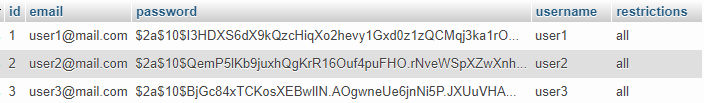


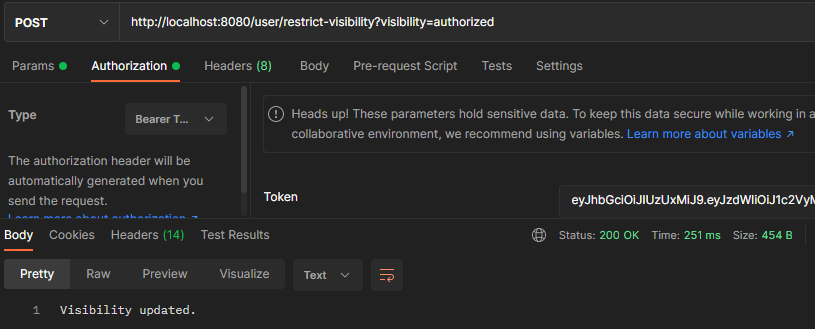
Since, visibility for all, we can see the info about user1.



Now, we’ll restrict visibility.

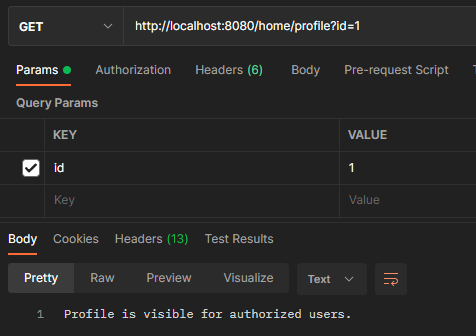
**URL to restrict user profile’s visibility:** <http://localhost:8080/user/restrict-visibility>

We need pass visibility value, and TOKEN of user. After that we get necessary message. We are going to make visible for “authorized”:

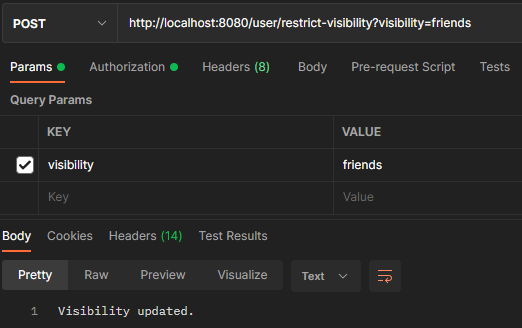


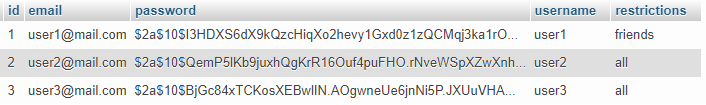


Getting info of user1 being a guest:



And also we make it visible to “friends”:



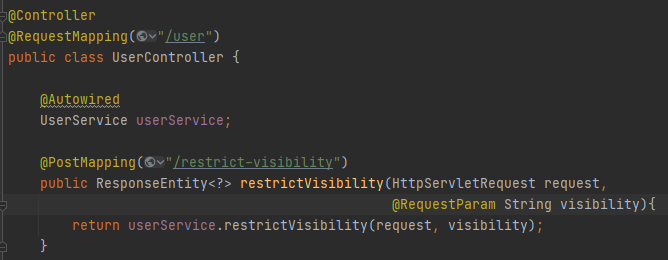


Now we login as user2 and check user1’s info:



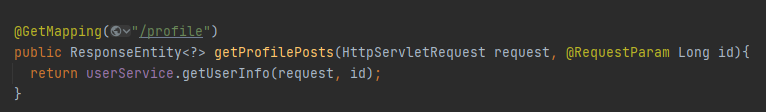
Code implementation:

UserController

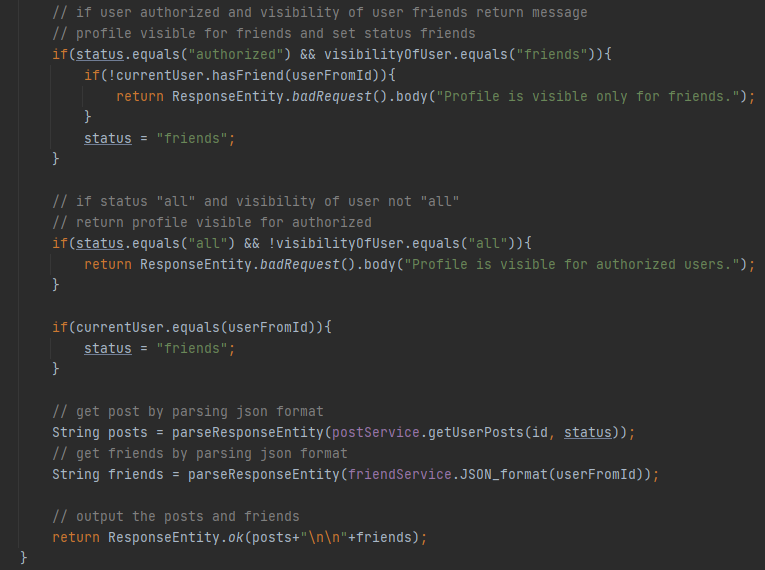
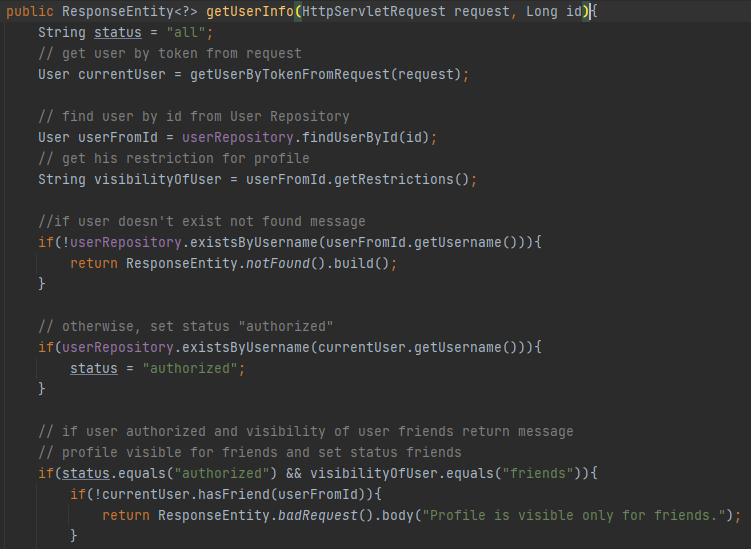


UserService

HomeController

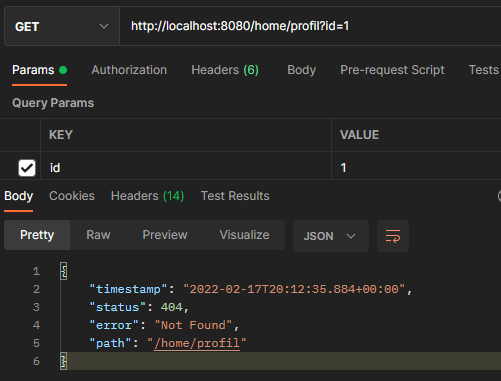


Also in **UserService** realization of getUserInfo()

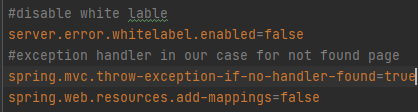


1. **Not found page:**

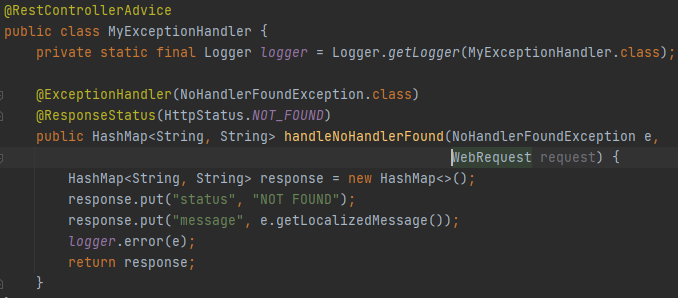
To check the existence of page URL, let’s send remove e from path “profile”, and we can see the following message in response:

****

This response reaches thanks for configuring properties:

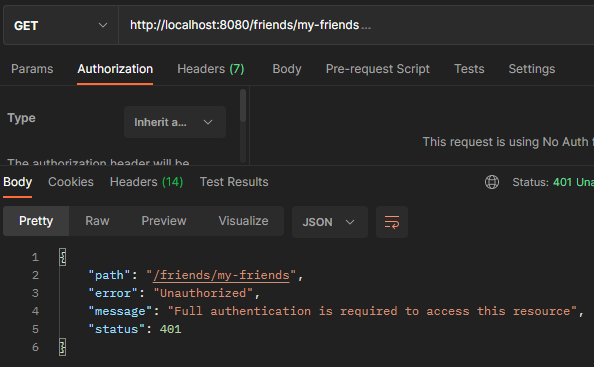


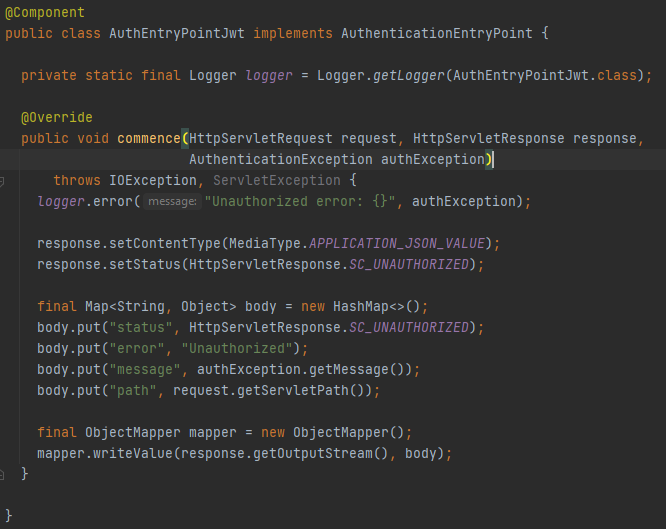
Code:



1. **Permission denied page:**

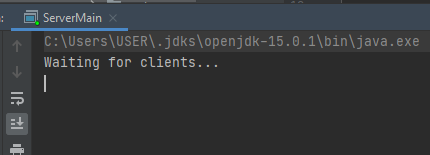
If User not authenticated we can sent UNAUTHORIZED status:



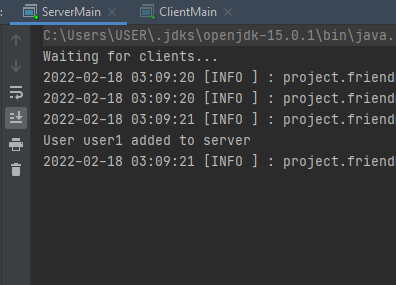


1. Socket Server

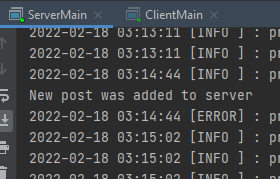
This part we decided to make console application. And next steps show the functionality with code:



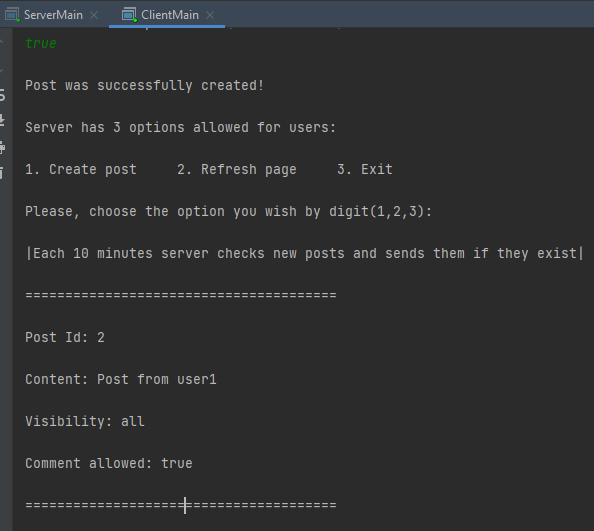
Server started and waiting for clients. After one user connected we can see the server’s message



From the server side when we add the post, message appears:

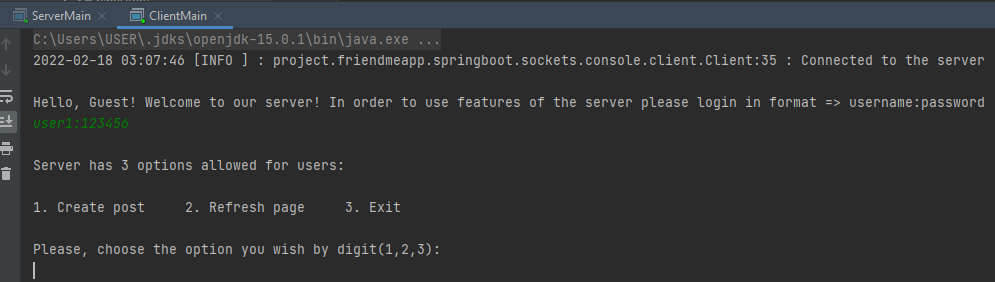


And also each 10 minutes client side get new post:



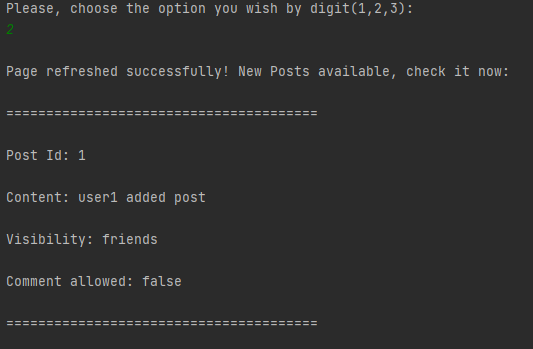
1. Client Server

We have already registered users in database, so we can login using username and password.

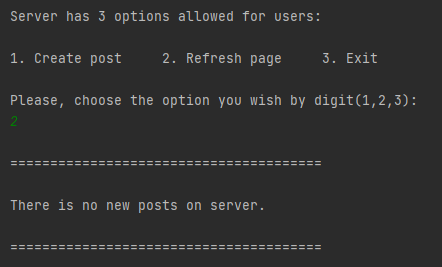


As you can see server proceed commands or 3 options: 1.create post, refresh page and exit

Let’s refresh the page and get new posts, we haven’t seen yet.

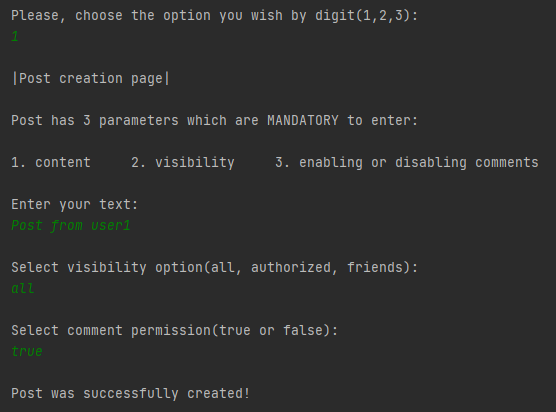


So, new post is seen, what if we send option 2 again, let’s see:

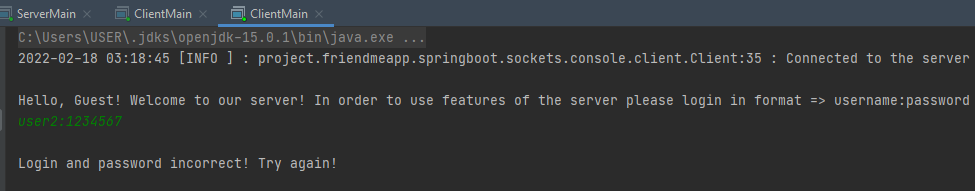


It’s logically correct.

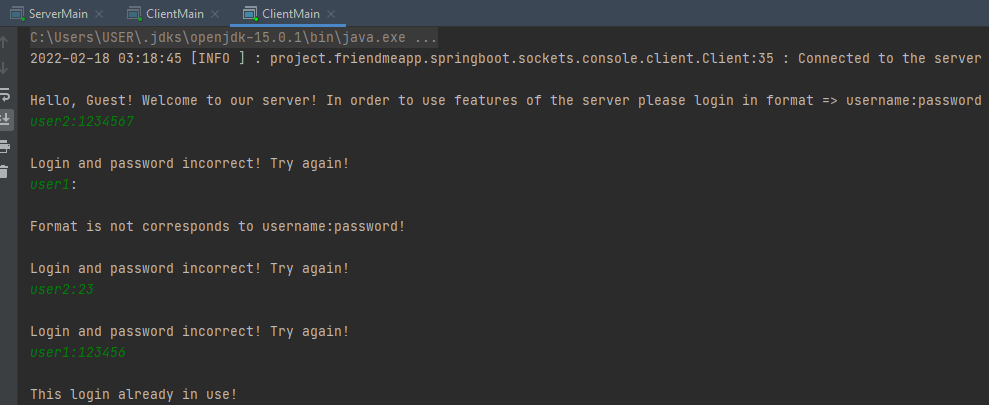
In that case we can add and see the post:

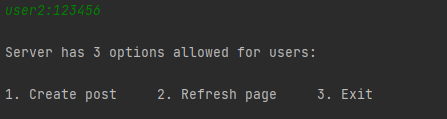


And now we are going to login as a 2nd user and see the posts:

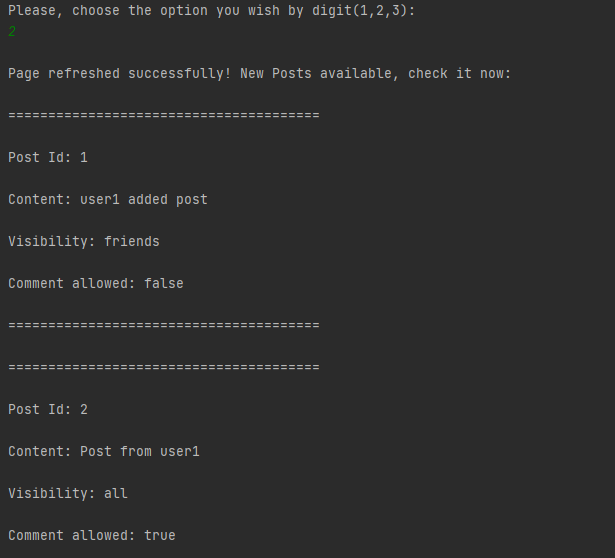


Checking the username and password also here coded. And message like, user already logged in and format is incorrect:





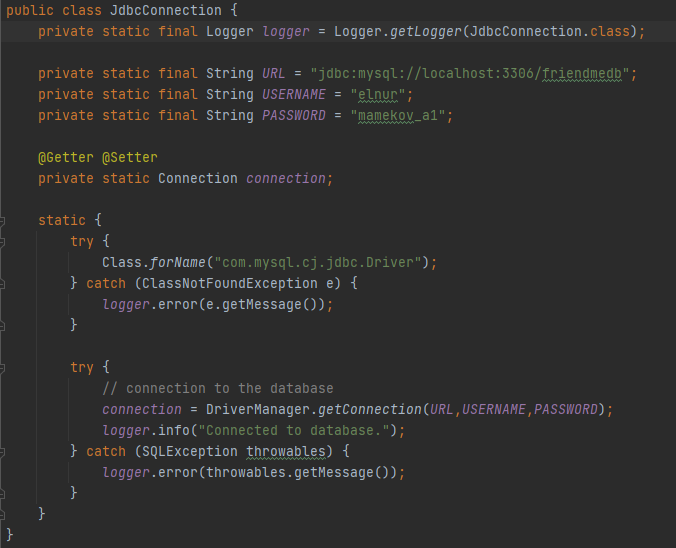
So, user2 logged in and we select 2 to see new posts for this user:



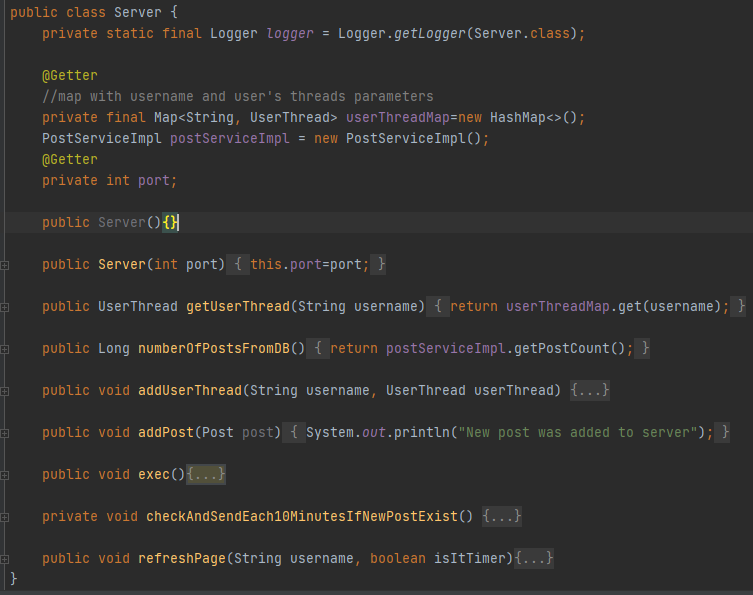
Here we are. Posts are new.

And last option is 3, let’s close the socket of user2: 

Coding:

So, from the source that is got from your lectures, we implemented the userThread classes, added the logic and changed some messages. In initial point, it’s important to mention that we used Jdbc connection: 

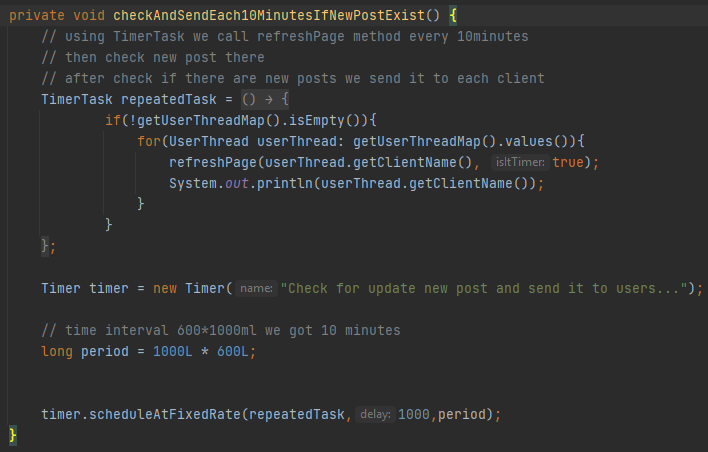
Server Class



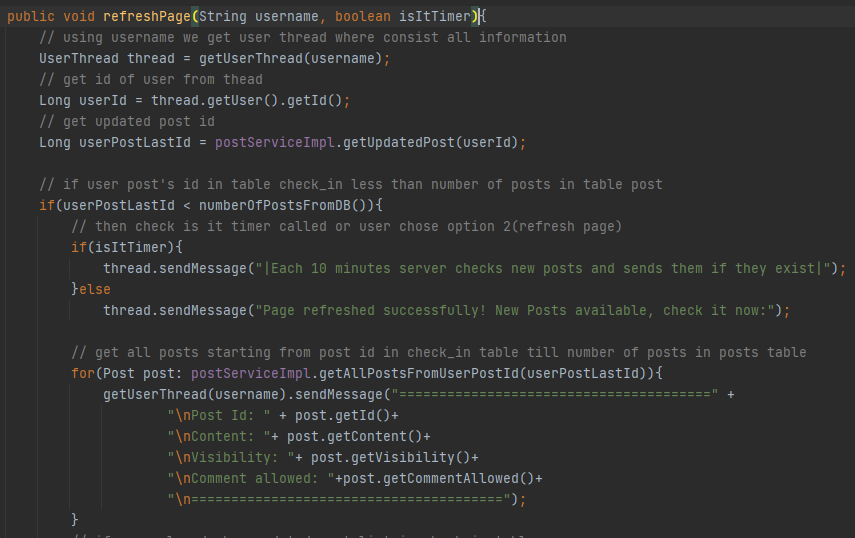
Server execute method



Checking and sending new posts method, here it’s used Timer class and TimerTask

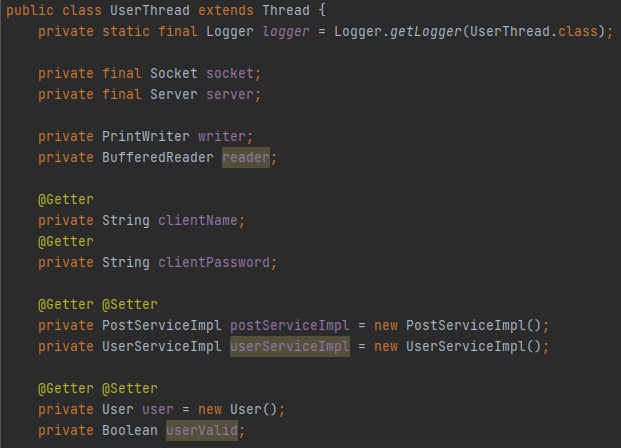


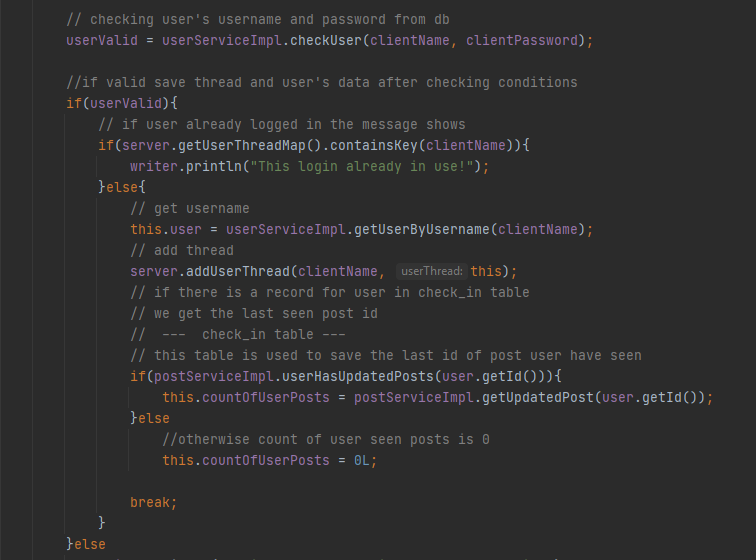
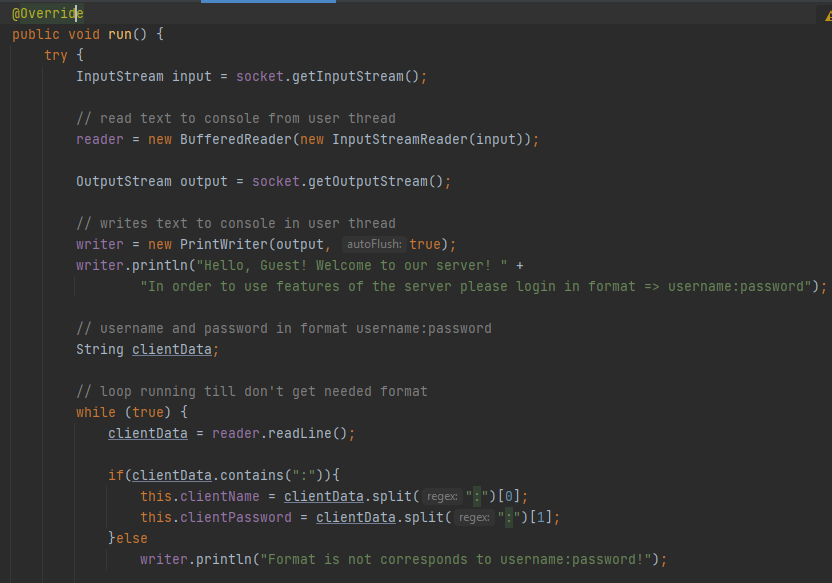
Refresh page which called we either user chooses the 2nd option or time was up and 10 minutes left.

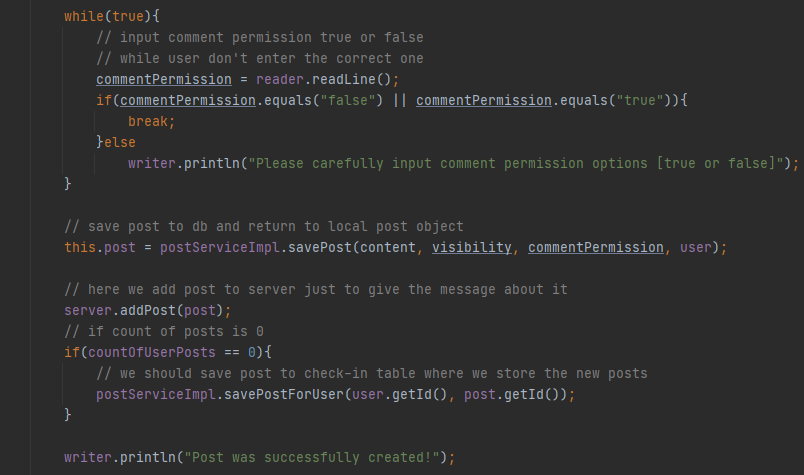
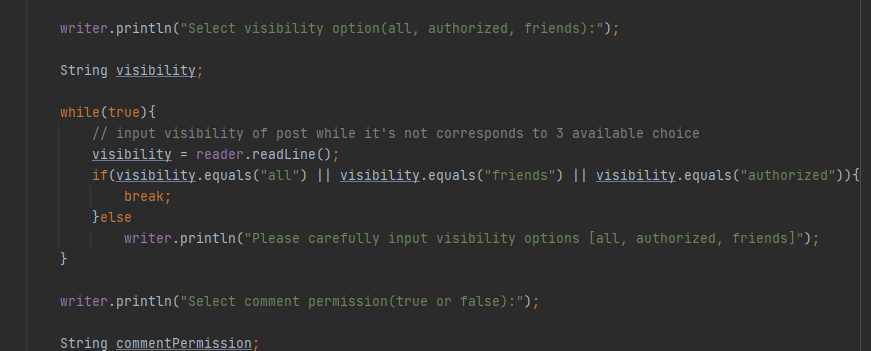
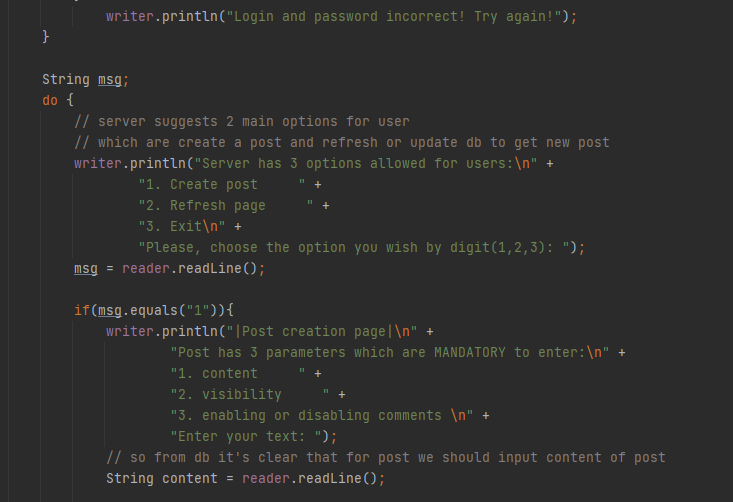


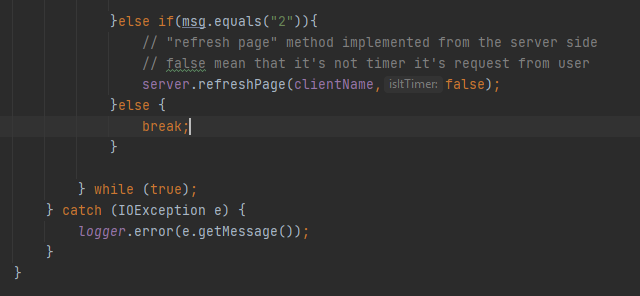
When we run the server it waits the clients. After client is found the server using port it runs the User Thread

The main which wrote for Client side is in UserThread class



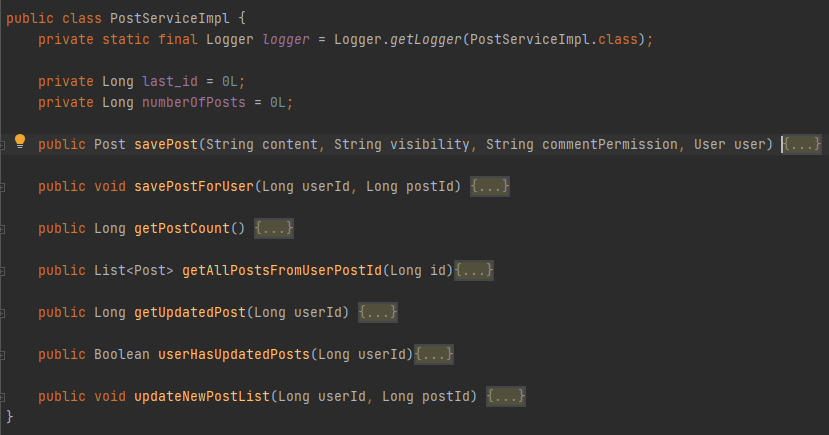






We already did a big amount of screenshots, in order to not make it harder:

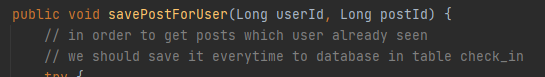
Also we will write only the idea of following method in PostService and UserService classes:



Save post:



SavePostForUser:



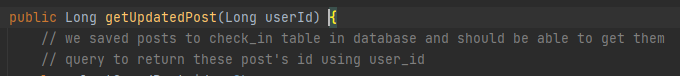
GetPostCount:



GetAllPostsFromUserPostId:



GetUpdatedPost:

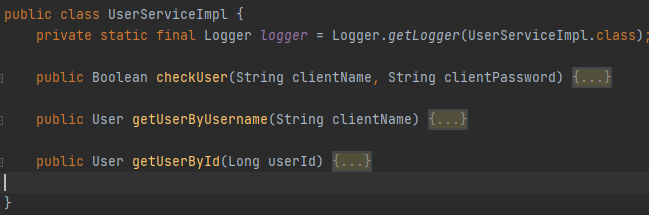


UserHasUpdatedPost:



Update new Post List:





CheckUser:



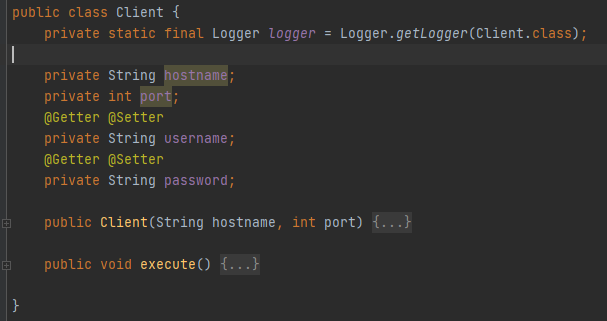
GetUserByUsername:



GetUserById:



Client Class



Client execute method

