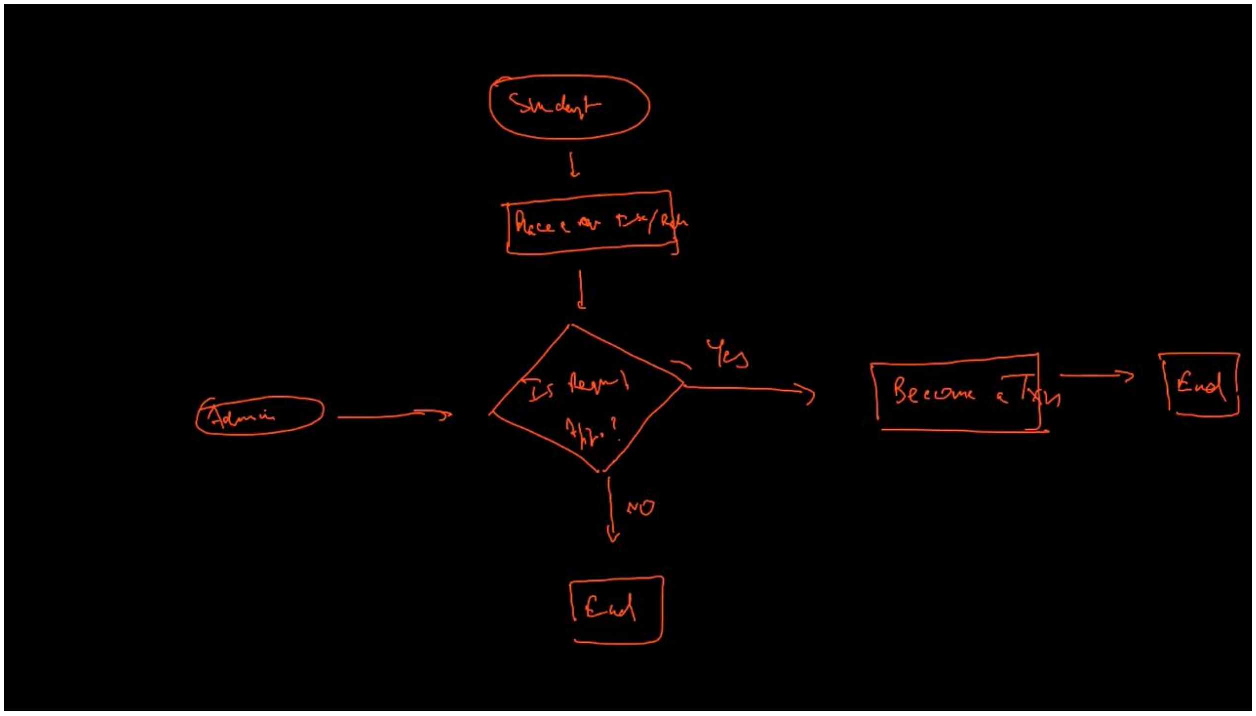
Contents

TASK 1: Let us check the Minor Project 1 and see if we can find any issues with it in the sense of security .
HINT : We need to observe the process request function , place request function ,getStudent function and also the getAdmin functions in StudentController and AdminController classes.

Below is the complete project workflow for Minor project 1 if anyone needs revision :



TASK 2: Now that we understand the issues , let's copy the project folder into another location and start adding Spring Security to the Minor Project 1 .
IMP : Change the db in the app.properties file.
HINT : cp -r ./Downloads/minor ./Downloads/minor2 (This command will copy the contents of the minor folder into the minor2 folder .)

TASK 3: Add the spring boot starter security maven dependency and the spring boot starter data redis dependency .(We will use Lettuce Core parent dependency of spring boot starter data redis this time .Last time we used Jedis.)

```
 <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
  </dependency>
```

TASK 4: Create Entity class MyUser which implements the UserDetails interface.

TASK 5: Let's decide on Data Modeling Relationships now .We know that both admin and student are myuser and will have their username , password and authority in MyUser table .But Which table will have PK and which table will have back referencing ?
HINT : Admin will be mapped to MyUser as One to One ,
Student will be mapped to MyUser as One to One

TASK 6: Make sure to add the JsonIgnoreProperties to the newly added attributes in Student,Admin and MyUser table to avoid infinite looping while getting .

TASK 7: Create a common config class which extends the WebSecurityConfigureAdaptor. Here we will add our function returning beans for Security and Redis both .Right now lets only write the logic for the Authentication configure method .Also do not forget to create the Bean for Password Encoder.

TASK 8: Create a MyUserService which implements the UserDetailsService interface.

TASK 9: Create interface MyUserRepository which extends the JpaRepository<>

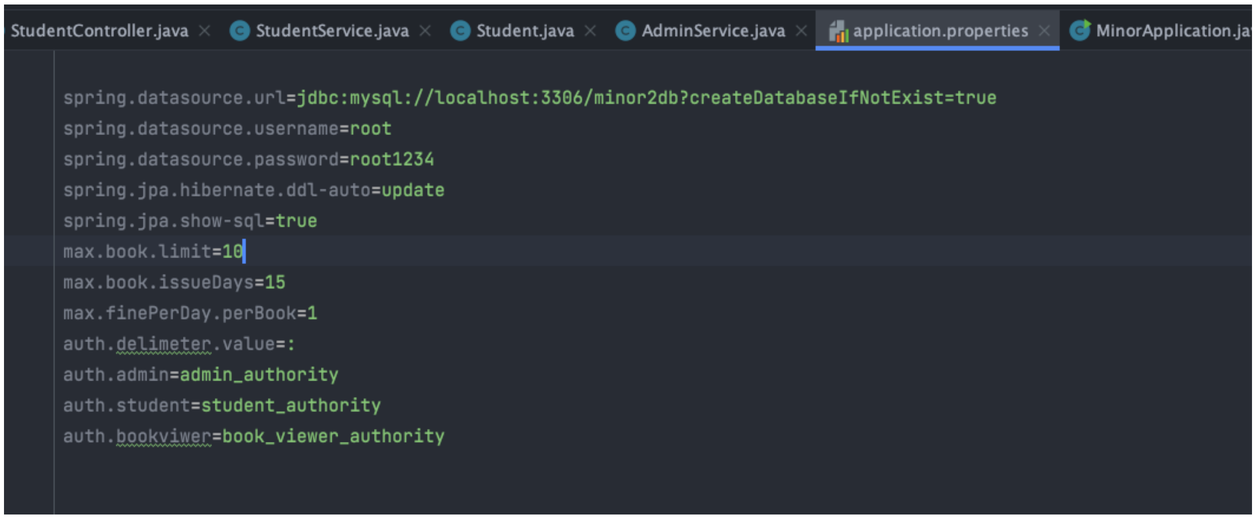
TASK 10: Now let's decide the authorities. For that let us open our controller class one by one and categorize who can hit which API or if any specific API should not be authenticated. Make sure to write new endpoints for apis in comments so as to generate correct antmatchers.

TASK 11: In Student controller , we have a get student API which takes a student id and returns the details of that student . This should only be allowed to hit by Admin . So we need to create a getStudent API here which does not take an id and returns the details of the logged in student by taking the id from the security context.

V IMP : The above getStudent api which will be accessed by that specific user will require use to get myuser object from the security context .But remember , this myuser is not a student . userId is not equal to studentId . So how can we fetch the student associated with the myuser object of our security context ?

Note : Only an admin should be allowed to create another admin . And no admin can view the details of other admin . So we will have to modify the getAdmin API in AdminController.Only Admin can create a new book .Both Student and admin can see the details of any book .

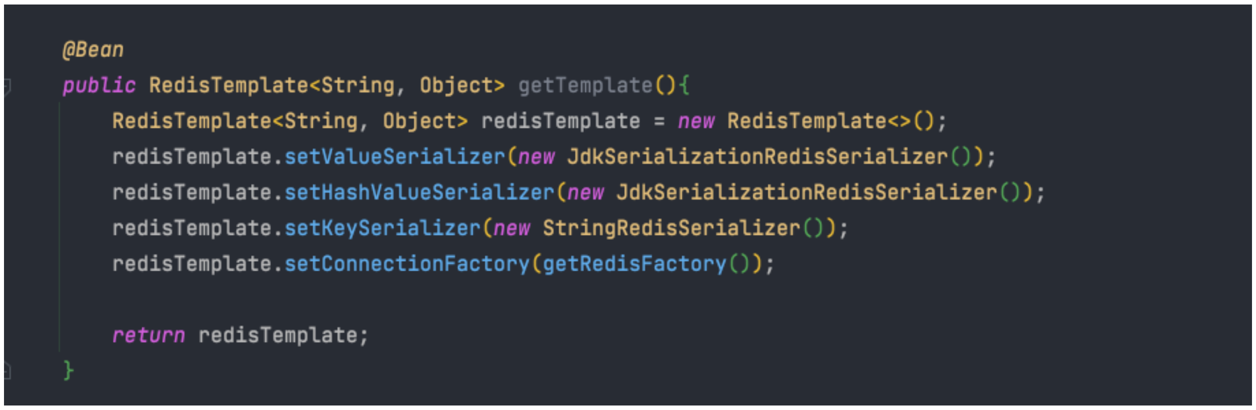
TASK 12: Let us define the authorities in our application.properties file according to actions .
HINT :



StudentController.java × StudentService.java × Student.java × AdminService.java × application.properties × MinorApplication.ja

```
spring.datasource.url=jdbc:mysql://localhost:3306/minor2db?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=root1234
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
max.book.limit=10
max.book.issueDays=15
max.finePerDay.perBook=1
auth.delimeter.value=:
auth.admin=admin_authority
auth.student=student_authority
auth.bookviwer=book_viewer_authority
```

TASK 13: Now let's create our ant matchers in the Authorisation configure method in the common class .



```
@Bean
public RedisTemplate<String, Object> getTemplate(){
    RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();
    redisTemplate.setValueSerializer(new JdkSerializationRedisSerializer());
    redisTemplate.setHashValueSerializer(new JdkSerializationRedisSerializer());
    redisTemplate.setKeySerializer(new StringRedisSerializer());
    redisTemplate.setConnectionFactory(getRedisFactory());

    return redisTemplate;
}
```

TASK 14: Let's modify our API's and request classes now .
Whenever a student /admin is created , we also need to create a myuser in the MyUser table . So in createRequests for admin and student , we will also take username and password .

IMP : The password of myUser should be encoded before storing in db.

TASK 15: While saving the student/admin in service class , make sure to create a myuser and save the myuser in MyUser table .Then before storing student/admin in their respective table , we need to attach a SAVED myuser object with id to student/admin.

IMP : Beware that the student we get from myUser.getStudent() is merely a proxy with studentId. Don't expect it to be a complete student object with bookList and all other things . Make sure to fetch the student id from myUser's student and then use this studentId to fetch complete student from studentService .

TASK 16: You will need to create an admin in the database (also user) through command line runner because it takes an admin to create another admin.

TASK 17: Let's test our application now for Spring Security and post this we will add caching . Let's check the desc of our tables .

Challenge : Why are we not able to make a successful POST request to create a student ?
HINT : csrf (Cross Site Request Forgery) and concept of safe(GET) and unsafe(POST,PUT) HTTP methods. The issue is likely due to CSRF protection. If users will not be using your application in a web browser, then it is safe to disable CSRF protection. Otherwise you should ensure to include the CSRF token in the request.Cross-site request forgery is a web security vulnerability that allows an attacker to induce users to perform actions that they do not intend to perform.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        // ...
        .csrf().disable();
}
```

https://www.geeksforgeeks.org/what-is-cross-site-request-forgery-csrf/

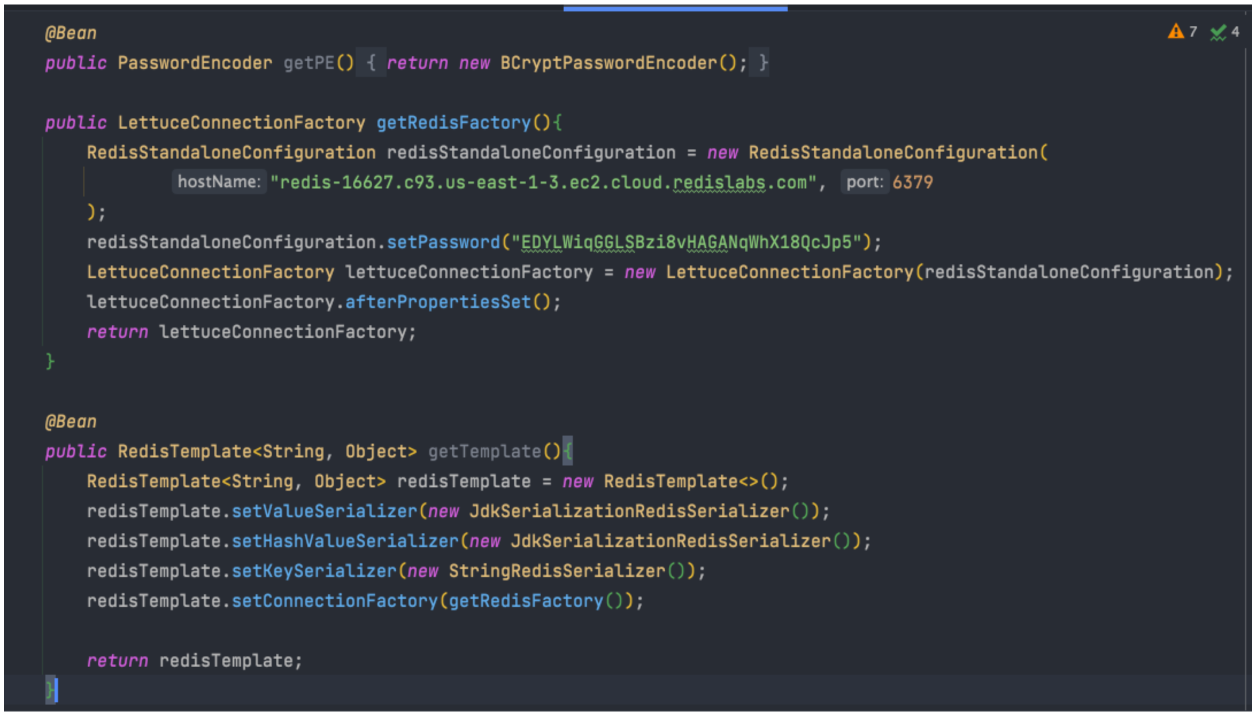
Challenge : How much time it takes to get an entity from cache ?

TASK 18: Lets Implement Cache Now !
Make sure all our entity classes implement Serializable interface .

Challenge : What is a marker interface ?
HINT : https://www.geeksforgeeks.org/marker-interface-java/

TASK 19: Define the getRedisFactory() function and getRedisTemplate() Bean function inside your common config class .

HINT :



```java
@Bean
public PasswordEncoder getPE() { return new BCryptPasswordEncoder(); }

public LettuceConnectionFactory getRedisFactory(){
    RedisStandaloneConfiguration redisStandaloneConfiguration = new RedisStandaloneConfiguration(
            hostName: "redis-16627.c93.us-east-1-3.ec2.cloud.redislabs.com",  port: 6379
    );
    redisStandaloneConfiguration.setPassword("EDYLWiqGGLSBzi8vHAGANqWhX18QcJp5");
    LettuceConnectionFactory lettuceConnectionFactory = new LettuceConnectionFactory(redisStandaloneConfiguration);
    lettuceConnectionFactory.afterPropertiesSet();
    return lettuceConnectionFactory;
}

@Bean
public RedisTemplate<String, Object> getTemplate(){
    RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();
    redisTemplate.setValueSerializer(new JdkSerializationRedisSerializer());
    redisTemplate.setHashValueSerializer(new JdkSerializationRedisSerializer());
    redisTemplate.setKeySerializer(new StringRedisSerializer());
    redisTemplate.setConnectionFactory(getRedisFactory());

    return redisTemplate;
}
```

TASK 20: Create a StudentCacheRepository class for caching the details of students .

TASK 21: Lets save the student to cache when its created .(saveStudentInCache function of StudentCacheRepository will be called in Student Service class)

Challenge : We should save in the cache before saving in the db or after saving in the db ?

IMP : Make sure to save in cache / attach a user object to a student/admin object only after getting the saved user object from db which will have an id .

Challenge : Will we try to get from the database first or from the cache first ?

Session 15

HINT :
/*
    1. Search in the cache, if found return from here itself, otherwise move to next steps
    2. Get from db and save it in the cache
    3. Return the retrieved object
*/

TASK 22: Now Let's get the student from cache when the mapping for student is hit .Create a getStudentFromCache(int studentId) function in StudentCacheRepository which will be called in Student Service class.

TASK 23: Perform the caching mechanism for Admin and Student as well .

TASK 24: Now let's test our application .Also , open in monitor mode for redis cli connection to redis labs server in terminal .
HINT : Open terminal and run the following

```
>redis-cli -h <host> -p <port> -a <default redis password>
>monitor
```

Challenge : How much time it takes to get an entity from cache ?

IMPORTANT ANNOUNCEMENT : In the next session , we are going to study OAuth2 with Github in Spring Boot . I would suggest coming to the next session after having a very basic knowledge of HTML , CSS and Javascript (JQuery) . You can watch short crash courses for beginners on these topics . They will be sufficient to understanding the frontend code we will be using : https://github.com/spring-guides/tut-spring-boot-oauth2
(Please make sure to clone this repo before the next session .)

HOMEWORK TIME :

1.Develop the functionality where Students can place a request to change password and admin will have to process it .

2.Develop the functionality where Student can place a request to delete his account.

3. Create StudentResponse instead of Student when GET /student and GET /student/id is hit so as to not return requestList of that student .

4. Right now in all service classes we are using Field Injection . How to do the same through Constructor Dependency Injection.

5. Are calls from the local database faster or outside located central cache server faster ? Why ?
HINT : Try to perform the get call with local redis server and local db and now compare .