Back End Development 1

Sesi 18

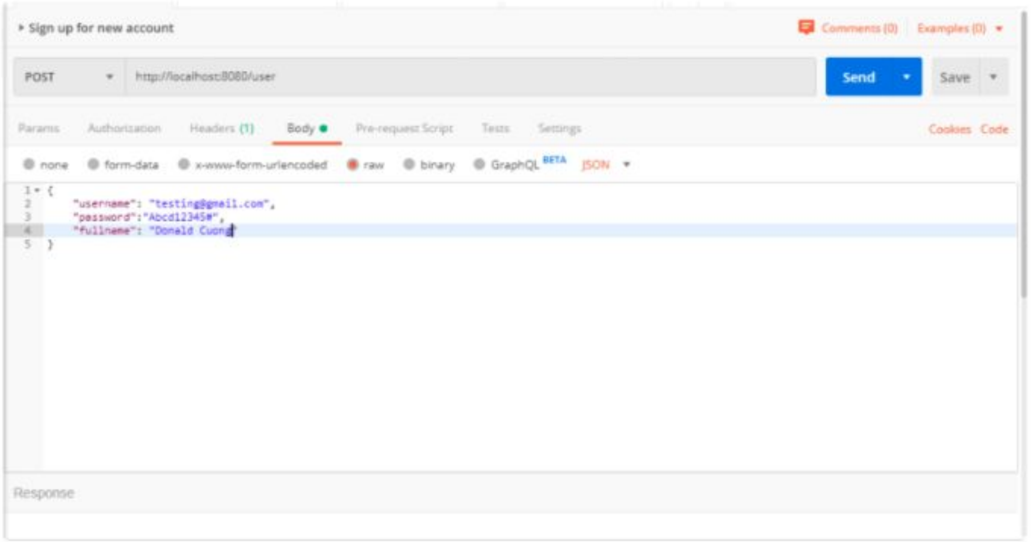
Springboot – OAuth2 With JWT

Springboot - OAUTH2 With JWT - Sesi 18

Implementations User Modules

Pertemuan sebelumnya kita sudah membahas step by step membuat otentikasi pada rest api kita, maka pada sesi kali ini kita akan membuat REST API Simple login dan request blog content dengan implementasi JWT.

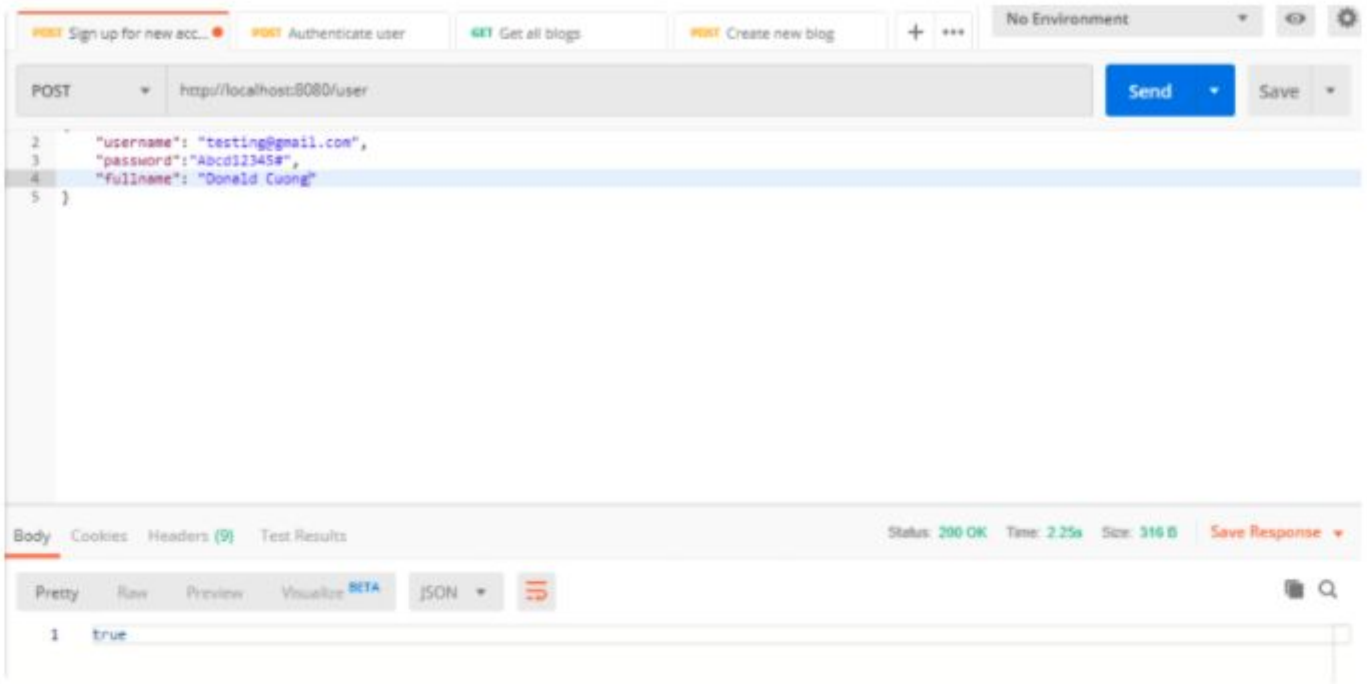
1. Create New User



POST http://localhost:8080/user

```
{
    "username": "testing@gmail.com",
    "password":"Abcd12345#",
    "fullname": "Donald Cuong"
}
```

2. Create New User Success



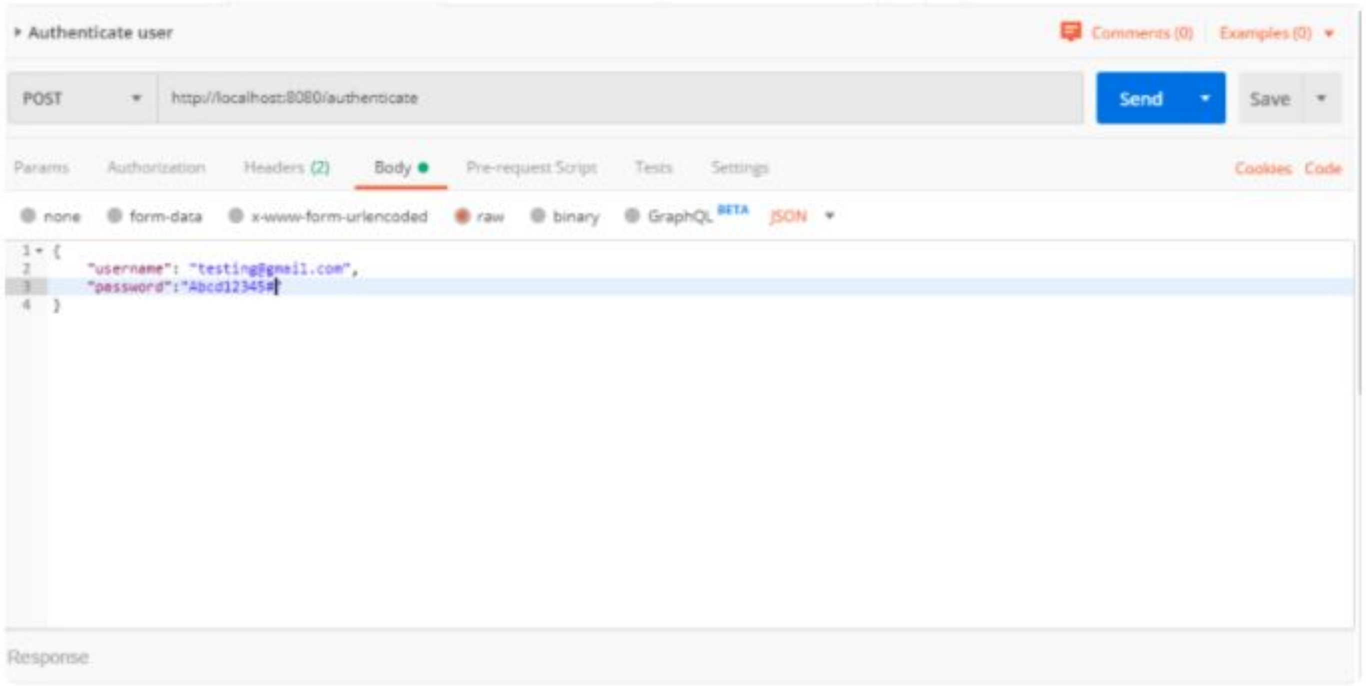
POST http://localhost:8080/user

```
    "username": "testing@gmail.com",
    "password":"Abcd12345#",
    "fullname": "Donald Cuong"
}
```

Status: 200 OK Time: 2.25s Size: 316 B

```
true
```

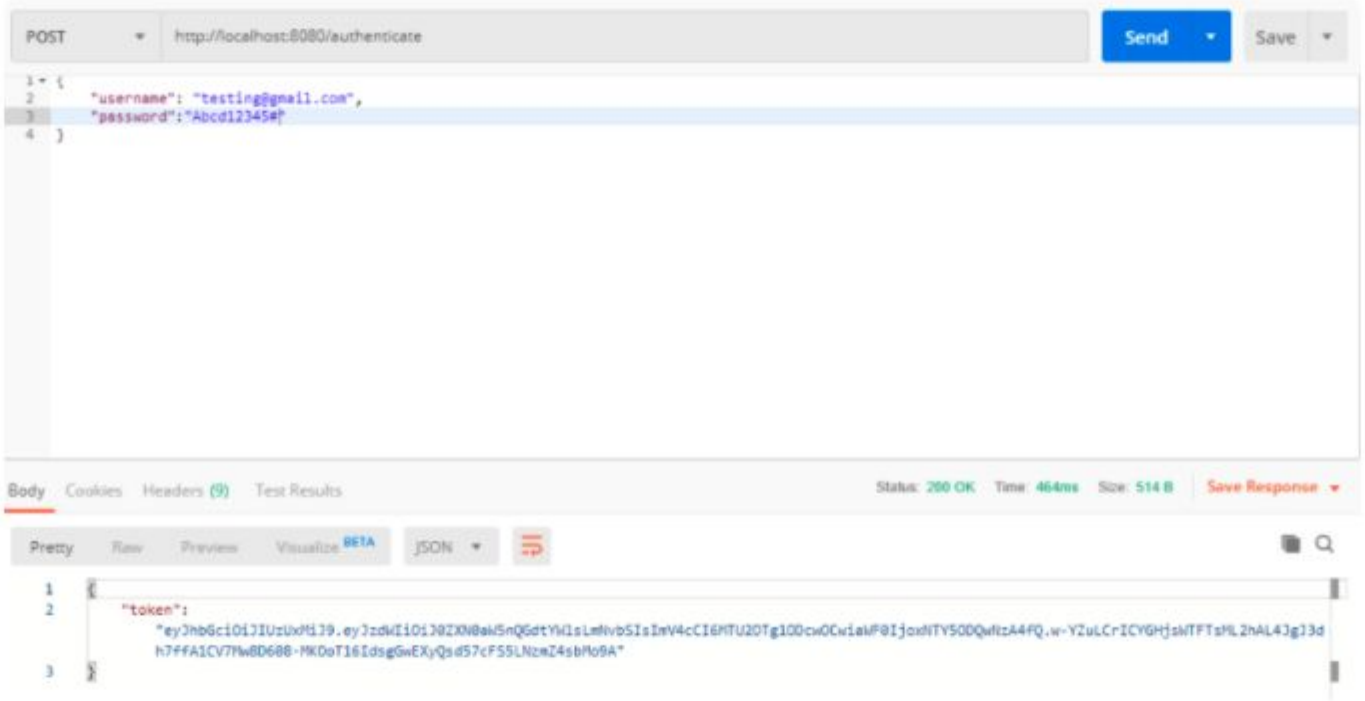
3. Authenticate user credentials :



POST http://localhost:8080/authenticate

```
{
    "username": "testing@gmail.com",
    "password":"Abcd12345#"
}
```

4. Authenticate user credentials success :



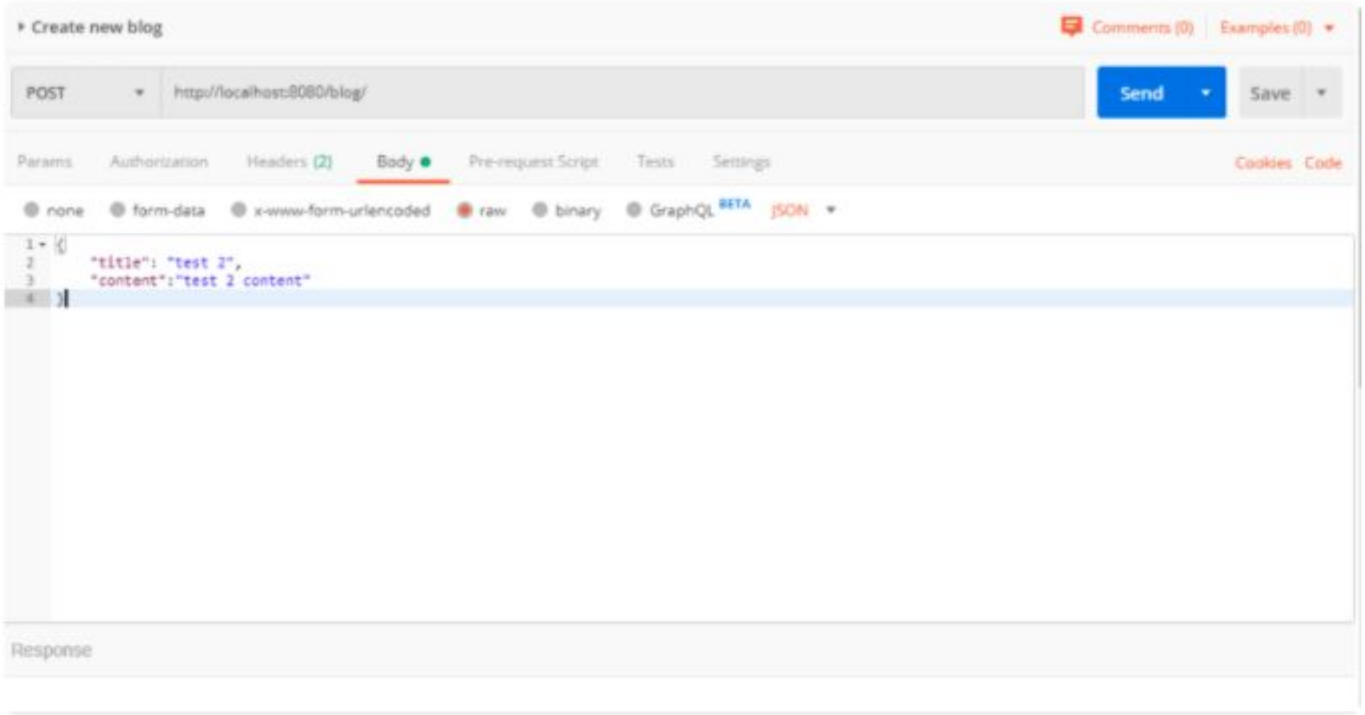
http://localhost:8080/authenticate

```
{
    "username": "testing@gmail.com",
    "password":"Abcd12345#"
}
```

Status: 200 OK   Time: 464ms   Size: 514 B

```
{
    "token":
        "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ0ZXN0aW5nQGdtYWlsLmNvbSIsImV4cCI6MTU2OTg1ODcwOCwiaWF0IjoxNTY5ODQwNzA4fQ.w-YZuLCrICYGHjsWTFTsML2hAL4JgJ3d
        h7ffA1CV7Mw8D608-MKDoT16IdsgGwEXyQsd57cF55LNzmZ4sbMo9A"
}
```

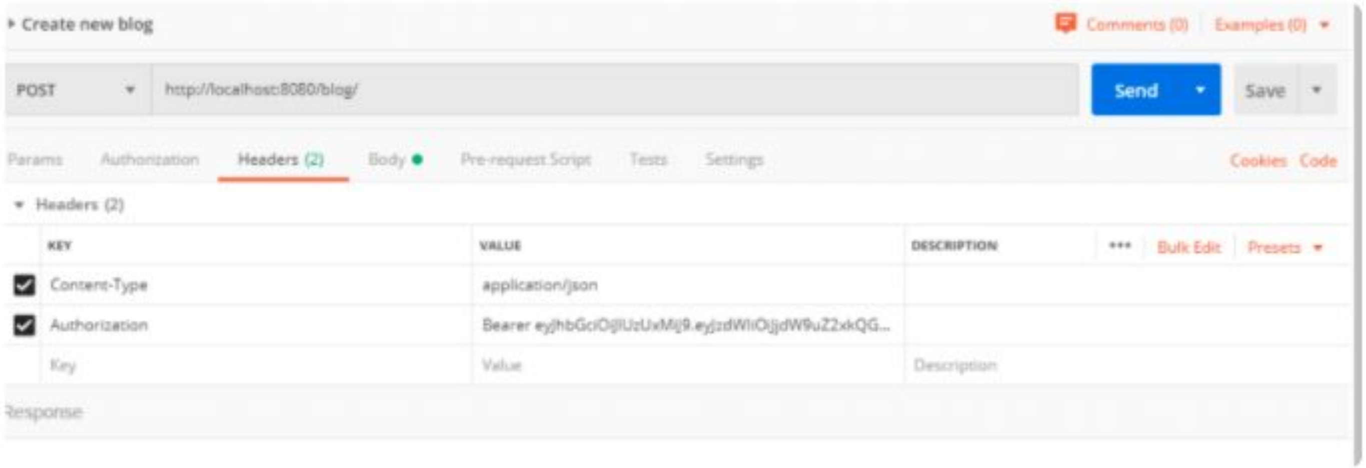
5. Blog Request Body



```
{
    "title": "test 2",
    "content":"test 2 content"
}
```

HACKTIV8

6. Blog Request Header

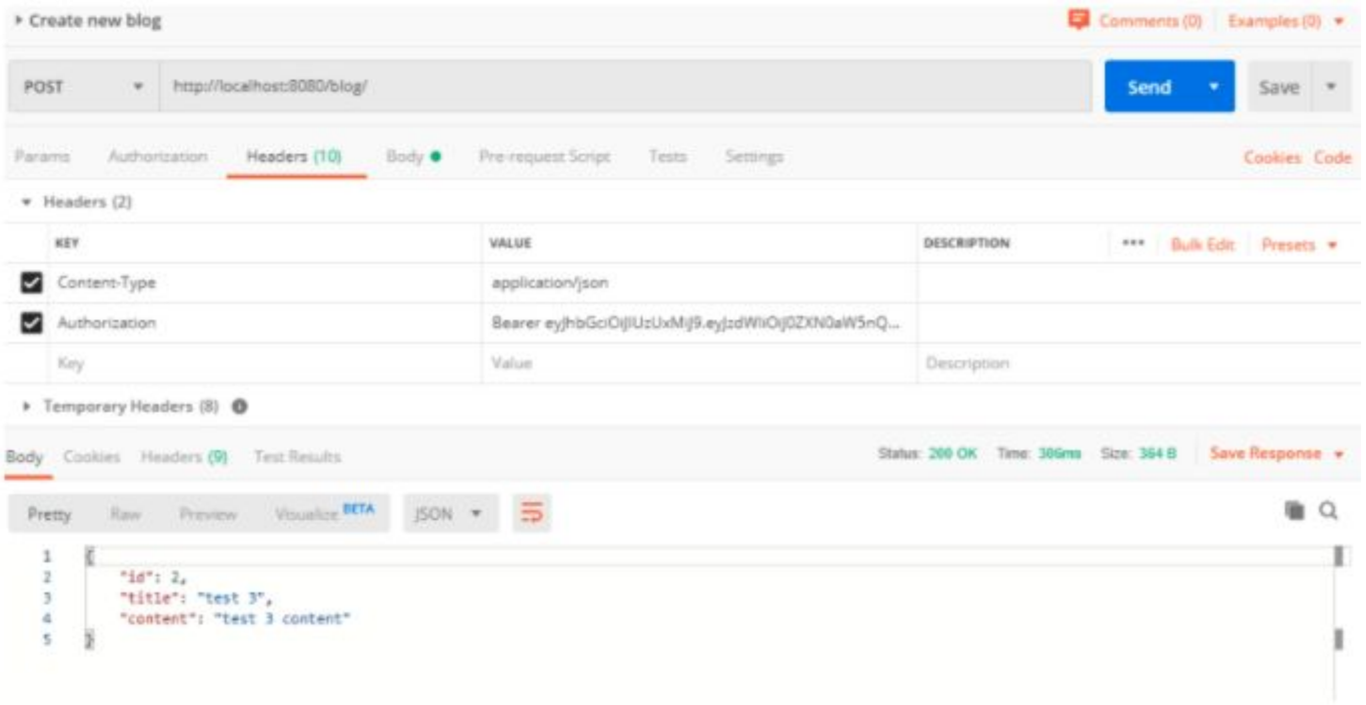


POST http://localhost:8080/blog/

Headers (2)

	KEY	VALUE	DESCRIPTION
✓	Content-Type	application/json	
✓	Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJjdW9uZ2xkQG...	
	Key	Value	Description

7. Blog Request Response


Create new blog

POST http://localhost:8080/blog/

Headers (2)

KEY	VALUE	DESCRIPTION
Content-Type	application/json	
Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ0ZXN0aW5nQ...	

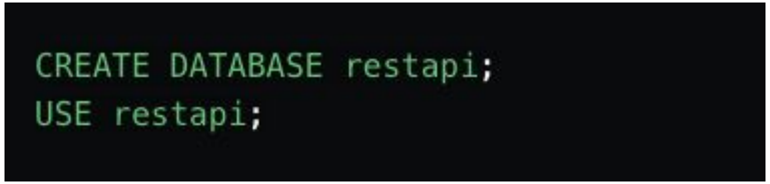
Temporary Headers (8)

Status: 200 OK Time: 306ms Size: 364 B

```
{
    "id": 2,
    "title": "test 3",
    "content": "test 3 content"
}
```

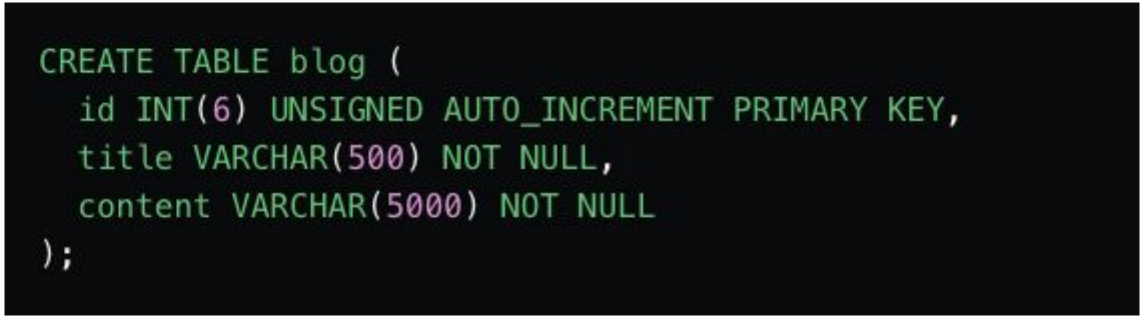
1. MySQL database dan table using SQL Query

- Create new database



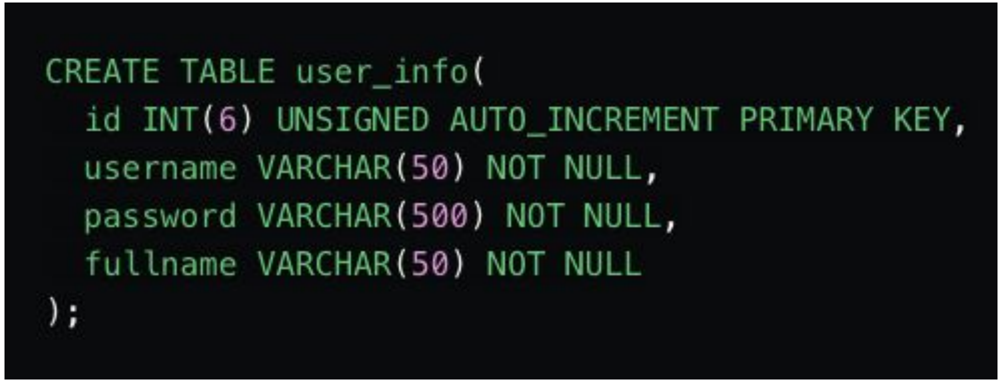
```
CREATE DATABASE restapi;
USE restapi;
```

- Create new table for blog



```
CREATE TABLE blog (
  id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(500) NOT NULL,
  content VARCHAR(5000) NOT NULL
);
```

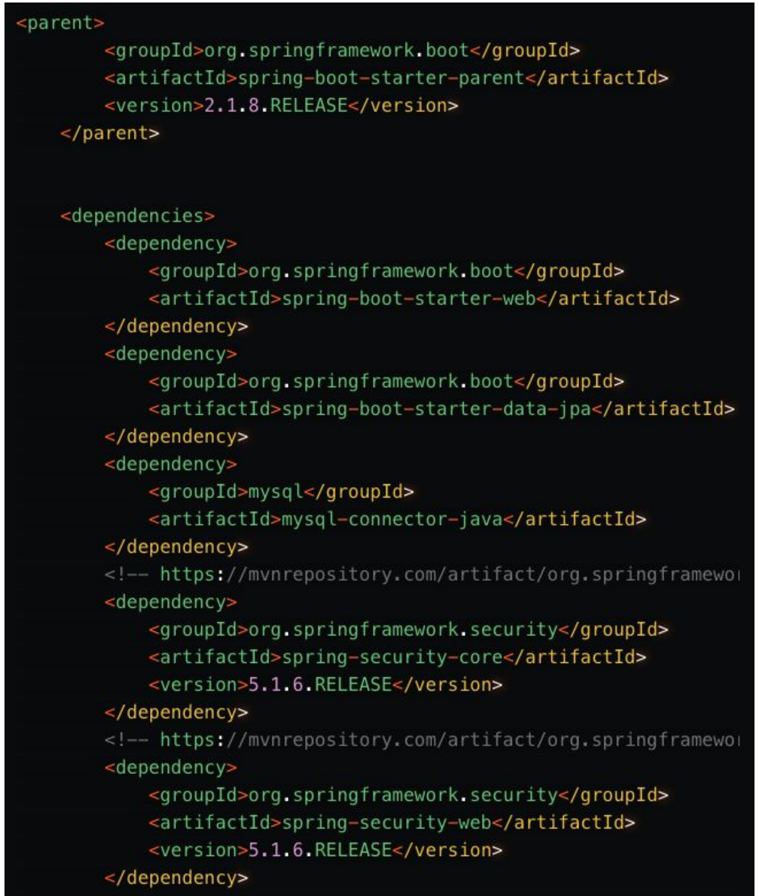
- Create new table for userinfo:



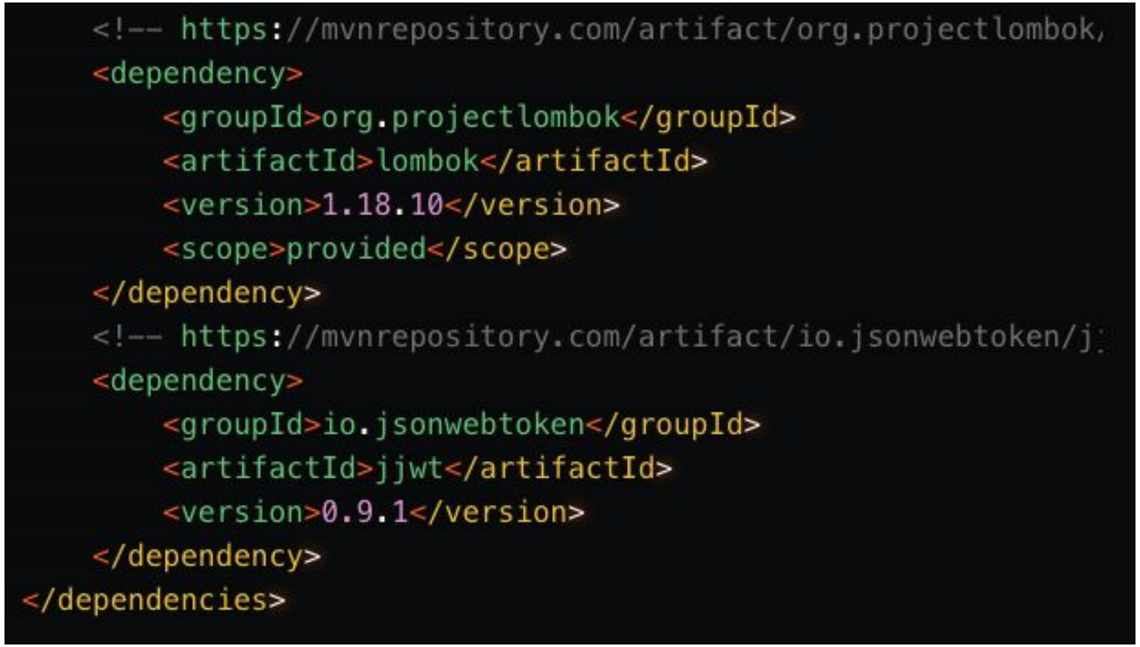
```
CREATE TABLE user_info(
  id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(50) NOT NULL,
  password VARCHAR(500) NOT NULL,
  fullname VARCHAR(50) NOT NULL
);
```

Springboot - OAUTH2 With JWT - Sesi 18

Dependencies



```
<parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.8.RELEASE</version>
    </parent>


    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.springframewo
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-core</artifactId>
            <version>5.1.6.RELEASE</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.springframewo
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
            <version>5.1.6.RELEASE</version>
        </dependency>
```



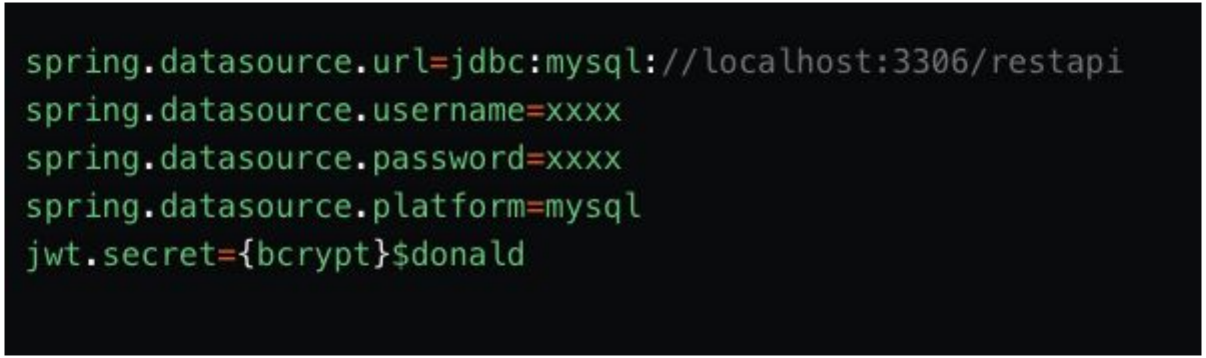
```
    <!-- https://mvnrepository.com/artifact/org.projectlombok,
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.10</version>
        <scope>provided</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/io.jsonwebtoken/j
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt</artifactId>
        <version>0.9.1</version>
    </dependency>
</dependencies>
```

Kita membutuhkan :

1. spring-boot-starter untuk create REST API
2. Mysql-connector-java untuk koneksi DB MySQL
3. Spring-Security untuk set up Auth
4. Jsonwebtoken untuk implementasi JWT dengan Auth

2. Project Structure

- resources: We will define the properties for our project in application.properties



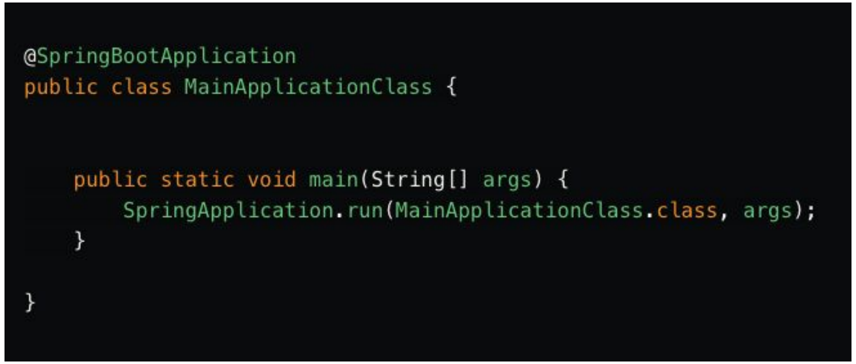
```
spring.datasource.url=jdbc:mysql://localhost:3306/restapi
spring.datasource.username=xxxx
spring.datasource.password=xxxx
spring.datasource.platform=mysql
jwt.secret={bcrypt}$donald
```

Spring.datasource digunakan untuk provide info terkait database pada app REST API yang akan kita buat, untuk meng-koneksi kan nya jangan lupa isi username dan password.

- project packages:

+) config:
Used to store config files for our project.
+) controller:
Used to define controller class for Authentication, CRUD for Blog content, Create new user
+) exceptions:
Define base error handles and exception for validate data
+) model:
Create model for Blog Entity, UserInfo Entity, JwtRequest and JwtResponse
+) repository:
Create Blog and UserInfo repository to interact with MySQL database using JPA
+) service:
Create JwtUserDetailsService to check whether the username is existed in database or not

- MainApplicationClass to run SpringBootApplication:



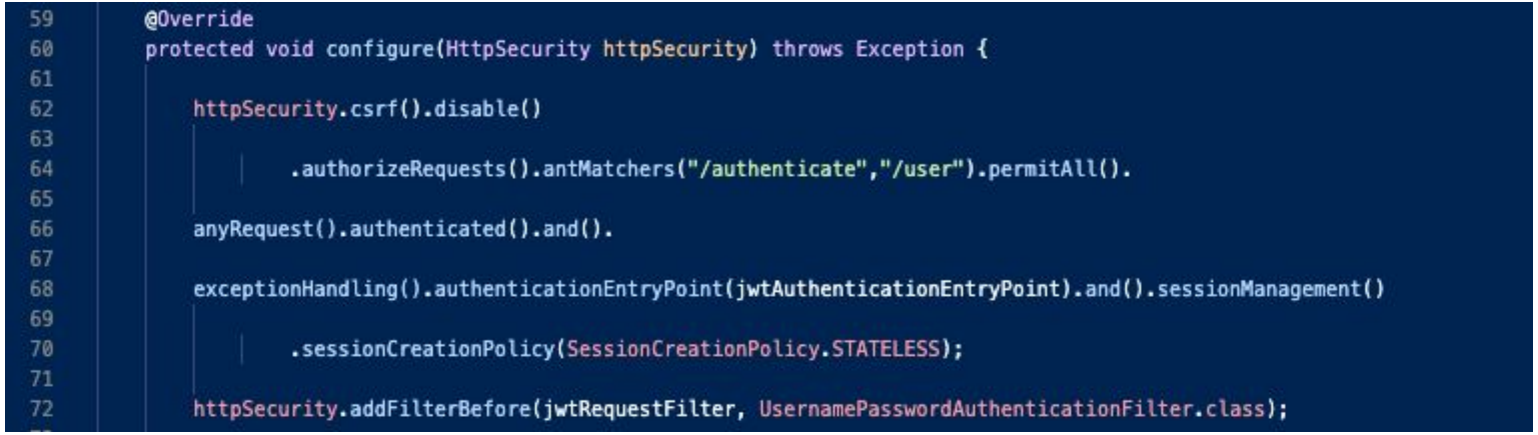
```java
@SpringBootApplication
public class MainApplicationClass {


    public static void main(String[] args) {
        SpringApplication.run(MainApplicationClass.class, args);
    }

}
```

3.What we will create:

- API to create new user in the application
- API to authen whether the user credentials is valid, if it is return token so that he or she can do other stuff
- API to create new blog post, view blog post, or update them.

So the API for create and authenticate credentials, will not have that authorization part to make sure anyone can access and perform these APIs.

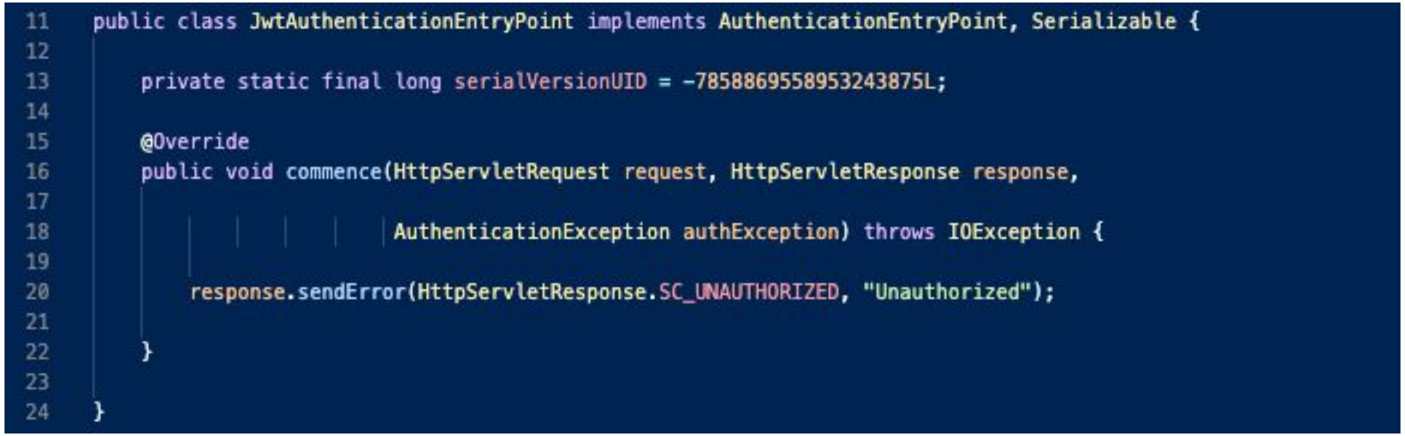
The API for interact with blogs will require authentication with jwt token.
To be able to do this, we would need to create configure method in our WebSecurityConfig class in config package:



```
    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {

        httpSecurity.csrf().disable()

                .authorizeRequests().antMatchers("/authenticate","/user").permitAll().

        anyRequest().authenticated().and().

        exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint).and().sessionManagement()

                .sessionCreationPolicy(SessionCreationPolicy.STATELESS);

        httpSecurity.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);
```

WebSecurityConfig.java

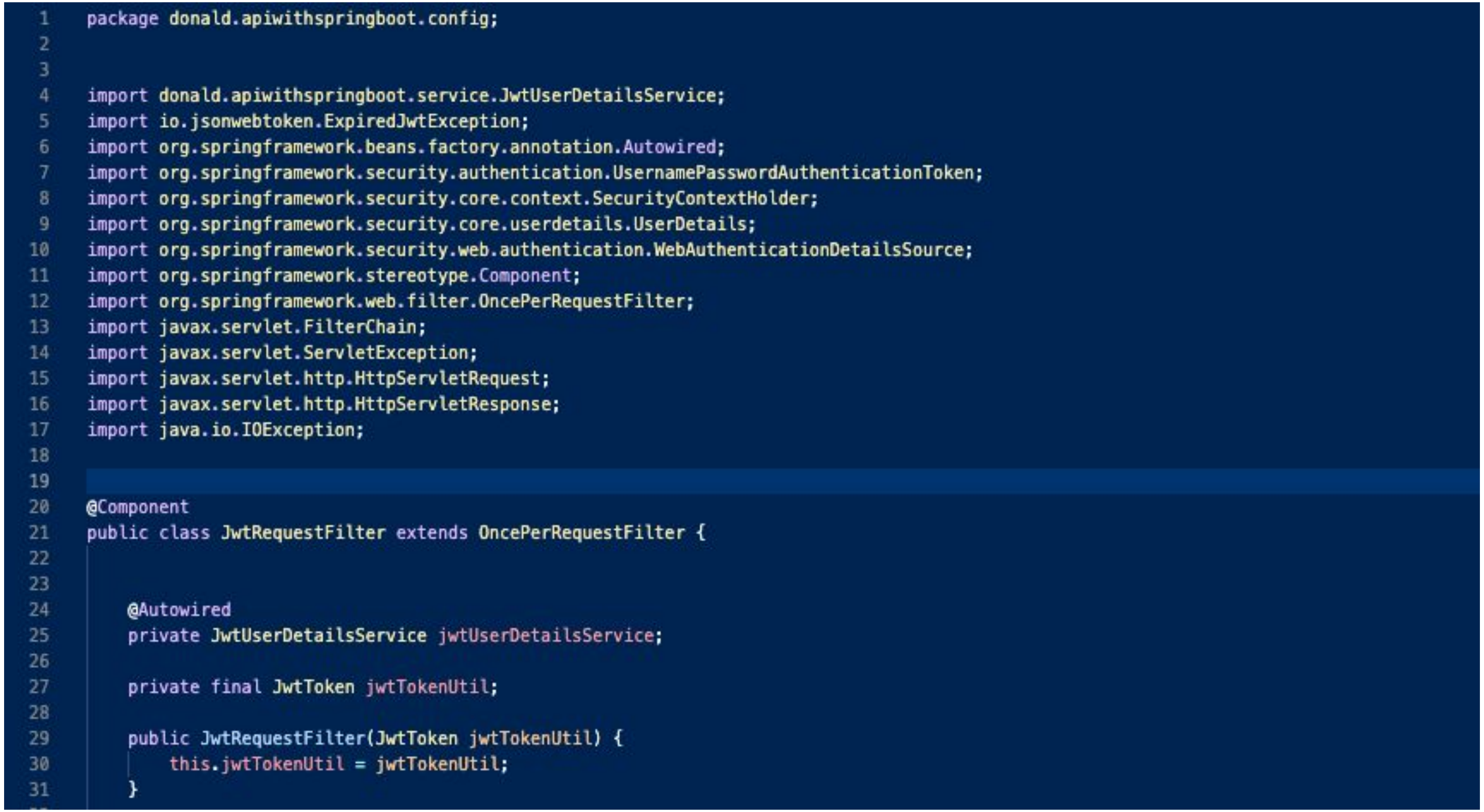
4. Configuration for jwt token
JwtAuthenticationEntryPoint to throw an unauthorized message if the user credential is NOT CORRECT.



```java
public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint, Serializable {

    private static final long serialVersionUID = -7858869558953243875L;

    @Override
    public void commence(HttpServletRequest request, HttpServletResponse response,

                        AuthenticationException authException) throws IOException {

        response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Unauthorized");

    }

}
```

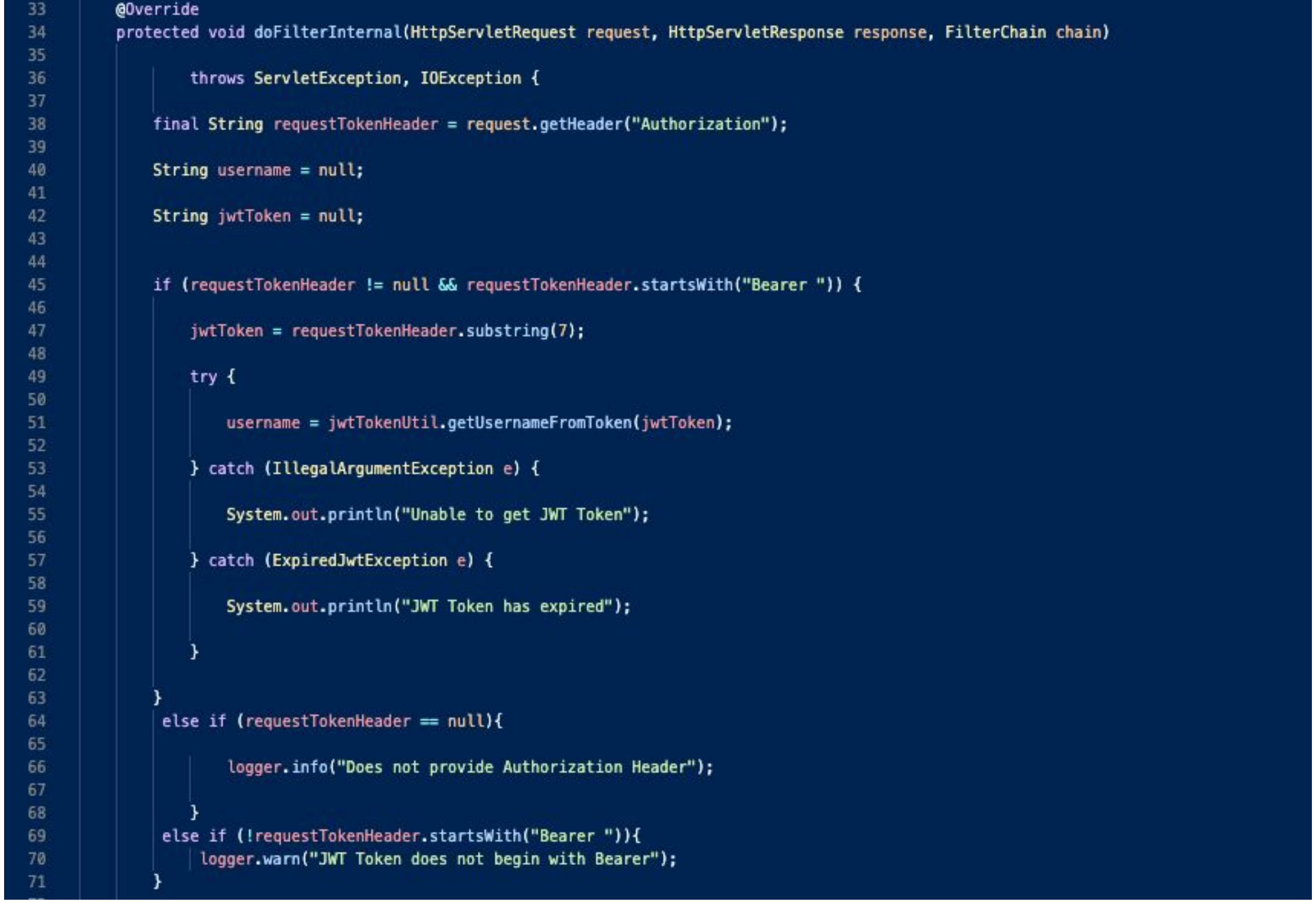
JwtAuthenticationEntryPoint.java

JwtRequestFilter to filter value of Authorization header: ***JwtRequestFilter.java***

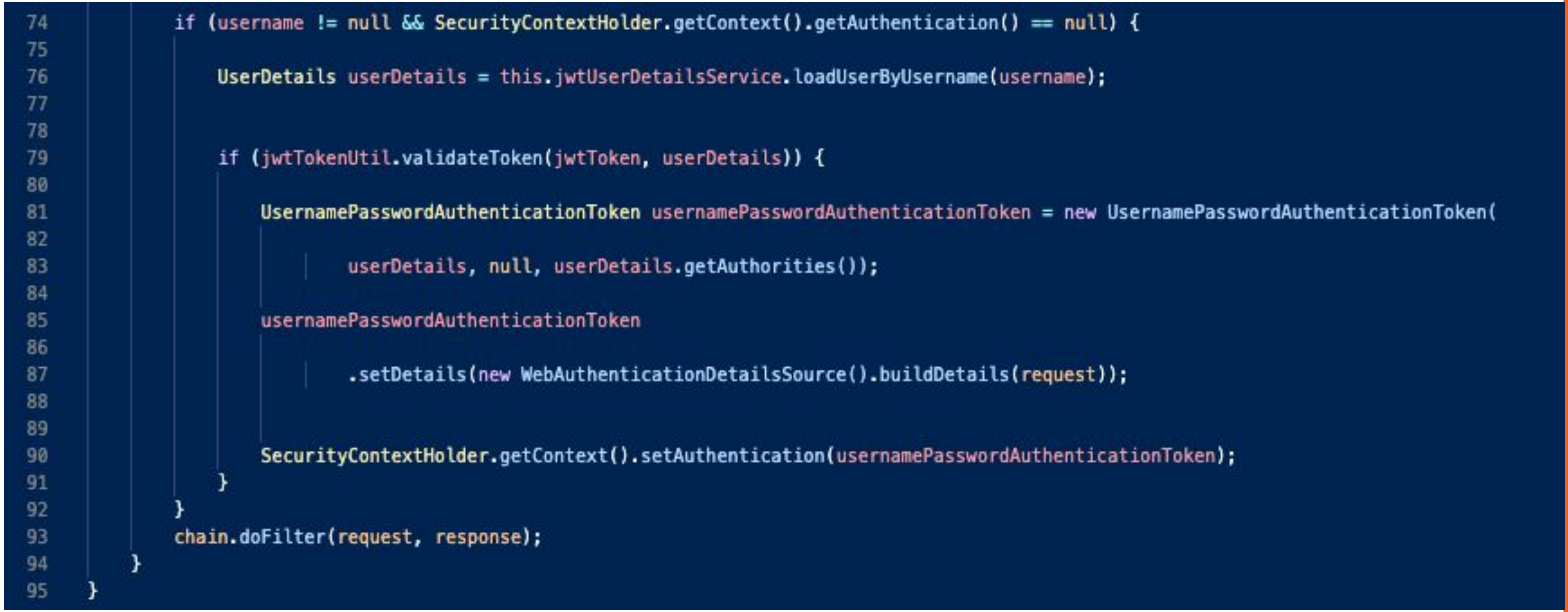


```java
package donald.apiwithspringboot.config;


import donald.apiwithspringboot.service.JwtUserDetailsService;
import io.jsonwebtoken.ExpiredJwtException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;


@Component
public class JwtRequestFilter extends OncePerRequestFilter {


    @Autowired
    private JwtUserDetailsService jwtUserDetailsService;

    private final JwtToken jwtTokenUtil;

    public JwtRequestFilter(JwtToken jwtTokenUtil) {
        this.jwtTokenUtil = jwtTokenUtil;
    }
```

HACKTIV8

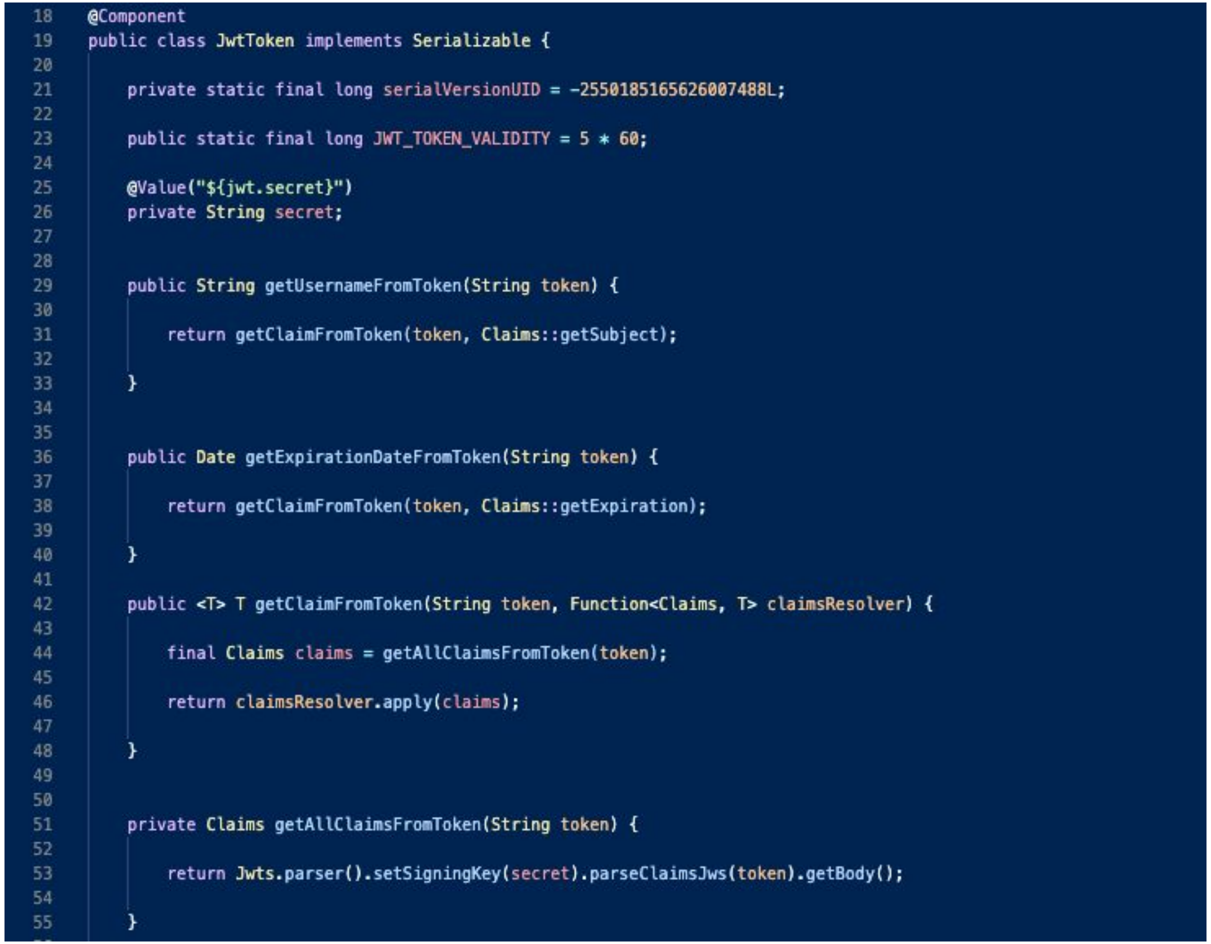


```java
    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)

            throws ServletException, IOException {

        final String requestTokenHeader = request.getHeader("Authorization");

        String username = null;

        String jwtToken = null;


        if (requestTokenHeader != null && requestTokenHeader.startsWith("Bearer ")) {

            jwtToken = requestTokenHeader.substring(7);

            try {

                username = jwtTokenUtil.getUsernameFromToken(jwtToken);

            } catch (IllegalArgumentException e) {

                System.out.println("Unable to get JWT Token");

            } catch (ExpiredJwtException e) {

                System.out.println("JWT Token has expired");

            }

        }
         else if (requestTokenHeader == null){

                logger.info("Does not provide Authorization Header");

            }
         else if (!requestTokenHeader.startsWith("Bearer ")){
              logger.warn("JWT Token does not begin with Bearer");
        }
```

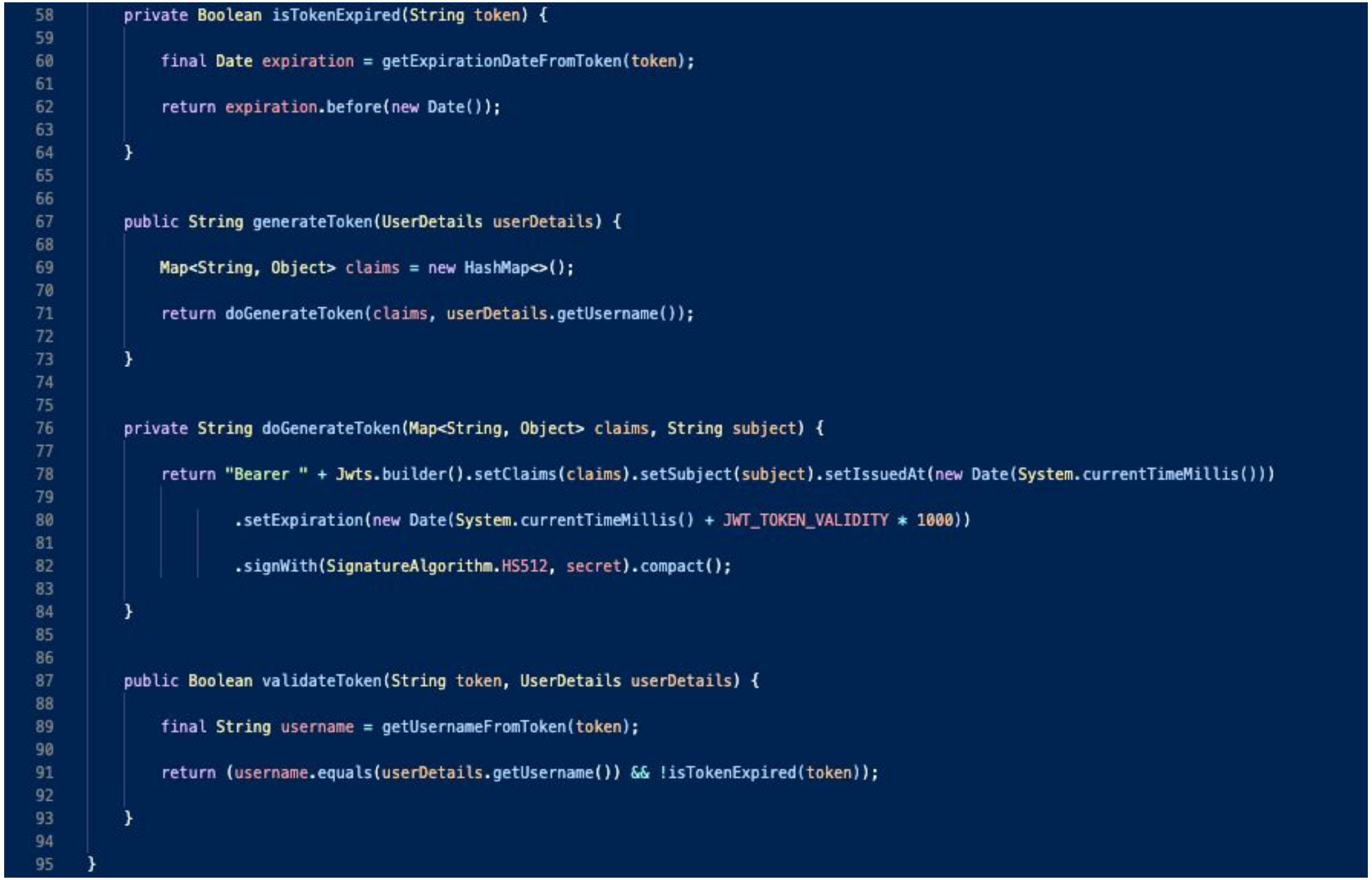


```
        if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {

            UserDetails userDetails = this.jwtUserDetailsService.loadUserByUsername(username);


            if (jwtTokenUtil.validateToken(jwtToken, userDetails)) {

                UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(

                        userDetails, null, userDetails.getAuthorities());

                usernamePasswordAuthenticationToken

                        .setDetails(new WebAuthenticationDetailsSource().buildDetails(request));


                SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);
            }
        }
        chain.doFilter(request, response);
    }
}
```

JwtToken class to generate jwt token: ***JwtToken.java***



```
@Component
public class JwtToken implements Serializable {

    private static final long serialVersionUID = -2550185165626007488L;

    public static final long JWT_TOKEN_VALIDITY = 5 * 60;

    @Value("${jwt.secret}")
    private String secret;


    public String getUsernameFromToken(String token) {

        return getClaimFromToken(token, Claims::getSubject);

    }


    public Date getExpirationDateFromToken(String token) {

        return getClaimFromToken(token, Claims::getExpiration);

    }

    public <T> T getClaimFromToken(String token, Function<Claims, T> claimsResolver) {

        final Claims claims = getAllClaimsFromToken(token);

        return claimsResolver.apply(claims);

    }


    private Claims getAllClaimsFromToken(String token) {

        return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();

    }
```

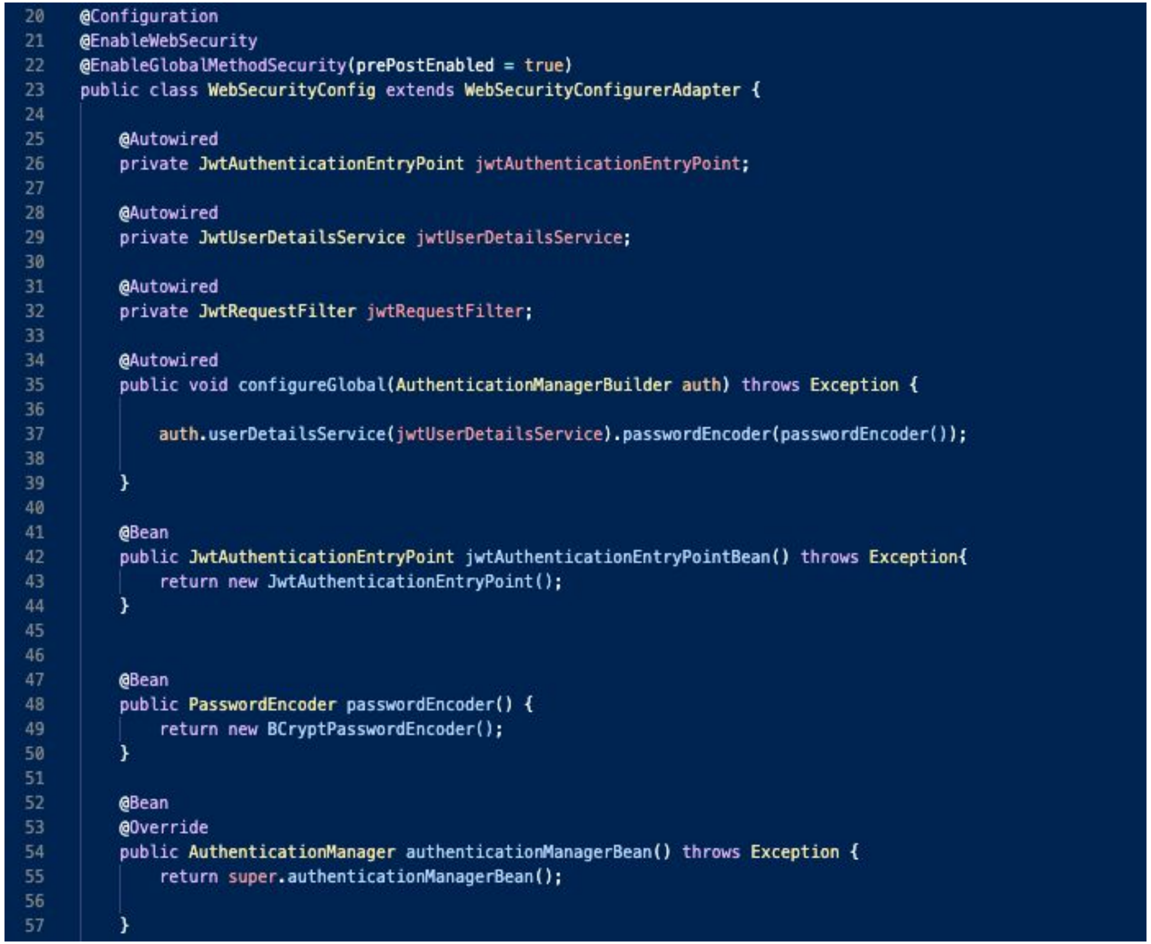


```java
    private Boolean isTokenExpired(String token) {

        final Date expiration = getExpirationDateFromToken(token);

        return expiration.before(new Date());

    }


    public String generateToken(UserDetails userDetails) {

        Map<String, Object> claims = new HashMap<>();

        return doGenerateToken(claims, userDetails.getUsername());

    }


    private String doGenerateToken(Map<String, Object> claims, String subject) {

        return "Bearer " + Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.currentTimeMillis()))

                .setExpiration(new Date(System.currentTimeMillis() + JWT_TOKEN_VALIDITY * 1000))

                .signWith(SignatureAlgorithm.HS512, secret).compact();

    }


    public Boolean validateToken(String token, UserDetails userDetails) {

        final String username = getUsernameFromToken(token);

        return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));

    }

}
```

HACKTIV8

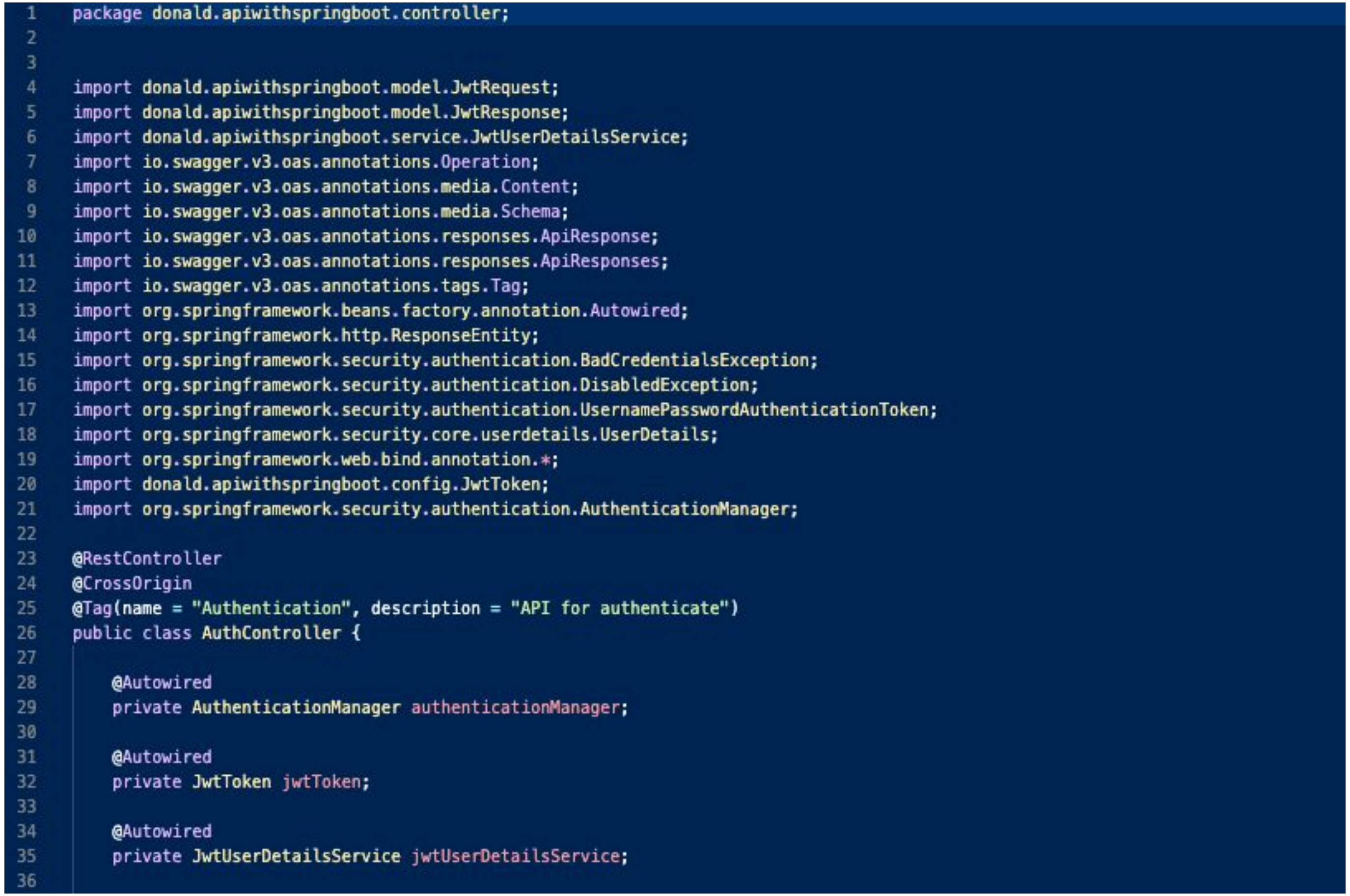
WebSecurityConfig to define the beans we would need and config path with authentication:

WebSecurityConfig.java

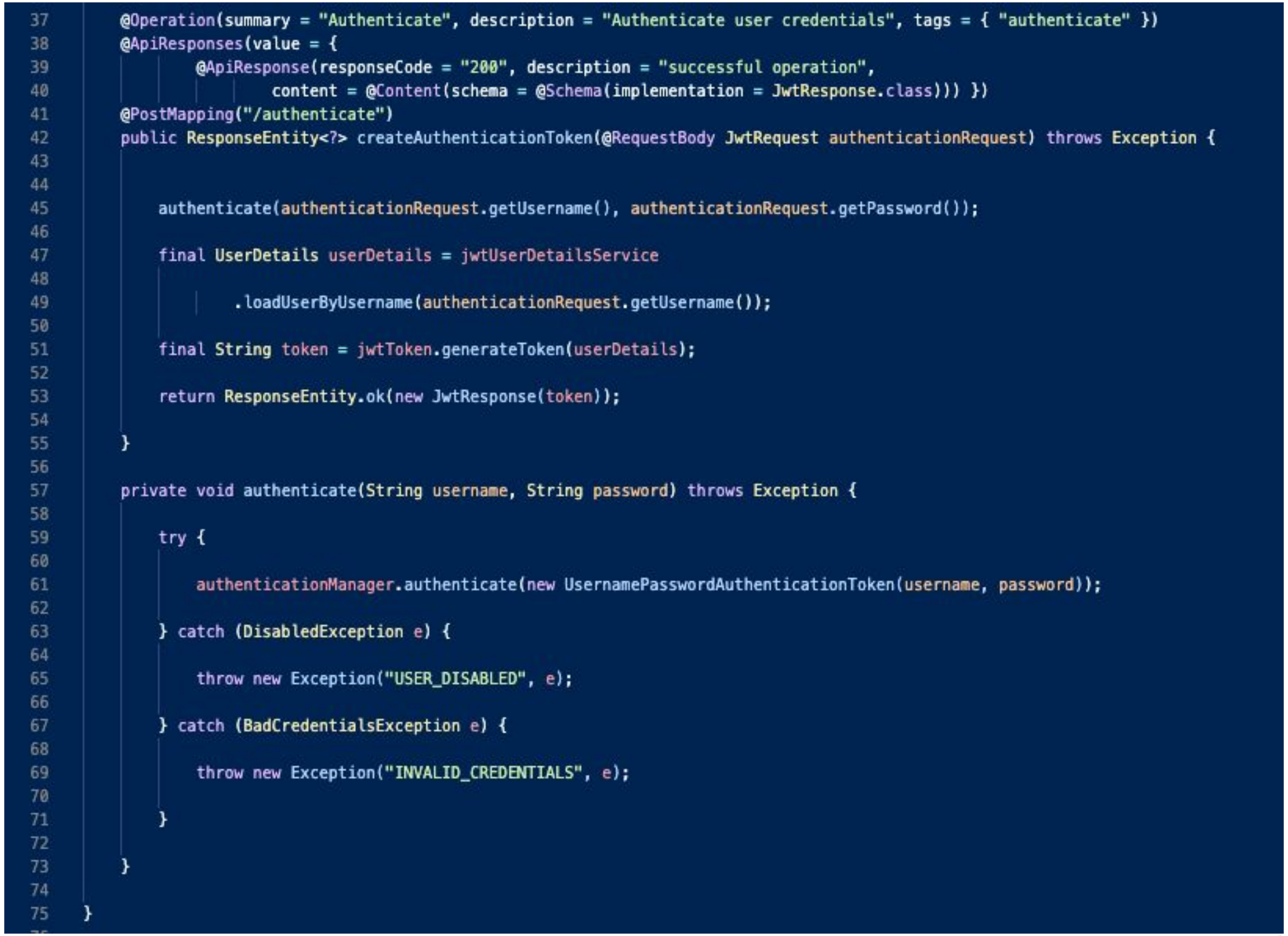


```java
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint;

    @Autowired
    private JwtUserDetailsService jwtUserDetailsService;

    @Autowired
    private JwtRequestFilter jwtRequestFilter;

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

        auth.userDetailsService(jwtUserDetailsService).passwordEncoder(passwordEncoder());

    }

    @Bean
    public JwtAuthenticationEntryPoint jwtAuthenticationEntryPointBean() throws Exception{
        return new JwtAuthenticationEntryPoint();
    }


    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();

    }
```

5.Controller:
AuthController to define the API to authenticate user credentials and response jwt token if correct:



```java
package donald.apiwithspringboot.controller;


import donald.apiwithspringboot.model.JwtRequest;
import donald.apiwithspringboot.model.JwtResponse;
import donald.apiwithspringboot.service.JwtUserDetailsService;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import io.swagger.v3.oas.annotations.tags.Tag;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.DisabledException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.web.bind.annotation.*;
import donald.apiwithspringboot.config.JwtToken;
import org.springframework.security.authentication.AuthenticationManager;

@RestController
@CrossOrigin
@Tag(name = "Authentication", description = "API for authenticate")
public class AuthController {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private JwtToken jwtToken;

    @Autowired
    private JwtUserDetailsService jwtUserDetailsService;

```

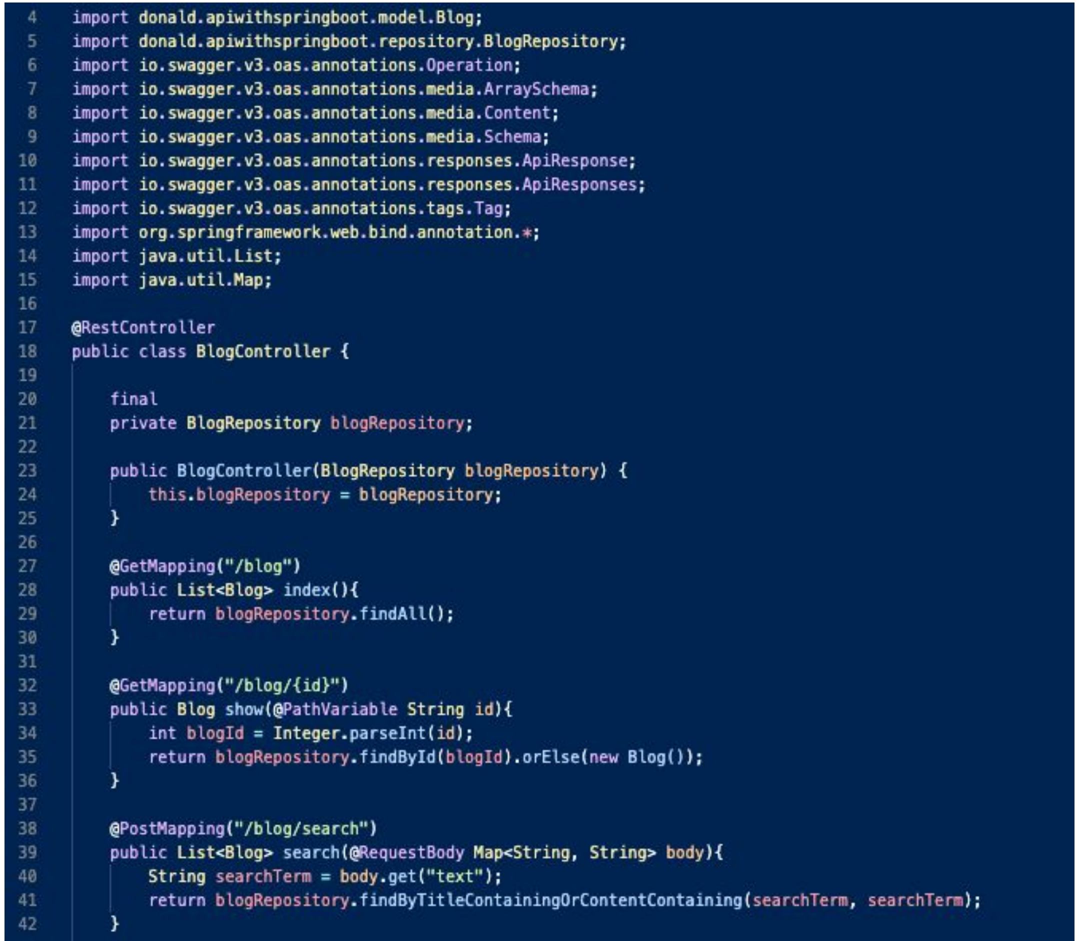


```java
    @Operation(summary = "Authenticate", description = "Authenticate user credentials", tags = { "authenticate" })
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "successful operation",
                    content = @Content(schema = @Schema(implementation = JwtResponse.class))) })
    @PostMapping("/authenticate")
    public ResponseEntity<?> createAuthenticationToken(@RequestBody JwtRequest authenticationRequest) throws Exception {


        authenticate(authenticationRequest.getUsername(), authenticationRequest.getPassword());

        final UserDetails userDetails = jwtUserDetailsService

                .loadUserByUsername(authenticationRequest.getUsername());

        final String token = jwtToken.generateToken(userDetails);

        return ResponseEntity.ok(new JwtResponse(token));

    }

    private void authenticate(String username, String password) throws Exception {

        try {

            authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username, password));

        } catch (DisabledException e) {

            throw new Exception("USER_DISABLED", e);

        } catch (BadCredentialsException e) {

            throw new Exception("INVALID_CREDENTIALS", e);

        }

    }

}
```

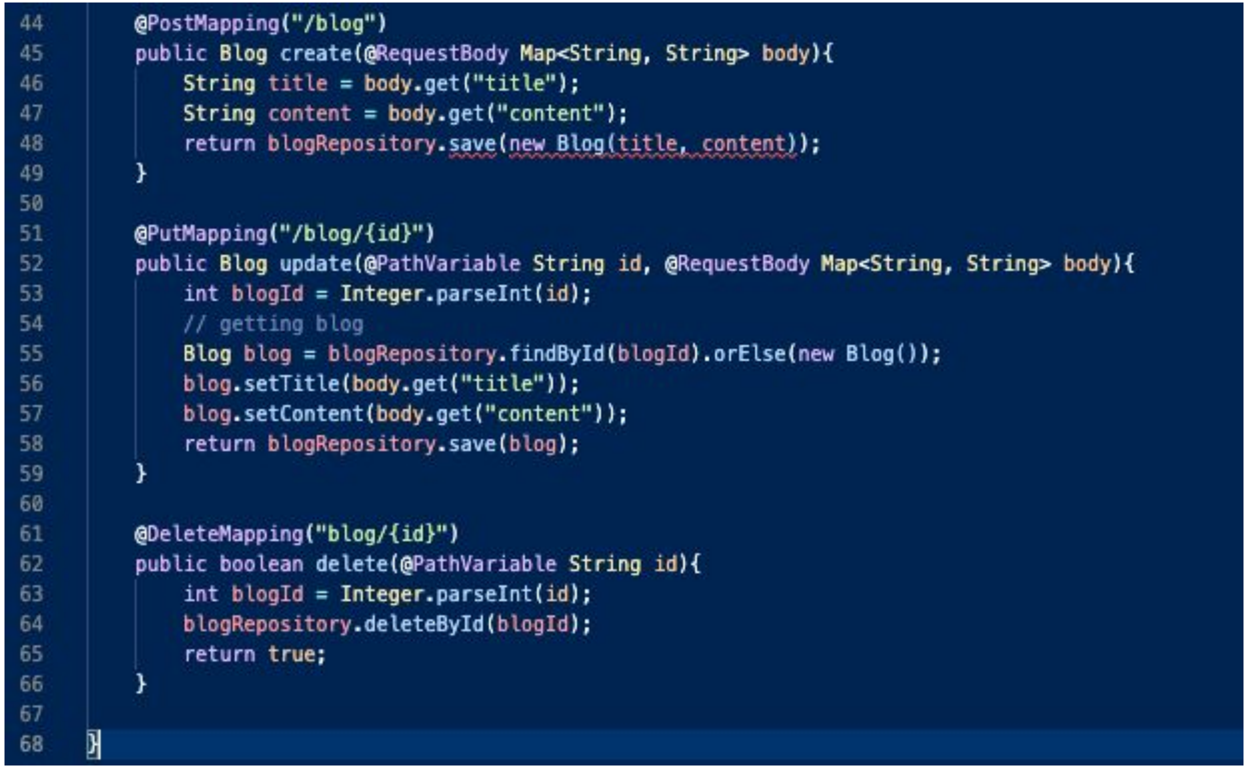
AuthController.java

BlogController class to create API for create new blog, modify blog content, view blog or update blog

BlogController.java



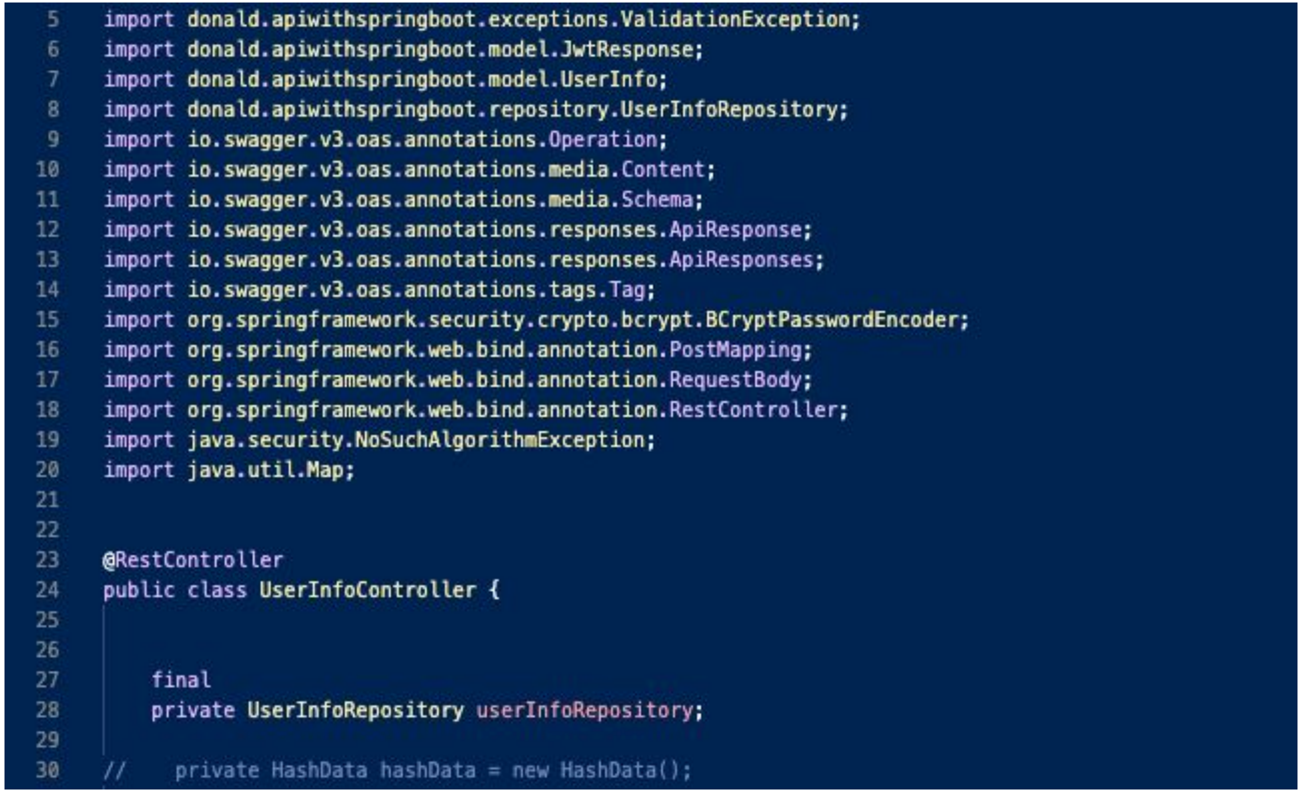
```java
import donald.apiwithspringboot.model.Blog;
import donald.apiwithspringboot.repository.BlogRepository;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.media.ArraySchema;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import io.swagger.v3.oas.annotations.tags.Tag;
import org.springframework.web.bind.annotation.*;
import java.util.List;
import java.util.Map;

@RestController
public class BlogController {

    final
    private BlogRepository blogRepository;

    public BlogController(BlogRepository blogRepository) {
        this.blogRepository = blogRepository;
    }

    @GetMapping("/blog")
    public List<Blog> index(){
        return blogRepository.findAll();
    }

    @GetMapping("/blog/{id}")
    public Blog show(@PathVariable String id){
        int blogId = Integer.parseInt(id);
        return blogRepository.findById(blogId).orElse(new Blog());
    }

    @PostMapping("/blog/search")
    public List<Blog> search(@RequestBody Map<String, String> body){
        String searchTerm = body.get("text");
        return blogRepository.findByTitleContainingOrContentContaining(searchTerm, searchTerm);
    }
```



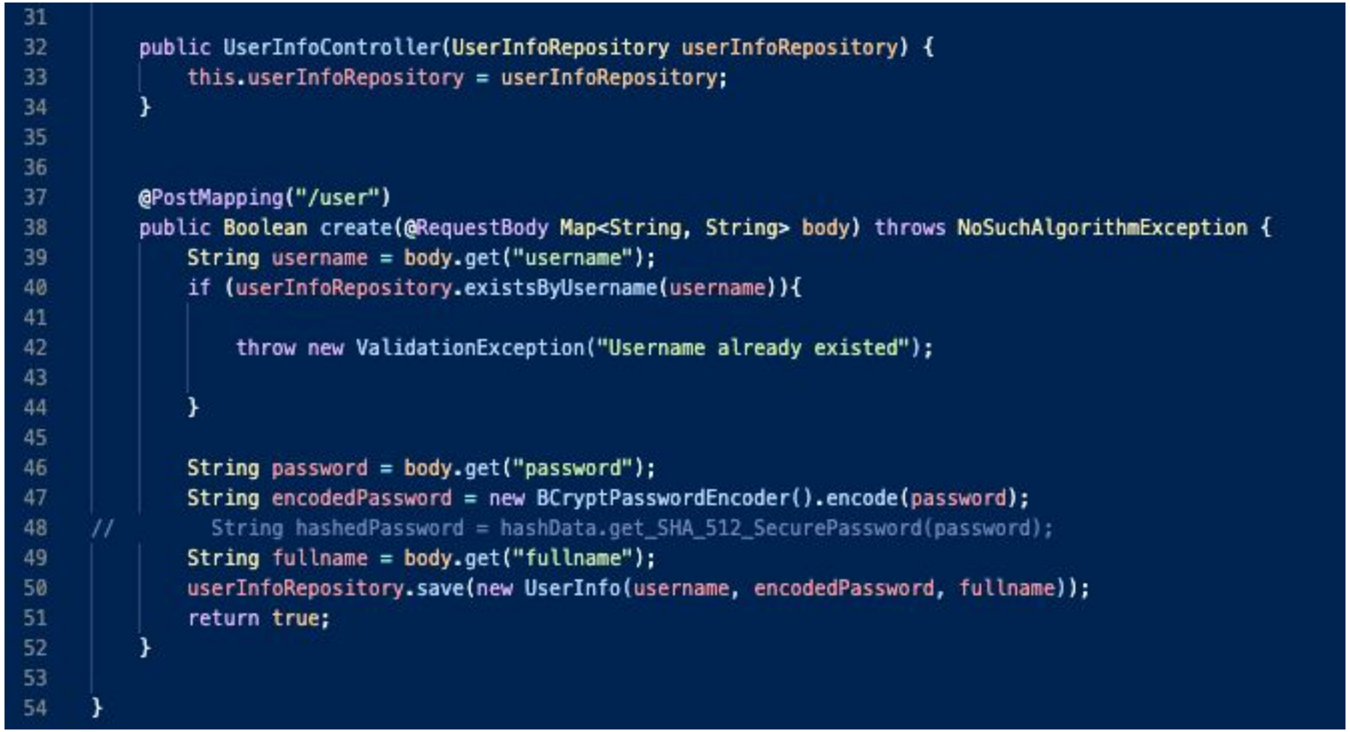
```java
    @PostMapping("/blog")
    public Blog create(@RequestBody Map<String, String> body){
        String title = body.get("title");
        String content = body.get("content");
        return blogRepository.save(new Blog(title, content));
    }

    @PutMapping("/blog/{id}")
    public Blog update(@PathVariable String id, @RequestBody Map<String, String> body){
        int blogId = Integer.parseInt(id);
        // getting blog
        Blog blog = blogRepository.findById(blogId).orElse(new Blog());
        blog.setTitle(body.get("title"));
        blog.setContent(body.get("content"));
        return blogRepository.save(blog);
    }

    @DeleteMapping("blog/{id}")
    public boolean delete(@PathVariable String id){
        int blogId = Integer.parseInt(id);
        blogRepository.deleteById(blogId);
        return true;
    }

}
```

UserInfoController to create API create new user and insert it into database with password is encoded with BCryptPasswordEncoder:

UserInfoController.java



```java
import donald.apiwithspringboot.exceptions.ValidationException;
import donald.apiwithspringboot.model.JwtResponse;
import donald.apiwithspringboot.model.UserInfo;
import donald.apiwithspringboot.repository.UserInfoRepository;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import io.swagger.v3.oas.annotations.tags.Tag;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
import java.security.NoSuchAlgorithmException;
import java.util.Map;


@RestController
public class UserInfoController {


    final
    private UserInfoRepository userInfoRepository;

//    private HashData hashData = new HashData();
```

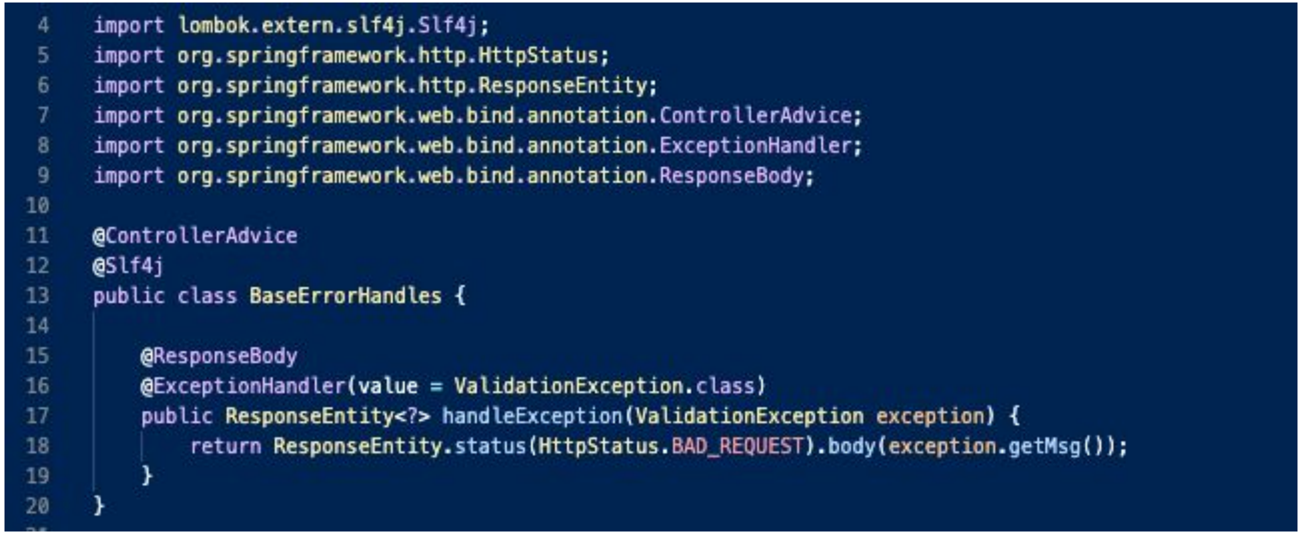


```java

    public UserInfoController(UserInfoRepository userInfoRepository) {
        this.userInfoRepository = userInfoRepository;
    }


    @PostMapping("/user")
    public Boolean create(@RequestBody Map<String, String> body) throws NoSuchAlgorithmException {
        String username = body.get("username");
        if (userInfoRepository.existsByUsername(username)){

            throw new ValidationException("Username already existed");

        }

        String password = body.get("password");
        String encodedPassword = new BCryptPasswordEncoder().encode(password);
//          String hashedPassword = hashData.get_SHA_512_SecurePassword(password);
        String fullname = body.get("fullname");
        userInfoRepository.save(new UserInfo(username, encodedPassword, fullname));
        return true;
    }

}
```

6.Exceptions:

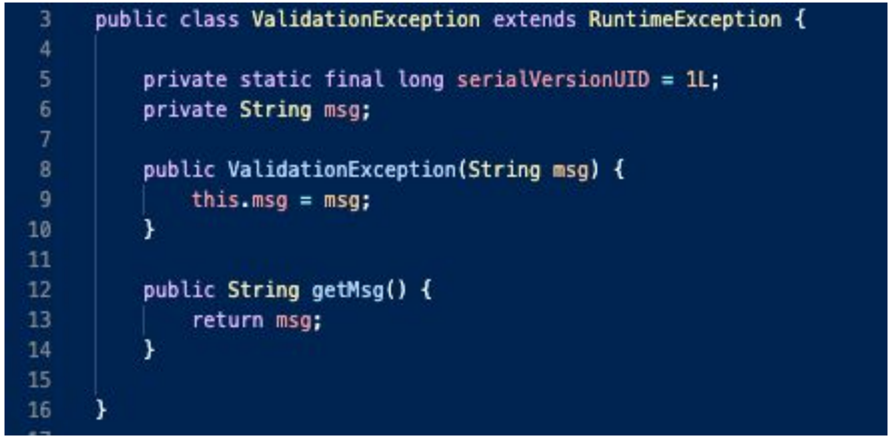
BaseErrorHandles class for handleException as BAD_REQUEST:

BaseErrorHandler.java



```java
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;

@ControllerAdvice
@Slf4j
public class BaseErrorHandles {

    @ResponseBody
    @ExceptionHandler(value = ValidationException.class)
    public ResponseEntity<?> handleException(ValidationException exception) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(exception.getMsg());
    }
}
```

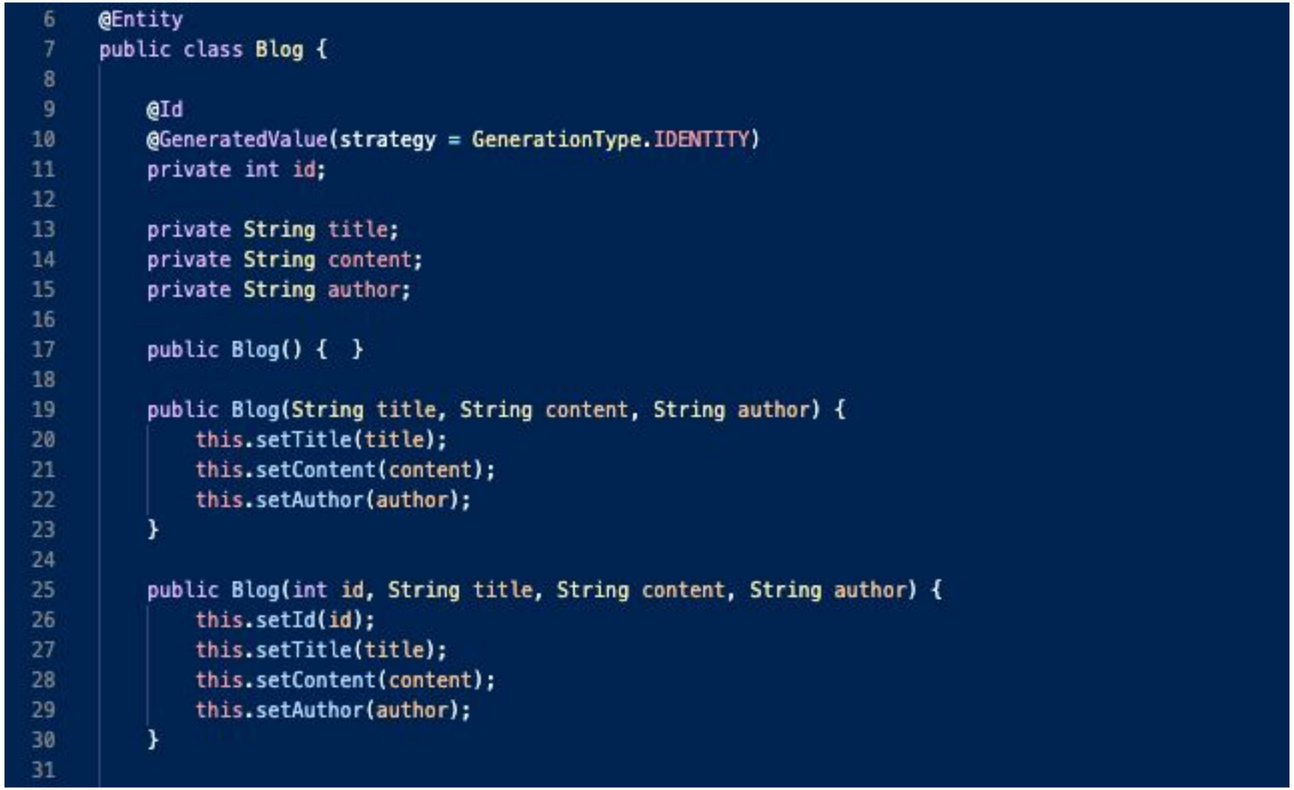
ValidationException.java



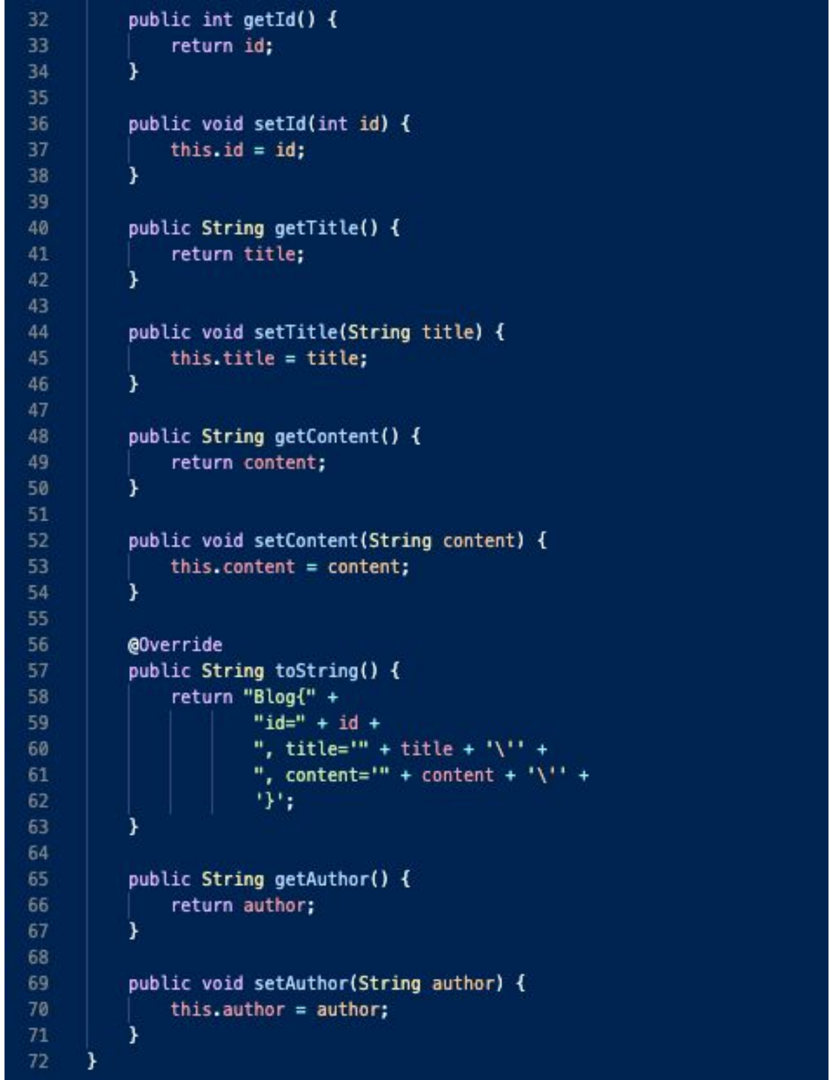
```java
public class ValidationException extends RuntimeException {

    private static final long serialVersionUID = 1L;
    private String msg;

    public ValidationException(String msg) {
        this.msg = msg;
    }

    public String getMsg() {
        return msg;
    }

}
```

7.Model:
Blog model: define blog entity

Blog.java



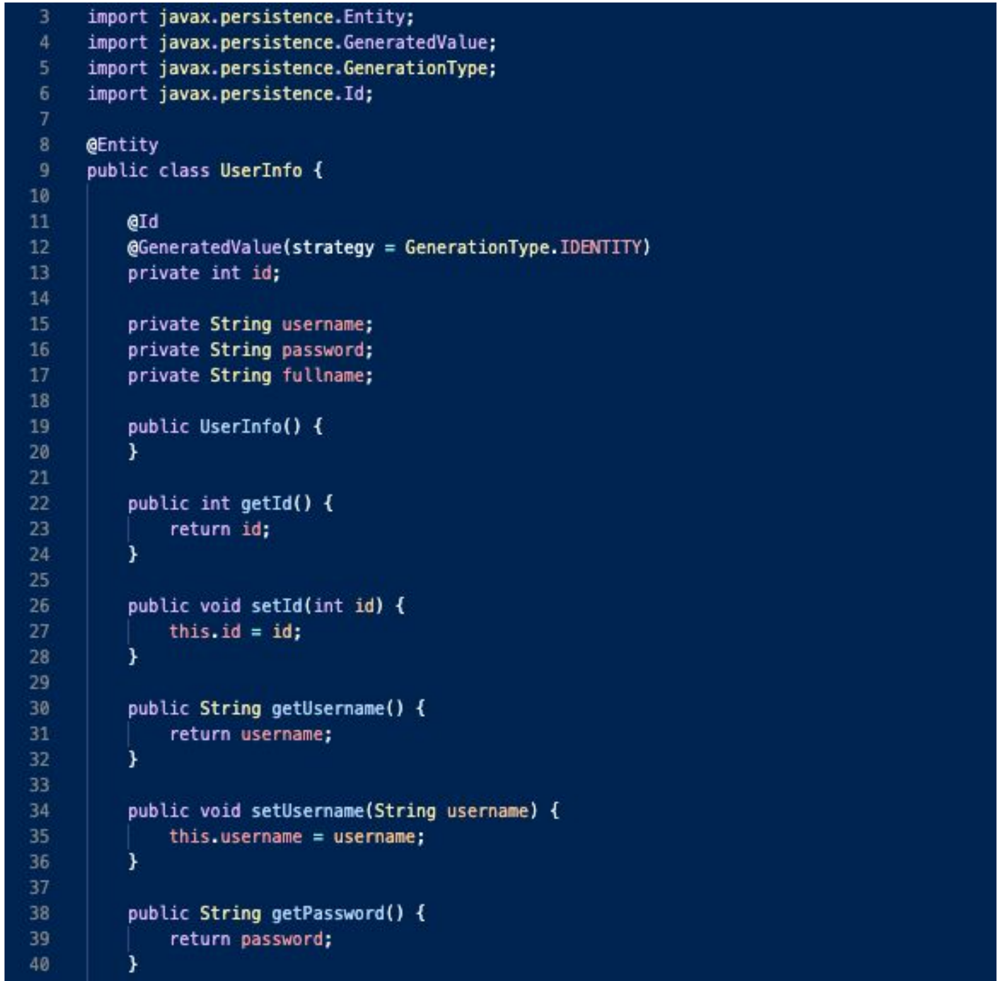
```java
@Entity
public class Blog {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String title;
    private String content;
    private String author;

    public Blog() {  }

    public Blog(String title, String content, String author) {
        this.setTitle(title);
        this.setContent(content);
        this.setAuthor(author);
    }

    public Blog(int id, String title, String content, String author) {
        this.setId(id);
        this.setTitle(title);
        this.setContent(content);
        this.setAuthor(author);
    }

```



```java
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

    @Override
    public String toString() {
        return "Blog{" +
                "id=" + id +
                ", title='" + title + '\'' +
                ", content='" + content + '\'' +
                '}';
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }
}
```

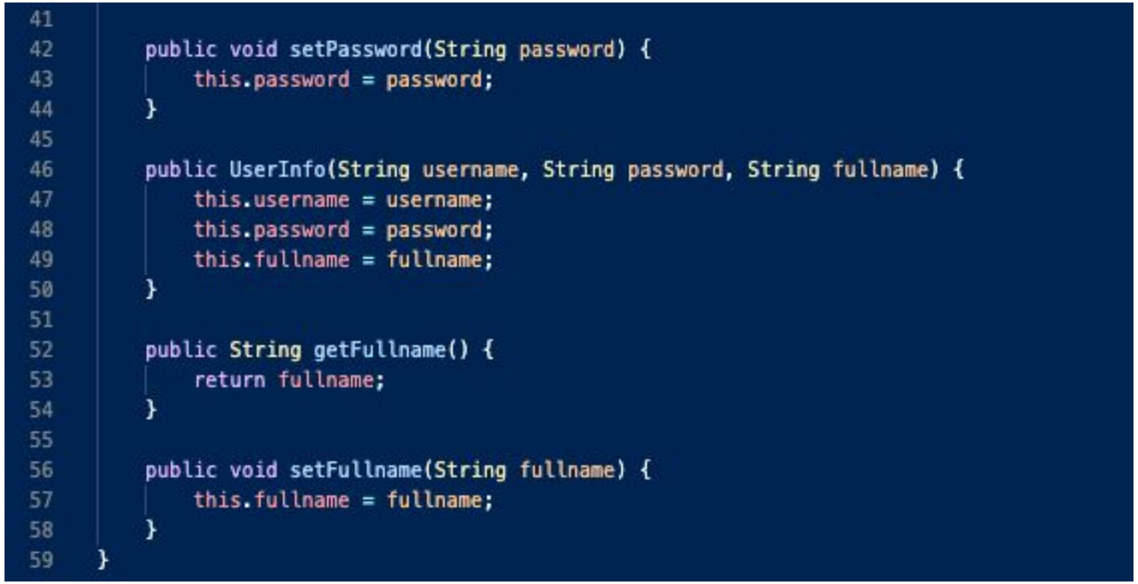
UserInfo class to define UserInfo enty:

UserInfo.java



```java
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class UserInfo {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String username;
    private String password;
    private String fullname;

    public UserInfo() {
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }
```

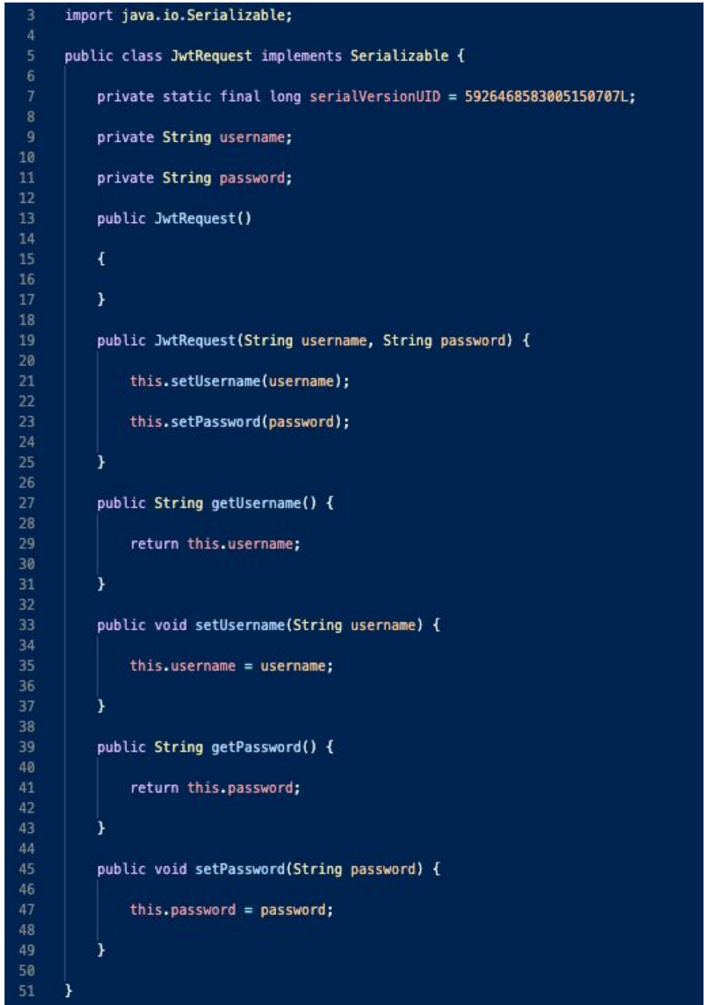
HACKTIV8



```java

    public void setPassword(String password) {
        this.password = password;
    }

    public UserInfo(String username, String password, String fullname) {
        this.username = username;
        this.password = password;
        this.fullname = fullname;
    }

    public String getFullname() {
        return fullname;
    }

    public void setFullname(String fullname) {
        this.fullname = fullname;
    }
}
```

JwtRequest model for authenticate username and password in the request in AuthController

JwtRequest.java

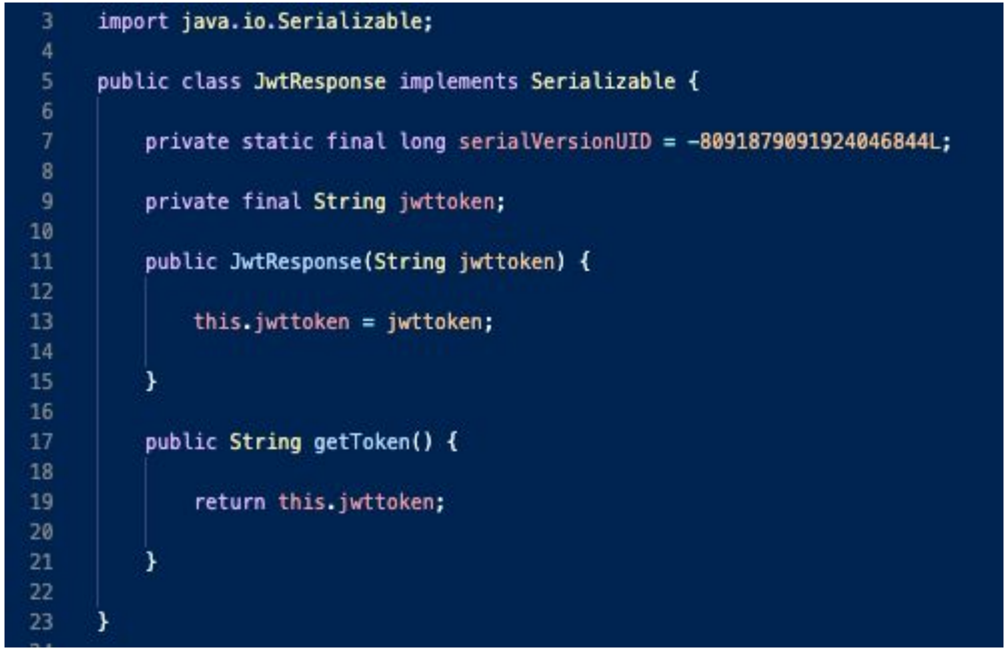


```java
import java.io.Serializable;

public class JwtRequest implements Serializable {

    private static final long serialVersionUID = 5926468583005150707L;

    private String username;

    private String password;

    public JwtRequest()

    {

    }

    public JwtRequest(String username, String password) {

        this.setUsername(username);

        this.setPassword(password);

    }

    public String getUsername() {

        return this.username;

    }

    public void setUsername(String username) {

        this.username = username;

    }

    public String getPassword() {

        return this.password;

    }

    public void setPassword(String password) {

        this.password = password;

    }

}
```

HACKTIV8

JwtResponse create model for token response

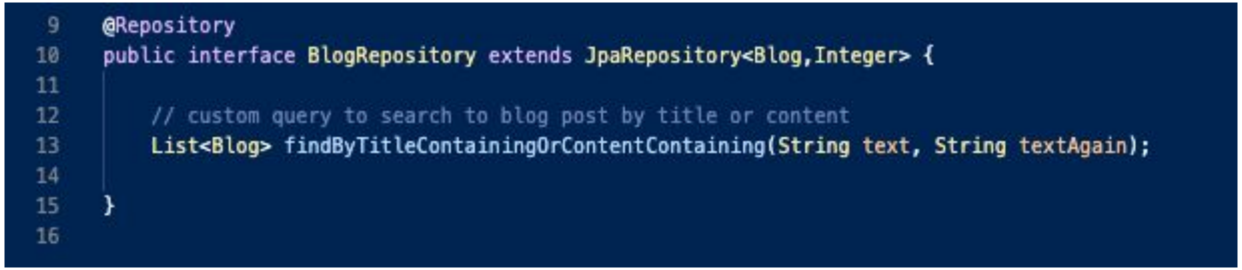
JwtResponse.java



```java
import java.io.Serializable;

public class JwtResponse implements Serializable {

    private static final long serialVersionUID = -8091879091924046844L;

    private final String jwttoken;

    public JwtResponse(String jwttoken) {

        this.jwttoken = jwttoken;

    }

    public String getToken() {

        return this.jwttoken;

    }

}
```

8.Repository:

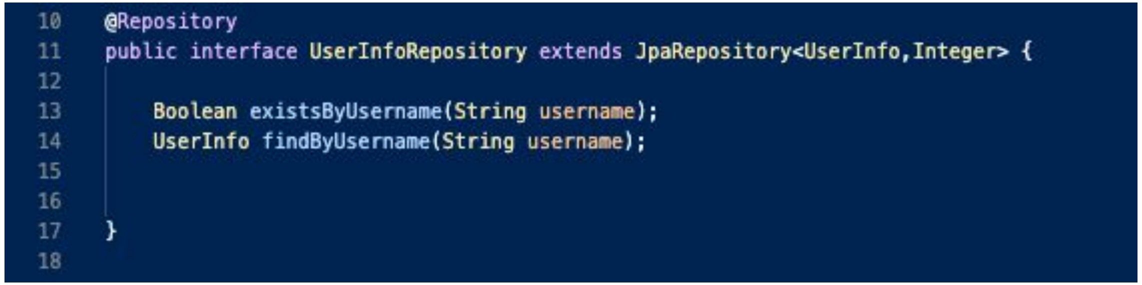
BlogRepository to work with MySQL database via JPA for blog table:



```java
@Repository
public interface BlogRepository extends JpaRepository<Blog,Integer> {

    // custom query to search to blog post by title or content
    List<Blog> findByTitleContainingOrContentContaining(String text, String textAgain);

}

```

BlogRepository.java

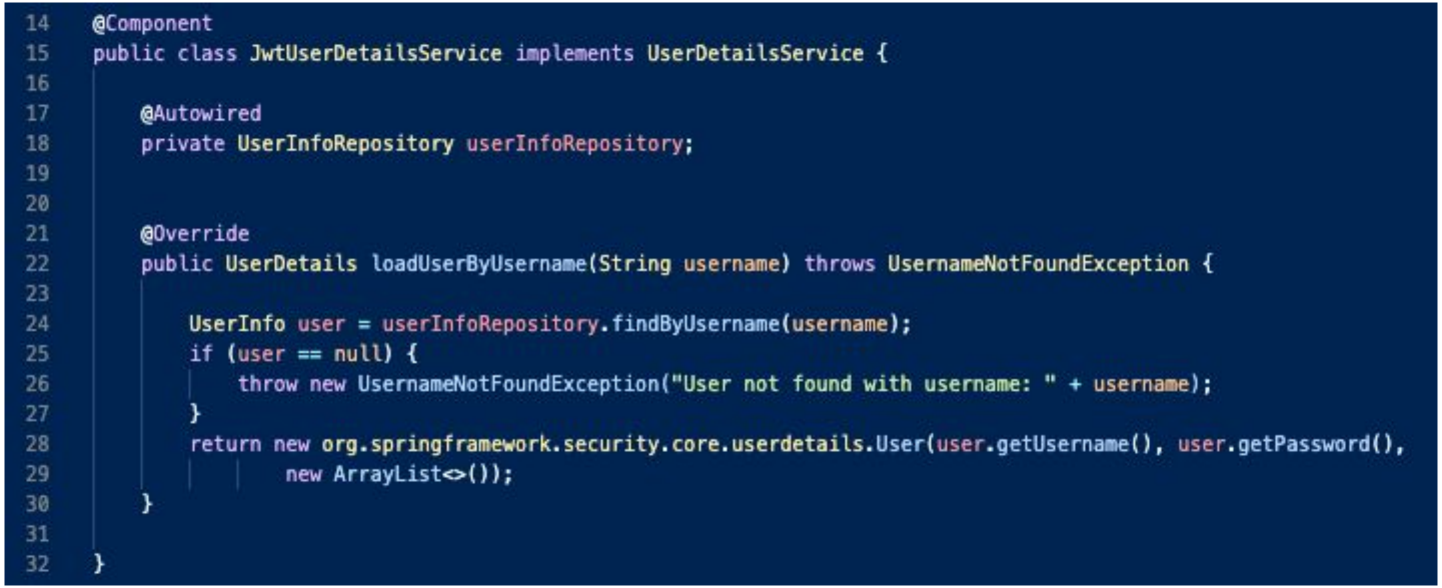
UserInfoRepository to work with MySQL database via JPA for user_info table



```java
@Repository
public interface UserInfoRepository extends JpaRepository<UserInfo,Integer> {

    Boolean existsByUsername(String username);
    UserInfo findByUsername(String username);


}

```

UserInfoRepository.java

9.Service:
Define JwtUserDetailsService for loadUserByUsername method:



```java
@Component
public class JwtUserDetailsService implements UserDetailsService {

    @Autowired
    private UserInfoRepository userInfoRepository;


    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        UserInfo user = userInfoRepository.findByUsername(username);
        if (user == null) {
            throw new UsernameNotFoundException("User not found with username: " + username);
        }
        return new org.springframework.security.core.userdetails.User(user.getUsername(), user.getPassword(),
                new ArrayList<>());
    }

}
```

JwtUserDetailsService.java

10.Run spring-boot application:

Simply run : mvn spring-boot:run

Do Interact with API using Postman to :

1. Create New User
2. Create New User Success
3. Authenticate User Credentials
4. Authenticate User Credentials Success
5. Create New Blog Request Body
6. Create New Blog header
7. Create New Blog response