

Full-Stack Software Development

Lab Sheet 11

D.R.R.C.Dasanayake

EC/2021/074

AuthController.java

```
package com.example.LibraryManagement.controller;

import com.example.LibraryManagement.model.User;
import com.example.LibraryManagement.payload.*;
import com.example.LibraryManagement.repository.UserRepository;
import com.example.LibraryManagement.security.JwtUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.*;
import org.springframework.security.core.Authentication;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/auth")
public class AuthController {

    @Autowired AuthenticationManager authenticationManager;
    @Autowired UserRepository userRepository;
    @Autowired PasswordEncoder encoder;
    @Autowired JwtUtils jwtUtils;

    @PostMapping("/signin")
    public ResponseEntity<?> authenticateUser(@RequestBody LoginRequest loginRequest) {
        Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(loginRequest.getUsername(),
                loginRequest.getPassword()));
    }
}
```

```
String jwt = jwtUtils.generateJwtToken(authentication);
return ResponseEntity.ok(new JwtResponse(jwt));
}

@PostMapping("/signup")
public ResponseEntity<?> registerUser(@RequestBody SignupRequest signUpRequest) {
    User user = new User(signUpRequest.getUsername(),
encoder.encode(signUpRequest.getPassword()));

    user.setRoles(signUpRequest.getRoles());
    userRepository.save(user);
    return ResponseEntity.ok("User registered successfully!");
}
}
```

BookController.java

```
package com.example.LibraryManagement.controller;

import com.example.LibraryManagement.model.Book;
import com.example.LibraryManagement.repository.BookRepository;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@CrossOrigin(origins = "*", maxAge = 3600) // Handles CORS issues for your React frontend
@RestController
```

```
@RequestMapping("/api/books")
public class BookController {

    private final BookRepository bookRepository;

    public BookController(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

    @GetMapping
    @PreAuthorize("hasRole('ADMIN') or hasRole('USER')")
    public List<Book> getAll() {
        return bookRepository.findAll();
    }

    @GetMapping("/{id}")
    @PreAuthorize("hasRole('ADMIN') or hasRole('USER')")
    public ResponseEntity<Book> getById(@PathVariable String id) {
        return bookRepository.findById(id)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
    }

    @PostMapping
    @PreAuthorize("hasRole('ADMIN')")
    public Book add(@RequestBody Book book) {
```

```
        return bookRepository.save(book);
    }

    @PutMapping("/{id}")
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<Book> update(@PathVariable String id, @RequestBody Book book) {
        return bookRepository.findById(id)
            .map(existingBook -> {
                existingBook.setTitle(book.getTitle());
                existingBook.setAuthor(book.getAuthor());
                existingBook.setPublicationYear(book.getPublicationYear());
                existingBook.setGenre(book.getGenre());
                existingBook.setCopiesAvailable(book.getCopiesAvailable());
                return ResponseEntity.ok(bookRepository.save(existingBook));
            })
            .orElse(ResponseEntity.notFound().build());
    }

    @DeleteMapping("/{id}")
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<?> delete(@PathVariable String id) {
        if (!bookRepository.existsById(id)) {
            return ResponseEntity.notFound().build();
        }
        bookRepository.deleteById(id);
        return ResponseEntity.ok("Book deleted successfully");
    }
}
```

```
 }  
}
```

Book.java

```
package com.example.LibraryManagement.model;  
  
import org.springframework.data.annotation.Id;  
import org.springframework.data.mongodb.core.mapping.Document;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
import lombok.AllArgsConstructor;  
  
@Document(collection = "books")  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class Book {  
    @Id  
    private String id;  
    private String title;  
    private String author;  
    private int publicationYear;  
    private String genre;  
    private int copiesAvailable; // add this field  
}
```

Role.java

```
package com.example.LibraryManagement.model;

public enum Role {

    ROLE_USER,
    ROLE_ADMIN

}
```

User.java

```
package com.example.LibraryManagement.model;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import java.util.Set;

@Document(collection = "users")

public class User {

    @Id
    private String id;
    private String username;
    private String password;
    private Set<String> roles; // ROLE_USER, ROLE_ADMIN

    public User() {}

    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }
}
```

```
// Getters and Setters

public String getId() { return id; }

public String getUsername() { return username; }

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

public String getPassword() { return password; }

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

public Set<String> getRoles() { return roles; }

public void setRoles(Set<String> roles) { this.roles = roles; }

}
```

JwtResponse.java

```
package com.example.LibraryManagement.payload;
```

```
public class JwtResponse {

    private String token;

    private String type;

    public JwtResponse(String token) {

        this.token = token;

        this.type = "Bearer";

    }

    public String getToken() {

        return token;

    }
```

```
public String getType() {  
    return type;  
}  
  
public void setToken(String token) {  
    this.token = token;  
}  
  
public void setType(String type) {  
    this.type = type;  
}  
}
```

LoginRequest.java

```
package com.example.LibraryManagement.payload;  
  
public class LoginRequest {  
    private String username;  
    private String password;  
    // Getters and Setters  
    public String getUsername() { return username; }  
    public String getPassword() { return password; }  
}
```

signupRequest.java

```
package com.example.LibraryManagement.payload;  
import java.util.Set;
```

```
public class SignupRequest {  
    private String username;  
    private String password;  
    private Set<String> roles;  
    // Getters and Setters  
    public String getUsername() { return username; }  
    public String getPassword() { return password; }  
    public Set<String> getRoles() { return roles; }  
}
```

BookRepository.java

```
package com.example.LibraryManagement.repository;  
  
import com.example.LibraryManagement.model.Book;  
  
import org.springframework.data.mongodb.repository.MongoRepository;  
  
import org.springframework.stereotype.Repository;  
  
  
import java.util.List;  
  
  
@Repository  
public interface BookRepository extends MongoRepository<Book, String> {  
    List<Book> findByPublicationYear(int publicationYear);  
}
```

UserRepository.java

```
package com.example.LibraryManagement.repository;  
  
import com.example.LibraryManagement.model.User;  
  
import org.springframework.data.mongodb.repository.MongoRepository;
```

```
import java.util.Optional;

public interface UserRepository extends MongoRepository<User, String> {
    Optional<User> findByUsername(String username);
}
```

JwtAuthFilter.java

```
package com.example.LibraryManagement.security;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
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 java.io.IOException;

@Component
public class JwtAuthFilter extends OncePerRequestFilter {
    @Autowired
    private JwtUtils jwtUtils;
```

```
@Autowired  
private UserDetailsServiceImpl userDetailsService;  
  
@Override  
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,  
FilterChain filterChain)  
throws ServletException, IOException {  
try {  
    String jwt = parseJwt(request);  
    if (jwt != null && jwtUtils.validateJwtToken(jwt)) {  
        String username = jwtUtils.getUserNameFromJwtToken(jwt);  
  
        UserDetails userDetails = userDetailsService.loadUserByUsername(username);  
        UsernamePasswordAuthenticationToken authentication = new  
        UsernamePasswordAuthenticationToken(  
            userDetails, null, userDetails.getAuthorities());  
  
        authentication.setDetails(new  
        WebAuthenticationDetailsSource().buildDetails(request));  
  
        SecurityContextHolder.getContext().setAuthentication(authentication);  
    }  
}  
} catch (Exception e) {  
    logger.error("Cannot set user authentication: {}", e);  
}
```

```
        filterChain.doFilter(request, response);

    }

private String parseJwt(HttpServletRequest request) {
    String headerAuth = request.getHeader("Authorization");
    if (headerAuth != null && headerAuth.startsWith("Bearer ")) {
        return headerAuth.substring(7);
    }
    return null;
}
```

JwtUtils.java

```
// Java

package com.example.LibraryManagement.security;

import io.jsonwebtoken.*;
import io.jsonwebtoken.security.Keys;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Component;

import java.nio.charset.StandardCharsets;
import java.security.Key;
import java.util.Date;
```

```
@Component
public class JwtUtils {
    private static final Logger logger = LoggerFactory.getLogger(JwtUtils.class);

    @Value("${library.app.jwtSecret:DefaultSecretKeyChangeMeDefaultSecretKey}")
    private String jwtSecret;

    @Value("${library.app.jwtExpirationMs:86400000}") // default 1 day
    private int jwtExpirationMs;

    public String generateJwtToken(Authentication authentication) {
        String username = authentication.getName();
        Date now = new Date();
        Date expiryDate = new Date(now.getTime() + jwtExpirationMs);

        return Jwts.builder()
            .setSubject(username)
            .setIssuedAt(now)
            .setExpiration(expiryDate)
            .signWith(getSigningKey(), SignatureAlgorithm.HS256)
            .compact();
    }

    public String getUserNameFromJwtToken(String token) {
        Claims claims = Jwts.parserBuilder()
```

```
.setSigningKey(getSigningKey())
.build()
.parseClaimsJws(token)
.getBody();

return claims.getSubject();
}

public boolean validateJwtToken(String authToken) {
try {
Jwts.parserBuilder()
.setSigningKey(getSigningKey())
.build()
.parseClaimsJws(authToken);

return true;
} catch (SecurityException | MalformedJwtException | UnsupportedJwtException e) {
logger.error("Invalid JWT token: {}", e.getMessage());
} catch (ExpiredJwtException e) {
logger.warn("JWT token is expired: {}", e.getMessage());
} catch (IllegalArgumentException e) {
logger.error("JWT claims string is empty: {}", e.getMessage());
} catch (JwtException e) {
logger.error("JWT exception: {}", e.getMessage());
}
return false;
}
```

```
private Key getSigningKey() {  
    byte[] keyBytes = jwtSecret.getBytes(StandardCharsets.UTF_8);  
    return Keys.hmacShaKeyFor(keyBytes);  
}  
}
```

securityConfig.java

```
// Java  
  
package com.example.LibraryManagement.security;  
  
  
import org.springframework.context.annotation.*;  
import org.springframework.security.authentication.*;  
  
import  
org.springframework.security.config.annotation.authentication.configuration.AuthenticationCo  
nfiguration;  
  
import  
org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;  
  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.crypto.bcrypt.*;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.security.web.SecurityFilterChain;  
  
import  
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;  
  
  
@Configuration  
@EnableMethodSecurity  
public class SecurityConfig {
```

```
@Bean  
public PasswordEncoder passwordEncoder() {  
    return new BCryptPasswordEncoder();  
}  
  
@Bean  
public AuthenticationManager authenticationManager(  
    AuthenticationConfiguration config) throws Exception {  
    return config.getAuthenticationManager();  
}  
  
@Bean  
public JwtAuthFilter jwtAuthFilter() {  
    return new JwtAuthFilter();  
}  
  
@Bean  
public SecurityFilterChain filterChain(HttpSecurity http,  
    JwtAuthFilter jwtFilter) throws Exception {  
  
    http.csrf(csrf -> csrf.disable());  
  
    http.authorizeHttpRequests(authorize ->  
        authorize  
            .requestMatchers("/api/auth/**").permitAll()  
            .anyRequest().authenticated()  
    );  
}
```

```
        http.addFilterBefore(jwtFilter,  
            UsernamePasswordAuthenticationFilter.class);  
  
        return http.build();  
    }  
  
}
```

UserDetailsServiceImpl.java

```
package com.example.LibraryManagement.security;  
  
import com.example.LibraryManagement.model.User;  
  
import com.example.LibraryManagement.repository.UserRepository;  
  
import org.springframework.beans.factory.annotation.Autowired;  
  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
  
import org.springframework.security.core.userdetails.*;  
  
import org.springframework.stereotype.Service;  
  
import java.util.stream.Collectors;
```

```
@Service
```

```
public class UserDetailsServiceImpl implements UserDetailsService {
```

```
    @Autowired
```

```
    UserRepository userRepository;
```

```
    @Override
```

```
    public UserDetails loadUserByUsername(String username) throws  
        UsernameNotFoundException {
```

```
        User user = userRepository.findByUsername(username)  
  
        .orElseThrow(() -> new UsernameNotFoundException("User Not Found: " +  
username));
```

```
        return new org.springframework.security.core.userdetails.User(  
            user.getUsername(),  
            user.getPassword(),  
            user.getRoles().stream().map(SimpleGrantedAuthority::new).collect(Collectors.toList())  
        );  
    }  
}
```

Application.properties

```
spring.application.name=LibraryManagement  
server.port=8081
```

```
spring.data.mongodb.uri=mongodb+srv://ridmichamoda_db_user:RmZ8pwmu3KKQALeC@cluster0.tfyzhgg.mongodb.net/LibraryManagement?retryWrites=true&w=majority&tls=true
```

```
spring.jpa.show-sql=true  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.properties.hibernate.format_sql=true
```

```
logging.level.org.springframework=INFO  
logging.level.com.mongodb=INFO  
logging.level.org.mongodb.driver=INFO
```

```
server.error.include-message=always
```

Register Users

The screenshot shows a POST request to `http://localhost:8081/api/auth/signup`. The request body is raw JSON:

```
1 {
2   "username": "admin1",
3   "password": "admin123",
4   "roles": ["ROLE_ADMIN"]
5 }
```

The response status is **200 OK**, time taken is 2.70 s, and size is 452 B. The response body is:

```
1 User registered successfully!
```

The screenshot shows a POST request to `http://localhost:8081/api/auth/signup`. The request body is raw JSON:

```
1 {
2   "username": "user1",
3   "password": "user123",
4   "roles": ["ROLE_USER"]
5 }
```

The response status is **200 OK**, time taken is 298 ms, and size is 452 B. The response body is:

```
1 User registered successfully!
```

Login to Get JWT Token

The screenshot shows a POST request to `http://localhost:8081/api/auth/signin`. The request body is JSON with the following content:

```
1 {  
2   "username": "admin1",  
3   "password": "admin123"  
4 }
```

The response status is 200 OK, with a response time of 360 ms and a response size of 583 B. The response body contains a JSON object with a token and type:

```
1 {  
2   "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbjEiLCJpYXQiOjE3NjY1NjI4NTAsImV4cCI6MTc2NjY0OTI1MH0.lPi1mkkTERVF5A2kakpqmU1kmZbA8Qh3hgpLxFTCtvE",  
3   "type": "Bearer"  
4 }
```

The screenshot shows a POST request to `http://localhost:8081/api/auth/signin`. The request body is JSON with the following content:

```
1 {  
2   "username": "user1",  
3   "password": "user123"  
4 }
```

The response status is 200 OK, with a response time of 283 ms and a response size of 582 B. The response body contains a JSON object with a token and type:

```
1 {  
2   "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMSIsImlhdiCI6MTc2NjU2Mjc0NSwiZXhwIjoxNzY2NjQ5MTQ1fQ.l0FI3a6-wn116k5Ft0Nkg5R1u_AU9uQjF6ZdcUGaIUY",  
3   "type": "Bearer"  
4 }
```

Test GET /api/books

HTTP <http://localhost:8081/api/books>

GET <http://localhost:8081/api/books>

Headers (11) Body Scripts Tests Settings Cookies

Headers (10 hidden)

Key	Value	Description	Bulk Edit	Presets
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzI1NiJ9eyJzdWIiOiJcJpYXQi...			

Body Cookies (1) Headers (14) Test Results

```
{
  "id": "693876052ee3a0428ee0421f",
  "title": "Modya mongodb connection string eka danawd github",
  "author": "F. Scott Fitzgerald",
  "publicationYear": 1819,
  "genre": "Adventure",
  "copiesAvailable": 0
},
{
  "id": "693876722ee3a0428ee04220",
  "title": "The Great Gatsby",
  "author": "F. Scott Fitzgerald",
  "publicationYear": 1925,
  "genre": "Classic",
  "copiesAvailable": 0
}
```

200 OK 412 ms 1.85 KB

HTTP <http://localhost:8081/api/books>

GET <http://localhost:8081/api/books>

Headers (11) Body Scripts Tests Settings Cookies

Headers (10 hidden)

Key	Value	Description	Bulk Edit	Presets
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzI1NiJ9eyJzdWIiOiJc2VyMSlsImhdCl6...			

Body Cookies (1) Headers (14) Test Results

```
{
  "id": "693876052ee3a0428ee0421f",
  "title": "Modya mongodb connection string eka danawd github",
  "author": "F. Scott Fitzgerald",
  "publicationYear": 1819,
  "genre": "Adventure",
  "copiesAvailable": 0
},
{
  "id": "693876722ee3a0428ee04220",
  "title": "The Great Gatsby",
  "author": "F. Scott Fitzgerald",
  "publicationYear": 1925,
  "genre": "Classic",
  "copiesAvailable": 0
}
```

200 OK 361 ms 1.85 KB

Test GET /api/books/{id}

HTTP <http://localhost:8081/api/books/6938767f2ee3a0428ee04221>

GET <http://localhost:8081/api/books/6938767f2ee3a0428ee04221>

Headers (11) Body Scripts Tests Settings Cookies

Headers (10 hidden)

Key	Value	Description	Bulk Edit	Presets
Authorization	Bearer eyJhbGciOiJIUzI1NiJ9eyJzdWIiOiJ1c2VyMSIsImhdCI6...			

Body Cookies (1) Headers (14) Test Results

```
{
  "id": "6938767f2ee3a0428ee04221",
  "title": "1984",
  "author": "George Orwell",
  "publicationYear": 1949,
  "genre": "Dystopian",
  "copiesAvailable": 0
}
```

200 OK • 353 ms • 559 B

HTTP <http://localhost:8081/api/books/6938767f2ee3a0428ee04221>

GET <http://localhost:8081/api/books/6938767f2ee3a0428ee04221>

Headers (11) Body Scripts Tests Settings Cookies

Headers (10 hidden)

Key	Value	Description	Bulk Edit	Presets
Authorization	Bearer eyJhbGciOiJIUzI1NiJ9eyJzdWIiOiJhZG1pbEiLCJpYXQi...			
Key	Value	Description		

Body Cookies (1) Headers (14) Test Results

```
{
  "id": "6938767f2ee3a0428ee04221",
  "title": "1984",
  "author": "George Orwell",
  "publicationYear": 1949,
  "genre": "Dystopian",
  "copiesAvailable": 0
}
```

200 OK • 364 ms • 559 B

POST /api/books

```

1 {
2   "title": "Java Programming",
3   "author": "James Gosling",
4   "publicationYear": 2020,
5   "genre": "Programming"
6 }
7
  
```

Body Cookies (1) Headers (14) Test Results

{} JSON ▾ Preview Visualize

```

1 {
2   "id": "694ba12fdaec23c97b205c59",
3   "title": "Java Programming",
4   "author": "James Gosling",
5   "publicationYear": 2020,
6   "genre": "Programming",
7   "copiesAvailable": 0
8 }
  
```

200 OK 382 ms 573 B

For users

Headers (11)

Key	Value	Description	Bulk Edit	Presets
Authorization	Bearer eyJhbGciOiJIUzI1NiJ9eyJzdWlOIjJc2VyMSIsImhdC16...			
Key	Value	Description		

Body Cookies (1) Headers (14) Test Results

Raw ▾ Try with different auth credentials

1

403 Forbidden

Test PUT /api/books/{id}**For admin**

HTTP <http://localhost:8081/api/books/693876722ee3a0428ee04220>

PUT http://localhost:8081/api/books/693876722ee3a0428ee04220

Body `{ "title": "Java Programming", "author": " Gosling", "publicationYear": 2025, "genre": "Programming" }`

200 OK 553 ms 568 B

For users

HTTP <http://localhost:8081/api/books/693876722ee3a0428ee04220>

PUT http://localhost:8081/api/books/693876722ee3a0428ee04220

Headers `Authorization: Bearer eyJhbGciOiJIUzI1NiJ9eyJzdWIiOiJ1c2VyMSIsImhdCI6...`

403 Forbidden 193 ms 389 B

Test DELETE /api/books/{id}

For admin

The screenshot shows a Postman request to `http://localhost:8081/api/books/693876722ee3a0428ee04220`. The method is set to `DELETE`. The `Body` tab is selected, showing a JSON payload:

```
1 {
2   "title": "Java Programming",
3   "author": "Gosling",
4   "publicationYear": 2025,
5   "genre": "Programming"
6 }
```

The response status is `200 OK`, and the body of the response is `1 Book deleted successfully`.

For users

The screenshot shows a Postman request to `http://localhost:8081/api/books/693876722ee3a0428ee04220`. The method is set to `DELETE`. The `Headers` tab is selected, showing an `Authorization` header with a value of `Bearer eyJhbGciOiJIUzI1NiJ9eyJzdWIiOiJ1c2VyMSIsImhdCl6...`.

The response status is `403 Forbidden`, and the body of the response is `1 Try with different auth credentials`.