

LAB 11

EC/2021/006

W.K.G.K JAYAWARDANA

The screenshot shows the Postman application interface. On the left, the sidebar displays 'Gamika Jayawardhana's Workspace' with collections like 'ApeBodima Api', 'Employee Api', 'Fitness Microservices', 'HarvestLink Api', 'Movies Api', and 'Vehicle Api'. The main workspace shows a POST request to 'http://localhost:9099/api/auth/signup'. The request body is set to JSON and contains the following data:

```
1 {  
2   "username": "admin",  
3   "email": "admin@test.com",  
4   "password": "password123",  
5   "roles": ["admin"]  
6 }
```

The response status is '200 OK' with a response time of 1.26 s and a size of 466 B. The response body is:

```
1 {  
2   "message": "User registered successfully!"  
3 }
```

At the bottom, there are tabs for 'Cloud View', 'Console', and 'Terminal'.

LAB 11

EC/2021/006

W.K.G.K JAYAWARDANA

The screenshot shows the API Network tool interface. On the left, there's a sidebar with 'Collections' (ApeBodima Api, Employee Api, Fitness Microservices, HarvestLink Api, Movies Api, Vehicle Api), 'Environments', 'History', 'Flows', and a 'Files (BETA)' section. The main area shows a request to `http://localhost:9099/api/auth/signup`. The 'Body' tab is selected, displaying the following JSON payload:

```
1 {
2   "username": "user",
3   "email": "user@test.com",
4   "password": "password123",
5   "roles": ["usex"]
6 }
```

Below the request, the response is shown: `200 OK`, 817 ms, 468 B. The response body is:

```
1 {
2   "message": "User registered successfully!"
3 }
```

The screenshot shows the API Network tool interface, similar to the previous one. The sidebar includes 'Collections' (ApeBodima Api, Employee Api, Fitness Microservices, HarvestLink Api, Movies Api, Vehicle Api), 'Environments', 'History', 'Flows', and a 'Files (BETA)' section. The main area shows a request to `http://localhost:9099/api/auth/signin`. The 'Body' tab is selected, displaying the following JSON payload:

```
1 {
2   "username": "admin",
3   "password": "password123"
4 }
```

Below the request, the response is shown: `200 OK`, 604 ms, 681 B. The response body is a complex JSON object containing user information and a token:

```
1 {
2   "id": "694e6de5d6fc5cd8934d5298",
3   "username": "admin",
4   "email": "admin@test.com",
5   "roles": [
6     "ROLE_ADMIN"
7   ],
8   "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZGibpItiSm1hdC16MTc2Njc0Nzc3MCwiZXhwIjoxNzY2ODM0MTcwfQ.LZNB-x9InW0oexYQyYoABXmFCH-diWm_MzAKThi24I",
9   "type": "Bearer"
10 }
```

LAB 11

EC/2021/006

W.K.G.K JAYAWARDANA

The screenshot shows the API Network tool interface. On the left, the sidebar lists collections: ApeBodima Api, Employee Api, Fitness Microservices, HarvestLink Api, Movies Api, and Vehicle Api. The main area shows a POST request to `http://localhost:9099/api/books`. The Body tab contains the following JSON payload:

```
1 {  
2   "title": "Spring Security In Action",  
3   "author": "Laurentiu Spilca",  
4   "isbn": "9783617297731",  
5   "year": 2020  
6 }
```

The response status is 200 OK, with a response time of 656 ms and a body size of 574 B. The response JSON is:

```
1 {  
2   "id": "694e6ed4d6fc5cd8934d5292",  
3   "title": "Spring Security In Action",  
4   "author": "laurentiu Spilca",  
5   "genre": null,  
6   "publicationYear": 0,  
7   "shelflocation": null  
8 }
```

The screenshot shows the API Network tool interface. The sidebar lists the same collections as the first screenshot. The main area shows a POST request to `http://localhost:9099/api/books`. The Headers tab is selected, showing the Auth Type set to Bearer Token with a Token input field containing "Token". The response status is 401 Unauthorized, with a response time of 28 ms and a body size of 462 B. The response JSON is:

```
1 {  
2   "path": "/api/books",  
3   "error": "Unauthorized",  
4   "message": "Full authentication is required to access this resource",  
5   "status": 401  
6 }
```

LAB 11

EC/2021/006

W.K.G.K JAYAWARDANA

The screenshot shows the API Network tool interface. On the left, there's a sidebar with 'Collections' (selected), 'Environments', 'History', and 'Flows'. The main area shows a list of collections: 'ApeBodima Api', 'Employee Api', 'Fitness Microservices', 'HarvestInk Api', 'Movies Api', and 'Vehicle Api'. A search bar at the top says 'Search collections'. Below it, a request card is open for 'POST http://localhost:9099/api/books'. The 'Auth' tab is selected, showing 'Bearer Token' is chosen. A note says: 'The authorization header will be automatically generated when you send the request.' Below this, a token field contains 'xYQyYoABXmFCH-diWm_MrAKTh[24]'. The response section shows a 200 OK status with a response body in JSON format:

```
1 {  
2   "id": "694e6ed4d6fc5cd8934d6292",  
3   "title": "Spring Security In Action",  
4   "author": "Laurentiu Spilca",  
5   "genre": null,  
6   "publicationYear": 0,  
7   "shelfLocation": null  
8 }
```

This screenshot shows the same API Network tool interface as the previous one, but with a different request. The request card now says 'GET http://localhost:9099/api/books'. The 'Params' tab is selected, showing a table with columns 'Key', 'Value', 'Descri...', and 'Bulk Edit'. There are two rows: 'id' with value '69186bd444862e3ee3275c16' and 'id' with value '694e6ed4d6fc5cd8934d5292'. The response section shows a 200 OK status with a response body in JSON format:

```
1 [  
2   {  
3     "id": "69186bd444862e3ee3275c16",  
4     "title": "Clean Code",  
5     "author": "Robert C. Martin",  
6     "genre": "Programming",  
7     "publicationYear": 2008,  
8     "shelfLocation": "B-07"  
9   },  
10   {  
11     "id": "694e6ed4d6fc5cd8934d5292",  
12     "title": "Spring Security In Action",  
13     "author": "Laurentiu Spilca",  
14     "genre": null,  
15     "publicationYear": 0,  
16     "shelfLocation": null  
17   }  
18 ]
```

AuthController.java

```
package com.lib.LiMS.controller;

import com.lib.LiMS.model.ERole;
import com.lib.LiMS.model.Role;
import com.lib.LiMS.model.User;
import com.lib.LiMS.payload.request.LoginRequest;
import com.lib.LiMS.payload.request.SignupRequest;
import com.lib.LiMS.payload.response.JwtResponse;
import com.lib.LiMS.payload.response.MessageResponse;
import com.lib.LiMS.repository.RoleRepository;
import com.lib.LiMS.repository.UserRepository;
import com.lib.LiMS.security.jwt.JwtUtils;
import com.lib.LiMS.security.services.UserDetailsImpl;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import
org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import
org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.*;

import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
@RequestMapping("/api/auth")
public class AuthController {
```

```
@Autowired
AuthenticationManager authenticationManager;

@Autowired
UserRepository userRepository;

@Autowired
RoleRepository roleRepository;

@Autowired
PasswordEncoder encoder;

@Autowired
JwtUtils jwtUtils;

@PostMapping("/signin")
public ResponseEntity<?> authenticateUser(@Valid @RequestBody
LoginRequest loginRequest) {

    Authentication authentication =
authenticationManager.authenticate(
        new
UsernamePasswordAuthenticationToken(loginRequest.getUsername(),
loginRequest.getPassword()));

    SecurityContextHolder.getContext().setAuthentication(authentication);
    String jwt = jwtUtils.generateJwtToken(authentication);

    UserDetailsImpl userDetails = (UserDetailsImpl)
authentication.getPrincipal();
    List<String> roles = userDetails.getAuthorities().stream()
        .map(item -> item.getAuthority())
        .collect(Collectors.toList());

    return ResponseEntity.ok(new JwtResponse(jwt,
        userDetails.getId(),
        userDetails.getUsername(),
```

```
        userDetails.getEmail(),
        roles));
    }

    @PostMapping("/signup")
    public ResponseEntity<?> registerUser(@Valid @RequestBody
SignupRequest signUpRequest) {
        if
(userRepository.existsByUsername(signUpRequest.getUsername())) {
            return ResponseEntity
                .badRequest()
                .body(new MessageResponse("Error: Username is
already taken!"));
        }

        if (userRepository.existsByEmail(signUpRequest.getEmail())) {
            return ResponseEntity
                .badRequest()
                .body(new MessageResponse("Error: Email is already
in use!"));
        }

        // Create new user's account
        User user = new User(signUpRequest.getUsername(),
            signUpRequest.getEmail(),
            encoder.encode(signUpRequest.getPassword()));

        Set<String> strRoles = signUpRequest.getRoles();
        Set<Role> roles = new HashSet<>();

        if (strRoles == null) {
            Role userRole = roleRepository.findByName(ERole.ROLE_USER)
                .orElseThrow(() -> new RuntimeException("Error:
Role is not found."));
            roles.add(userRole);
        } else {
            strRoles.forEach(role -> {
                switch (role) {
```

```
        case "admin":  
            Role adminRole =  
roleRepository.findByName(ERole.ROLE_ADMIN)  
                .orElseThrow(() -> new  
RuntimeException("Error: Role is not found."));  
            roles.add(adminRole);  
  
            break;  
        default:  
            Role userRole =  
roleRepository.findByName(ERole.ROLE_USER)  
                .orElseThrow(() -> new  
RuntimeException("Error: Role is not found."));  
            roles.add(userRole);  
        }  
    }  
  
    user.setRoles(roles);  
    userRepository.save(user);  
  
    return ResponseEntity.ok(new MessageResponse("User registered  
successfully!"));  
}  
}
```

BookController.java

```
package com.lib.LiMS.controller;  
  
import com.lib.LiMS.model.Book;  
import com.lib.LiMS.service.BookService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.access.prepost.PreAuthorize;  
import org.springframework.web.bind.annotation.*;
```

```
import java.util.List;
import java.util.Optional;

@RestController
@CrossOrigin(origins = "*")
@RequestMapping("api/books")
public class BookController {

    @Autowired
    private BookService bookService;

    @PostMapping
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<Book> createBook(@RequestBody Book book) {
        Book newBook = bookService.createBook(book);
        return ResponseEntity.ok(newBook);
    }

    @GetMapping
    @PreAuthorize("hasRole('USER') or hasRole('ADMIN')")
    public ResponseEntity<List<Book>> getAllBooks() {
        List<Book> books = bookService.getAllBooks();
        return ResponseEntity.ok(books);
    }

    @GetMapping("/{id}")
    @PreAuthorize("hasRole('USER') or hasRole('ADMIN')")
    public ResponseEntity<Book> getBookByID(@PathVariable String id) {
        Book book = bookService.getBookById(id);
        return book != null ? ResponseEntity.ok(book) :
ResponseEntity.notFound().build();
    }

    @PutMapping("/{id}")
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<Book> updateBook(@PathVariable String id,
    @RequestBody Book book) {
```

```
        Book updatedBook = bookService.updateBook(id, book);
        return updatedBook != null ? ResponseEntity.ok(updatedBook) :
    ResponseEntity.notFound().build();
    }

    @DeleteMapping("/{id}")
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<Book> deleteBook(@PathVariable String id) {
        bookService.deleteBookById(id);
        return ResponseEntity.noContent().build();
    }

    @GetMapping("/year/{year}")
    @PreAuthorize("hasRole('USER') or hasRole('ADMIN')")
    public ResponseEntity<List<Book>> getBooksByYear(@PathVariable int
year) {
        List<Book> books = bookService.getBooksByYear(year);
        return ResponseEntity.ok(books);
    }

    @GetMapping("/{id}/genre")
    @PreAuthorize("hasRole('USER') or hasRole('ADMIN')")
    public ResponseEntity<String> getGenre(@PathVariable String id) {
        Optional<String> genre = bookService.getGenreById(id);
        return genre.map(ResponseEntity::ok).orElseGet(() ->
    ResponseEntity.notFound().build());
    }

    @DeleteMapping("/year/{year}")
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<String> deleteBookByYear(@PathVariable int
year) {
        bookService.deleteByYear(year);
        return ResponseEntity.ok("Successfully deleted all service
record for the year: " + year);
    }

}
```

BookService.java

```
package com.lib.LiMS.service;

import com.lib.LiMS.model.Book;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public interface BookService {
    Book createBook(Book book);
    List<Book> getAllBooks();
    Book getBookById(String id);
    Book updateBook(String id, Book book);
    void deleteBookById(String id);
    List<Book> getBooksByYear(int year);
    Optional<String> getGenreById(String id);
    void deleteByYear(int publicationYear);

}
```

BookServiceImpl.java

```
package com.lib.LiMS.service.impl;

import com.lib.LiMS.model.Book;
import com.lib.LiMS.repository.BookRepository;
import com.lib.LiMS.service.BookService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
import java.util.List;
import java.util.Optional;

@Service
public class BookServiceImpl implements BookService {

    @Autowired
    private BookRepository bookRepository;

    @Override
    public Book createBook(Book book) {
        return bookRepository.save(book);
    }

    @Override
    public List<Book> getAllBooks() {
        return bookRepository.findAll();
    }

    @Override
    public Book getBookById(String id) {
        Optional<Book> book = bookRepository.findById(id);
        return book.orElse(null);
        //return null or throw an exception
    }

    @Override
    public Book updateBook(String id, Book book) {
        if(bookRepository.existsById(id)){
            book.setId(id);
            return bookRepository.save(book);
        }
        return null;
    }

    @Override
    public void deleteBookById(String id) {
```

```
        bookRepository.deleteById(id);

    }

@Override
public List<Book> getBooksByYear(int year) {
    return bookRepository.findByPublicationYear(year);

}

@Override
public Optional<String> getGenreById(String id) {
    Optional<Book> book = bookRepository.findGenreById(id);
    return book.map(Book::getGenre);
}

@Override
public void deleteByYear(int publicationYear) {
    bookRepository.deleteByPublicationYear(publicationYear);

}

}
```

UserDetailsService.java

```
package com.lib.LiMS.security.services;

import com.lib.LiMS.model.User;
import com.lib.LiMS.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.core.userdetails.UserDetailsService;
```

```
import  
org.springframework.security.core.userdetails.UsernameNotFoundException;  
  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
@Service  
public class UserDetailsServiceImpl implements UserDetailsService {  
    @Autowired  
    UserRepository userRepository;  
  
    @Override  
    @Transactional  
    public UserDetails loadUserByUsername(String username) throws  
        UsernameNotFoundException {  
        User user = userRepository.findByUsername(username)  
            .orElseThrow(() -> new UsernameNotFoundException("User  
Not Found with username: " + username));  
  
        return UserDetailsImpl.build(user);  
    }  
}
```

UserDetailsServiceImpl.java

```
package com.lib.LiMS.security.services;  
  
import com.fasterxml.jackson.annotation.JsonIgnore;  
import com.lib.LiMS.model.User;  
import org.springframework.security.core.GrantedAuthority;  
import  
org.springframework.security.core.authority.SimpleGrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
  
import java.util.Collection;  
import java.util.List;
```

```
import java.util.Objects;
import java.util.stream.Collectors;

public class UserDetailsImpl implements UserDetails {
    private static final long serialVersionUID = 1L;

    private String id;

    private String username;

    private String email;

    @JsonIgnore
    private String password;

    private Collection<? extends GrantedAuthority> authorities;

    public UserDetailsImpl(String id, String username, String email,
String password,
            Collection<? extends GrantedAuthority> authorities) {
        this.id = id;
        this.username = username;
        this.email = email;
        this.password = password;
        this.authorities = authorities;
    }

    public static UserDetailsImpl build(User user) {
        List<GrantedAuthority> authorities = user.getRoles().stream()
            .map(role -> new
SimpleGrantedAuthority(role.getName().name()))
            .collect(Collectors.toList());

        return new UserDetailsImpl(
            user.getId(),
            user.getUsername(),
            user.getEmail(),
            user.getPassword(),
        );
    }
}
```

```
        authorities);  
    }  
  
    @Override  
    public Collection<? extends GrantedAuthority> getAuthorities() {  
        return authorities;  
    }  
  
    public String getId() {  
        return id;  
    }  
  
    public String getEmail() {  
        return email;  
    }  
  
    @Override  
    public String getPassword() {  
        return password;  
    }  
  
    @Override  
    public String getUsername() {  
        return username;  
    }  
  
    @Override  
    public boolean isAccountNonExpired() {  
        return true;  
    }  
  
    @Override  
    public boolean isAccountNonLocked() {  
        return true;  
    }  
  
    @Override  
    public boolean isCredentialsNonExpired() {
```

```
        return true;
    }

@Override
public boolean isEnabled() {
    return true;
}

@Override
public boolean equals(Object o) {
    if (this == o)
        return true;
    if (o == null || getClass() != o.getClass())
        return false;
    UserDetailsImpl user = (UserDetailsImpl) o;
    return Objects.equals(id, user.id);
}
}
```

BookRepository.java

```
package com.lib.LiMS.repository;

import com.lib.LiMS.model.Book;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Optional;

public interface BookRepository extends MongoRepository<Book, String>
{
    List<Book> findByPublicationYear(int publicationYear);

    Optional<Book> findGenreById(String id);
```

```
@Transactional  
void deleteByPublicationYear(int publicationYear);  
}
```

UserRepository.java

```
package com.lib.LiMS.repository;  
  
import com.lib.LiMS.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);  
  
    Boolean existsByUsername(String username);  
  
    Boolean existsByEmail(String email);  
}
```

LoginRequest.java

```
package com.lib.LiMS.payload.request;  
  
import jakarta.validation.constraints.NotBlank;  
import lombok.Data;  
  
@Data  
public class LoginRequest {  
    @NotBlank  
    private String username;
```

```
@NotBlank  
    private String password;  
}
```

SignupRequest.java

```
package com.lib.LiMS.payload.request;  
  
import jakarta.validation.constraints.Email;  
import jakarta.validation.constraints.NotBlank;  
import jakarta.validation.constraints.Size;  
import lombok.Data;  
  
import java.util.Set;  
  
@Data  
public class SignupRequest {  
    @NotBlank  
    @Size(min = 3, max = 20)  
    private String username;  
  
    @NotBlank  
    @Size(max = 50)  
    @Email  
    private String email;  
  
    private Set<String> roles;  
  
    @NotBlank  
    @Size(min = 6, max = 40)  
    private String password;  
}
```

jwtResponse.java

```
package com.lib.LiMS.payload.response;

import lombok.Data;

import java.util.List;

@Data
public class JwtResponse {
    private String token;
    private String type = "Bearer";
    private String id;
    private String username;
    private String email;
    private List<String> roles;

    public JwtResponse(String accessToken, String id, String username,
String email, List<String> roles) {
        this.token = accessToken;
        this.id = id;
        this.username = username;
        this.email = email;
        this.roles = roles;
    }
}
```

Book.java

```
package com.lib.LiMS.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
```

```
@Document(collection = "books")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Book {
    @Id
    private String id;

    private String title;
    private String author;
    private String genre;
    private int publicationYear;
    private String shelfLocation;
}
```

User.java

```
package com.lib.LiMS.model;

import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.DBRef;
import org.springframework.data.mongodb.core.mapping.Document;

import java.util.HashSet;
import java.util.Set;

@Document(collection = "users")
@Data
@NoArgsConstructor
public class User {
    @Id
    private String id;
```

```
private String username;  
  
private String email;  
  
private String password;  
  
@DBRef  
private Set<Role> roles = new HashSet<>();  
  
public User(String username, String email, String password) {  
    this.username = username;  
    this.email = email;  
    this.password = password;  
}  
}  
}
```

WebSecurityConfig.java

```
package com.lib.LiMS.security;  
  
import com.lib.LiMS.security.jwt.AuthEntryPointJwt;  
import com.lib.LiMS.security.jwt.AuthTokenFilter;  
import com.lib.LiMS.security.services.UserDetailsServiceImpl;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import  
org.springframework.security.authentication.AuthenticationManager;  
import  
org.springframework.security.authentication.dao.DaoAuthenticationProvider;  
import  
org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
```

```
import  
org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;  
import  
org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.http.SessionCreationPolicy;  
import  
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.security.web.SecurityFilterChain;  
import  
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;  
  
@Configuration  
@EnableMethodSecurity  
public class WebSecurityConfig {  
    @Autowired  
    UserDetailsServiceImpl userDetailsService;  
  
    @Autowired  
    private AuthEntryPointJwt unauthorizedHandler;  
  
    @Bean  
    public AuthTokenFilter authenticationJwtTokenFilter() {  
        return new AuthTokenFilter();  
    }  
  
    @Bean  
    public DaoAuthenticationProvider authenticationProvider() {  
        DaoAuthenticationProvider authProvider = new  
        DaoAuthenticationProvider();  
  
        authProvider.setUserDetailsService(userDetailsService);  
        authProvider.setPasswordEncoder(passwordEncoder());  
  
        return authProvider;  
    }  
}
```

```
    }

    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguration authConfig) throws
Exception {
    return authConfig.getAuthenticationManager();
}

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

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
        http.csrf(csrf -> csrf.disable())
            .exceptionHandling(exception ->
exception.authenticationEntryPoint(unauthorizedHandler))
            .sessionManagement(session ->
session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(auth ->
auth.requestMatchers("/api/auth/**").permitAll()
                .requestMatchers("/api/test/**").permitAll()
                .requestMatchers("/error").permitAll()
                .anyRequest().authenticated());

        http.authenticationProvider(authenticationProvider());

        http.addFilterBefore(authenticationJwtTokenFilter(),
UsernamePasswordAuthenticationFilter.class);

    return http.build();
}
}
```

LAB 11

EC/2021/006

W.K.G.K JAYAWARDANA