Spring Boot JWT Security-Mr.RAGHU

Spring Boot Application

a. application.properties

```
server.port=9900

server.servlet.context-path=/api

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/bootdb
spring.datasource.username=root
spring.datasource.password=root

spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=create
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

raghu.app.jwtSecret=NITRAGHU
raghu.app.jwtExpTimeInMs=88800000
```

b. Setup SQLs

```
create database bootdb;
use bootdb;
insert into roles_tab(name) values('ROLE_USER');
insert into roles_tab(name) values('ROLE_MODERATOR');
insert into roles_tab(name) values('ROLE_ADMIN');
```

c. Request and Response classes

```java
package in.nareshit.raghu.request;

import javax.validation.constraints.NotBlank;

import lombok.Data;
```

```
@Data
public class LoginRequest {

	@NotBlank
	private String username;

	@NotBlank
	private String password;
}


package in.nareshit.raghu.request;

import java.util.Set;

import javax.validation.constraints.Email;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Size;

import lombok.Data;

@Data
public class SignupRequest {

	@NotBlank
	@Size(min = 3, max = 20)
	private String username;

	@NotBlank
	@Size(max=50)
	@Email
	private String email;

	@NotBlank
	@Size(min = 4, max = 40)
	private String password;

	private Set<String> role;
}



package in.nareshit.raghu.response;

import java.util.Set;

import lombok.Data;
import lombok.NonNull;

@Data
public class JwtResponse {

	@NonNull
	private String token;

	private String type = "Bearer";

	@NonNull
	private Long id;

	@NonNull
	private String username;

	@NonNull
	private String email;

	@NonNull
	private Set<String> roles;
}


package in.nareshit.raghu.response;

import lombok.AllArgsConstructor;
import lombok.Data;
```

```
@Data
@AllArgsConstructor
public class MessageResponse {

	private String message;
}
```

d. Models

```
package in.nareshit.raghu.model;

public enum ERole {
	ROLE_USER,
	ROLE_MODERATOR,
	ROLE_ADMIN
}

package in.nareshit.raghu.model;

import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.Data;

@Data
@Entity
@Table(name="roles_tab")
public class Role {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Integer id;

	@Enumerated(EnumType.STRING)
	private ERole name;
```

```
}


package in.nareshit.raghu.model;

import java.util.Set;

import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;
import javax.validation.constraints.Email;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Size;

import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;

@Data
@NoArgsConstructor
@RequiredArgsConstructor
@Entity
@Table(
		name="users_tab",
		uniqueConstraints = {
				@UniqueConstraint(columnNames = "username"),
				@UniqueConstraint(columnNames = "email")
		})
public class User {

	@Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank
    @Size(max=20)
    @NonNull
    private String username;

    @NotBlank
    @Size(max=50)
    @Email
    @NonNull
    private String email;

    @NotBlank
    @Size(max=120)
    @NonNull
    private String password;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(
            name="users_roles_tab",
            joinColumns = @JoinColumn(name="user_id"),
            inverseJoinColumns = @JoinColumn(name="role_id")
            )
    private Set<Role> roles;

}

package in.nareshit.raghu.model;

import java.util.Collection;
import java.util.stream.Collectors;

import org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.authority.SimpleGrantedAuthori
ty;
```

```
import
org.springframework.security.core.userdetails.UserDetails;

import com.fasterxml.jackson.annotation.JsonIgnore;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class UserDetailsImpl implements UserDetails {

     private static final long serialVersionUID = 1L;

     private Long id;
     private String username;
     private String email;

     @JsonIgnore
     private String password;

     private Collection<? extends GrantedAuthority> authorities;

     public static UserDetailsImpl build(User user) {
          return new UserDetailsImpl(
                         user.getId(),
                         user.getUsername(),
                         user.getEmail(),
                         user.getPassword(),
                         user.getRoles()
                         .stream()
                         .map(role -> new
SimpleGrantedAuthority(role.getName().name()))
                         .collect(Collectors.toList())
                         );
     }

     public Collection<? extends GrantedAuthority>
getAuthorities() {
		return authorities;
	}

	public String getPassword() {
		return password;
	}

	public String getUsername() {
		return username;
	}

	public boolean isAccountNonExpired() {
		return true;
	}

	public boolean isAccountNonLocked() {
		return true;
	}

	public boolean isCredentialsNonExpired() {
		return true;
	}

	public boolean isEnabled() {
		return true;
	}

}
```

e. Repository Interfaces

```
package in.nareshit.raghu.repo;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;

import in.nareshit.raghu.model.ERole;
import in.nareshit.raghu.model.Role;

public interface RoleRepository extends JpaRepository<Role,
Integer>{

     Optional<Role> findByName(ERole name);
}
```

```
package in.nareshit.raghu.repo;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;

import in.nareshit.raghu.model.User;

public interface UserRepository extends JpaRepository<User,
Long>{

     Optional<User> findByUsername(String username);
     boolean existsByUsername(String username);
     boolean existsByEmail(String email);
}
```

---

f. Utils

```
package in.nareshit.raghu.util;

import java.util.Set;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.ERole;
import in.nareshit.raghu.model.Role;
import in.nareshit.raghu.repo.RoleRepository;

@Component
```

```
public class RolesUtils {

	@Autowired
	private RoleRepository repository;

	public void mapRoles(Set<String> userRoles, Set<Role> 
dbRoles) {

		if(userRoles == null || userRoles.isEmpty()) {
			Role userRole = 
repository.findByName(ERole.ROLE_USER)
					.orElseThrow(()->new 
RuntimeException("Error: Role is not found"));
			dbRoles.add(userRole);
		} else {
			userRoles.forEach(role->{
				switch (role) {
				case "admin":
					Role adminRole = 
repository.findByName(ERole.ROLE_ADMIN)
					.orElseThrow(()->new 
RuntimeException("Error: Role is not found"));
					dbRoles.add(adminRole);
					break;

				case "mod":
					Role modRole = 
repository.findByName(ERole.ROLE_MODERATOR)
					.orElseThrow(()->new 
RuntimeException("Error: Role is not found"));
					dbRoles.add(modRole);
					break;

				default:
					Role userRole = 
repository.findByName(ERole.ROLE_USER)
					.orElseThrow(()->new 
RuntimeException("Error: Role is not found"));
					dbRoles.add(userRole);
```

```
						break;
					}
				});
			}

		}
}
```

```
package in.nareshit.raghu.util;

import java.util.Date;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.UserDetailsImpl;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;

@Component
public class JwtUtils {

	@Value("${raghu.app.jwtSecret}")
	private String jwtSecret;

	@Value("${raghu.app.jwtExpTimeInMs}")
	private int jwtExpTimeInMs;

	public String generateToken(Authentication authentication)
{
		UserDetailsImpl userDetails =
(UserDetailsImpl)authentication.getPrincipal();

		return Jwts.builder()
				.setSubject(userDetails.getUsername())
				.setIssuedAt(new Date())
```

```
                    .setExpiration(new 
Date(System.currentTimeMillis() + jwtExpTimeInMs))
                    .signWith(SignatureAlgorithm.HS512, 
jwtSecret)
                    .compact()
                    ;
     }

     public String getUsernameFromJwtToken(String token) {
          return Jwts.parser()
                    .setSigningKey(jwtSecret)
                    .parseClaimsJws(token)
                    .getBody()
                    .getSubject();
     }


     public boolean validateToken(String token){
          try {
               
     Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token
);
               return true;
          } catch (Exception e) {
               e.printStackTrace();
          }
          return false;
     }

}
```

g. UserDetailsServiceImpl

```
package in.nareshit.raghu.service;

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.UsernameNotFoundEx
ception;
import org.springframework.stereotype.Service;

import in.nareshit.raghu.model.User;
import in.nareshit.raghu.model.UserDetailsImpl;
import in.nareshit.raghu.repo.UserRepository;

@Service
public class UserDetailsServiceImpl implements
UserDetailsService {

	@Autowired
	private UserRepository repository;

	@Override
	public UserDetails loadUserByUsername(String username)
			throws UsernameNotFoundException
	{
		//loading model class user object
		User user = repository.findByUsername(username)
				.orElseThrow(()->new
UsernameNotFoundException("User not exist" + username));
		//converting into Spring Security User object
		return UserDetailsImpl.build(user);
	}

}
```

h. Authentication Entry Point

```
package in.nareshit.raghu.security;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import
org.springframework.security.core.AuthenticationException;
import
org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;

/**
 * Logic is executed if login(JWT) failed
 */
@Component
public class AuthenticationEntryPointJwt implements
AuthenticationEntryPoint {

     @Override
     public void commence(HttpServletRequest request,
HttpServletResponse response,
               AuthenticationException authException) throws
IOException, ServletException {

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

     }

}
```

i. RestControllers

```
package in.nareshit.raghu.rest;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

```java
@RestController
@RequestMapping("/home")
public class HomeRestController {

	@GetMapping("/all")
	public String allowAll() {
		return "Permit All Data!";
	}

	@GetMapping("/user")
	public String userData() {
		return "User Data!";
	}

	@GetMapping("/mod")
	public String moderatorData() {
		return "Moderator Data!";
	}

	@GetMapping("/admin")
	public String adminData() {
		return "Admin Data!";
	}


}

package in.nareshit.raghu.rest;

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

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
```

```
import
org.springframework.security.authentication.AuthenticationManage
r;
import
org.springframework.security.authentication.UsernamePasswordAuth
enticationToken;
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.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import in.nareshit.raghu.model.Role;
import in.nareshit.raghu.model.User;
import in.nareshit.raghu.model.UserDetailsImpl;
import in.nareshit.raghu.repo.UserRepository;
import in.nareshit.raghu.request.LoginRequest;
import in.nareshit.raghu.request.SignupRequest;
import in.nareshit.raghu.response.JwtResponse;
import in.nareshit.raghu.response.MessageResponse;
import in.nareshit.raghu.util.JwtUtils;
import in.nareshit.raghu.util.RolesUtils;

@RestController
@RequestMapping("/auth")
public class AuthenticationRestController {

	@Autowired
	private AuthenticationManager authenticationManager;

	@Autowired
	private JwtUtils jwtUtils;

	@Autowired
	private UserRepository userRepository;
```

```
    @Autowired
    private RolesUtils rolesUtils;

    @Autowired
    private PasswordEncoder encoder;

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

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

         // set as SecurityContext(Authentication)

    SecurityContextHolder.getContext().setAuthentication(authen
tication);

         // Generate JWT Token
         String jwt = jwtUtils.generateToken(authentication);

         //current user object
         UserDetailsImpl userDetails=
(UserDetailsImpl)authentication.getPrincipal();

         // return response
         return ResponseEntity.ok(
                   new JwtResponse(
                             jwt, //token
                             userDetails.getId(), //id
```

```
                              userDetails.getUsername(),
//username
                              userDetails.getEmail(),  //email
                              userDetails.getAuthorities()
                              .stream()
                              .map(auth->auth.getAuthority())
                              .collect(Collectors.toSet())
//Set<String>
                              )
                    );
     }

     //register
     @PostMapping("/register")
     public ResponseEntity<?> createUser(
               @Valid @RequestBody SignupRequest signupRequest
               )
     {
          // check username exist

     if(userRepository.existsByUsername(signupRequest.getUsernam
e())) {
               return ResponseEntity
                         .badRequest()
                         .body(new MessageResponse("Error  :
Username already exist"));
          }
          //check email exist

     if(userRepository.existsByEmail(signupRequest.getEmail()))
{
               return ResponseEntity
                         .badRequest()
                         .body(new MessageResponse("Error  :
EmailId already exist"));
          }

          //create user
          User user = new User(
                    signupRequest.getUsername(),
                    signupRequest.getEmail(),
                    encoder.encode(signupRequest.getPassword())
                    );
          //roles given by UI
          Set<String> usrRoles = signupRequest.getRole();
          //roles need to be stored in DB
          Set<Role> dbRoles = new HashSet<>();

          rolesUtils.mapRoles(usrRoles, dbRoles);
          user.setRoles(dbRoles);
          userRepository.save(user);

          return ResponseEntity.ok(new MessageResponse("User
Created Successfully!"));
     }


}
```

j. Security Filter

```
package in.nareshit.raghu.filter;

import java.io.IOException;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.authentication.UsernamePasswordAuth
enticationToken;
```

```
import
org.springframework.security.core.context.SecurityContextHolder;
import
org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.web.authentication.WebAuthenticatio
nDetailsSource;
import org.springframework.util.StringUtils;
import org.springframework.web.filter.OncePerRequestFilter;

import in.nareshit.raghu.service.UserDetailsServiceImpl;
import in.nareshit.raghu.util.JwtUtils;

public class SecurityFilter extends OncePerRequestFilter {

	@Autowired
	private JwtUtils jwtUtils;
	@Autowired
	private UserDetailsServiceImpl userDetailsService;

	protected void doFilterInternal(
			HttpServletRequest request,
			HttpServletResponse response,
			FilterChain filterChain)
			throws ServletException, IOException
	{
		try {
			String jwt = readJwtToken(request);
			if(jwt!=null && jwtUtils.validateToken(jwt)) {
				String username =
jwtUtils.getUsernameFromJwtToken(jwt);
				//load user object from db
				UserDetails userDetails =
userDetailsService.loadUserByUsername(username);
				UsernamePasswordAuthenticationToken
authentication =
						new
UsernamePasswordAuthenticationToken(
							userDetails,
                                         null,

    userDetails.getAuthorities()
                                         );
                    authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));

    SecurityContextHolder.getContext().setAuthentication(authen
tication);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            filterChain.doFilter(request, response);

    }

    private String readJwtToken(HttpServletRequest request) {
            String headerAuth = request.getHeader("Authorization");
            if(StringUtils.hasText(headerAuth) &&
headerAuth.startsWith("Bearer ")) {
                return headerAuth.substring(7,
headerAuth.length());
            }
            return null;
    }

}
```

k. Security Config

```
package in.nareshit.raghu.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.authentication.AuthenticationManage
r;
```

```
import
org.springframework.security.config.annotation.authentication.bu
ilders.AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.Http
Security;
import
org.springframework.security.config.annotation.web.configuration
.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration
.WebSecurityConfigurerAdapter;
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.AuthenticationEntryPoint;
import
org.springframework.security.web.authentication.UsernamePassword
AuthenticationFilter;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import
org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import in.nareshit.raghu.filter.SecurityFilter;
import in.nareshit.raghu.service.UserDetailsServiceImpl;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter
{

    @Autowired
    private UserDetailsServiceImpl userDetailsService;
    @Autowired
    private AuthenticationEntryPoint authenticationEntryPoint;

    @Bean
    public SecurityFilter securityFilter() {
        return new SecurityFilter();
    }

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

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

    protected void configure(AuthenticationManagerBuilder auth)
throws Exception {
        auth.userDetailsService(userDetailsService)
        .passwordEncoder(passwordEncoder());
    }

    protected void configure(HttpSecurity http) throws
Exception {
        http.cors().and().csrf().disable()
        .authorizeRequests()
        .antMatchers("/auth/**").permitAll()
        .antMatchers("/home/all").permitAll()

    .antMatchers("/home/mod").hasAuthority("ROLE_MODERATOR")
        .antMatchers("/home/admin").hasAuthority("ROLE_ADMIN")
        .anyRequest().authenticated()

        .and()
```

```
          .exceptionHandling()
          .authenticationEntryPoint(authenticationEntryPoint)

          .and()
          .sessionManagement()
          .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
          ;

          http.addFilterBefore(securityFilter(), 
UsernamePasswordAuthenticationFilter.class);
     }

     @Bean
     public CorsConfigurationSource corsConfigurationSource() {
          UrlBasedCorsConfigurationSource source = new 
                    UrlBasedCorsConfigurationSource();
          source.registerCorsConfiguration("/**", 
                    new 
CorsConfiguration().applyPermitDefaultValues());
          return source;
     }

}
```

Angular Application

```
ng new angular-spring-jwt

ng g class models/login-model --skipTests=true
ng g class models/user-model --skipTests=true
ng g class models/message-model --skipTests=true
ng g class models/jwt-model --skipTests=true
ng g s services/authentication --skipTests=true
ng g s services/token-storage --skipTests=true
ng g s services/user --skipTests=true

ng g c pages/login-page --skipTests=true
ng g c pages/register-page --skipTests=true
ng g c pages/home-page --skipTests=true
ng g c pages/profile-page --skipTests=true
ng g c pages/admin-page --skipTests=true
ng g c pages/moderator-page --skipTests=true
ng g c pages/user-page --skipTests=true

ng g interceptor security/auth --skipTests=true
```

a. models

```
export class UserModel {
    constructor(
        public username: String,
        public email: String,
        public password: String
    ) {

    }
}


export class MessageModel {

    constructor(public message:string) {

    }
}
```

```
export class LoginModel {
    constructor(
        public username: String,
        public password: String
    ) {

    }
}


export class JwtModel {

    constructor(
        public token:string,
        public type:string,
        public id:number,
        public username:string,
        public email:string,
        public roles:string[])
        {

    }
}
```

B. SERVICES

AuthenticationService.ts

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { JwtModel } from '../models/jwt-model';
import { LoginModel } from '../models/login-model';
import { MessageModel } from '../models/message-model';
import { UserModel } from '../models/user-model';

const baseUrl = 'http://localhost:9900/api/auth';

@Injectable({
  providedIn: 'root',
})
export class AuthenticationService {
  constructor(private http: HttpClient) {}

  // register
  registerUser(user: UserModel): Observable<MessageModel> {
    return this.http.post<MessageModel>(`${baseUrl}/register`, u
ser);
  }

  // login
  loginUser(login: LoginModel): Observable<JwtModel> {
    return this.http.post<JwtModel>(`${baseUrl}/login`, login);
  }
}
```

UserService.ts

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';

const baseUrl = 'http://localhost:9900/api/home';

@Injectable({
  providedIn: 'root',
})
export class UserService {
  constructor(private http: HttpClient) {}

  getPublicContent(): Observable<any> {
    return this.http.get(baseUrl + '/all', { responseType: 'text
' });
  }

  getUserData(): Observable<any> {
    return this.http.get(baseUrl + '/user', { responseType: 'tex
t' });
  }

  getModeratorData(): Observable<any> {
    return this.http.get(baseUrl + '/mod', { responseType: 'text
' });
  }

  getAdminData(): Observable<any> {
    return this.http.get(baseUrl + '/admin', { responseType: 'te
xt' });
  }
}
```

TokenStorageService.ts

```
import { Injectable } from '@angular/core';

const TOKEN_KEY = 'auth-token';
const USER_KEY = 'auth-user';

@Injectable({
  providedIn: 'root',
})
export class TokenStorageService {
  constructor() {}

  signOut(): void {
    window.sessionStorage.clear();
  }

  public saveToken(token: string): void {
    window.sessionStorage.removeItem(TOKEN_KEY);
    window.sessionStorage.setItem(TOKEN_KEY, token);
  }

  public getToken(): string | null {
    return window.sessionStorage.getItem(TOKEN_KEY);
  }

  public saveUser(user: any): void {
    window.sessionStorage.removeItem(USER_KEY);
    window.sessionStorage.setItem(USER_KEY, JSON.stringify(user)
);
  }

  public getUser(): any {
    const user = window.sessionStorage.getItem(USER_KEY);
    if (user) {
      return JSON.parse(user);
    }

    return {};
  }
}
```

C. PAGES

<u>RegisterPageComponent.ts</u>

```
import { Component, OnInit } from '@angular/core';
import { UserModel } from 'src/app/models/user-model';
import { AuthenticationService } from 'src/app/services/authenti
cation.service';

@Component({
  selector: 'app-register-page',
  templateUrl: './register-page.component.html',
  styleUrls: ['./register-page.component.css']
})
export class RegisterPageComponent implements OnInit {

  user: UserModel = new UserModel('', '', '');
  isSuccessful: boolean = false;
  message: string = '';

  constructor(private auth: AuthenticationService) { }

  ngOnInit(): void {
  }

  onSubmit() {
    this.auth.registerUser(this.user).subscribe(
      data => {
        console.log(data);
        this.isSuccessful = true;
        this.message = data.message;
      }, err => {
        console.log(err);
        this.isSuccessful = false;
        this.message = err.error.message;
      });
  }
}
```

<u>RegisterPageComponent.html</u>

```
<h3>User Register Form</h3>
<form
  *ngIf="!isSuccessful"
  name="form"
  (ngSubmit)="registerForm.form.valid && onSubmit()"
  #registerForm="ngForm"
>
  <div class="form-group">
    <label for="username">Username</label>
    <input
      type="text"
      class="form-control"
      name="username"
      [(ngModel)]="user.username"
      required
      minlength="3"
      maxlength="20"
      #username="ngModel"
    />
    <div
      class="alert-danger"
      *ngIf="username.errors && (username.touched || registerFor
m.submitted)"
    >
      <div *ngIf="username.errors.required">Username is required
</div>
      <div *ngIf="username.errors.minlength">
        Username must be at least 3 characters
      </div>
      <div *ngIf="username.errors.maxlength">
```

```
        Username must be at most 20 characters
      </div>
    </div>
  </div>

  <div class="form-group">
    <label for="email">Email</label>
    <input
      type="email"
      class="form-control"
      name="email"
      [(ngModel)]="user.email"
      required
      email
      #email="ngModel"
    />
    <div
      class="alert-danger"
      *ngIf="email.invalid && (email.touched || registerForm.sub
mitted)"
    >
      <div *ngIf="email.errors?.required">Email is required</div
>
      <div *ngIf="email.errors?.email">Email must be a valid ema
il address</div>
    </div>
  </div>

  <div class="form-group">
    <label for="password">Password</label>
    <input
      type="password"
      class="form-control"
      name="password"
      [(ngModel)]="user.password"
      required
      minlength="6"
      #password="ngModel"
    />
    <div
      class="alert-danger"
      *ngIf="password.invalid && (password.touched || registerFo
rm.submitted)"
    >
      <div *ngIf="password.errors?.required">Password is require
d</div>
      <div *ngIf="password.errors?.minlength">
        Password must be at least 6 characters
      </div>
    </div>
  </div>

  <div class="form-group">
    <button [disabled]="!registerForm.form.valid" class="btn btn
-success btn-block">Sign Up</button>
  </div>

  <div
    class="alert alert-warning"
    *ngIf="registerForm.submitted && !isSuccessful && message !=
 ''"
  >
    {{ message }}
  </div>
</form>

<div class="alert alert-success" *ngIf="isSuccessful">
  {{ message }}
```

```
</div>
```

LoginPageComponent.ts

```
import { Component, OnInit } from '@angular/core';
import { LoginModel } from 'src/app/models/login-model';
import { AuthenticationService } from 'src/app/services/authenti
cation.service';
import { TokenStorageService } from 'src/app/services/token-
storage.service';

@Component({
  selector: 'app-login-page',
  templateUrl: './login-page.component.html',
  styleUrls: ['./login-page.component.css'],
})
export class LoginPageComponent implements OnInit {
  //------variables----------------------
  isLoggedIn = false;
  isLoginFailed = false;
  errorMessage = '';
  roles: string[] = [];
  loginModel : LoginModel = new LoginModel('','');
  //---------DI-------------------
  constructor(
    private authService: AuthenticationService,
    private tokenStorage: TokenStorageService
  ) {}
  //---------OnInit-------------------
  ngOnInit(): void {
    if (this.tokenStorage.getToken()) {
      this.isLoggedIn = true;
      this.roles = this.tokenStorage.getUser().roles;
    }
  }

  onSubmit(): void {

    this.authService.loginUser(this.loginModel).subscribe(
      data => {
        this.tokenStorage.saveToken(data.token);
        this.tokenStorage.saveUser(data);

        this.isLoginFailed = false;
        this.isLoggedIn = true;
        this.roles = this.tokenStorage.getUser().roles;
        this.reloadPage();
      },
      err => {
        this.errorMessage = err.error.message;
        this.isLoginFailed = true;
      }
    );
  }

  reloadPage(): void {
    window.location.reload();
  }
}
```

<u>LoginPageComponent.html</u>

```
<h3>User Login Form</h3>
<form
  *ngIf="!isLoggedIn"
  name="form"
  (ngSubmit)="loginForm.form.valid && onSubmit()"
  #loginForm="ngForm"
```

```
>
  <div class="form-group">
    <label for="username">Username</label>
    <input
      type="text"
      class="form-control"
      name="username"
      [(ngModel)]="loginModel.username"
      required
      #username="ngModel"
    />
    <div
      class="alert alert-danger"
      role="alert"
      *ngIf="username.errors && (username.touched || loginForm.s
ubmitted)"
    >
      Username is required!
    </div>
  </div>
  <div class="form-group">
    <label for="password">Password</label>
    <input
      type="password"
      class="form-control"
      name="password"
      [(ngModel)]="loginModel.password"
      required
      minlength="6"
      #password="ngModel"
    />
    <div
      class="alert alert-danger"
      role="alert"
      *ngIf="password.errors && (password.touched || loginForm.s
ubmitted)"
    >
      <div *ngIf="password.errors.required">Password is required
</div>
      <div *ngIf="password.errors.minlength">
        Password must be at least 6 characters
      </div>
    </div>
  </div>
  <div class="form-group">
    <button [disabled]="!loginForm.form.valid" class="btn btn-
success btn-block">Login</button>
  </div>
  <div class="form-group">
    <div
      class="alert alert-danger"
      role="alert"
      *ngIf="loginForm.submitted && isLoginFailed"
    >
      Login failed: {{ errorMessage }}
    </div>
  </div>
</form>

<div class="alert alert-success" *ngIf="isLoggedIn">
  Logged in as {{ roles }}.
</div>
```

HomePageComponent.ts

```
import { Component, OnInit } from '@angular/core';
import { UserService } from 'src/app/services/user.service';

@Component({
  selector: 'app-home-page',
  templateUrl: './home-page.component.html',
  styleUrls: ['./home-page.component.css'],
})
export class HomePageComponent implements OnInit {
  content?: string;

  constructor(private userService: UserService) {}

  ngOnInit(): void {
    this.userService.getPublicContent().subscribe(
      (data) => {
        this.content = data;
      },
      (err) => {
        console.log(err);
      }
    );
  }
}
```

HomePageComponent.html

```
<div class="container">
  <header class="jumbotron">
    <p>{{ content }}</p>
  </header>
</div>
```

ProfilePageComponent.ts

```
import { Component, OnInit } from '@angular/core';
import { TokenStorageService } from 'src/app/services/token-
storage.service';
```

```
@Component({
  selector: 'app-profile-page',
  templateUrl: './profile-page.component.html',
  styleUrls: ['./profile-page.component.css']
})
export class ProfilePageComponent implements OnInit {

  currentUser: any;

  constructor(private token: TokenStorageService) { }

  ngOnInit(): void {
    this.currentUser = this.token.getUser();
  }

}
```

ProfilePageComponent.html

```
<div class="container" *ngIf="currentUser; else loggedOut">
  <header class="jumbotron">
    <h3>
      <strong>{{ currentUser.username }}</strong> Profile
    </h3>
  </header>
  <p>
    <strong>Token:</strong>
    {{ currentUser.token.substring(0, 20) }} ...
    {{ currentUser.token.substr(currentUser.token.length -
 20) }}
  </p>
  <p>
    <strong>Email:</strong>
    {{ currentUser.email }}
  </p>
  <strong>Roles:</strong>
  <ul>
    <li *ngFor="let role of currentUser.roles">
      {{ role }}
    </li>
  </ul>
</div>

<ng-template #loggedOut> Please login. </ng-template>
```

UserPageComponent.ts

```
import { Component, OnInit } from '@angular/core';
import { UserService } from 'src/app/services/user.service';

@Component({
  selector: 'app-user-page',
  templateUrl: './user-page.component.html',
  styleUrls: ['./user-page.component.css']
})
export class UserPageComponent implements OnInit {
  content?: string;

  constructor(private userService: UserService) { }

  ngOnInit(): void {
    this.userService.getUserData().subscribe(
      data => {
        this.content = data;
      },
      err => {
        console.log(err);
      }
    );
  }

}
```

UserPageComponent.ts

```
<div class="container">
  <header class="jumbotron">
    <p>{{ content }}</p>
  </header>
</div>
```

AdminPageComponent.ts

```
import { Component, OnInit } from '@angular/core';
import { UserService } from 'src/app/services/user.service';

@Component({
  selector: 'app-admin-page',
  templateUrl: './admin-page.component.html',
  styleUrls: ['./admin-page.component.css'],
})
export class AdminPageComponent implements OnInit {
  content?: string;

  constructor(private userService: UserService) {}

  ngOnInit(): void {
    this.userService.getAdminData().subscribe(
      (data) => {
        this.content = data;
      },
      (err) => {
        console.log(err);
      }
    );
  }
}
```

```
AdminPageComponent.html
<div class="container">
  <header class="jumbotron">
    <p>{{ content }}</p>
  </header>
</div>
```

```
ModeratorPageComponent.ts

import { Component, OnInit } from '@angular/core';
import { UserService } from 'src/app/services/user.service';

@Component({
  selector: 'app-moderator-page',
  templateUrl: './moderator-page.component.html',
  styleUrls: ['./moderator-page.component.css']
})
export class ModeratorPageComponent implements OnInit {

  content?: string;

  constructor(private userService: UserService) { }

  ngOnInit(): void {
    this.userService.getModeratorData().subscribe(
      data => {
        this.content = data;
      },
      err => {
        console.log(err);
      }
    );
  }

}
```

```
ModeratorPageComponent.html
<div class="container">
    <header class="jumbotron">
      <p>{{ content }}</p>
    </header>
  </div>
```

```
d. Security Interceptor
AuthInterceptor.ts

import { Injectable } from '@angular/core';
import {
  HttpRequest,
  HttpHandler,
  HttpEvent,
  HttpInterceptor,
  HTTP_INTERCEPTORS,
} from '@angular/common/http';
import { Observable } from 'rxjs';
import { TokenStorageService } from '../services/token-
storage.service';

const TOKEN_HEADER_KEY = 'Authorization';
```

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {
  //inject token service
  constructor(private tokenService: TokenStorageService) {}

  intercept(
    request: HttpRequest<any>,
    next: HttpHandler
  ): Observable<HttpEvent<any>> {
    let authReq = request;
    const token = this.tokenService.getToken();
    if (token != null) {
      authReq = request.clone({
        headers: request.headers.set(TOKEN_HEADER_KEY, 'Bearer '
 + token),
      });
    }
    return next.handle(authReq);
  }
}

export const authInterceptorProviders = [
  { provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi
: true },
];
```

AppRouting.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { AdminPageComponent } from './pages/admin-page/admin-
page.component';
import { HomePageComponent } from './pages/home-page/home-
page.component';
```

```
import { LoginPageComponent } from './pages/login-page/login-
page.component';
import { ModeratorPageComponent } from './pages/moderator-
page/moderator-page.component';
import { ProfilePageComponent } from './pages/profile-
page/profile-page.component';
import { RegisterPageComponent } from './pages/register-
page/register-page.component';
import { UserPageComponent } from './pages/user-page/user-
page.component';

const routes: Routes = [
  { path: 'home', component: HomePageComponent },
  { path: 'login', component: LoginPageComponent },
  { path: 'register', component: RegisterPageComponent },
  { path: 'profile', component: ProfilePageComponent },
  { path: 'user', component: UserPageComponent },
  { path: 'mod', component: ModeratorPageComponent },
  { path: 'admin', component: AdminPageComponent },
  { path: '', redirectTo: 'home', pathMatch: 'full' },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

AppComponent.html

```
<div id="app">
    <nav class="navbar navbar-expand navbar navbar-dark bg-
primary">
      <a href="#" class="navbar-brand">NareshIT</a>
      <ul class="navbar-nav mr-auto" routerLinkActive="active">
```

```
        <li class="nav-item">
          <a href="/home" class="nav-link text-
white" routerLink="home">Home </a>
        </li>
        <li class="nav-item" *ngIf="showAdminPage">
          <a href="/admin" class="nav-link text-
white" routerLink="admin">Admin</a>
        </li>
        <li class="nav-item" *ngIf="showModeratorPage">
          <a href="/mod" class="nav-link text-
white" routerLink="mod">Moderator</a>
        </li>
        <li class="nav-item">
          <a href="/user" class="nav-link text-
white" *ngIf="isLoggedIn" routerLink="user">User</a>
        </li>
      </ul>

      <ul class="navbar-nav ml-auto" *ngIf="!isLoggedIn">
        <li class="nav-item">
          <a href="/register" class="nav-link text-
white" routerLink="register">Sign Up</a>
        </li>
        <li class="nav-item">
          <a href="/login" class="nav-link text-
white" routerLink="login">Login</a>
        </li>
      </ul>

      <ul class="navbar-nav ml-auto" *ngIf="isLoggedIn">
        <li class="nav-item">
          <a href="/profile" class="nav-link text-
white" routerLink="profile">{{ username }}</a>
        </li>
        <li class="nav-item">
          <a href class="nav-link text-
white" (click)="logout()">LogOut</a>
        </li>
      </ul>
    </nav>

    <div class="container">
      <router-outlet></router-outlet>
    </div>
  </div>


import { Component } from '@angular/core';
import { TokenStorageService } from './services/token-
storage.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
})
export class AppComponent {
  private roles: string[] = [];
  isLoggedIn = false;
  showAdminPage = false;
  showModeratorPage = false;
  username?: string;

  constructor(private tokenStorageService: TokenStorageService)
{}

  ngOnInit(): void {
    this.isLoggedIn = !!this.tokenStorageService.getToken();
    if (this.isLoggedIn) {
      const user = this.tokenStorageService.getUser();
      this.roles = user.roles;

      this.showAdminPage = this.roles.includes('ROLE_ADMIN');
      this.showModeratorPage = this.roles.includes('ROLE_MODERAT
OR');

      this.username = user.username;
    }
  }

  logout(): void {
    this.tokenStorageService.signOut();
    window.location.reload();
  }
}
```

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { LoginPageComponent } from './pages/login-page/login-
page.component';
import { RegisterPageComponent } from './pages/register-
page/register-page.component';
import { HomePageComponent } from './pages/home-page/home-
page.component';
import { ProfilePageComponent } from './pages/profile-
page/profile-page.component';
import { AdminPageComponent } from './pages/admin-page/admin-
page.component';
```

```
import { ModeratorPageComponent } from './pages/moderator-
page/moderator-page.component';
import { UserPageComponent } from './pages/user-page/user-
page.component';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { authInterceptorProviders } from './security/auth.interc
eptor';

@NgModule({
  declarations: [
    AppComponent,
    LoginPageComponent,
    RegisterPageComponent,
    HomePageComponent,
    ProfilePageComponent,
    AdminPageComponent,
    ModeratorPageComponent,
    UserPageComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    HttpClientModule
  ],
  providers: [authInterceptorProviders],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

```
index.html
<!doctype html>
```

```
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>Angularjwtapp</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-
scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/b
ootstrap/4.0.0/css/bootstrap.min.css"
    >
  <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/
1.12.9/umd/popper.min.js"

    crossorigin></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/j
s/bootstrap.min.js"

> </script>
</head>

<body>
  <div class="container">
    <app-root></app-root>
  </div>
</body>

</html>
```