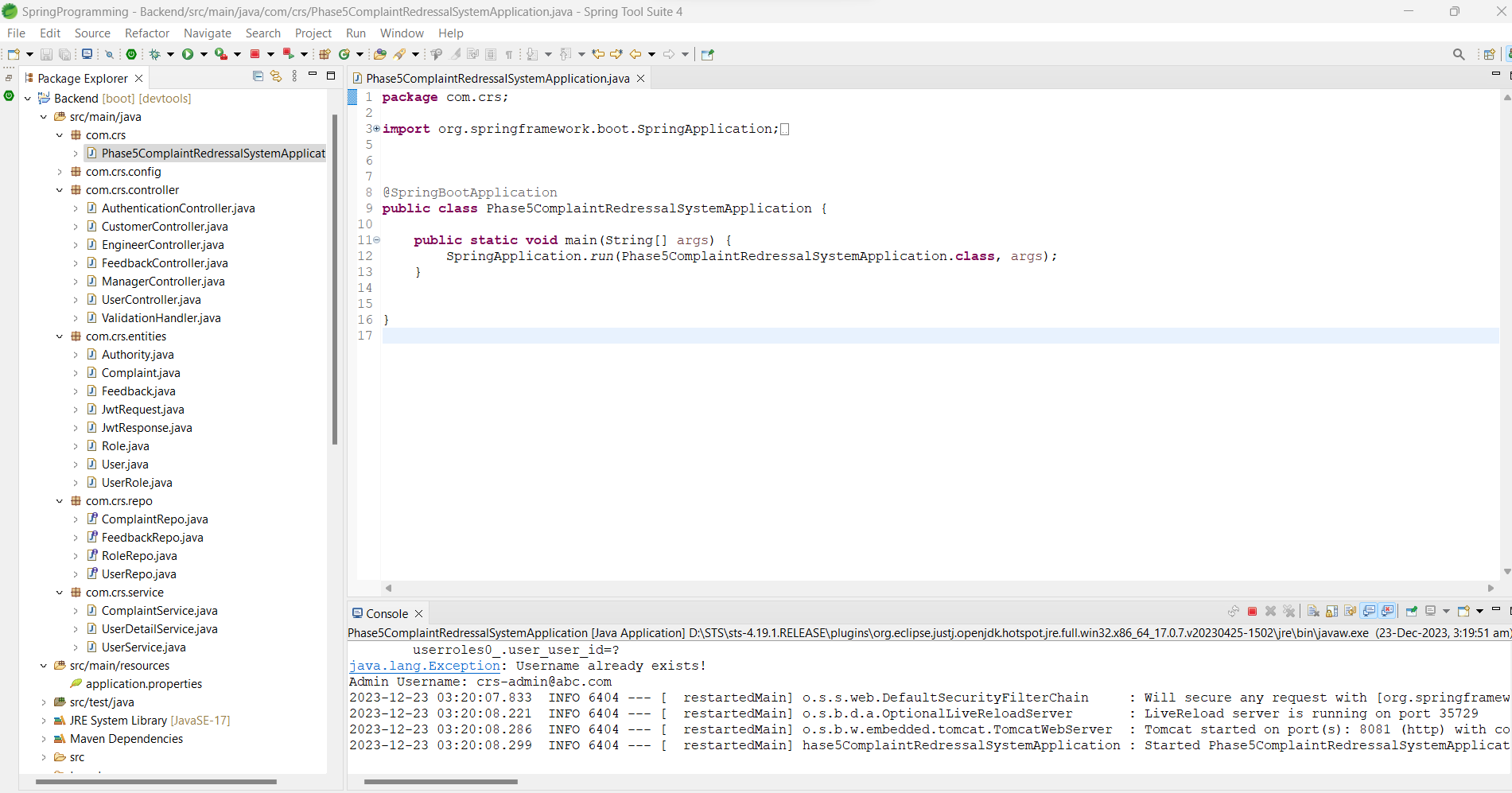
GitHub Link:

Backend



1) package com.crs.config;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import java.util.function.Function;

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

import org.springframework.stereotype.Component;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

@Component

public class JwtUtil {

private String SECRET\_KEY = "secret";

public String extractUsername(String token) {

return extractClaim(token, Claims::getSubject);

}

public Date extractExpiration(String token) {

return extractClaim(token, Claims::getExpiration);

}

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

final Claims claims = extractAllClaims(token);

return claimsResolver.apply(claims);

}

private Claims extractAllClaims(String token) {

return Jwts.parser().setSigningKey(SECRET\_KEY).parseClaimsJws(token).getBody();

}

private boolean isTokenExpired(String token) {

return extractExpiration(token).before(new Date());

}

public String generateToken(UserDetails userDetails) {

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

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

}

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

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

.setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 60 \* 10))

.signWith(SignatureAlgorithm.HS256, SECRET\_KEY).compact();

}

public boolean validateToken(String token, UserDetails userDetails) {

final String username = extractUsername(token);

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

}

2) package com.crs.config;

import java.io.IOException;

import java.util.Iterator;

import java.util.LinkedList;

import java.util.List;

import org.springframework.security.core.GrantedAuthority;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import com.fasterxml.jackson.core.JacksonException;

import com.fasterxml.jackson.core.JsonParser;

import com.fasterxml.jackson.databind.DeserializationContext;

import com.fasterxml.jackson.databind.JsonDeserializer;

import com.fasterxml.jackson.databind.JsonNode;

import com.fasterxml.jackson.databind.ObjectMapper;

public class CustomAuthorityDeserializer extends JsonDeserializer<Object>{

@Override

public Object deserialize(JsonParser jp, DeserializationContext ctxt) throws IOException, JacksonException {

ObjectMapper mapper = (ObjectMapper) jp.getCodec();

JsonNode jsonNode = mapper.readTree(jp);

List<GrantedAuthority> grantedAuthorities = new LinkedList<>();

Iterator<JsonNode> elements = jsonNode.elements();

while (elements.hasNext()) {

JsonNode next = elements.next();

JsonNode authority = next.get("authority");

grantedAuthorities.add(new SimpleGrantedAuthority(authority.asText()));

}

return grantedAuthorities;

}

}

3)

package com.crs.config;

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.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 com.crs.service.UserDetailService;

import io.jsonwebtoken.ExpiredJwtException;

@Component

public class JwtAuthFilter extends OncePerRequestFilter{

@Autowired

private UserDetailService userDetailService;

@Autowired

private JwtUtil jwtUtil;

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)

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 = this.jwtUtil.extractUsername(jwtToken);

}catch(ExpiredJwtException e) {

e.printStackTrace();

System.out.println("Token Expired!");

}catch(Exception e) {

e.printStackTrace();

}

}else {

System.out.println("Invalid token! Not starting from bearer string!");

}

// validated

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

final UserDetails userDetails = this.userDetailService.loadUserByUsername(username);

if(this.jwtUtil.validateToken(jwtToken, userDetails)) {

//token is valid

UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(userDetails,null,userDetails.getAuthorities());

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

SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);

}

}else {

System.out.println("Token is not valid! Please generate a new token!");

}

filterChain.doFilter(request, response);

}

}

5) package com.crs.config;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import java.util.function.Function;

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

import org.springframework.stereotype.Component;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

@Component

public class JwtUtil {

private String SECRET\_KEY = "secret";

public String extractUsername(String token) {

return extractClaim(token, Claims::getSubject);

}

public Date extractExpiration(String token) {

return extractClaim(token, Claims::getExpiration);

}

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

final Claims claims = extractAllClaims(token);

return claimsResolver.apply(claims);

}

private Claims extractAllClaims(String token) {

return Jwts.parser().setSigningKey(SECRET\_KEY).parseClaimsJws(token).getBody();

}

private boolean isTokenExpired(String token) {

return extractExpiration(token).before(new Date());

}

public String generateToken(UserDetails userDetails) {

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

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

}

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

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

.setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 60 \* 10))

.signWith(SignatureAlgorithm.HS256, SECRET\_KEY).compact();

}

public boolean validateToken(String token, UserDetails userDetails) {

final String username = extractUsername(token);

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

}

}

6) package com.crs.config;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.http.HttpMethod;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

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.web.authentication.UsernamePasswordAuthenticationFilter;

import com.crs.service.UserDetailService;

@SuppressWarnings("deprecation")

@EnableWebSecurity

@Configuration

public class SecurityConfig extends WebSecurityConfigurerAdapter{

@Autowired

private UserDetailService userDetailService;

@Autowired

private AuthEntryPoint authEntryPoint;

@Autowired

private JwtAuthFilter jwtAuthFilter;

@Bean

public BCryptPasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Override

@Bean

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(this.userDetailService).passwordEncoder(passwordEncoder());

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.csrf()

.disable()

.cors()

.disable()

.authorizeRequests()

.antMatchers("/generate-token").permitAll()

.antMatchers(HttpMethod.OPTIONS).permitAll()

.antMatchers("/user/\*\*").hasAuthority("ADMIN")

.antMatchers("/customer/\*\*").hasAuthority("CUSTOMER")

.antMatchers("/manager/\*\*").hasAuthority("MANAGER")

.antMatchers("/engineer/\*\*").hasAuthority("ENGINEER")

.anyRequest().authenticated()

.and().exceptionHandling().authenticationEntryPoint(authEntryPoint)

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

http.addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);

}

}

7) package com.crs.controller;

import java.security.Principal;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.security.authentication.AuthenticationManager;

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.security.core.userdetails.UsernameNotFoundException;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.web.bind.annotation.CrossOrigin;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.PutMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RestController;

import com.crs.config.JwtUtil;

import com.crs.entities.JwtRequest;

import com.crs.entities.JwtResponse;

import com.crs.entities.User;

import com.crs.repo.UserRepo;

import com.crs.service.UserDetailService;

@RestController

@CrossOrigin("http://localhost:4200")

public class AuthenticationController {

@Autowired

private AuthenticationManager authenticationManager;

@Autowired

private UserDetailService userDetailService;

@Autowired

private JwtUtil jwtUtil;

@Autowired

private UserRepo repo;

@Autowired

private BCryptPasswordEncoder passwordEncoder;

//generate token

@PostMapping("/generate-token")

public ResponseEntity<?> generateToken(@RequestBody JwtRequest jwtRequest) throws Exception{

try {

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

}catch(UsernameNotFoundException e) {

e.printStackTrace();

throw new Exception("User does not exist!");

}

//validated

UserDetails userDetails = this.userDetailService.loadUserByUsername(jwtRequest.getUsername());

String token = this.jwtUtil.generateToken(userDetails);

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

}

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

try {

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

} catch (BadCredentialsException e) {

throw new Exception("Invalid Credentials! "+e.getMessage());

}catch(DisabledException e) {

throw new Exception("User Disabled! "+e.getMessage());

}

}

//return the details of current user

@GetMapping("/current-user")

public User getCurrentUser(Principal principal) {

return ((User)this.userDetailService.loadUserByUsername(principal.getName()));

}

@PutMapping("/change-password")

public ResponseEntity<?> changePassword(@RequestBody User user){

User u = this.repo.findByUsername(user.getUsername());

if(u!=null) {

u.setPassword(this.passwordEncoder.encode(user.getPassword()));

this.repo.save(u);

return ResponseEntity.status(HttpStatus.CREATED).build();

}else {

return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).build();

}

}

}

8) package com.crs.controller;

import java.text.DateFormat;

import java.util.Calendar;

import java.util.List;

import javax.validation.Valid;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.CrossOrigin;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

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 com.crs.entities.Complaint;

import com.crs.entities.Feedback;

import com.crs.service.ComplaintService;

@RestController

@CrossOrigin("http://localhost:4200")

@RequestMapping("/customer")

public class CustomerController {

@Autowired

private ComplaintService complaintService;

@PostMapping("/create-complaint")

public ResponseEntity<Complaint> createComplaint(@Valid @RequestBody Complaint complaint) throws Exception{

DateFormat df = DateFormat.getDateInstance();

Calendar cl = Calendar.getInstance();

String complaintDate = df.format(cl.getTime());

complaint.setDate(complaintDate);

complaint.setStatus("RAISED");

complaint.setActive(true);

complaint.setAssigned(false);

complaint.setRemark("Ticket Raised.");

Complaint newComplaint = this.complaintService.createComplaint(complaint);

// URI location = ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}").buildAndExpand(newComplaint.getId()).toUri();

// return ResponseEntity.created(location).build();

return ResponseEntity.ok(newComplaint);

}

@GetMapping("/get-complaint/{username}")

public ResponseEntity<?> getComplaintByUsername(@PathVariable("username") String username){

List<Complaint> complaints = this.complaintService.findComplaintByUsername(username);

if(complaints.isEmpty()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}else {

return ResponseEntity.ok(complaints);

}

}

@GetMapping("/complaint-feedback/{id}")

public ResponseEntity<?> getComplaintById(@PathVariable("id") int id){

Complaint complaint = this.complaintService.getComplaint(id);

return ResponseEntity.ok(complaint);

}

@PostMapping("/save-feedback")

public ResponseEntity<?> saveFeedback(@RequestBody Feedback feedback) throws Exception{

Feedback savedFeedback = this.complaintService.saveFeedback(feedback);

return ResponseEntity.ok(savedFeedback);

}

}

9) package com.crs.controller;

import java.util.List;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.CrossOrigin;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PutMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.crs.entities.Complaint;

import com.crs.service.ComplaintService;

@RestController

@CrossOrigin("http://localhost:4200")

@RequestMapping("/engineer")

public class EngineerController {

@Autowired

private ComplaintService complaintService;

@GetMapping("/get-all-complaints/{assignedEngineer}")

public ResponseEntity<?> getAssignedComplaints(@PathVariable("assignedEngineer") String assignedEngineer){

List<Complaint> complaints = this.complaintService.assignedComplaints(assignedEngineer);

if(complaints.isEmpty()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}else {

return ResponseEntity.ok(complaints);

}

}

@PutMapping("/update-status/{id}")

public ResponseEntity<?> updateComplaintStatus(@PathVariable("id") int id, @RequestBody Complaint complaint){

this.complaintService.updateStatus(id, complaint);

return ResponseEntity.status(HttpStatus.CREATED).build();

}

}

10) package com.crs.controller;

import java.util.List;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.CrossOrigin;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.crs.entities.Feedback;

import com.crs.service.ComplaintService;

@RestController

@CrossOrigin("http://localhost:4200")

@RequestMapping("/feedback")

public class FeedbackController {

@Autowired

private ComplaintService complaintService;

@GetMapping("/get-feedback")

public ResponseEntity<?> getFeedback(){

List<Feedback> feedbacks = this.complaintService.findAllFeedback();

if(feedbacks.isEmpty()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}else {

return ResponseEntity.ok(feedbacks);

}

}

11) package com.crs.controller;

import java.util.List;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.CrossOrigin;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PutMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.crs.entities.Complaint;

import com.crs.entities.User;

import com.crs.service.ComplaintService;

import com.crs.service.UserService;

@RestController

@CrossOrigin("http://localhost:4200")

@RequestMapping("/manager")

public class ManagerController {

@Autowired

private ComplaintService complaintService;

@Autowired

private UserService userService;

@GetMapping("/get-complaints")

public ResponseEntity<?> getAllComplaints(){

List<Complaint> complaints = this.complaintService.findAllComplaint();

if(complaints.isEmpty()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}else {

return ResponseEntity.ok(complaints);

}

}

@GetMapping("/complaints/{isAssigned}")

public ResponseEntity<?> getAllAssignedComplaints(@PathVariable("isAssigned") boolean isAssigned){

List<Complaint> complaints = this.complaintService.findAssignedComplaint(isAssigned);

if(complaints.isEmpty()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}else {

return ResponseEntity.ok(complaints);

}

}

@GetMapping("/unassigned-complaint/{pinCode}")

public ResponseEntity<?> getUnassignedComplaints(@PathVariable("pinCode") int pinCode){

List<Complaint> complaints = this.complaintService.getComplaintByPinCode(pinCode, false);

if(complaints.isEmpty()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}else {

return ResponseEntity.ok(complaints);

}

}

@GetMapping("/assigned-complaint/{pinCode}")

public ResponseEntity<?> getAssignedComplaints(@PathVariable("pinCode") int pinCode){

List<Complaint> complaints = this.complaintService.getComplaintByPinCode(pinCode, true);

if(complaints.isEmpty()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}else {

return ResponseEntity.ok(complaints);

}

}

@PutMapping("/assign-engineer/{id}")

public ResponseEntity<?> complaintAssignEngineer(@PathVariable("id") int id, @RequestBody Complaint complaint){

this.complaintService.assignEngineer(id, complaint);

return ResponseEntity.status(HttpStatus.CREATED).build();

}

@GetMapping("/get-engineers")

public ResponseEntity<?> getAllEngineers(){

List<User> engineers = this.userService.getUserByRole("ENGINEER");

if(engineers.isEmpty()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}else {

return ResponseEntity.ok(engineers);

}

}

}

13)

package com.crs.controller;

import java.net.URI;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import javax.validation.Valid;

import javax.annotation.PostConstruct;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.CrossOrigin;

import org.springframework.web.bind.annotation.DeleteMapping;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.PutMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.servlet.support.ServletUriComponentsBuilder;

import com.crs.entities.Role;

import com.crs.entities.User;

import com.crs.entities.UserRole;

import com.crs.service.UserService;

@RestController

@CrossOrigin("http://localhost:4200")

@RequestMapping("/user")

public class UserController {

@Autowired

private UserService userService;

//create user

@PostMapping("/create-user")

public ResponseEntity<User> createNewUser(@Valid @RequestBody User user){

Set<UserRole> userRole = new HashSet<>();

Role role = new Role();

if(user.getRoleName().contentEquals("CUSTOMER")) {

role.setRoleId(102);

role.setRoleName(user.getRoleName());

}else if(user.getRoleName().contentEquals("MANAGER")) {

role.setRoleId(104);

role.setRoleName(user.getRoleName());

}else if(user.getRoleName().contentEquals("ENGINEER")) {

role.setRoleId(106);

role.setRoleName(user.getRoleName());

}

UserRole uR = new UserRole();

uR.setUser(user);

uR.setRole(role);

userRole.add(uR);

if(this.userService.getUserName(user.getUsername())!=null) {

System.out.println("Username already exist!");

return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).build();

}else {

User createUser = this.userService.createUser(user, userRole);

URI location = ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}").buildAndExpand(createUser.getUserId()).toUri();

return ResponseEntity.created(location).build();

}

}

//create admin

@PostConstruct

public void createAdmin() {

User admin = new User();

admin.setUsername("crs-admin@abc.com");

admin.setPassword("admin@crs");

// admin.setFirstName("Twarit");

// admin.setLastName("Soni");

// admin.setEmail("twarit.soni@gmail.com");

admin.setFirstName("Chhaya");

admin.setLastName("Askar");

admin.setEmail("chhaya.askar@gmail.com");

admin.setPinCode(110001);

admin.setPhone("+916265458854");

admin.setRoleName("ADMIN");

Role role = new Role();

role.setRoleId(101);

role.setRoleName(admin.getRoleName());

Set<UserRole> userRole = new HashSet<>();

UserRole uR = new UserRole();

uR.setUser(admin);

uR.setRole(role);

userRole.add(uR);

User userAdmin = this.userService.createUser(admin, userRole);

System.out.println("Admin Username: "+userAdmin.getUsername());

}

//get user by username

@GetMapping("/get-user/{username}")

public ResponseEntity<User> getUserByUsername(@PathVariable("username") String username){

User user = this.userService.getUserName(username);

if(user!=null) {

return ResponseEntity.ok(user);

}else {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

}

//delete user by userid

@DeleteMapping("/delete-user/{userId}")

public ResponseEntity<?> deleteUser(@PathVariable("userId") Integer userId){

this.userService.deleteUserById(userId);

return ResponseEntity.status(HttpStatus.OK).build();

}

//update user by username

@PutMapping("/update-user/{username}")

public ResponseEntity<User> updateUser(@PathVariable("username") String username,@RequestBody User user){

this.userService.updateUserByUsername(username, user);

return ResponseEntity.status(HttpStatus.CREATED).build();

}

//get user by role name

@GetMapping("/get-all/{roleName}")

public ResponseEntity<?> getAllUserByRole(@PathVariable("roleName") String roleName){

List<User> users = this.userService.getUserByRole(roleName);

if(users.isEmpty()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}else {

return ResponseEntity.ok(users);

}

}

}

14) package com.crs.controller;

import java.util.HashMap;

import java.util.Map;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.validation.FieldError;

import org.springframework.web.bind.MethodArgumentNotValidException;

import org.springframework.web.bind.annotation.ControllerAdvice;

import org.springframework.web.context.request.WebRequest;

import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;

@ControllerAdvice

public class ValidationHandler extends ResponseEntityExceptionHandler{

@Override

protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex,

HttpHeaders headers, HttpStatus status, WebRequest request) {

Map<String, String> errors = new HashMap<>();

ex.getBindingResult().getAllErrors().forEach((error) ->{

String fieldName = ((FieldError) error).getField();

String message = error.getDefaultMessage();

errors.put(fieldName, message);

});

return new ResponseEntity<Object>(errors, HttpStatus.BAD\_REQUEST);

}

}

FrontEnd:

