Config

AuthEntryPoint

```
package com.crs.config;

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;

@Component
public class AuthEntryPoint implements AuthenticationEntryPoint{

	@Override
	public void commence(HttpServletRequest request, HttpServletResponse response,
				AuthenticationException authException) throws IOException, ServletException {
		response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Unauthorized");
	}

}
```

CustomAuthorityDeserializer

```
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;
        }
```

}

JwtAuthFilter

```
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);


    }


}
```

JwtUtil

```
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));
    }

}
```

SecurityConfig

```
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);
        }


}
```

Controller

AuthenticationController

```
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(origins = "*")
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();
        }

    }


}
```

CustomerController

```
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(origins = "*")
@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);
	}


}
```

EngineerController

```
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(origins = "*")
@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();

	}




}
```

FeedbackController

```
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(origins = "*")
@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);
		}
	}
```

}

ManagerController

```
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(origins = "*")
@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);
		}
	}

}
```

UserController

```
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(origins = "*")
@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.getUserI
d()).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.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);
		}
	}

}
```

ValidationHandler

```
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);
	}
```

```
}
```

Entities

Authority

```
package com.crs.entities;

import org.springframework.security.core.GrantedAuthority;

public class Authority implements GrantedAuthority{

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
	private String authority;

	public Authority(String authority) {
		this.authority = authority;
	}

	@Override
	public String getAuthority() {
		return this.authority;
	}


}
```

Complaint

```
package com.crs.entities;
```

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;

@Entity
public class Complaint {
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private int id;

	@NotNull(message = "username field is required")
	private String username;

	@NotNull(message = "first name field is required")
	private String firstName;

	@NotNull(message = "last name field is required")
	private String lastName;

	@NotNull(message = "address field is required")
	private String address;

	@NotNull(message = "enter 6 digit pincode")
	@Min(100000)
	private int pinCode;
```

```
	@NotNull(message = "state field is required")
	private String state;

	@NotNull(message = "contact field is required")
	private String contact;

	@NotNull(message = "complaint field is required")
	private String complaint;

	private String status;

	private String assignedEngineer;

	private String remark;

	private String date;

	private boolean isActive;

	private boolean isAssigned;

	public Complaint() {

	}

	public Complaint(int id, String username, String firstName, String lastName, String address, int pinCode,
			String state, String contact, String complaint, String status, String assignedEngineer, String remark, String date, boolean isActive, boolean isAssigned) {
```

```
        super();
        this.id = id;
        this.username = username;
        this.firstName = firstName;
        this.lastName = lastName;
        this.address = address;
        this.pinCode = pinCode;
        this.state = state;
        this.contact = contact;
        this.complaint = complaint;
        this.status = status;
        this.assignedEngineer = assignedEngineer;
        this.remark = remark;
        this.date = date;
        this.isActive = isActive;
        this.isAssigned = isAssigned;
}

public int getId() {
        return id;
}

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

public String getUsername() {
        return username;
}
```

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

public String getFirstName() {
	return firstName;
}

public void setFirstName(String firstName) {
	this.firstName = firstName;
}

public String getLastName() {
	return lastName;
}

public void setLastName(String lastName) {
	this.lastName = lastName;
}

public String getAddress() {
	return address;
}

public void setAddress(String address) {
	this.address = address;
}
```

```
public int getPinCode() {
	return pinCode;
}

public void setPinCode(int pinCode) {
	this.pinCode = pinCode;
}

public String getState() {
	return state;
}

public void setState(String state) {
	this.state = state;
}

public String getContact() {
	return contact;
}

public void setContact(String contact) {
	this.contact = contact;
}

public String getComplaint() {
	return complaint;
}

public void setComplaint(String complaint) {
```

```
        this.complaint = complaint;
}

public String getStatus() {
        return status;
}

public void setStatus(String status) {
        this.status = status;
}

public String getAssignedEngineer() {
        return assignedEngineer;
}

public void setAssignedEngineer(String assignedEngineer) {
        this.assignedEngineer = assignedEngineer;
}

public String getRemark() {
        return remark;
}

public void setRemark(String remark) {
        this.remark = remark;
}

public String getDate() {
        return date;
```

```
	}

	public void setDate(String date) {
		this.date = date;
	}

	public boolean isActive() {
		return isActive;
	}

	public void setActive(boolean isActive) {
		this.isActive = isActive;
	}

	public boolean isAssigned() {
		return isAssigned;
	}

	public void setAssigned(boolean isAssigned) {
		this.isAssigned = isAssigned;
	}

}
```

Feedback

```
package com.crs.entities;

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

```
import javax.persistence.Id;

@Entity
public class Feedback {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private int fid;

	private int cid;

	private String username;

	private String complaint;

	private String feedback;

	public Feedback() {

	}

	public Feedback(int fid, int cid, String username, String complaint, String feedback) {
		super();
		this.fid = fid;
		this.cid = cid;
		this.username = username;
		this.complaint = complaint;
		this.feedback = feedback;
	}
```

```
public int getFid() {
	return fid;
}

public void setFid(int fid) {
	this.fid = fid;
}

public int getCid() {
	return cid;
}

public void setCid(int cid) {
	this.cid = cid;
}

public String getUsername() {
	return username;
}

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

public String getComplaint() {
	return complaint;
}
```

```
	public void setComplaint(String complaint) {
		this.complaint = complaint;
	}

	public String getFeedback() {
		return feedback;
	}

	public void setFeedback(String feedback) {
		this.feedback = feedback;
	}


}
```

JwtRequest

```
package com.crs.entities;

public class JwtRequest {

	String username;
	String password;

	public JwtRequest() {

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

```
	}
	public String getUsername() {
		return username;
	}
	public void setUsername(String username) {
		this.username = username;
	}
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}


}
```

JwtResponse

```
package com.crs.entities;

public class JwtResponse {

	String token;

	public JwtResponse() {

	}

	public JwtResponse(String token) {
		super();
```

```
		this.token = token;
	}

	public String getToken() {
		return token;
	}

	public void setToken(String token) {
		this.token = token;
	}



}
```

Role

```
package com.crs.entities;

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

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;

@Entity
```

```
@Table(name="roles")
public class Role {
	@Id
	private int roleId;

	@NotNull(message = "role name field is required")
	private String roleName;

	@OneToMany(cascade = CascadeType.ALL,fetch = FetchType.LAZY,mappedBy = "role")
	private Set<UserRole> userRoles = new HashSet<>();

	public Role() {

	}

	public Role(int roleId, String roleName, Set<UserRole> userRoles) {
		super();
		this.roleId = roleId;
		this.roleName = roleName;
		this.userRoles = userRoles;
	}

	public int getRoleId() {
		return roleId;
	}

	public void setRoleId(int roleId) {
		this.roleId = roleId;
	}
```

```
	public String getRoleName() {
		return roleName;
	}

	public void setRoleName(String roleName) {
		this.roleName = roleName;
	}

	public Set<UserRole> getUserRoles() {
		return userRoles;
	}

	public void setUserRoles(Set<UserRole> userRoles) {
		this.userRoles = userRoles;
	}



}
```

User

```
package com.crs.entities;

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

import javax.persistence.CascadeType;
import javax.persistence.Entity;
```

```
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

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

import com.crs.config.CustomAuthorityDeserializer;
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.databind.annotation.JsonDeserialize;

@Entity
@Table(name="users")
public class User implements UserDetails{
	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private int userId;
```

```
@NotNull(message = "username field is required")
private String username;

@NotNull(message = "password field is required")
@Size(min=8, message = "enter minimum six character password")
private String password;

@NotNull(message = "first name field is required")
private String firstName;

@NotNull(message = "last name field is required")
private String lastName;

@NotNull(message = "enter 6 digit pincode")
@Min(100000)
private int pinCode;

@NotNull(message = "email field is required")
private String email;

@NotNull(message = "phone field is required")
private String phone;

@NotNull(message = "role name field is required")
private String roleName;

private boolean enabled = true;

@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER, mappedBy = "user")
```

```
	@JsonIgnore
	private Set<UserRole> userRoles = new HashSet<>();

	public User() {

	}

	public User(int userId, String username, String password, String firstName, String lastName, int pinCode,
			String email, String phone, String roleName, boolean enabled, Set<UserRole> userRoles) {
		super();
		this.userId = userId;
		this.username = username;
		this.password = password;
		this.firstName = firstName;
		this.lastName = lastName;
		this.pinCode = pinCode;
		this.email = email;
		this.phone = phone;
		this.roleName = roleName;
		this.enabled = enabled;
		this.userRoles = userRoles;
	}

	public int getUserId() {
		return userId;
	}
```

```
public void setUserId(int userId) {
	this.userId = userId;
}

public String getUsername() {
	return username;
}

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

public String getPassword() {
	return password;
}

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

public String getFirstName() {
	return firstName;
}

public void setFirstName(String firstName) {
	this.firstName = firstName;
}

public String getLastName() {
```

```
        return lastName;
}

public void setLastName(String lastName) {
        this.lastName = lastName;
}

public int getPinCode() {
        return pinCode;
}

public void setPinCode(int pinCode) {
        this.pinCode = pinCode;
}

public String getEmail() {
        return email;
}

public void setEmail(String email) {
        this.email = email;
}

public String getPhone() {
        return phone;
}

public void setPhone(String phone) {
        this.phone = phone;
```

```
}

public boolean isEnabled() {
	return enabled;
}

public void setEnabled(boolean enabled) {
	this.enabled = enabled;
}

public Set<UserRole> getUserRoles() {
	return userRoles;
}

public void setUserRoles(Set<UserRole> userRoles) {
	this.userRoles = userRoles;
}

public String getRoleName() {
	return roleName;
}

public void setRoleName(String roleName) {
	this.roleName = roleName;
}
@JsonDeserialize(using = CustomAuthorityDeserializer.class)
@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
	Set<Authority> set = new HashSet<>();
```

```
            this.userRoles.forEach(userRole -> {
                set.add(new Authority(userRole.getRole().getRoleName()));
            });
            return set;
        }

        @Override
        public boolean isAccountNonExpired() {

            return true;
        }

        @Override
        public boolean isAccountNonLocked() {

            return true;
        }

        @Override
        public boolean isCredentialsNonExpired() {

            return true;
        }



}
```

UserRole

```
package com.crs.entities;
```

```
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;

@Entity
public class UserRole {
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private int userRoleId;

	@ManyToOne(fetch = FetchType.EAGER)
	private User user;

	@ManyToOne
	private Role role;

	public UserRole() {

	}

	public UserRole(int userRoleId, User user, Role role) {
		super();
		this.userRoleId = userRoleId;
		this.user = user;
		this.role = role;
```

```
	}

	public int getUserRoleId() {
		return userRoleId;
	}

	public void setUserRoleId(int userRoleId) {
		this.userRoleId = userRoleId;
	}

	public User getUser() {
		return user;
	}

	public void setUser(User user) {
		this.user = user;
	}

	public Role getRole() {
		return role;
	}

	public void setRole(Role role) {
		this.role = role;
	}

}
```

Repo

ComplaintRepo

```
package com.crs.repo;

import java.util.List;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import com.crs.entities.Complaint;

@Repository
public interface ComplaintRepo extends CrudRepository<Complaint, Integer>{
	public List<Complaint> findByUsername(String username);
	public List<Complaint> findByAssignedEngineer(String assignedEngineer);
	public Complaint findByComplaintAndUsernameAndIsActive(String complaint, String username,
boolean isActive);
	public List<Complaint> findByIsAssigned(boolean isAssigned);
	public List<Complaint> findByPinCodeAndIsAssigned(int pinCode, boolean isAssigned);
}
```

FeedbackRepo

```
package com.crs.repo;

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

import com.crs.entities.Feedback;

@Repository
```

```
public interface FeedbackRepo extends JpaRepository<Feedback, Integer>{
	public Feedback findByCid(int cid);
}
```

RoleRepo

```
package com.crs.repo;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.crs.entities.Role;

@Repository
public interface RoleRepo extends JpaRepository<Role, Integer>{

}
```

UserRepo

```
package com.crs.repo;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.crs.entities.User;


@Repository
public interface UserRepo extends JpaRepository<User, Integer>{
	public User findByUsername(String username);
	public List<User> findByRoleName(String roleName);
```

}

ComplaintService

```
package com.crs.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.util.Streamable;
import org.springframework.stereotype.Service;

import com.crs.entities.Complaint;
import com.crs.entities.Feedback;
import com.crs.repo.ComplaintRepo;
import com.crs.repo.FeedbackRepo;

@Service
public class ComplaintService {

	@Autowired
	private ComplaintRepo complaintRepo;

	@Autowired
	private FeedbackRepo feedbackRepo;

	public ComplaintService(ComplaintRepo complaintRepo) {
		this.complaintRepo = complaintRepo;
	}

	//create a new complaint
```

```
	public Complaint createComplaint(Complaint complaint) throws Exception {
		Complaint ticket = this.complaintRepo.findByComplaintAndUsernameAndIsActive(complaint.getComplaint(), complaint.getUsername(), complaint.isActive());
		if(ticket!=null) {
			throw new Exception("Complaint is already registered!");
		}else {
			ticket = this.complaintRepo.save(complaint);
		}
		return ticket;
	}

	//get complaint by username
	public List<Complaint> findComplaintByUsername(String username){
		List<Complaint> complaints = this.complaintRepo.findByUsername(username);
		return complaints;
	}

	//get all complaints
	public List<Complaint> findAllComplaint(){
		Iterable<Complaint> complaints = this.complaintRepo.findAll();
		List<Complaint> tickets = Streamable.of(complaints).toList();
		return tickets;
	}

	//get all isAssigned complaints
	public List<Complaint> findAssignedComplaint(boolean isAssigned){
		List<Complaint> complaints = this.complaintRepo.findByIsAssigned(isAssigned);
		return complaints;
```

```
	}

	//find complaint by id
	public Complaint getComplaint(int id) {
		Complaint complaint = this.complaintRepo.findById(id).get();
		return complaint;
	}

	//find assigned complaint by PinCode
	public List<Complaint> getComplaintByPinCode(int pinCode, boolean isAssigned){
		List<Complaint> complaints =
this.complaintRepo.findByPinCodeAndIsAssigned(pinCode, isAssigned);
		return complaints;
	}

	//assign engineer
	public Complaint assignEngineer(int id, Complaint complaint) {
		Complaint updateComplaint = this.complaintRepo.findById(id).get();
		updateComplaint.setAssigned(true);
		updateComplaint.setAssignedEngineer(complaint.getAssignedEngineer());
		updateComplaint.setRemark("Assigned to Engineer");
		Complaint assignedComplaint = this.complaintRepo.save(updateComplaint);
		return assignedComplaint;
	}

	//find complaint by assigned engineer
	public List<Complaint> assignedComplaints(String assignedEngineer){
		List<Complaint> complaints =
this.complaintRepo.findByAssignedEngineer(assignedEngineer);
```

```
        return complaints;
}

//update status by engineer
public Complaint updateStatus(int id, Complaint complaint) {
        Complaint updateComplaint = this.complaintRepo.findById(id).get();
        if(complaint.getStatus().contentEquals("WIP")) {
                updateComplaint.setStatus(complaint.getStatus());
                updateComplaint.setRemark(complaint.getRemark());
                updateComplaint.setActive(true);
        }else {
                updateComplaint.setStatus(complaint.getStatus());
                updateComplaint.setRemark(complaint.getRemark());
                updateComplaint.setActive(false);
        }

        Complaint resolveComplaint = this.complaintRepo.save(updateComplaint);
        return resolveComplaint;
}

//save feedback
public Feedback saveFeedback(Feedback feedback) throws Exception {
        Feedback getFeedback = this.feedbackRepo.findByCid(feedback.getCid());
        if(getFeedback==null) {
                Feedback save = this.feedbackRepo.save(feedback);
                return save;
        }else {
                throw new Exception("Feedback already registered!");
        }
```

```
	}

	//get all feedback
	public List<Feedback> findAllFeedback(){
		List<Feedback> feedbacks = this.feedbackRepo.findAll();
		return feedbacks;
	}


}
```

UserDetailService

```
package com.crs.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.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import com.crs.entities.User;
import com.crs.repo.UserRepo;

@Service
public class UserDetailService implements UserDetailsService{

	@Autowired
	private UserRepo userRepo;

	@Override
```

```
	public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException
{
		User user = this.userRepo.findByUsername(username);
		if(user == null) {
			System.out.println("User not found!");
			throw new UsernameNotFoundException("User does not exist!");
		}
		return user;
	}

}
```

UserService

```
package com.crs.service;

import java.util.List;
import java.util.Set;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;
import com.crs.entities.User;
import com.crs.entities.UserRole;
import com.crs.repo.RoleRepo;
import com.crs.repo.UserRepo;

@Service
public class UserService {
	@Autowired
	private UserRepo userRepo;
```

```
@Autowired
private RoleRepo roleRepo;

@Autowired
private BCryptPasswordEncoder passwordEncoder;

public UserService(UserRepo userRepo, RoleRepo roleRepo) {
	super();
	this.userRepo = userRepo;
	this.roleRepo = roleRepo;
}

// creating new user
public User createUser(User user, Set<UserRole> userRole){
	User local = this.userRepo.findByUsername(user.getUsername());
	try {
		if(local!=null) {
			throw new Exception("Username already exists!");
		}
		else {
			//user creation

			//saving role from userRole
			for(UserRole ur : userRole) {
				roleRepo.save(ur.getRole());
			}
			//assign userRole in user
			user.getUserRoles().addAll(userRole);
```

```
                        //encode password
                        user.setPassword(this.passwordEncoder.encode(user.getPassword()));

                        local = this.userRepo.save(user);
                }
        }catch(Exception e) {
                System.out.println(e);
        }
        return local;

}
//find user by username
public User getUserName(String username) {
        User findUser = this.userRepo.findByUsername(username);
        return findUser;
}

//find user by role
public List<User> getUserByRole(String roleName){
        return this.userRepo.findByRoleName(roleName);
}

//delete user by userid
public void deleteUserById(Integer userId) {
        this.userRepo.deleteById(userId);
}

//update user by username
public User updateUserByUsername(String username,User user) {
```

```
            User u = this.userRepo.findByUsername(username);
            u.setFirstName(user.getFirstName());
            u.setLastName(user.getLastName());
            u.setPhone(user.getPhone());
            u.setEmail(user.getEmail());
            u.setPinCode(user.getPinCode());
            User updatedUser = this.userRepo.save(u);
            return updatedUser;
        }

}
```