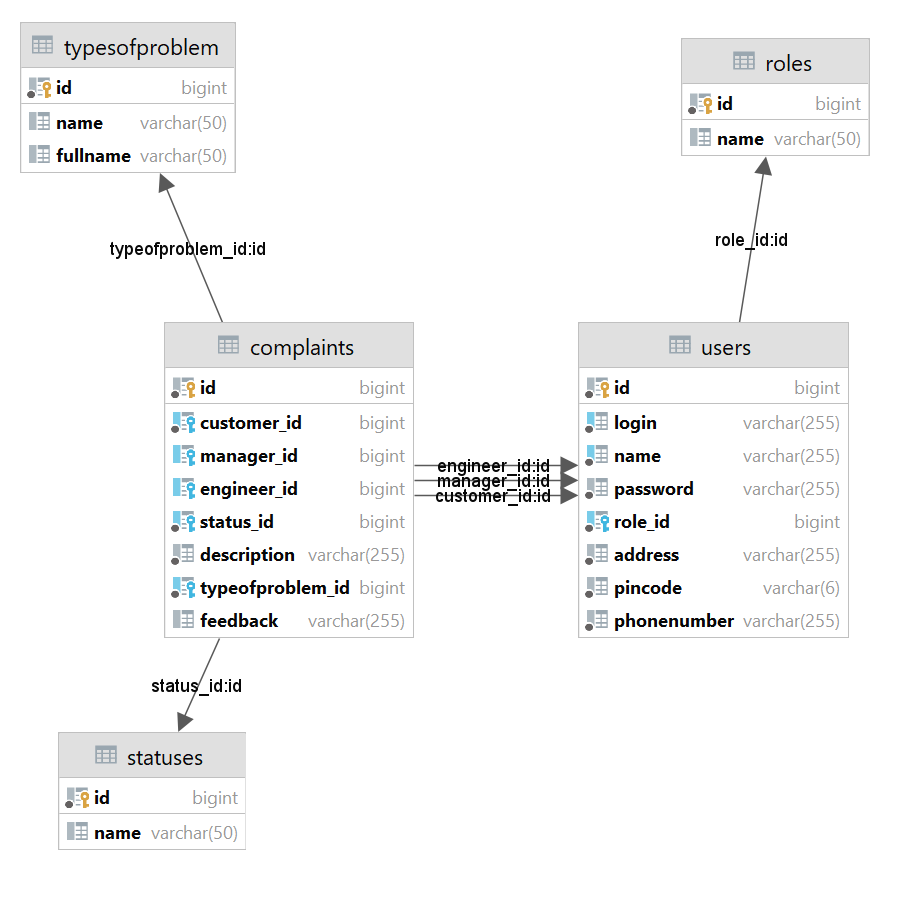
**Back end**

SQL Server side.

Database diagram:



Script for creating and populating database:

use complaints\_management;

CREATE TABLE `roles` (

`id` BIGINT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(50) NULL,

PRIMARY KEY (`id`));

INSERT INTO `roles` VALUES (1,'ADMINISTRATOR'),(2,'CUSTOMER'),(3,'MANAGER'),(4,'ENGINEER');

CREATE TABLE `users` (

`id` BIGINT NOT NULL AUTO\_INCREMENT,

`login` VARCHAR(255) NOT NULL UNIQUE,

`name` VARCHAR(255) NOT NULL UNIQUE,

`password` VARCHAR(255) NOT NULL,

`role\_id` BIGINT NOT NULL,

`address` VARCHAR(255) NOT NULL,

`pincode` VARCHAR(6) NOT NULL,

`phonenumber` VARCHAR(255) NOT NULL,

CONSTRAINT FK\_RoleUser FOREIGN KEY (role\_id) REFERENCES roles(id),

PRIMARY KEY (`id`));

INSERT INTO `users` VALUES (1,'Admin','Admin the greatest','$2a$12$rkHr/WWvtedt1q1rZ61jQe8/1WinsglTEmeZUziQcn8KLz6xwEx6a',1,'New York, 15yh street', '101121','+7848 4848 488'),

(2,'Customer1','User first','$2a$12$rkHr/WWvtedt1q1rZ61jQe8/1WinsglTEmeZUziQcn8KLz6xwEx6a',2,'Los Angeles, Left coast 14', '101100','+333 333 333'),

(3,'Customer2','User second','$2a$12$rkHr/WWvtedt1q1rZ61jQe8/1WinsglTEmeZUziQcn8KLz6xwEx6a',2,'Los Angeles, Down town 3', '101100','+5465454 444'),

(4,'Manager','Manager super','$2a$12$rkHr/WWvtedt1q1rZ61jQe8/1WinsglTEmeZUziQcn8KLz6xwEx6a',3,'New York, Peace avenue 15', '101121','+7848 4848 488'),

(5,'Engineer1','Engineer first','$2a$12$rkHr/WWvtedt1q1rZ61jQe8/1WinsglTEmeZUziQcn8KLz6xwEx6a',4,'New York, freedom square 41', '101100','+999 777 123'),

(6,'Engineer2','Engineer second','$2a$12$rkHr/WWvtedt1q1rZ61jQe8/1WinsglTEmeZUziQcn8KLz6xwEx6a',4,'New York, freedom square 5', '101101','+858 222 321');

CREATE TABLE `statuses` (

`id` BIGINT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(50) NULL,

PRIMARY KEY (`id`));

INSERT INTO `statuses` VALUES (1,'Raised'),(2,'Assigned'),(3,'WIP'),(4,'Resolved'),(5,'Escalated');

CREATE TABLE `typesofproblem` (

`id` BIGINT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(50) NULL,

`fullname` VARCHAR(50) NULL,

PRIMARY KEY (`id`));

INSERT INTO `typesofproblem` VALUES (1,'NotMakeCanRecieve','cannot make a call, but receive a call'),(2,'CanMakeNotRecieve','can make calls, but cannot receive calls'),(3,'NotMakeNotRecieve','neither make nor receive calls');

CREATE TABLE `complaints` (

`id` BIGINT NOT NULL AUTO\_INCREMENT,

`customer\_id` BIGINT NOT NULL,

`manager\_id` BIGINT,

`engineer\_id` BIGINT,

`status\_id` BIGINT NOT NULL,

`description` VARCHAR(255) NOT NULL,

`typeofproblem\_id` BIGINT NOT NULL,

`feedback` VARCHAR(255),

CONSTRAINT FK\_UserComplaintCustomer FOREIGN KEY (customer\_id) REFERENCES users(id),

CONSTRAINT FK\_UserComplaintManager FOREIGN KEY (manager\_id) REFERENCES users(id),

CONSTRAINT FK\_UserComplaintEngineer FOREIGN KEY (engineer\_id) REFERENCES users(id),

CONSTRAINT FK\_UserComplaintStatus FOREIGN KEY (status\_id) REFERENCES statuses(id),

CONSTRAINT FK\_UserComplaintProblem FOREIGN KEY (typeofproblem\_id) REFERENCES typesofproblem(id),

PRIMARY KEY (`id`));

INSERT INTO `complaints` VALUES

(1,2,null,null,1,'Something wrong',2,''),

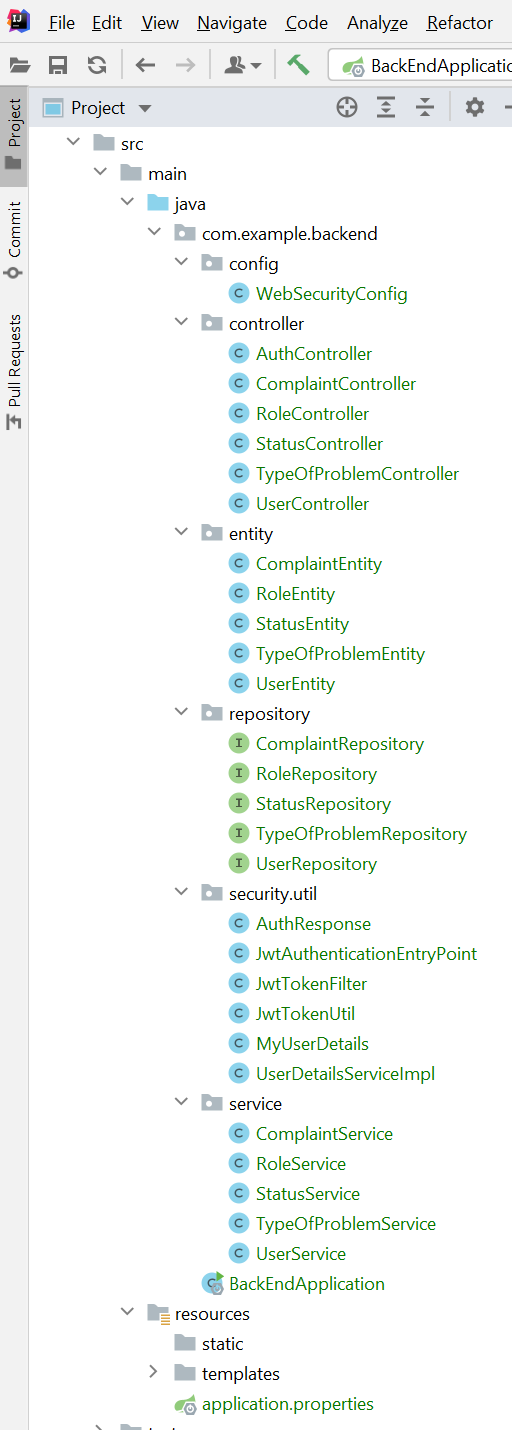
(2,3,4,5,3,'Looks like my phone is dead',3,''),

(3,3,4,5,4,'I cant use it',3,''),

(4,2,4,5,3,'Something realy strange. I hear noise',3,'Aga'),

(5,2,null,null,1,'I can call, but noone can call me',2,'');

Project structure:



Pom.xml:

*<?*xml version="1.0" encoding="UTF-8"*?>*<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.7.1</version>  
 <relativePath/> *<!-- lookup parent from repository -->* </parent>  
 <groupId>com.example</groupId>  
 <artifactId>BackEnd</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>BackEnd</name>  
 <description>Demo project for Spring Boot</description>  
  
 <properties>  
 <java.version>1.8</java.version>  
 </properties>  
  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <optional>true</optional>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.hibernate</groupId>  
 <artifactId>hibernate-ehcache</artifactId>  
 <version>5.6.8.Final</version>  
 </dependency>  
  
 <dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt</artifactId>  
 <version>0.9.1</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-thymeleaf</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.thymeleaf.extras</groupId>  
 <artifactId>thymeleaf-extras-springsecurity5</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>javax.validation</groupId>  
 <artifactId>validation-api</artifactId>  
 <version>2.0.1.Final</version>  
 </dependency>  
  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 <configuration>  
 <excludes>  
 <exclude>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 </exclude>  
 </excludes>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

application.properties:

spring.jpa.hibernate.ddl-auto=update  
spring.datasource.url=jdbc:mysql://localhost:3306/complaints\_management?useSSL=false  
spring.datasource.username=root  
spring.datasource.password=vestaeagle  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
server.port=8081  
  
*# Show all queries*spring.jpa.show-sql=true  
spring.jpa.properties.hibernate.format\_sql=true  
logging.level.org.hibernate.type=trace  
spring.jpa.properties.hibernate.connection.isolation=2  
  
  
*#L2 Cache  
#1. enable second level cache*spring.jpa.properties.hibernate.cache.use\_second\_level\_cache=true  
*#2. specify the caching framework - EhCache*spring.jpa.properties.hibernate.cache.region.factory\_class=org.hibernate.cache.ehcache.EhCacheRegionFactory  
*#3. Only cache what I tell to cache.*spring.jpa.properties.javax.persistence.sharedCache.mode=ENABLE\_SELECTIVE  
  
  
*# Your desired user name*spring.security.user.name=web  
*# password*spring.security.user.password=sparrow  
  
jwttoken.secret=mysecretkey  
*# in milliseconds (5 mins)*jwttoken.expiration=300000

Application:

package com.example.backend;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class BackEndApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(BackEndApplication.class, args);  
 }  
  
}

Repositories.

UserRepository:

package com.example.backend.repository;  
  
  
import com.example.backend.entity.UserEntity;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.JpaSpecificationExecutor;  
import org.springframework.stereotype.Repository;  
  
  
  
@Repository  
public interface UserRepository extends JpaRepository<UserEntity,Long>, JpaSpecificationExecutor<UserEntity> {  
 UserEntity findUserEntityByLogin(String name);  
  
 }

RoleRepository:

package com.example.backend.repository;  
  
import com.example.backend.entity.RoleEntity;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface RoleRepository extends JpaRepository<RoleEntity,Long> {  
}

ComplaintRepository:

package com.example.backend.repository;  
  
import com.example.backend.entity.ComplaintEntity;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
  
public interface ComplaintRepository extends JpaRepository<ComplaintEntity,Long> {  
}

StatusRepository:

package com.example.backend.repository;  
  
import com.example.backend.entity.StatusEntity;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface StatusRepository extends JpaRepository<StatusEntity,Long> {  
}

TypeOfProblemRepository:

package com.example.backend.repository;  
  
import com.example.backend.entity.TypeOfProblemEntity;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface TypeOfProblemRepository extends JpaRepository<TypeOfProblemEntity,Long> {  
}

Security util classes.

AuthResponse

package com.example.backend.security.util;  
  
import lombok.Getter;  
import lombok.Setter;  
  
@Getter  
@Setter  
  
public class AuthResponse {  
 private String token;  
 private String role;  
 private Long id;  
 private String name;  
}

JwtAuthenticationEntryPoint

package com.example.backend.security.util;  
  
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;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
@Component  
public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint {  
 private static final Logger *logger* = LoggerFactory.*getLogger*(JwtAuthenticationEntryPoint.class);  
 public void commence(HttpServletRequest request, HttpServletResponse response,  
 AuthenticationException authException) throws IOException, ServletException {  
 *logger*.error("Unauthorized access error : " + authException.getMessage());  
 response.sendError(HttpServletResponse.*SC\_UNAUTHORIZED*, "Unauthorized Access");  
 }  
}

JwtTokenFilter

package com.example.backend.security.util;  
  
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.util.StringUtils;  
import org.springframework.web.filter.OncePerRequestFilter;  
  
  
public class JwtTokenFilter extends OncePerRequestFilter{  
 @Autowired  
 private JwtTokenUtil jwtTokenUtil;  
 @Autowired  
 private UserDetailsServiceImpl userDetailsService;  
 @Override  
 protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)  
 throws ServletException, IOException {  
 try {  
 String token = getTokenFromRequest(request);  
 *//System.out.println("Token-- " + token);* if (token != null && jwtTokenUtil.validateJwtToken(token)) {  
 String username = jwtTokenUtil.getUserNameFromJwtToken(token);  
 *//System.out.println("User Name--JwtTokenFilter-- " + username);* UserDetails userDetails = userDetailsService.loadUserByUsername(username);  
 *//System.out.println("Authorities--JwtTokenFilter-- " + userDetails.getAuthorities());* UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(  
 userDetails, null, userDetails.getAuthorities());  
 authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));  
  
 SecurityContextHolder.*getContext*().setAuthentication(authentication);  
 }  
 } catch (Exception e) {  
 *//logger.error("Cannot set user authentication: {}", e);* throw new RuntimeException("Cannot set user authentication" + e.getMessage());  
 }  
 filterChain.doFilter(request, response);  
 }  
  
 private String getTokenFromRequest(HttpServletRequest request) {  
 String token = request.getHeader("Authorization");  
  
 if (StringUtils.*hasText*(token) && token.startsWith("Bearer ")) {  
 *// remove "Bearer "* return token.substring(7, token.length());  
 }  
 return null;  
 }  
}

JwtTokenUtil

package com.example.backend.security.util;  
  
import java.util.Date;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.security.core.Authentication;  
import org.springframework.stereotype.Component;  
import io.jsonwebtoken.Claims;  
import io.jsonwebtoken.ExpiredJwtException;  
import io.jsonwebtoken.Jwts;  
import io.jsonwebtoken.MalformedJwtException;  
import io.jsonwebtoken.SignatureAlgorithm;  
import io.jsonwebtoken.SignatureException;  
import io.jsonwebtoken.UnsupportedJwtException;  
  
@Component  
public class JwtTokenUtil {  
 @Value("${jwttoken.secret}")  
 private String jwtTokenSecret;  
 @Value("${jwttoken.expiration}")  
 private long jwtTokenExpiration;  
  
 public String generateJwtToken(Authentication authentication) {  
 MyUserDetails userPrincipal = (MyUserDetails) authentication.getPrincipal();  
 return Jwts.*builder*()  
 .setSubject(userPrincipal.getUsername())  
 .setIssuedAt(new Date(System.*currentTimeMillis*()))  
 .setExpiration(new Date(System.*currentTimeMillis*() + jwtTokenExpiration))  
 .signWith(SignatureAlgorithm.*HS512*, jwtTokenSecret)  
 .compact();  
 }  
  
 public boolean validateJwtToken(String token) {  
 try {  
 Jwts.*parser*()  
 .setSigningKey(jwtTokenSecret)  
 .parseClaimsJws(token);  
 return true;  
 }catch(UnsupportedJwtException exp) {  
 System.*out*.println("claimsJws argument does not represent Claims JWS" + exp.getMessage());  
 }catch(MalformedJwtException exp) {  
 System.*out*.println("claimsJws string is not a valid JWS" + exp.getMessage());  
 }catch(SignatureException exp) {  
 System.*out*.println("claimsJws JWS signature validation failed" + exp.getMessage());  
 }catch(ExpiredJwtException exp) {  
 System.*out*.println("Claims has an expiration time before the method is invoked" + exp.getMessage());  
 }catch(IllegalArgumentException exp) {  
 System.*out*.println("claimsJws string is null or empty or only whitespace" + exp.getMessage());  
 }  
 return false;  
 }  
  
 public String getUserNameFromJwtToken(String token) {  
 Claims claims =Jwts.*parser*()  
 .setSigningKey(jwtTokenSecret)  
 .parseClaimsJws(token)  
 .getBody();  
 return claims.getSubject();  
  
 }  
}

MyUserDetails

package com.example.backend.security.util;  
  
import com.example.backend.entity.UserEntity;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
  
import java.util.ArrayList;  
import java.util.Collection;  
import java.util.List;  
  
public class MyUserDetails implements UserDetails {  
  
 private UserEntity user;  
  
 public MyUserDetails(UserEntity user) {  
 this.user = user;  
 }  
  
 @Override  
 public Collection<? extends GrantedAuthority> getAuthorities() {  
 List<SimpleGrantedAuthority> authorities = new ArrayList<>();  
 authorities.add(new SimpleGrantedAuthority(user.getRole().getName()));  
 return authorities;  
 }  
  
 @Override  
 public String getPassword() {  
 return user.getPassword();  
 }  
  
 @Override  
 public String getUsername() {  
 return user.getLogin();  
 }  
  
 @Override  
 public boolean isAccountNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isAccountNonLocked() {  
 return true;  
 }  
  
 @Override  
 public boolean isCredentialsNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isEnabled() {  
 return true;  
 }  
  
 public Long getUserId() {  
 return user.getId();  
 }  
  
 public String getName() {  
 return user.getName();  
 }  
  
}

UserDetailsServiceImpl

package com.example.backend.security.util;  
  
import com.example.backend.entity.UserEntity;  
import com.example.backend.repository.UserRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.stereotype.Service;  
  
@Service  
public class UserDetailsServiceImpl implements UserDetailsService {  
  
 @Autowired  
 private UserRepository userRepository;  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 UserEntity user = userRepository.findUserEntityByLogin(username);  
  
 if (user == null) {  
 throw new UsernameNotFoundException("Could not find user");  
 }  
 return new MyUserDetails(user);  
 }  
  
}

Entities.

StatusEntity:

import lombok.\*;  
  
import javax.persistence.\*;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@ToString  
  
@Cacheable  
@Entity(name = "statuses")  
public class StatusEntity {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Column(nullable = false)  
 private String name;  
}

ComplaintEntity:

package com.example.backend.entity;  
  
  
import lombok.\*;  
  
import javax.persistence.\*;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@ToString  
  
@Entity(name = "complaints")  
public class ComplaintEntity {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @OneToOne  
 @JoinColumn(name = "customer\_id")  
 private UserEntity customer;  
  
 @OneToOne  
 @JoinColumn(name = "manager\_id")  
 private UserEntity manager;  
  
 @OneToOne  
 @JoinColumn(name = "engineer\_id")  
 private UserEntity engineer;  
  
 @OneToOne  
 @JoinColumn(name = "status\_id")  
 private StatusEntity status;  
  
 @Column(nullable = false)  
 private String description;  
  
 @OneToOne  
 @JoinColumn(nullable = false,name = "typeofproblem\_id")  
 private TypeOfProblemEntity typeOfProblem;  
  
 @Column  
 private String feedback;  
  
 public ComplaintEntity(TypeOfProblemEntity typeOfProblem, String description,StatusEntity status, UserEntity customer) {  
 this.typeOfProblem = typeOfProblem;  
 this.description = description;  
 this.status=status;  
 this.customer=customer;  
 }  
  
}

TypeOfProblemEntity:

package com.example.backend.entity;  
  
import lombok.\*;  
  
import javax.persistence.\*;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@ToString  
  
@Cacheable  
@Entity(name = "typesofproblem")  
public class TypeOfProblemEntity {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Column(nullable = false)  
 private String name;  
  
 @Column(nullable = false)  
 private String fullname;  
}

RoleEntity:

package com.example.backend.entity;  
  
  
import lombok.\*;  
  
import javax.persistence.\*;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@ToString  
  
@Cacheable  
@Entity(name = "roles")  
public class RoleEntity {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Column(nullable = false)  
 private String name;  
  
}

UserEntity:

package com.example.backend.entity;  
  
import lombok.\*;  
  
import javax.persistence.\*;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@ToString  
  
@Entity(name = "users")  
public class UserEntity {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Column(nullable = false)  
 private String login;  
 @Column(nullable = false)  
 private String password;  
 @Column(nullable = false)  
 private String name;  
 @Column(nullable = false)  
 private String address;  
  
 @Column(nullable = false, name="phonenumber")  
 private String phoneNumber;  
  
 @Column(nullable = false, name = "pincode")  
 private String pinCode;  
  
 @OneToOne  
 @JoinColumn(nullable = false)  
 private RoleEntity role;  
  
 public UserEntity(String login, String password) {  
 this.login = login;  
 this.password = password;  
 }  
}

Services.

UserService:

package com.example.backend.service;  
  
import com.example.backend.entity.RoleEntity;  
import com.example.backend.entity.UserEntity;  
import com.example.backend.repository.UserRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class UserService {  
 @Autowired  
 UserRepository userRepository;  
 @Autowired  
 BCryptPasswordEncoder bCryptPasswordEncoder;  
  
  
  
 public boolean isUserExist(String login){  
 UserEntity userEntity=userRepository.findUserEntityByLogin(login);  
 if (userEntity==null)  
 return false;  
 else  
 return true;  
 }  
  
  
  
  
 public RoleEntity getUserRole(String login){  
  
 UserEntity userEntity=userRepository.findUserEntityByLogin(login);  
  
 if (userEntity==null)  
 return null;  
 else  
 return userEntity.getRole();  
  
 }  
  
 public List<UserEntity> getUsersList() {  
 return userRepository.findAll();  
 }  
  
 public void Save(UserEntity user) {  
 userRepository.save(user);  
 }  
}

PurchaseService:

package com.example.sportyshoes.service;  
  
  
import com.example.sportyshoes.entity.CategoryEntity;  
import com.example.sportyshoes.entity.PurchaseEntity;  
import com.example.sportyshoes.repository.PurchaseRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.data.domain.Sort;  
import org.springframework.stereotype.Service;  
  
import java.time.LocalDateTime;  
import java.util.List;  
  
@Service  
public class PurchaseService {  
  
 @Autowired  
 PurchaseRepository purchaseRepository;  
  
 public List<PurchaseEntity> getPurchaseReportByDateIntervalAndCategory(LocalDateTime from, LocalDateTime to, CategoryEntity category){  
 return purchaseRepository.getPurchaseReportByDateIntervalAndCategory(from, to, category);  
 }  
  
  
}

RoleService:

package com.example.backend.service;  
  
import com.example.backend.entity.RoleEntity;  
import com.example.backend.repository.RoleRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.security.Principal;  
import java.util.List;  
  
@Service  
public class RoleService {  
 @Autowired  
 RoleRepository roleRepository;  
  
 public List<RoleEntity> getRolesList() {  
 return roleRepository.findAll();  
 }  
  
}

ComplaintService:

package com.example.backend.service;  
  
import com.example.backend.entity.ComplaintEntity;  
import com.example.backend.repository.ComplaintRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class ComplaintService {  
  
 @Autowired  
 ComplaintRepository complaintRepository;  
  
 public List<ComplaintEntity> getAll() {  
 return complaintRepository.findAll();  
 }  
  
 public void Save(ComplaintEntity complaintEntity) {  
 complaintRepository.save(complaintEntity);  
 }  
  
}

StatusService:

package com.example.backend.service;  
  
import com.example.backend.entity.StatusEntity;  
import com.example.backend.repository.StatusRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class StatusService {  
  
 @Autowired  
 StatusRepository statusRepository;  
  
 public List<StatusEntity> getAll() {  
 return statusRepository.findAll();  
 }  
  
}

TypeOfProblemService:

package com.example.backend.service;  
  
import com.example.backend.entity.TypeOfProblemEntity;  
import com.example.backend.repository.TypeOfProblemRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class TypeOfProblemService {  
  
 @Autowired  
 TypeOfProblemRepository typeOfProblemRepository;  
  
 public List<TypeOfProblemEntity> getAll() {  
 return typeOfProblemRepository.findAll();  
 }  
  
 public TypeOfProblemEntity getTypeOfProblemById(Long id){  
 return typeOfProblemRepository.findById(id).get();  
 }  
}

Controllers.

UserController:

package com.example.backend.controller;  
import com.example.backend.entity.UserEntity;  
import com.example.backend.service.UserService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.access.prepost.PreAuthorize;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@CrossOrigin  
@RequestMapping("/users")  
public class UserController {  
  
 @Autowired  
 UserService userService;  
 @Autowired  
 BCryptPasswordEncoder bCryptPasswordEncoder;  
  
 @ResponseBody  
 @GetMapping("/all")  
 @PreAuthorize("hasAuthority('ADMINISTRATOR')")  
 public List<UserEntity> getAllUsers() {  
 List<UserEntity >result=userService.getUsersList();  
 for (UserEntity userEntity : result) {  
 userEntity.setPassword("");  
 }  
 return result;  
 }  
  
 @PutMapping("/update")  
 @PreAuthorize("hasAuthority('ADMINISTRATOR')")  
 @ResponseBody  
 public ResponseEntity<?> updateUser(@RequestBody UserEntity user) {  
  
 if (!user.getPassword().isEmpty()){  
 *//user change password* user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));  
 }  
 userService.Save(user);  
 return ResponseEntity.*ok*(user);  
 }  
  
 @PostMapping("/new")  
 @PreAuthorize("hasAuthority('ADMINISTRATOR')")  
 @ResponseBody  
 public ResponseEntity<?> newUser(@RequestBody UserEntity user) {  
  
 if (!userService.isUserExist(user.getLogin())){  
 user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));  
 userService.Save(user);  
 return ResponseEntity.*ok*(user);  
 }  
 else{  
 return ResponseEntity.*badRequest*().body("Username is already taken");  
 }  
  
 }  
  
}

AuthController:

package com.example.backend.controller;  
  
  
import java.util.stream.Collectors;  
import javax.validation.Valid;  
  
import com.example.backend.entity.UserEntity;  
import com.example.backend.repository.RoleRepository;  
import com.example.backend.repository.UserRepository;  
import com.example.backend.security.util.AuthResponse;  
import com.example.backend.security.util.JwtTokenUtil;  
import com.example.backend.security.util.MyUserDetails;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.web.bind.annotation.\*;  
  
  
@RestController  
@RequestMapping("/auth")  
@CrossOrigin  
public class AuthController {  
 @Autowired  
 UserRepository userRepository;  
 @Autowired  
 RoleRepository roleRepository;  
 @Autowired  
 PasswordEncoder encoder;  
 @Autowired  
 AuthenticationManager authenticationManager;  
 @Autowired  
 JwtTokenUtil jwtTokenUtil;  
  
 @PostMapping("/login")  
 public ResponseEntity<?> userLogin(@Valid @RequestBody UserEntity user) {  
 Authentication authentication = authenticationManager.authenticate(  
 new UsernamePasswordAuthenticationToken(user.getLogin(), user.getPassword()));  
  
 SecurityContextHolder.*getContext*().setAuthentication(authentication);  
 String token = jwtTokenUtil.generateJwtToken(authentication);  
 MyUserDetails userBean = (MyUserDetails) authentication.getPrincipal();  
  
 String role = userBean.getAuthorities().stream()  
 .map(auth -> auth.getAuthority())  
 .collect(Collectors.*toList*()).get(0);  
  
 AuthResponse authResponse = new AuthResponse();  
 authResponse.setToken(token);  
 authResponse.setRole(role);  
 authResponse.setId(userBean.getUserId());  
 authResponse.setName(userBean.getName());  
 return ResponseEntity.*ok*(authResponse);  
 }  
}

ComplaintController:

package com.example.backend.controller;  
  
import com.example.backend.entity.ComplaintEntity;  
import com.example.backend.service.ComplaintService;  
import com.example.backend.service.TypeOfProblemService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@CrossOrigin  
@RequestMapping("/complaints")  
public class ComplaintController {  
  
 @Autowired  
 ComplaintService complaintService;  
 @Autowired  
 TypeOfProblemService typeOfComplaintService;  
  
 @ResponseBody  
 @GetMapping("/all")  
 public List<ComplaintEntity> getAllComplaints() {  
 List<ComplaintEntity>result=complaintService.getAll();  
 return result;  
 }  
  
 @PutMapping("/update")  
 @ResponseBody  
 public ResponseEntity<?> updateComplaint(@RequestBody ComplaintEntity complaint) {  
 complaintService.Save(complaint);  
 return ResponseEntity.*ok*(complaint);  
 }  
  
 @PostMapping("/new")  
 @ResponseBody  
 public ResponseEntity<?> newComplaint(@RequestBody ComplaintEntity complaint) {  
 System.*out*.println(complaint);  
 complaintService.Save(complaint);  
 return ResponseEntity.*ok*(complaint);  
 }  
}

RoleController:

package com.example.backend.controller;  
import com.example.backend.entity.RoleEntity;  
import com.example.backend.service.RoleService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@CrossOrigin  
@RequestMapping("/roles")  
public class RoleController {  
  
 @Autowired  
 RoleService roleService;  
  
 @ResponseBody  
 @GetMapping("/all")  
 public List<RoleEntity> displayUsers() {  
 return roleService.getRolesList();  
 }  
  
}

StatusController:

package com.example.backend.controller;  
  
import com.example.backend.entity.StatusEntity;  
import com.example.backend.service.StatusService;  
import org.springframework.beans.factory.annotation.Autowired;  
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 java.util.List;  
  
@RestController  
@CrossOrigin  
@RequestMapping("/statuses")  
public class StatusController {  
  
 @Autowired  
 StatusService statusService;  
  
 @GetMapping("/all")  
 public List<StatusEntity> getAllStatuses() {  
  
 List<StatusEntity >result=statusService.getAll();  
  
 return result;  
 }  
  
}

TypeOfProblemController:

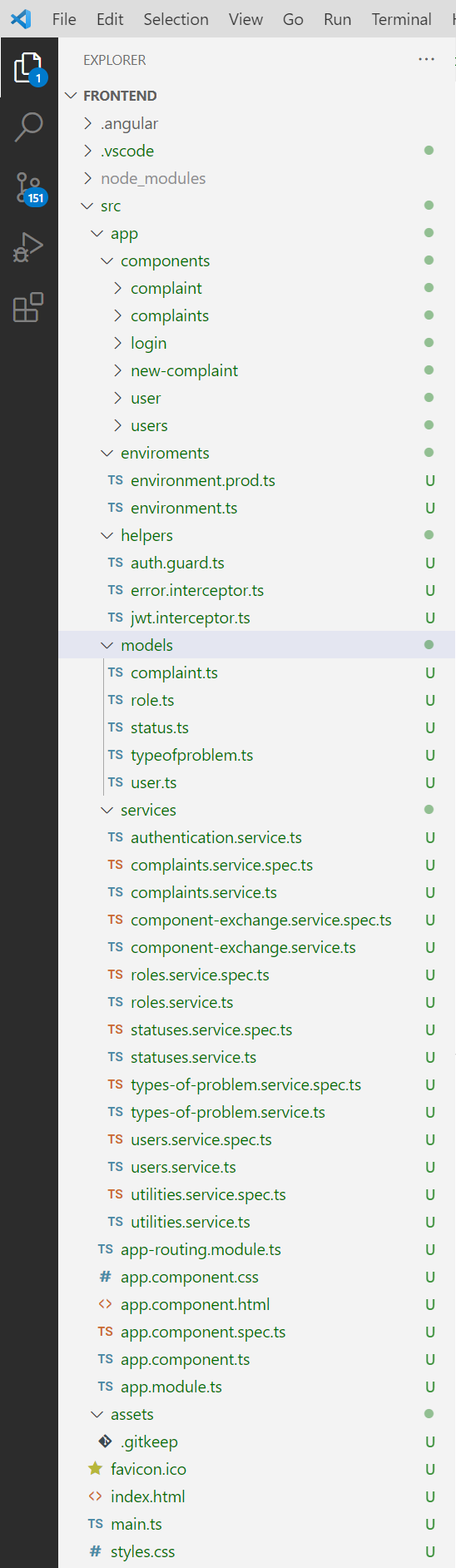
package com.example.backend.controller;  
  
  
import com.example.backend.entity.TypeOfProblemEntity;  
import com.example.backend.service.TypeOfProblemService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.CrossOrigin;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.util.List;  
  
@RestController  
@CrossOrigin  
@RequestMapping("/typesofproblem")  
public class TypeOfProblemController {  
  
 @Autowired  
 TypeOfProblemService typeOfProblemService;  
  
 @GetMapping("/all")  
 public List<TypeOfProblemEntity> getAllTypesOfProblem() {  
  
 List<TypeOfProblemEntity>result=typeOfProblemService.getAll();  
  
 return result;  
 }  
  
}

WebSecurityConfig:

package com.example.backend.config;  
  
import com.example.backend.security.util.JwtAuthenticationEntryPoint;  
import com.example.backend.security.util.JwtTokenFilter;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;  
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;  
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;  
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.core.userdetails.UserDetailsService;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;  
import org.springframework.web.servlet.config.annotation.CorsRegistry;  
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  
  
@Configuration  
@EnableWebSecurity  
@EnableGlobalMethodSecurity(prePostEnabled = true)  
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {  
  
  
 @Autowired  
 public UserDetailsService userDetailsService;  
 @Autowired  
 JwtAuthenticationEntryPoint authenticationEntryPoint;  
  
 @Bean  
 public BCryptPasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
  
 @Bean  
 public DaoAuthenticationProvider authenticationProvider() {  
 DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();  
 authProvider.setUserDetailsService(userDetailsService);  
 authProvider.setPasswordEncoder(passwordEncoder());  
 return authProvider;  
 }  
  
 @Override  
 protected void configure(AuthenticationManagerBuilder auth) throws Exception {  
 auth.authenticationProvider(authenticationProvider());  
 }  
  
 @Bean  
 @Override  
 public AuthenticationManager authenticationManagerBean() throws Exception {  
 return super.authenticationManagerBean();  
 }  
 @Bean  
 public JwtTokenFilter jwtTokenFilter(){  
 return new JwtTokenFilter();  
 }  
  
  
 @Override  
 protected void configure(HttpSecurity httpSecurity) throws Exception {  
 *// We don't need CSRF for this example* httpSecurity.cors().and().csrf().disable()  
 .exceptionHandling().authenticationEntryPoint(authenticationEntryPoint).and()  
 .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.*STATELESS*).and()  
 .authorizeRequests().antMatchers("/auth/\*\*").permitAll()  
 .anyRequest().authenticated();  
  
 httpSecurity.addFilterBefore(jwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);  
 }  
  
  
}

**Front end**

Project structure:



Components:

Complaint

*complaint.component.css*

.container div {

  float: left;

  margin-bottom: 15px;

}

.col-1{

    width: 20%;

  }

  .col-2{

    width: 80%;

  }

  .text-box1{

    width: 600px;

  }

  .text-box2{

    width: 600px;

  }

  .button{

    bottom: 20px;

    height: 55px;

    width: 250px;

  }

*complaint.component.html*

<div class="container">

    <br>

    <h1>

      Complaint management

    </h1>

    <br>

    <form #myForm="ngForm">

        <div class="col-1">

            <mat-form-field appearance="fill">

                <mat-label>Engineer</mat-label>

                <mat-select placeholder="Select engineer" [disabled]="!(isAdministrator||isManager)" required="{{isManager}}" [(ngModel)]="selectedEngineerId" name="engineer">

                    <mat-option \*ngFor="let user of engineerList" [value]="user.id">

                        {{user.login}}

                    </mat-option>

                </mat-select>

            </mat-form-field>

            <mat-form-field appearance="fill">

                <mat-label>Status</mat-label>

                <mat-select placeholder="Select status" [disabled]="!(isAdministrator||isEngineer)" [(ngModel)]="selectedStatusId" name="status">

                    <mat-option \*ngFor="let status of statuses" [value]="status.id">

                        {{status.name}}

                    </mat-option>

                </mat-select>

            </mat-form-field>

            <mat-form-field appearance="fill">

                <mat-label>Manager</mat-label>

                <mat-select placeholder="Select manager" [disabled]="!isAdministrator"  [(ngModel)]="selectedManagerId" name="manager">

                    <mat-option \*ngFor="let user of managerList" [value]="user.id">

                        {{user.login}}

                    </mat-option>

                </mat-select>

            </mat-form-field>

            <mat-form-field appearance="fill">

                <mat-label>Customer</mat-label>

                <mat-select placeholder="Select customer" [disabled]="!isAdministrator" [(ngModel)]="selectedCustomerId" name="customer">

                    <mat-option \*ngFor="let user of customerList" [value]="user.id">

                        {{user.login}}

                    </mat-option>

                </mat-select>

            </mat-form-field>

             <mat-form-field appearance="fill">

                <mat-label>Type of problem</mat-label>

                <mat-select placeholder="Select type of problem" [disabled]="!isAdministrator"  [(ngModel)]="selectedTypeOfProblemId" name="typeOfProblem">

                    <mat-option \*ngFor="let typeOfProblem of typesOfProblem" [value]="typeOfProblem.id">

                        {{typeOfProblem.fullname}}

                    </mat-option>

                </mat-select>

            </mat-form-field>

        </div>

        <div class="col-2">

            <mat-form-field appearance="fill" class="text-box1">

                <mat-label>Description</mat-label>

                <textarea [disabled]="!isAdministrator" minlength="15" matInput [(ngModel)]="complaint.description" name="description" #descr="ngModel"></textarea>

                <mat-error \*ngIf="!descr?.valid && (descr?.dirty || descr?.touched)">Description must be at least 15 symbols</mat-error>

            </mat-form-field>

            <br>

            <mat-form-field appearance="fill" class="text-box2">

                <mat-label>Feedback</mat-label>

                <textarea [disabled]="!(isAdministrator||isCustomer)" matInput [(ngModel)]="complaint.feedback" name="feedback"></textarea>

            </mat-form-field>

            <br>

            <br>

            <button (click)="onSaveChanges()"  [disabled]="!myForm.dirty" mat-raised-button color="primary" class="button">{{isManager? "Save changes. Set status to Assigned":"Save changes"}}</button>

        </div>

    </form>

</div>

*complaint.component.ts*

import { Component, OnInit } from '@angular/core';

import { Router } from '@angular/router';

import { first } from 'rxjs';

import { administratorRole, customerRole, engineerAccesseableStatusesId, engineerRole, managerRole, statusAssigned } from 'src/app/enviroments/environment';

import { Complaint } from 'src/app/models/complaint';

import { Status } from 'src/app/models/status';

import { TypeOfProblem } from 'src/app/models/typeofproblem';

import { User } from 'src/app/models/user';

import { AuthenticationService } from 'src/app/services/authentication.service';

import { CompliantsService } from 'src/app/services/complaints.service';

import { ComponentExchangeService } from 'src/app/services/component-exchange.service';

import { StatusesService } from 'src/app/services/statuses.service';

import { TypesOfProblemService } from 'src/app/services/types-of-problem.service';

import { UsersService } from 'src/app/services/users.service';

@Component({

  selector: 'app-complaint',

  templateUrl: './complaint.component.html',

  styleUrls: ['./complaint.component.css']

})

export class ComplaintComponent implements OnInit{

  isAdministrator=false;

  isCustomer=false;

  isManager=false;

  isEngineer=false;

  error:string="";

  loading:boolean=false;

  complaint!:Complaint;

  initialComplaint!:Complaint;

  users:User[]=[];

  engineerList:User[]=[];

  managerList:User[]=[];

  customerList:User[]=[];

  statuses:Status[]=[];

  typesOfProblem:TypeOfProblem[]=[];

  selectedEngineerId:number|undefined;

  selectedManagerId:number|undefined;

  selectedCustomerId:number|undefined;

  selectedStatusId:number|undefined;

  selectedTypeOfProblemId:number|undefined;

  ngOnInit(): void {

    this.initialComplaint=this.componentExchangeService.getCurrentComplaint();

    //this.complaint=Object.create(this.initialComplient);

    this.complaint=JSON.parse(JSON.stringify(this.initialComplaint));

    const role=String(this.authenticationService.userValue?.role);

    this.isAdministrator=(role==administratorRole);

    this.isCustomer=(role==customerRole);

    this.isManager=(role==managerRole);

    this.isEngineer=(role==engineerRole);

    let counter=1;

    this.loading = true;

    this.usersService.getAllUsers().pipe(first()).subscribe(users => {

        this.users = users;

        this.engineerList=JSON.parse(JSON.stringify(this.users));

        this.managerList=JSON.parse(JSON.stringify(this.users));

        this.customerList=JSON.parse(JSON.stringify(this.users));

        //&&engineer.pinCode==this.initialComplaint.customer?.pinCode

        this.engineerList=this.engineerList.filter(engineer=>

          engineer.role?.name===engineerRole||engineer.id===this.initialComplaint.engineer?.id

        )

        this.managerList=this.users.filter(manager=>

          (manager.role?.name==managerRole)||manager.id==this.initialComplaint.manager?.id

        )

        if (this.isManager){

          this.engineerList=this.users.filter(engineer=>

            (engineer.pinCode==this.initialComplaint.customer?.pinCode && engineer.role?.name===engineerRole)||engineer.id==this.initialComplaint.engineer?.id

        )

        }

        this.selectedEngineerId=this.initialComplaint.engineer?.id;

        this.selectedManagerId=this.initialComplaint.manager?.id;

        this.selectedCustomerId=this.initialComplaint.customer?.id;

        //for manager need filtering engineerList

        counter++;

        if (counter===3) this.loading=false;

   });

    this.statusesService.getStatuses().pipe(first()).subscribe(statuses => {

      this.statuses = statuses

      //engineer filter

      if (this.isEngineer){

        this.statuses = statuses.filter(status=>

          engineerAccesseableStatusesId.includes(status.id)

          || status.id==this.initialComplaint.status?.id);

      }

      else{

        this.statuses = statuses

      }

      this.selectedStatusId=this.initialComplaint.status?.id;

      counter++;

      if (counter===3) this.loading=false;

    });

    this.typesOfProblemService.getTypesOfProblem().pipe(first()).subscribe(types => {

      this.typesOfProblem = types;

      this.selectedTypeOfProblemId=this.initialComplaint.typeOfProblem?.id;

      counter++;

      if (counter===3) this.loading=false;

    });

  }

  onSaveChanges(){

    //Change status Assigned if this is manager

    if (this.isManager) {

      this.selectedStatusId=statusAssigned.id;

    }

    this.complaint.customer=this.customerList?.find(customer => customer.id === this.selectedCustomerId);

    this.complaint.manager=this.managerList?.find(manager => manager.id === this.selectedManagerId);

    this.complaint.engineer=this.engineerList?.find(engineer => engineer.id === this.selectedEngineerId);

    this.complaint.status=this.statuses?.find(status => status.id === this.selectedStatusId);

    this.complaint.typeOfProblem=this.typesOfProblem?.find(typeOfProblem => typeOfProblem.id === this.selectedTypeOfProblemId);

    this.loading=true;

    this.complaintsService.modifyComplaint(this.complaint).pipe(first()).subscribe(

      {

      next: () => {

          this.loading=false;

          this.router.navigateByUrl('/complaints');

      },

      error: error => {

          this.error = error;

          this.loading = false;

      }

    });

  }

  constructor(

    private componentExchangeService:ComponentExchangeService,

    private statusesService:StatusesService,

    private typesOfProblemService:TypesOfProblemService,

    private usersService:UsersService,

    private complaintsService:CompliantsService,

    private authenticationService:AuthenticationService,

    private router:Router

    ){

  }

}

Complaints

*complaints.component.css*

.container div {

    float: left;

    margin-bottom: 15px;

  }

  .button-togle{

    width: 150px;

  }

  .button-raise{

    left: 2px;

    width: 150px;

    height: 55px;

  }

  .button-group{

    top: 22px;

  }

  .col-1{

    width: 15%;

  }

  .col-2 {

    width: 85%;

  }

  table {

    width: 100%;

  }

  .fr{

    width: 170px;

    float: right;

  }

  tr.example-element-row:not(.example-expanded-row):hover {

    background: whitesmoke;

  }

  tr.example-element-row:not(.example-expanded-row):active {

    background: #efefef;

  }

  .example-element-row td {

    border-bottom-width: 0;

  }

*complaints.component.html*

  <div class="container">

    <br> <br>

    <div class="col-1">

    <button mat-raised-button color="primary" class="button-raise" (click)="raiseComplaint()">Raise a complaint</button>

     <section>

        <mat-button-toggle-group vertical="true" class="button-group" (change)="onStatusButtonClick()"  [(ngModel)]="currentStatus" aria-label="Current status">

          <mat-button-toggle class="button-togle" value="All">All</mat-button-toggle>

          <mat-button-toggle value="Raised">Raised</mat-button-toggle>

          <mat-button-toggle value="Assigned">Assigned</mat-button-toggle>

          <mat-button-toggle value="WIP">WIP</mat-button-toggle>

          <mat-button-toggle value="Resolved">Resolved</mat-button-toggle>

          <mat-button-toggle value="Escalated">Escalated</mat-button-toggle>

        </mat-button-toggle-group>

      </section>

    </div>

    <div class="col-2">

      <mat-form-field class="fr">

        <mat-label>Full-text search</mat-label>

        <input matInput (keyup)="applyFilter($event)" placeholder="Text for search" #input>

      </mat-form-field>

      <table mat-table [dataSource]="dataSource" class="mat-elevation-z8">

      <!--- Note that these columns can be defined in any order.

              The actual rendered columns are set as a property on the row definition" -->

        <!-- Customer Column -->

        <ng-container matColumnDef="customer">

          <th mat-header-cell \*matHeaderCellDef> Customer </th>

          <td mat-cell \*matCellDef="let element"> {{element.customer?.name}} </td>

        </ng-container>

        <!-- Pin code  Column -->

        <ng-container matColumnDef="pinCode">

          <th mat-header-cell \*matHeaderCellDef> Pin </th>

          <td mat-cell \*matCellDef="let element"> {{element.customer?.pinCode}} </td>

        </ng-container>

        <!-- Description Column -->

        <ng-container matColumnDef="description">

          <th mat-header-cell \*matHeaderCellDef> Description </th>

          <td mat-cell \*matCellDef="let element"> {{element.description}} </td>

        </ng-container>

        <!-- Type of problem Column -->

        <ng-container matColumnDef="typeOfProblem">

          <th mat-header-cell \*matHeaderCellDef> Type </th>

          <td mat-cell \*matCellDef="let element"> {{element.typeOfProblem.fullname}} </td>

        </ng-container>

        <!-- Status Column -->

        <ng-container matColumnDef="status">

          <th mat-header-cell \*matHeaderCellDef> Status </th>

          <td mat-cell \*matCellDef="let element"> {{element.status.name}} </td>

        </ng-container>

        <!-- Engineer Column -->

        <ng-container matColumnDef="engineer">

          <th mat-header-cell \*matHeaderCellDef> Engineer </th>

          <td mat-cell \*matCellDef="let element"> {{element.engineer?.name}} </td>

        </ng-container>

         <!-- Manager Column -->

         <ng-container matColumnDef="manager">

          <th mat-header-cell \*matHeaderCellDef> Manager </th>

          <td mat-cell \*matCellDef="let element"> {{element.manager?.name}} </td>

        </ng-container>

        <tr \*matRowDef="let row; columns: displayedColumns;" (click)="onRowClick(row)" mat-row class="example-element-row"></tr>

        <tr mat-header-row \*matHeaderRowDef="displayedColumns"></tr>

      </table>

      <mat-paginator [pageSizeOptions]="[15,30,50]" showFirstLastButtons aria-label="Select page of complaints"></mat-paginator>

      </div>

  </div>

*complaints.component.ts*

import { AfterViewInit, Component, OnInit, ViewChild } from '@angular/core';

import { Complaint } from 'src/app/models/complaint';

import { CompliantsService } from 'src/app/services/complaints.service';

import { MatPaginator } from '@angular/material/paginator';

import { MatTableDataSource } from '@angular/material/table';

import { Router } from '@angular/router';

import { ComponentExchangeService } from 'src/app/services/component-exchange.service';

import { first } from 'rxjs';

import { AuthenticationService } from 'src/app/services/authentication.service';

import { customerRole, managerRole } from 'src/app/enviroments/environment';

export interface Tile {

  color: string;

  cols: number;

  rows: number;

  text: string;

}

@Component({

  selector: 'app-complaints',

  templateUrl: './complaints.component.html',

  styleUrls: ['./complaints.component.css']

})

export class ComplaintsComponent implements OnInit, AfterViewInit{

  @ViewChild(MatPaginator) paginator!: MatPaginator ;

  complaints:Complaint[]=[];

  currentStatus: string="All";

  displayedColumns: string[] = ['customer', 'pinCode', 'description', 'typeOfProblem', 'status', 'engineer','manager'];

  dataSource=new MatTableDataSource<Complaint>();

  loading:boolean=false;

  filterComplaintsByStatus(status:string){

    this.dataSource.data=this.complaints.filter(str=>str.status?.name===status);

  }

  raiseComplaint(){

    this.router.navigateByUrl('new-complaint');

  }

  onStatusButtonClick(){

     switch(this.currentStatus) {

      case "All":{

        this.dataSource.data=this.complaints;

        break

      }

      case "Raised":{

        this.filterComplaintsByStatus("Raised");

        break

      }

      case "Assigned":{

        this.filterComplaintsByStatus("Assigned");

        break;

      }

      case "WIP":{

        this.filterComplaintsByStatus("WIP");

        break;

      }

      case "Resolved":{

        this.filterComplaintsByStatus("Resolved");

        break;

      }

      case "Escalated":{

        this.filterComplaintsByStatus("Escalated");

        break;

      }

      default:{

        break;

      }

    }

   }

  onRowClick(complaint: Complaint) {

    this.componentExchangeService.setCurrentComplaint(complaint);

    this.router.navigateByUrl('complaint');

  }

  constructor(private complaintsService:CompliantsService

    ,private router: Router

    ,private componentExchangeService:ComponentExchangeService

    ,private authenticationService:AuthenticationService){

  }

  ngAfterViewInit(): void {

    this.dataSource.paginator = this.paginator;

  }

  ngOnInit(): void {

    const role=String(this.authenticationService.userValue?.role);

    const needFilterByManagerId=(role==managerRole);

    const needFilterByCustomerId=(role==customerRole);

    this.loading = true;

        this.complaintsService.getAllCompliants().pipe(first()).subscribe(complaints => {

            if (needFilterByManagerId){

              this.complaints=complaints.filter(complaint=>complaint.manager?.id==this.authenticationService.userValue?.id)

            }

            else if (needFilterByCustomerId){

              this.complaints=complaints.filter(complaint=>complaint.customer?.id==this.authenticationService.userValue?.id)

            }

            else{

              this.complaints = complaints;

            }

            this.loading = false;

            this.dataSource.data=this.complaints;

        });

     //this.complaints=this.dataSource.data.slice();

  }

  applyFilter(event: Event) {

    const filterValue = (event.target as HTMLInputElement).value;

    this.dataSource.filter = filterValue.trim().toLowerCase();

    if (this.dataSource.paginator) {

      this.dataSource.paginator.firstPage();

    }

  }

}

Login

*login.component.html*

<div class="container">

<br>

<mat-card class="example-card">

    <mat-card-header>

      <mat-card-title>Login</mat-card-title>

    </mat-card-header>

    <mat-card-content>

      <form #myForm="ngForm" class="example-form">

        <table class="example-full-width" cellspacing="0">

          <tr>

            <td>

              <mat-form-field class="example-full-width">

              <input matInput placeholder="Username" [(ngModel)]="username" name="username" required>

              </mat-form-field>

            </td>

          </tr>

          <tr>

          <td><mat-form-field class="example-full-width">

            <input required="true" matInput placeholder="Password" [(ngModel)]="password"type="password" name="password" required>

        </mat-form-field></td>

        </tr></table>

        <mat-error \*ngIf="error">Invalid credential</mat-error>

    </form>

    </mat-card-content>

    <mat-card-actions>

        <button [disabled]="loading" mat-raised-button (click)="login()" [disabled]="!myForm.valid" color="primary">

            <span \*ngIf="loading" class="spinner-border spinner-border-sm me-1"></span>

            Login

        </button>

    </mat-card-actions>

 </mat-card>

</div>

*login.component.ts*

import { Component } from '@angular/core';

import { ActivatedRoute, Router } from '@angular/router';

import { first } from 'rxjs';

import { AuthenticationService } from 'src/app/services/authentication.service';

@Component({

  selector: 'app-login',

  templateUrl: './login.component.html',

  styleUrls: ['./login.component.css']

})

export class LoginComponent {

  username: string ='';

  password: string ='';

  showSpinner=false;

  loading:boolean=false;

  error:string='';

  constructor(

    private route: ActivatedRoute,

    private router: Router,

    private authenticationService: AuthenticationService)

    {

       // redirect to home if already logged in

       if (this.authenticationService.userValue)

       {

          this.router.navigate(['/']);

       }

    }

  ngOnInit() {

  }

  login() : void {

    this.loading = true;

    this.authenticationService.login(this.username, this.password)

        .pipe(first())

        .subscribe({

            next: () => {

                // get return url from query parameters or default to home page

                const returnUrl = this.route?.snapshot.queryParams['returnUrl'] || '/';

                this.router.navigateByUrl(returnUrl);

            },

            error: error => {

                this.error = error;

                this.loading = false;

            }

        });

}

}

New-complaint

*new-complaint.component.css*

.select{

    width: 500px;

}

.text-box1{

    width: 500px;

}

.button{

  width: 150px;

  height: 55px;

}

*new-complaint.component.html*

<div class="container">

    <br>

    <h1>

      New complaint

    </h1>

    <br>

    <form #myForm="ngForm">

        <mat-form-field appearance="fill" class="select">

            <mat-label>Type of problem</mat-label>

            <mat-select  required="true" placeholder="Select type of problem" #typeOfProblem="ngModel" [(ngModel)]="selectedTypeOfProblem" name="typeOfProblem">

                <mat-option \*ngFor="let typeOfProblem of typesOfProblem" [value]="typeOfProblem">

                    {{typeOfProblem.fullname}}

                </mat-option>

            </mat-select>

        </mat-form-field>

        <br>

        <mat-form-field appearance="fill" class="text-box1">

            <mat-label>Description</mat-label>

            <textarea  required="true" minlength="15" matInput [(ngModel)]="description" #descr="ngModel" name="description" ></textarea>

            <mat-error \*ngIf="!descr?.valid && (descr?.dirty || descr?.touched)">Description must be at least 15 symbols</mat-error>

        </mat-form-field>

        <br>

        <br>

        <button (click)="onClickRise()" [disabled]="!(myForm.valid)" mat-raised-button color="primary" class="button">Raise</button>

    </form>

</div>

*new-complaint.component.ts*

import { Component, OnInit } from '@angular/core';

import { Router } from '@angular/router';

import { first } from 'rxjs';

import { statusRaised } from 'src/app/enviroments/environment';

import { Complaint } from 'src/app/models/complaint';

import { TypeOfProblem } from 'src/app/models/typeofproblem';

import { User } from 'src/app/models/user';

import { AuthenticationService } from 'src/app/services/authentication.service';

import { CompliantsService } from 'src/app/services/complaints.service';

import { TypesOfProblemService } from 'src/app/services/types-of-problem.service';

@Component({

  selector: 'app-new-complaint',

  templateUrl: './new-complaint.component.html',

  styleUrls: ['./new-complaint.component.css']

})

export class NewComplaintComponent implements OnInit{

  typesOfProblem:TypeOfProblem[]=[];

  selectedTypeOfProblem:TypeOfProblem|undefined;

  loading:boolean=false;

  error:string="";

  description:string="";

constructor(private typesOfProblemService:TypesOfProblemService,

  private complaintService:CompliantsService,

  private authenticationService:AuthenticationService,

  private router: Router){

}

  ngOnInit(): void {

    this.loading=true;

    this.typesOfProblemService.getTypesOfProblem().pipe(first()).subscribe(types => {

      this.typesOfProblem = types;

      this.loading=false;

    });

  }

  onClickRise(){

    this.loading=true;

    const currentComplaint:Complaint=new Complaint(new User(this.authenticationService.userValue?.id),this.description,this.selectedTypeOfProblem, statusRaised);

    this.complaintService.newComplaint(currentComplaint).pipe(first()).subscribe(

      {

      next: () => {

          this.loading=false;

          this.router.navigateByUrl('/complaints');

      },

      error: error => {

         this.error = error;

         this.loading = false;

      }

  });

  }

}

User

*user.component.css*

.container div {

    float: left;

    margin-bottom: 15px;

  }

.col-1{

    width: 20%;

 }

.col-2{

    width: 20%;

}

.col-3{

    width: 20%;

}

.button{

    bottom: 20px;

    height: 55px;

    width: 150px;

}

*user.component.html*

<div class="container">

    <br>

    <h1>

      {{isNew? "New user":"User management"}}

    </h1>

    <br>

    <form #myForm="ngForm">

        <mat-form-field appearance="fill">

            <mat-label>User name</mat-label>

            <input required="true" minlength="10" matInput [(ngModel)]="user.name" name="name" #name="ngModel" >

            <mat-error \*ngIf="!name?.valid && (name?.dirty || name?.touched)">Must be at least 10 symbols</mat-error>

        </mat-form-field>

        <br>

        <br>

        <div class="col-1">

            <mat-form-field appearance="fill">

                <mat-label>Login</mat-label>

                <input required="true" minlength="4" matInput [(ngModel)]="user.login" name="login" #login="ngModel">

                <mat-error \*ngIf="!login?.valid && (login?.dirty || login?.touched)">Must be at least 4 symbols</mat-error>

            </mat-form-field>

            <mat-form-field appearance="fill">

                <mat-label>Password</mat-label>

                <input required={{isNew}}  minlength="4" type="password" matInput [(ngModel)]="user.password" name="password" #password="ngModel">

                <mat-error \*ngIf="!password?.valid && (password?.dirty || password?.touched)">Must be at least 4 symbols</mat-error>

            </mat-form-field>

            <br>

            <br>

            <button (click)="onSaveChanges()"  [disabled]="!myForm.valid||loading" mat-raised-button color="primary" class="button">

                <span \*ngIf="loading" class="spinner-border spinner-border-sm me-1"></span>

                {{isNew? "Save user":"Save changes"}}

            </button>

        </div>

        <div class="col-2">

            <mat-form-field appearance="fill">

                <mat-label>Role</mat-label>

                <mat-select placeholder="Select role" required="true" [(ngModel)]="selectedRoleId" name="role">

                    <mat-option \*ngFor="let role of roles" [value]="role.id">

                        {{role.name}}

                    </mat-option>

                </mat-select>

            </mat-form-field>

            <mat-form-field appearance="fill">

                <mat-label>Pin code</mat-label>

                <input required="true" minlength="6" maxlength="6" matInput [(ngModel)]="user.pinCode" name="pinCode" #pinCode="ngModel">

                <mat-error \*ngIf="!pinCode?.valid && (pinCode?.dirty || pinCode?.touched)">Must have 6 symbols</mat-error>

            </mat-form-field>

        </div>

        <div class="col-3">

             <mat-form-field appearance="fill">

                <mat-label>Address</mat-label>

                <input required="true" minlength="5" matInput [(ngModel)]="user.address" name="address" #address="ngModel">

                <mat-error \*ngIf="!address?.valid && (address?.dirty || address?.touched)">Must be at least 5 symbols</mat-error>

            </mat-form-field>

            <mat-form-field appearance="fill">

                <mat-label>Phone</mat-label>

                <input required="true" minlength="6" matInput [(ngModel)]="user.phoneNumber" name="phoneNumber" #phone="ngModel">

                <mat-error \*ngIf="!phone?.valid && (phone?.dirty || phone?.touched)">Must be at least 6 symbols</mat-error>

            </mat-form-field>

       </div>

    </form>

</div>

*user.component.ts*

import { Component, OnInit } from '@angular/core';

import { Router } from '@angular/router';

import { first } from 'rxjs';

import { Role } from 'src/app/models/role';

import { User } from 'src/app/models/user';

import { ComponentExchangeService } from 'src/app/services/component-exchange.service';

import { RolesService } from 'src/app/services/roles.service';

import { UsersService } from 'src/app/services/users.service';

@Component({

  selector: 'app-user',

  templateUrl: './user.component.html',

  styleUrls: ['./user.component.css']

})

export class UserComponent implements OnInit{

  isNew:boolean=false;

  user!:User;

  initialUser!:User;

  loading:boolean=false;

  error:string='';

  selectedRoleId:number|undefined;

  roles:Role[]=[];

  constructor(private componentExchangeService:ComponentExchangeService,

    private usersService:UsersService,

    private rolesService:RolesService,

    private router:Router

    ){

  }

  ngOnInit(): void {

    this.initialUser=this.componentExchangeService.getCurrentUser();

    this.selectedRoleId=this.initialUser.role?.id;

    this.isNew=this.initialUser.login===undefined;

    this.user=JSON.parse(JSON.stringify(this.initialUser))

    this.loading = true;

    this.rolesService.getRoles().pipe(first()).subscribe(roles =>

      {

        this.roles = roles;

        this.loading = false;

      });

  }

  onSaveChanges(){

    this.loading=true;

    this.user.role=this.roles?.find(role => role.id === this.selectedRoleId);

    if (this.isNew){

      this.usersService.newUser(this.user).pipe(first()).subscribe(

        {

        next: () => {

            this.loading=false;

            this.router.navigateByUrl('/users');

        },

        error: error => {

           this.error = error;

           this.loading = false;

        }

    });

    }

    else{

      this.usersService.modifyUser(this.user).pipe(first()).subscribe(

        {

        next: () => {

            this.loading=false;

            this.router.navigateByUrl('/users');

        },

        error: error => {

            this.error = error;

            this.loading = false;

        }

      });

    }

  }

}

Users

*users.component.css*

.container div {

    float: left;

    margin-bottom: 15px;

  }

  .button-raise{

    width: 150px;

    height: 55px;

  }

  .fr{

    float: right;

  }

  table {

    width: 100%;

  }

  tr.example-element-row:not(.example-expanded-row):hover {

    background: whitesmoke;

  }

  tr.example-element-row:not(.example-expanded-row):active {

    background: #efefef;

  }

  .example-element-row td {

    border-bottom-width: 0;

  }

*users.component.html*

<div class="container">

    <div \*ngIf="loading" class="spinner-border spinner-border-sm"></div>

    <br> <br>

    <button mat-raised-button color="primary" class="button-raise" (click)="newUser()">New user</button>

    <mat-form-field class="fr">

        <mat-label>Full-text search</mat-label>

        <input matInput (keyup)="applyFilter($event)" placeholder="Text for search" #input>

    </mat-form-field>

    <table mat-table [dataSource]="dataSource" class="mat-elevation-z8">

        <!--- Note that these columns can be defined in any order.

              The actual rendered columns are set as a property on the row definition" -->

        <!-- Name Column -->

        <ng-container matColumnDef="name">

          <th mat-header-cell \*matHeaderCellDef> Name </th>

          <td mat-cell \*matCellDef="let element"> {{element.name}} </td>

        </ng-container>

        <!-- Login code  Column -->

        <ng-container matColumnDef="login">

          <th mat-header-cell \*matHeaderCellDef> Login </th>

          <td mat-cell \*matCellDef="let element"> {{element.login}} </td>

        </ng-container>

        <!-- Role Column -->

        <ng-container matColumnDef="role">

          <th mat-header-cell \*matHeaderCellDef> Role </th>

          <td mat-cell \*matCellDef="let element"> {{element.role.name}} </td>

        </ng-container>

        <!-- Pin code Column -->

        <ng-container matColumnDef="pinCode">

          <th mat-header-cell \*matHeaderCellDef> Pin </th>

          <td mat-cell \*matCellDef="let element"> {{element.pinCode}} </td>

        </ng-container>

        <!-- Address Column -->

        <ng-container matColumnDef="address">

          <th mat-header-cell \*matHeaderCellDef> Address </th>

          <td mat-cell \*matCellDef="let element"> {{element.address}} </td>

        </ng-container>

        <!-- Phone Column -->

        <ng-container matColumnDef="phoneNumber">

          <th mat-header-cell \*matHeaderCellDef> Phone </th>

          <td mat-cell \*matCellDef="let element"> {{element.phoneNumber}} </td>

        </ng-container>

        <tr \*matRowDef="let row; columns: displayedColumns;" (click)="onRowClick(row)" mat-row class="example-element-row"></tr>

        <tr mat-header-row \*matHeaderRowDef="displayedColumns"></tr>

      </table>

      <mat-paginator [pageSizeOptions]="[15,30,50]" showFirstLastButtons aria-label="Select page of users"></mat-paginator>

*users.component.ts*

import { Component, ViewChild } from '@angular/core';

import { MatPaginator } from '@angular/material/paginator';

import { MatTableDataSource } from '@angular/material/table';

import { Router } from '@angular/router';

import { first } from 'rxjs';

import { User } from 'src/app/models/user';

import { ComponentExchangeService } from 'src/app/services/component-exchange.service';

import { UsersService } from 'src/app/services/users.service';

@Component({

  selector: 'app-users',

  templateUrl: './users.component.html',

  styleUrls: ['./users.component.css']

})

export class UsersComponent {

  @ViewChild(MatPaginator) paginator!: MatPaginator ;

  loading:boolean=false;

  users:User[]=[];

  displayedColumns: string[] = ['name', 'login', 'role', 'pinCode', 'address', 'phoneNumber'];

  dataSource=new MatTableDataSource<User>();

  newUser(){

    this.componentExchangeService.setCurrentUser(new User());

    this.router.navigateByUrl('user');

  }

  onRowClick(user: User) {

    this.componentExchangeService.setCurrentUser(user);

    this.router.navigateByUrl('user');

  }

  constructor(private usersService:UsersService

    ,private router: Router

    ,private componentExchangeService:ComponentExchangeService){

  }

  ngAfterViewInit(): void {

    this.dataSource.paginator = this.paginator;

  }

  ngOnInit(): void {

    this.loading = true;

        this.usersService.getAllUsers().pipe(first()).subscribe(users => {

          this.loading = false;

            this.users = users;

            this.dataSource.data=this.users;

        });

  }

  applyFilter(event: Event) {

    const filterValue = (event.target as HTMLInputElement).value;

    this.dataSource.filter = filterValue.trim().toLowerCase();

    if (this.dataSource.paginator) {

      this.dataSource.paginator.firstPage();

    }

  }

}

Environments:

*environment.prod.ts*

export const environment = {

    production: true,

    apiUrl: 'http://localhost:4000'

};

*environment.ts*

import { Status } from "../models/status";

export const environment = {

    production: false,

    apiUrl: 'http://localhost:8081'

};

export const statusRaised = new Status(1,"Raised");

export const statusAssigned = new Status(2,"Assigned");

export const engineerAccesseableStatusesId:number[]=[3,4,5];

export const administratorRole:string='ADMINISTRATOR';

export const managerRole:string='MANAGER';

export const customerRole:string='CUSTOMER';

export const engineerRole:string='ENGINEER';

Helpers:

*auth.guard.ts*

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

import { Router, CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';

import { AuthenticationService } from '../services/authentication.service';

@Injectable({ providedIn: 'root' })

export class AuthGuard implements CanActivate {

    constructor(

        private router: Router,

        private authenticationService: AuthenticationService

    ) { }

    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {

        const user = this.authenticationService.userValue;

        if (user) {

            // check if route is restricted by role

            const { roles } = route.data;

            if (roles && !roles.includes(user.role)) {

                // role not authorized so redirect to home page

                this.router.navigate(['/']);

                return false;

            }

            // authorized so return true

            return true;

        }

        // not logged in so redirect to login page with the return url

        this.router.navigate(['/login'], { queryParams: { returnUrl: state.url } });

        return false;

    }

}

*error.interceptor.ts*

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

import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from '@angular/common/http';

import { Observable, throwError } from 'rxjs';

import { catchError } from 'rxjs/operators';

import { AuthenticationService } from '../services/authentication.service';

@Injectable()

export class ErrorInterceptor implements HttpInterceptor {

    constructor(private authenticationService: AuthenticationService) { }

    intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

        return next.handle(request).pipe(catchError(err => {

            if ([401, 403].includes(err.status) && this.authenticationService.userValue) {

                // auto logout if 401 Unauthorized or 403 Forbidden response returned from api

                this.authenticationService.logout();

            }

            const error = err.error.message || err.statusText;

            return throwError(() => error);

        }))

    }

}

*jwt.interceptor.ts*

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

import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from '@angular/common/http';

import { Observable } from 'rxjs';

import { AuthenticationService } from '../services/authentication.service';

import { environment } from '../enviroments/environment';

@Injectable()

export class JwtInterceptor implements HttpInterceptor {

    constructor(private authenticationService: AuthenticationService) { }

    intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

        // add auth header with jwt if user is logged in and request is to api url

        const user = this.authenticationService.userValue;

        const isLoggedIn = user?.token;

        const isApiUrl = request.url.startsWith(environment.apiUrl);

        if (isLoggedIn && isApiUrl) {

            request = request.clone({

                setHeaders: {

                    Authorization: `Bearer ${user.token}`

                }

            });

        }

        return next.handle(request);

    }

}

Models:

*complaint.ts*

import { Status } from "./status";

import { TypeOfProblem } from "./typeofproblem";

import { User } from "./user";

export class Complaint{

    customer:User | undefined | null;

    manager:User| undefined;

    engineer:User| undefined;

    status:Status| undefined;

    description:string| undefined;

    typeOfProblem:TypeOfProblem|undefined;

    feedback:string| undefined;

    constructor(customer?:User,

        description?:string,

        typeOfProblem?:TypeOfProblem,

        status?:Status,

        feedback?:string,

        engineer?:User,

        manager?:User

        ){

            this.description=description;

            this.status=status;

            this.feedback=feedback;

            this.typeOfProblem=typeOfProblem;

            this.customer=customer;

            this.engineer=engineer;

            this.manager=manager;

    }

}

*role.ts*

export class Role {

    id:number | undefined;

    name:string| undefined;

    constructor(id:number,name:string){

        this.id=id;

        this.name=name;

    }

}

*status.ts*

export class Status {

    id:number;

    name:string;

    constructor(id:number,name:string){

        this.id=id;

        this.name=name;

    }

}

*typeofproblem.ts*

//cannot make a call, but receive a call or can make calls, but cannot receive calls, or neither make nor receive calls

export class TypeOfProblem{

    id:number | undefined;

    name:string| undefined;

    fullname:string|undefined;

    constructor(id:number,name:string, fullname:string){

        this.id=id;

        this.name=name;

        this.fullname=fullname;

    }

}

*user.ts*

import { Role } from "./role";

export class User{

    id:number|undefined;

    login:string|undefined;

    password:string|undefined;

    role:Role|undefined;

    name:string|undefined;

    address:string|undefined;

    pinCode:string|undefined;

    phoneNumber:string|undefined;

    token: string|undefined;

    constructor(

        id?:number,

        login?:string,

        password?:string,

        role?:Role,

        name?:string,

        address?:string,

        pinCode?:string,

        phoneNumber?:string,

        token?:string){

            this.id=id;

            this.login=login;

            this.password=password;

            this.role=role;

            this.name=name;

            this.address=address;

            this.pinCode=pinCode;

            this.phoneNumber=phoneNumber;

            this.token=token;

    }

}

Services:

*authentication.service.ts*

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

import { Router } from '@angular/router';

import { HttpClient } from '@angular/common/http';

import { BehaviorSubject, Observable } from 'rxjs';

import { map } from 'rxjs/operators';

import { User } from '../models/user';

import { environment } from '../enviroments/environment';

@Injectable({ providedIn: 'root' })

export class AuthenticationService {

    private userSubject: BehaviorSubject<User | null>;

    public user: Observable<User | null>;

    constructor(

        private router: Router,

        private http: HttpClient

    ) {

        this.userSubject = new BehaviorSubject(JSON.parse(localStorage.getItem('user')!));

        this.user = this.userSubject.asObservable();

    }

    public get userValue() {

        return this.userSubject.value;

    }

    login(login: string, password: string) {

        return this.http.post<any>(`${environment.apiUrl}/auth/login`, { login, password })

            .pipe(map(user => {

                // store user details and jwt token in local storage to keep user logged in between page refreshes

                localStorage.setItem('user', JSON.stringify(user));

                this.userSubject.next(user);

                return user;

            }));

    }

    logout() {

        // remove user from local storage to log user out

        localStorage.removeItem('user');

        this.userSubject.next(null);

        this.router.navigate(['/login']);

    }

}

*complaints.service.ts*

import { HttpClient } from '@angular/common/http';

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

import { environment } from '../enviroments/environment';

import { Complaint } from '../models/complaint';

@Injectable({

  providedIn: 'root'

})

export class CompliantsService {

  constructor(private http: HttpClient) {

   }

  getAllCompliants() {

    return this.http.get<Complaint[]>(`${environment.apiUrl}/complaints/all`);

  }

  modifyComplaint(complaint:Complaint){

    return this.http.put<any>(`${environment.apiUrl}/complaints/update`, complaint);

  }

  newComplaint(complaint:Complaint){

    return this.http.post<any>(`${environment.apiUrl}/complaints/new`, complaint);

  }

}

*component-exchange.service.ts*

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

import { Complaint } from '../models/complaint';

import { User } from '../models/user';

@Injectable({

  providedIn: 'root'

})

export class ComponentExchangeService {

  currentComplaint!: Complaint;

  currentUser!: User;

  setCurrentComplaint(currentComplaint:Complaint){

    this.currentComplaint=currentComplaint;

  }

  getCurrentComplaint():Complaint{

    return this.currentComplaint;

  }

  setCurrentUser(user:User){

    this.currentUser=user;

  }

  getCurrentUser():User{

    return this.currentUser;

  }

  constructor() { }

}

*roles.service.ts*

import { HttpClient } from '@angular/common/http';

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

import { withCache } from '@ngneat/cashew';

import { environment } from '../enviroments/environment';

import { Role } from '../models/role';

@Injectable({

  providedIn: 'root'

})

export class RolesService {

  constructor(private http: HttpClient) { }

    getRoles() {

      return this.http.get<Role[]>(`${environment.apiUrl}/roles/all`,{

        context: withCache()

      });

    }

}

*statuses.service.ts*

import { HttpClient } from '@angular/common/http';

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

import { withCache } from '@ngneat/cashew';

import { environment } from '../enviroments/environment';

import { Status } from '../models/status';

@Injectable({

  providedIn: 'root'

})

export class StatusesService {

  constructor(private http: HttpClient) { }

  getStatuses() {

    return this.http.get<Status[]>(`${environment.apiUrl}/statuses/all`,{

      context: withCache()

    });

  }

}

*types-of-problem.service.ts*

import { HttpClient } from '@angular/common/http';

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

import { withCache } from '@ngneat/cashew';

import { environment } from '../enviroments/environment';

import { TypeOfProblem } from '../models/typeofproblem';

@Injectable({

  providedIn: 'root'

})

export class TypesOfProblemService {

  constructor(private http: HttpClient) { }

  getTypesOfProblem() {

    return this.http.get<TypeOfProblem[]>(`${environment.apiUrl}/typesofproblem/all`,{

      context: withCache()

    });

  }

}

*users.service.ts*

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

import { HttpClient, HttpErrorResponse } from '@angular/common/http';

import { User } from '../models/user';

import { environment } from '../enviroments/environment';

@Injectable({ providedIn: 'root' })

export class UsersService {

    constructor(private http: HttpClient) { }

    getAllUsers() {

      return this.http.get<User[]>(`${environment.apiUrl}/users/all`);

    }

    modifyUser(user:User){

      return this.http.put<any>(`${environment.apiUrl}/users/update`, user);

    }

    newUser(user:User){

      return this.http.post<any>(`${environment.apiUrl}/users/new`, user);

    }

}

Application level:

*app-routing.module.ts*

import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { ComplaintComponent } from './components/complaint/complaint.component';

import { ComplaintsComponent } from './components/complaints/complaints.component';

import { LoginComponent } from './components/login/login.component';

import { NewComplaintComponent } from './components/new-complaint/new-complaint.component';

import { UserComponent } from './components/user/user.component';

import { UsersComponent } from './components/users/users.component';

import { AuthGuard } from './helpers/auth.guard';

const routes: Routes = [

  {

    path: '',

    redirectTo: '/complaints',

    pathMatch: 'full'

  },

  {

    path:'complaints',

    component:ComplaintsComponent,

        canActivate: [AuthGuard]

  },

  {

    path:'complaint',

    component:ComplaintComponent,

    canActivate: [AuthGuard]

  },

  {

    path:'new-complaint',

    component:NewComplaintComponent,

    canActivate: [AuthGuard]

  },

  {

    path:'users',

    component:UsersComponent,

    canActivate: [AuthGuard]

    ,    data: { roles: ["ADMINISTRATOR"] }

  },

  {

    path:'user',

    component:UserComponent,

    canActivate: [AuthGuard],

     data: { roles: ["ADMINISTRATOR"] }

  },

  {

    path:'login',

    component: LoginComponent },

        // otherwise redirect to home

  {

    path: '\*\*', redirectTo: ''

  }

];

@NgModule({

  imports: [RouterModule.forRoot(routes)],

  exports: [RouterModule]

})

export class AppRoutingModule {

 }

*app.component.css*

.nav{

    max-width: 400px;

}

*app.component.html*

<mat-toolbar color="primary">

    <mat-toolbar-row>

      <span \*ngIf="!user">Complaints management application</span>

      <span class="example-fill-remaining-space"></span>

      <span class="example-fill-remaining-space"></span>

      <span class="align-center"></span>

      <span class="example-spacer"></span>

      <button (click)="goComlaints()" \*ngIf="user" mat-button>Complaints</button>

      <button (click)="goUsers()" \*ngIf="isAdmin" mat-button>Users</button>

      <button (click)="logout()" \*ngIf="user" mat-button>Logout {{user.name}}</button>

    </mat-toolbar-row>

  </mat-toolbar>

<router-outlet></router-outlet>

*app.component.ts*

import { Component } from '@angular/core';

import { Router } from '@angular/router';

import { User } from './models/user';

import { AuthenticationService } from './services/authentication.service';

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  title = 'angular-app';

  user?: User | null;

    constructor(private authenticationService: AuthenticationService,

      private router:Router) {

        this.authenticationService.user.subscribe(x => this.user = x);

    }

    get isAdmin() {

      return this.user?.role?.toString()==="ADMINISTRATOR";

    }

    logout() {

        this.authenticationService.logout();

    }

    goComlaints() {

      this.router.navigateByUrl('/complaints');

    }

    goUsers() {

      this.router.navigateByUrl('/users');

    }

}

*app.module.ts*

import { NgModule } from '@angular/core';

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

import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';

import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { ComplaintsComponent } from './components/complaints/complaints.component';

import { CompliantsService } from './services/complaints.service';

import { UsersComponent } from './components/users/users.component';

import { UserComponent } from './components/user/user.component';

import {MatButtonToggleModule} from '@angular/material/button-toggle';

import {MatButtonModule} from '@angular/material/button';

import {FormsModule, ReactiveFormsModule } from '@angular/forms';

import {MatTableModule} from '@angular/material/table';

import {MatPaginatorModule} from '@angular/material/paginator';

import {MatFormFieldModule} from '@angular/material/form-field';

import {MatInputModule} from '@angular/material/input';

import {ComplaintComponent } from './components/complaint/complaint.component';

import {MatSelectModule } from '@angular/material/select';

import {NewComplaintComponent } from './components/new-complaint/new-complaint.component';

import {MatTabsModule} from '@angular/material/tabs';

import {MatCardModule} from '@angular/material/card';

import {MatProgressSpinnerModule} from '@angular/material/progress-spinner';

import {MatToolbarModule} from '@angular/material/toolbar';

import {MatMenuModule} from '@angular/material/menu';

import {MatIconModule} from '@angular/material/icon';

import { JwtInterceptor } from './helpers/jwt.interceptor';

import { HttpClientModule, HTTP\_INTERCEPTORS } from "@angular/common/http";

import { ErrorInterceptor } from './helpers/error.interceptor';

import { LoginComponent } from './components/login/login.component';

import { HttpCacheInterceptorModule } from '@ngneat/cashew';

@NgModule({

  declarations: [

    AppComponent,

    ComplaintsComponent,

    ComplaintComponent,

    NewComplaintComponent,

    UsersComponent,

    UserComponent,

    LoginComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule,

    BrowserAnimationsModule,

    MatButtonToggleModule,

    FormsModule,

    ReactiveFormsModule,

    MatTableModule,

    MatPaginatorModule,

    MatFormFieldModule,

    MatInputModule,

    MatSelectModule,

    MatButtonModule,

    MatTabsModule,

    MatCardModule,

    MatProgressSpinnerModule,

    MatToolbarModule,

    MatMenuModule,

    MatIconModule,

    HttpClientModule,

    HttpCacheInterceptorModule.forRoot()

  ],

  providers: [

    CompliantsService,

    { provide: HTTP\_INTERCEPTORS, useClass: JwtInterceptor, multi: true },

    { provide: HTTP\_INTERCEPTORS, useClass: ErrorInterceptor, multi: true },

  ],

  bootstrap: [AppComponent]

})

export class AppModule { }

*angular.json*

{

  "$schema": "./node\_modules/@angular/cli/lib/config/schema.json",

  "version": 1,

  "newProjectRoot": "projects",

  "projects": {

    "angular-app": {

      "projectType": "application",

      "schematics": {},

      "root": "",

      "sourceRoot": "src",

      "prefix": "app",

      "architect": {

        "build": {

          "builder": "@angular-devkit/build-angular:browser",

          "options": {

            "outputPath": "dist/angular-app",

            "index": "src/index.html",

            "main": "src/main.ts",

            "polyfills": [

              "zone.js"

            ],

            "tsConfig": "tsconfig.app.json",

            "assets": [

              "src/favicon.ico",

              "src/assets"

            ],

            "styles": [

              "@angular/material/prebuilt-themes/deeppurple-amber.css",

              "./node\_modules/bootstrap/dist/css/bootstrap.css",

              "src/styles.css"

            ],

            "scripts": []

          },

          "configurations": {

            "production": {

              "budgets": [

                {

                  "type": "initial",

                  "maximumWarning": "500kb",

                  "maximumError": "1mb"

                },

                {

                  "type": "anyComponentStyle",

                  "maximumWarning": "2kb",

                  "maximumError": "4kb"

                }

              ],

              "outputHashing": "all"

            },

            "development": {

              "buildOptimizer": false,

              "optimization": false,

              "vendorChunk": true,

              "extractLicenses": false,

              "sourceMap": true,

              "namedChunks": true

            }

          },

          "defaultConfiguration": "production"

        },

        "serve": {

          "builder": "@angular-devkit/build-angular:dev-server",

          "configurations": {

            "production": {

              "browserTarget": "angular-app:build:production"

            },

            "development": {

              "browserTarget": "angular-app:build:development"

            }

          },

          "defaultConfiguration": "development"

        },

        "extract-i18n": {

          "builder": "@angular-devkit/build-angular:extract-i18n",

          "options": {

            "browserTarget": "angular-app:build"

          }

        },

        "test": {

          "builder": "@angular-devkit/build-angular:karma",

          "options": {

            "polyfills": [

              "zone.js",

              "zone.js/testing"

            ],

            "tsConfig": "tsconfig.spec.json",

            "assets": [

              "src/favicon.ico",

              "src/assets"

            ],

            "styles": [

              "@angular/material/prebuilt-themes/pink-bluegrey.css",

              "src/styles.css"

            ],

            "scripts": []

          }

        }

      }

    }

  },

  "cli": {

    "analytics": false

  }

}