**BackEnd Code**

**Entity**

**User**

package com.example.demo.model;

import java.util.Date;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.EnumType;

import javax.persistence.Enumerated;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name = "users")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@Column(name = "name", nullable = false)

private String name;

private String email;

private String password;

@Enumerated(EnumType.STRING)

@Column(nullable = false)

private UserType usertype = UserType.CUSTOMER;

@Column(name = "phoneno", length = 10)

private long phoneno;

@Column(name = "last\_update", columnDefinition = "TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP")

private Date last\_update = new Date();

public User() {}

public User(String name, String email, String password, UserType usertype, long phoneno) {

super();

this.name = name;

this.email = email;

this.password = password;

this.usertype = usertype;

this.phoneno = phoneno;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public UserType getUsertype() {

return usertype;

}

public void setUsertype(UserType usertype) {

this.usertype = usertype;

}

public long getPhoneno() {

return phoneno;

}

public void setPhoneno(long phoneno) {

this.phoneno = phoneno;

}

public Date getLast\_update() {

return last\_update;

}

public void setLast\_update(Date last\_update) {

this.last\_update = last\_update;

}

}

**Usertype**

**package** com.example.demo.model;

**public** **enum** UserType {

***CUSTOMER***,

***MANAGER***,

***ENGINEER***

}

**Complaint**

package com.example.demo.model;

import java.util.Date;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.ManyToOne;

import javax.persistence.Table;

@Entity

@Table(name = "complaints")

public class Complaint {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

private String subject;

private String detail;

@ManyToOne(cascade = CascadeType.MERGE)

@JoinColumn(name = "user\_id")

private User user;

private boolean isResolved = false;

@Column(name = "last\_update", columnDefinition = "TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP")

private Date last\_update = new Date();

public Complaint() {}

public Complaint(String subject, String detail, User user, boolean isResolved, Date last\_update) {

super();

this.subject = subject;

this.detail = detail;

this.user = user;

this.isResolved = isResolved;

this.last\_update = last\_update;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getSubject() {

return subject;

}

public void setSubject(String subject) {

this.subject = subject;

}

public String getDetail() {

return detail;

}

public void setDetail(String detail) {

this.detail = detail;

}

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

public boolean isResolved() {

return isResolved;

}

public void setResolved(boolean isResolved) {

this.isResolved = isResolved;

}

public Date getLast\_update() {

return last\_update;

}

public void setLast\_update(Date last\_update) {

this.last\_update = last\_update;

}

}

**ComplaintUpdate**

package com.example.demo.model;

import java.util.Date;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.ManyToOne;

import javax.persistence.Table;

@Entity

@Table(name = "complaint\_updates")

public class ComplaintUpdates {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@ManyToOne(cascade = CascadeType.MERGE)

@JoinColumn(name = "complaint\_id")

private Complaint complaint;

@ManyToOne(cascade = CascadeType.MERGE)

@JoinColumn(name = "user\_id")

private User user;//assigned engineer's id

// If both are false then again complaint pass to manager to assign another engineer

private boolean isWorkingOn = true;

private boolean isResolved = false;

private String statusRemark;

@Column(name = "last\_update", columnDefinition = "TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP")

private Date last\_update = new Date();

public ComplaintUpdates() {}

public ComplaintUpdates(Complaint complaint, User user, boolean isWorkingOn, boolean isResolved, String statusRemark, Date last\_update) {

super();

this.complaint = complaint;

this.user = user;

this.isWorkingOn = isWorkingOn;

this.isResolved = isResolved;

this.statusRemark = statusRemark;

this.last\_update = last\_update;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public Complaint getComplaint() {

return complaint;

}

public void setComplaint(Complaint complaint) {

this.complaint = complaint;

}

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

public boolean isWorkingOn() {

return isWorkingOn;

}

public void setWorkingOn(boolean isWorkingOn) {

this.isWorkingOn = isWorkingOn;

}

public boolean isResolved() {

return isResolved;

}

public void setResolved(boolean isResolved) {

this.isResolved = isResolved;

}

public String getStatusRemark() {

return statusRemark;

}

public void setStatusRemark(String statusRemark) {

this.statusRemark = statusRemark;

}

public Date getLast\_update() {

return last\_update;

}

public void setLast\_update(Date last\_update) {

this.last\_update = last\_update;

}

}

**Repository:-**

**ComplaintRepo:-**

package com.example.demo.repo;

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

import com.example.demo.model.Complaint;

public interface ComplaintRepository extends JpaRepository<Complaint, Integer>{

}

**ComplaintUpdateRepo:-**

package com.example.demo.repo;

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

import com.example.demo.model.ComplaintUpdates;

public interface ComplaintUpdateRepository extends JpaRepository<ComplaintUpdates, Integer> {

}

**UserRepo:-**

package com.example.demo.repo;

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

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

import com.example.demo.model.User;

import com.example.demo.model.UserType;

public interface UserRepository extends JpaRepository<User, Integer>{

@Query("FROM User WHERE name=?1 and password = ?2 and usertype=?3")

public User getUserByUsernameAndPaswordAndRole(String name, String password, UserType usertype);

}

**Service:-**

**ComplaintService:-**

package com.example.demo.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import com.example.demo.model.Complaint;

import com.example.demo.repo.ComplaintRepository;

@Service

public class ComplaintService {

@Autowired

ComplaintRepository repo;

public Complaint addComplaint(Complaint comp) {

comp = repo.save(comp);

if (comp != null) {

comp = getComplaintById(comp.getId());

}

return comp;

}

public List<Complaint> getAllComplaints() {

return repo.findAll();

}

public List<Complaint> getUserRoleComplaints(int userid) {

return repo.findAll();

}

public Complaint getComplaintById(int id) {

if (repo.findById(id).isPresent())

return repo.findById(id).get();

else

return null;

}

}

**ComplaintUpdateService:-**

package com.example.demo.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import com.example.demo.model.ComplaintUpdates;

import com.example.demo.repo.ComplaintUpdateRepository;

@Service

public class ComplaintUpdateService {

@Autowired

ComplaintUpdateRepository repo;

public ComplaintUpdates addComplaintUpdates(ComplaintUpdates compUpdate) {

compUpdate = repo.save(compUpdate);

if (compUpdate != null) {

compUpdate = getComplaintUpdatesById(compUpdate.getId());

}

return compUpdate;

}

public List<ComplaintUpdates> getAllComplaintUpdates() {

return repo.findAll();

}

public ComplaintUpdates getComplaintUpdatesById(int id) {

if (repo.findById(id).isPresent())

return repo.findById(id).get();

else

return null;

}

}

**UserService:-**

package com.example.demo.service;

import java.util.ArrayList;

import java.util.List;

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

import org.springframework.stereotype.Service;

import com.example.demo.model.User;

import com.example.demo.model.UserType;

import com.example.demo.repo.UserRepository;

@Service

public class UserService {

@Autowired

UserRepository repo;

public User authenticateUser(String username, String password, UserType usertype) {

return repo.getUserByUsernameAndPaswordAndRole(username, password, usertype);

}

public User addUser(User user) {

user = repo.save(user);

if (user != null) {

user = getUserById(user.getId());

}

return user;

}

public List<User> getAllUsers() {

return repo.findAll();

}

public List<User> getAllEngineers() {

List<User> lu = new ArrayList<User>();

for(User u : repo.findAll()) {

if(u.getUsertype() == UserType.ENGINEER) {

lu.add(u);

}

}

return lu;

}

public User getUserById(int id) {

if (repo.findById(id).isPresent())

return repo.findById(id).get();

else

return null;

}

public User updateUser(User user, int id) {

if (repo.findById(id).isPresent()) {

User old = repo.findById(id).get();

old.setName(user.getName());

old.setEmail(user.getEmail());

old.setPassword(user.getPassword());

old.setUsertype(user.getUsertype());

old.setPhoneno(user.getPhoneno());

return repo.save(old);

} else

return null;

}

public boolean deleteUser(int id) {

if (repo.findById(id).isPresent()) {

repo.deleteById(id);

return true;

}

return false;

}

}

**Controller:-**

**MainController:-**

package com.example.demo.controller;

import java.util.Date;

import java.util.List;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

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

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

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

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

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

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

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

import com.example.demo.model.Complaint;

import com.example.demo.model.ComplaintUpdates;

import com.example.demo.model.User;

import com.example.demo.model.UserType;

import com.example.demo.service.ComplaintService;

import com.example.demo.service.ComplaintUpdateService;

import com.example.demo.service.UserService;

@RestController

public class MainController {

@Autowired

UserService userService;

@Autowired

ComplaintService compService;

@Autowired

ComplaintUpdateService compUpdateService;

//

@PostMapping("/login")

public User authUser(@RequestParam String username,@RequestParam String password,@RequestParam String usertype) {

System.out.println("Login Params : " + username + "\_" + password + "\_" + usertype);

UserType ut = UserType.CUSTOMER;

if(usertype.equalsIgnoreCase("MANAGER")) {

ut = UserType.MANAGER;

} else if(usertype.equalsIgnoreCase("ENGINEER")) {

ut = UserType.ENGINEER;

}

User user = userService.authenticateUser(username, password, ut);

System.out.println("User : " + user.toString());

//

return authUser(null, null, null);

}

@GetMapping("/user")

public List<User> getAllUsers() {

return userService.getAllUsers();

}

@GetMapping("/user/engineers")

public List<User> getAllEngineers() {

return userService.getAllEngineers();

}

@GetMapping("/user/{id}")

public ResponseEntity<Object> getUser(@PathVariable int id) {

User user = userService.getUserById(id);

if (user != null)

return new ResponseEntity<Object>(user, HttpStatus.FOUND);

else

return new ResponseEntity<Object>("User is not available with given id", HttpStatus.NOT\_FOUND);

}

@PutMapping("/user/{id}")

public ResponseEntity<Object> updateUser(@PathVariable int id, @RequestBody User user) {

User updatedUser = userService.updateUser(user, id);

if (updatedUser != null)

return new ResponseEntity<Object>(updatedUser, HttpStatus.FOUND);

else

return new ResponseEntity<Object>("Updated Operation Failed..", HttpStatus.NOT\_FOUND);

}

@PostMapping("/user")

public ResponseEntity<Object> addUser(@RequestBody User user) {

User addedUser = userService.addUser(user);

if (addedUser != null)

return new ResponseEntity<Object>(addedUser, HttpStatus.CREATED);

else

return new ResponseEntity<Object>("Error while adding user", HttpStatus.INTERNAL\_SERVER\_ERROR);

}

@DeleteMapping("/user/{id}")

public ResponseEntity<Object> deleteUser(@PathVariable int id) {

if(userService.deleteUser(id))

return new ResponseEntity<Object>("User Is Successfully Deleted", HttpStatus.FOUND);

else

return new ResponseEntity<Object>("User is not available with given id : " + id, HttpStatus.NOT\_FOUND);

}

@PostMapping("/complaint")

public ResponseEntity<Object> addComplaint(@RequestBody Complaint comp){

Complaint newComplaint = new Complaint(comp.getSubject(), comp.getDetail(), getUserByObject(comp.getUser()), comp.isResolved(), new Date());

newComplaint = compService.addComplaint(comp);

if (newComplaint != null)

return new ResponseEntity<Object>(newComplaint, HttpStatus.CREATED);

else

return new ResponseEntity<Object>("Error while adding complaint.", HttpStatus.INTERNAL\_SERVER\_ERROR);

}

@GetMapping("/complaint")

public List<Complaint> getAllComplaints(){

return compService.getAllComplaints();

}

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

public List<Complaint> getUserRoleComplaints(@PathVariable int userid){

System.out.println("Request User ID : " + userid);

return compService.getUserRoleComplaints(userid);

}

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

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

Complaint comp = compService.getComplaintById(id);

if (comp != null)

return new ResponseEntity<Object>(comp, HttpStatus.CREATED);

else

return new ResponseEntity<Object>("Error while getting complaint by ID.", HttpStatus.INTERNAL\_SERVER\_ERROR);

}

@PostMapping("/complaint/update")

public ResponseEntity<Object> addComplaintUpdate(@RequestBody ComplaintUpdates compUpdate){

ComplaintUpdates addCompUpdate = new ComplaintUpdates(getComplaintByObject(compUpdate.getComplaint()), getUserByObject(compUpdate.getUser()),

compUpdate.isWorkingOn(),compUpdate.isResolved(), compUpdate.getStatusRemark(), new Date());

addCompUpdate = compUpdateService.addComplaintUpdates(compUpdate);

if (addCompUpdate != null)

return new ResponseEntity<Object>(addCompUpdate, HttpStatus.CREATED);

else

return new ResponseEntity<Object>("Error while adding complaint.", HttpStatus.INTERNAL\_SERVER\_ERROR);

}

@GetMapping("/complaint/update")

public List<ComplaintUpdates> getAllComplaintUpdates(){

return compUpdateService.getAllComplaintUpdates();

}

public User getUserByObject(User user) {

User userOb = userService.getUserById(user.getId());

if(userOb == null || userOb.getId()== 0) {

userOb = new User(user.getName(), user.getEmail(), user.getPassword(), user.getUsertype(), user.getPhoneno());

}

return userOb;

}

public Complaint getComplaintByObject(Complaint comp) {

Complaint compOb = compService.getComplaintById(comp.getId());

if(compOb == null || compOb.getId()== 0) {

compOb = new Complaint(comp.getSubject(), comp.getDetail(), getUserByObject(comp.getUser()), comp.isResolved(), new Date());

}

return compOb;

}

}

**Config:-**

package com.example.demo.config;

import java.io.IOException;

import javax.servlet.Filter;

import javax.servlet.FilterChain;

import javax.servlet.FilterConfig;

import javax.servlet.ServletException;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.stereotype.Component;

@Component

public class SimpleCORSFilter implements Filter{

private final Logger log = LoggerFactory.getLogger(SimpleCORSFilter.class);

public SimpleCORSFilter() {

log.info("SimpleCORSFilter init");

}

@Override

public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)

throws IOException, ServletException {

HttpServletRequest request = (HttpServletRequest) req;

HttpServletResponse response = (HttpServletResponse) res;

response.setHeader("Access-Control-Allow-Origin", request.getHeader("Origin"));

response.setHeader("Access-Control-Allow-Credentials", "true");

response.setHeader("Access-Control-Allow-Methods", "POST, PUT, GET, OPTIONS, DELETE");

response.setHeader("Access-Control-Max-Age", "3600");

response.setHeader("Access-Control-Allow-Headers", "Content-Type, Accept, X-Requested-With, remember-me");

chain.doFilter(req, res);

}

@Override

public void init(FilterConfig filterConfig) {

}

@Override

public void destroy() {

}

}