# spring\QrmsSpringApplication.java

package com.qrms.spring;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.scheduling.annotation.EnableAsync;  
  
@EnableAsync  
@SpringBootApplication  
public class QrmsSpringApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.run(QrmsSpringApplication.class, args);  
 }  
  
}

# spring\comparators\CourseComparatorByYear.java

package com.qrms.spring.comparators;  
  
import java.util.Map;  
  
import java.util.Comparator;  
  
class CourseComparatorByYear implements Comparator<String>  
{  
 Map<String, Integer> map;  
 public CourseComparatorByYear(Map<String, Integer> map)  
 {  
 this.map = map;  
 }  
   
 public int compare(String x, String y)  
 {  
 if (map.containsKey(x) && map.containsKey(y))  
 return map.get(x) - map.get(y);  
 else if (map.containsKey(y))  
 return 1;  
 else if (map.containsKey(x))  
 return -1;  
// else  
// return x - y;  
 return 0;  
 }  
}

# spring\comparators\DivisionsChainedComparator.java

package com.qrms.spring.comparators;  
  
import java.util.Arrays;  
import java.util.Comparator;  
import java.util.List;  
  
import com.qrms.spring.model.Divisions;  
  
public class DivisionsChainedComparator implements Comparator <Divisions>{  
 private List<Comparator<Divisions>> listComparators;  
  
  
 @Override  
 public int compare(Divisions arg0, Divisions arg1) {  
 // TODO Auto-generated method stub  
 for (Comparator<Divisions> comparator : listComparators) {  
 int result = comparator.compare(arg0, arg1);  
 if (result != 0) {  
 return result;  
 }  
 }  
 return 0;  
 }  
   
   
 @SafeVarargs  
 public DivisionsChainedComparator(Comparator<Divisions>... comparators) {  
 this.listComparators = Arrays.asList(comparators);  
 }  
}

# spring\comparators\DivisionsYearComparator.java

package com.qrms.spring.comparators;  
  
import java.util.Arrays;  
import java.util.Comparator;  
import java.util.List;  
  
import com.qrms.spring.model.Divisions;  
  
  
//1 - 2 for ascending  
//2 - 1 for descending  
public class DivisionsYearComparator implements Comparator<Divisions> {  
   
 @Override  
 public int compare(Divisions d1, Divisions d2) {  
   
 List<String> years= Arrays.asList("ME1","ME2","BE","TE","SE","FE");  
 return years.indexOf(d1.getYear())-years.indexOf(d2.getYear());  
 }  
}

# spring\comparators\FacultyPrefChainedComparator.java

package com.qrms.spring.comparators;  
  
import java.util.Arrays;  
import java.util.Comparator;  
import java.util.List;  
  
import com.qrms.spring.model.FacultyPref;  
  
public class FacultyPrefChainedComparator implements Comparator <FacultyPref>{  
 private List<Comparator<FacultyPref>> listComparators;  
  
  
 @Override  
 public int compare(FacultyPref arg0, FacultyPref arg1) {  
 // TODO Auto-generated method stub  
 for (Comparator<FacultyPref> comparator : listComparators) {  
 int result = comparator.compare(arg0, arg1);  
 if (result != 0) {  
 return result;  
 }  
 }  
 return 0;  
 }  
   
   
 @SafeVarargs  
 public FacultyPrefChainedComparator(Comparator<FacultyPref>... comparators) {  
 this.listComparators = Arrays.asList(comparators);  
 }  
}

# spring\comparators\FacultyPrefCourseExpComparator.java

package com.qrms.spring.comparators;  
  
import java.util.Comparator;  
  
import com.qrms.spring.model.FacultyPref;  
  
public class FacultyPrefCourseExpComparator implements Comparator<FacultyPref> {  
   
 @Override  
 public int compare(FacultyPref f1, FacultyPref f2) {  
 return f2.getCourseExp() - f1.getCourseExp();  
 }  
}

# spring\comparators\FacultyPrefNoComparator.java

package com.qrms.spring.comparators;  
  
import java.util.Comparator;  
  
import com.qrms.spring.model.FacultyPref;  
  
public class FacultyPrefNoComparator implements Comparator<FacultyPref> {  
   
 @Override  
 public int compare(FacultyPref f1, FacultyPref f2) {  
 return f1.getPrefNo() - f2.getPrefNo();  
 }  
}

# spring\comparators\FacultyPrefPrereqExp1Comparator.java

package com.qrms.spring.comparators;  
  
import java.util.Comparator;  
  
import com.qrms.spring.model.FacultyPref;  
  
public class FacultyPrefPrereqExp1Comparator implements Comparator<FacultyPref> {  
   
 @Override  
 public int compare(FacultyPref f1, FacultyPref f2) {  
 return f2.getPrereq1Exp() - f1.getPrereq1Exp();  
 }  
}

# spring\comparators\FacultyPrefPrereqExp2Comparator.java

package com.qrms.spring.comparators;  
  
import java.util.Comparator;  
  
import com.qrms.spring.model.FacultyPref;  
  
public class FacultyPrefPrereqExp2Comparator implements Comparator<FacultyPref> {  
   
 @Override  
 public int compare(FacultyPref f1, FacultyPref f2) {  
 return f2.getPrereq2Exp() - f1.getPrereq2Exp();  
 }  
}

# spring\config\SecurityConfiguration.java

package com.qrms.spring.config;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;  
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.builders.WebSecurity;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.security.web.authentication.AuthenticationSuccessHandler;  
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;  
  
import com.qrms.spring.repository.UsersRepository;  
import com.qrms.spring.service.CustomUserDetailsService;  
  
@EnableGlobalMethodSecurity(prePostEnabled = true)  
@EnableWebSecurity  
@EnableJpaRepositories(basePackageClasses = UsersRepository.class)  
@Configuration  
public class SecurityConfiguration extends WebSecurityConfigurerAdapter{  
   
 @Autowired  
 private CustomUserDetailsService userDetailsService;  
  
   
 @Override  
 protected void configure(AuthenticationManagerBuilder auth) throws Exception{  
   
 auth.userDetailsService(userDetailsService).passwordEncoder(getPasswordEncoder());  
 }  
   
 @Bean  
 public AuthenticationSuccessHandler myAuthenticationSuccessHandler(){  
 return new UserAuthenticationSuccessHandler();  
 }  
   
 @Override  
 public void configure(WebSecurity web) throws Exception {  
 web.ignoring().antMatchers("/validateToken\*","/updatePassword\*","/forgotPassword\*");  
 }  
   
 @Override  
 protected void configure(HttpSecurity http) throws Exception{  
 http.csrf().disable();  
 http.authorizeRequests()  
 .antMatchers("/updatePassword\*",  
 "/validateToken\*")  
 .hasAuthority("CHANGE\_PASSWORD\_PRIVILEGE")  
 .antMatchers("/u/admin/\*").hasRole("ADMIN")  
 .antMatchers("/u/student/\*").hasRole("STUDENT")  
 .antMatchers("/u/faculty/\*").hasRole("FACULTY")  
 .and()  
 .formLogin()  
 .successHandler(myAuthenticationSuccessHandler()).permitAll()  
 .loginPage("/login")  
 .and()  
 .logout().logoutRequestMatcher(new AntPathRequestMatcher("/logout")).logoutSuccessUrl("/login");  
  
 }  
   
 private PasswordEncoder getPasswordEncoder() {  
 return new PasswordEncoder() {  
   
 @Override  
 public boolean matches(CharSequence rawPassword, String encodedPassword) {  
   
 return true;  
// return new BCryptPasswordEncoder().encode(rawPassword.toString())  
// .equals(encodedPassword);  
 }  
   
 @Override  
 public String encode(CharSequence rawPassword) {  
 return rawPassword.toString();  
// return new BCryptPasswordEncoder().encode(rawPassword.toString());  
 }  
 };  
 }  
   
   
}

# spring\config\UserAuthenticationSuccessHandler.java

package com.qrms.spring.config;  
import java.io.IOException;  
import java.util.Collection;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import javax.servlet.http.HttpSession;  
  
import org.springframework.security.web.authentication.AuthenticationSuccessHandler;  
//import org.hibernate.validator.internal.util.logging.Log\_.logger;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.web.DefaultRedirectStrategy;  
import org.springframework.security.web.RedirectStrategy;  
import org.springframework.security.web.WebAttributes;  
import org.springframework.stereotype.Component;  
  
@Component  
public class UserAuthenticationSuccessHandler implements AuthenticationSuccessHandler {  
   
 private RedirectStrategy redirectStrategy = new DefaultRedirectStrategy();  
   
   
 @Override  
 public void onAuthenticationSuccess(HttpServletRequest request,HttpServletResponse response, Authentication authentication)  
 throws IOException, ServletException {  
 handle(request, response, authentication);  
 clearAuthenticationAttributes(request);   
 }  
   
 protected void handle(HttpServletRequest request, HttpServletResponse response, Authentication authentication) throws IOException {  
   
 String targetUrl = determineTargetUrl(authentication);  
 redirectStrategy.sendRedirect(request, response, targetUrl);  
   
 }  
   
 protected String determineTargetUrl(Authentication authentication) {  
 boolean isFaculty = false;  
 boolean isAdmin = false;  
 boolean isStudent = false;  
   
 Collection<? extends GrantedAuthority> authorities = authentication.getAuthorities();  
 for (GrantedAuthority grantedAuthority : authorities) {  
 if (grantedAuthority.getAuthority().equals("ROLE\_STUDENT")) {  
 isStudent = true;  
 break;  
 } else if (grantedAuthority.getAuthority().equals("ROLE\_ADMIN")) {  
 isAdmin = true;  
 break;  
 } else if (grantedAuthority.getAuthority().equals("ROLE\_FACULTY")) {  
 isFaculty = true;  
 break;  
 }  
 }  
   
 if (isStudent) {  
 return "/u/student/home";  
 } else if (isFaculty) {  
 return "/u/faculty/home";  
 } else if (isAdmin) {  
 return "/u/admin/home";  
 }  
 else {  
 throw new IllegalStateException();  
 }  
 }  
   
 protected void clearAuthenticationAttributes(HttpServletRequest request) {  
 HttpSession session = request.getSession(false);  
 if (session == null) {  
 return;  
 }  
 session.removeAttribute(WebAttributes.AUTHENTICATION\_EXCEPTION);  
 }  
   
 public void setRedirectStrategy(RedirectStrategy redirectStrategy) {  
 this.redirectStrategy = redirectStrategy;  
 }  
   
 protected RedirectStrategy getRedirectStrategy() {  
 return redirectStrategy;  
 }  
}

# spring\model\CompanionCourse.java

package com.qrms.spring.model;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.Table;  
  
/\*  
 \* Columns-  
 \* course  
 \* companionCourse  
 \*/  
@Entity  
@Table(name="companion\_course")  
public class CompanionCourse {  
  
 public CompanionCourse(String course, String companionCourse) {  
 this.course = course;  
 this.companionCourse = companionCourse;  
 }  
   
 public CompanionCourse() {  
   
 }  
   
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name="id")  
 private int id;  
  
 @Column(name="course\_id")  
 private String course;  
   
 @Column(name = "companion\_course")  
 private String companionCourse;  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getCourse() {  
 return course;  
 }  
  
 public void setCourse(String course) {  
 this.course = course;  
 }  
  
 public String getCompanionCourse() {  
 return companionCourse;  
 }  
  
 public void setCompanionCourse(String companionCourse) {  
 this.companionCourse = companionCourse;  
 }  
}

# spring\model\Course.java

package com.qrms.spring.model;  
  
import java.util.HashSet;  
import java.util.Set;  
  
import javax.persistence.CascadeType;  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.ManyToOne;  
import javax.persistence.OneToMany;  
import javax.persistence.Table;  
  
/\*  
 \* Columns-  
 \*   
 \* courseId  
 \* courseName  
 \* courseCredits  
 \* department  
 \* courseType  
 \* courseYear  
 \* courseSem  
 \* studAllocFlag  
 \* isTheory  
 \* noOfHours  
 \*   
 \*/  
@Entity  
@Table(name = "course")  
public class Course {  
  
 public Course(String courseId, String courseName, Integer courseCredits, Department department, char courseType,  
 String courseYear, int courseSem, int studAllocFlag, int isTheory, int noOfHours) {  
 super();  
 this.courseId = courseId;  
 this.courseName = courseName;  
 this.courseCredits = courseCredits;  
 this.department = department;  
 this.courseType = courseType;  
 this.courseYear = courseYear;  
 this.courseSem = courseSem;  
 this.studAllocFlag = studAllocFlag;  
 this.isTheory = isTheory;  
 this.noOfHours = noOfHours;  
 }  
  
 public Course() {  
  
 }  
   
 @Id  
 @Column(name = "course\_id")  
 private String courseId;  
  
 @Column(name = "course\_name")  
 private String courseName;  
  
 @Column(name = "course\_credits")  
 private Integer courseCredits;  
  
 // Child of FK relation to Department -- do not cascade on delete/update  
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "dept\_id")  
 private Department department;  
  
 // Parent of FK relation to Electives -- do not cascade on delete/update  
 @OneToMany(mappedBy = "course", cascade = CascadeType.ALL)  
 Set<Electives> electives = new HashSet<Electives>();  
  
 // Parent of FK relation to StudentAllocs -- do not cascade on  
 // delete/update  
 @OneToMany(mappedBy = "courseId", cascade = CascadeType.ALL)  
 Set<StudentAllocCourse> course\_Ids = new HashSet<StudentAllocCourse>();  
   
 // Parent of FK relation to CourseCompanion -- do not cascade on  
 // delete/update  
 @OneToMany(mappedBy = "companionCourse", cascade = CascadeType.ALL)  
 Set<CompanionCourse> companionCourses = new HashSet<CompanionCourse>();  
  
 // Parent of FK relation to CourseCompanion -- do not cascade on  
 // delete/update  
 @OneToMany(mappedBy = "course", cascade = CascadeType.ALL)  
 Set<CompanionCourse> courses = new HashSet<CompanionCourse>();  
   
 //E: Elective R: regular course  
 @Column(name = "course\_type")  
 private char courseType;  
  
 @Column(name = "course\_year")  
 private String courseYear;  
  
 @Column(name = "course\_sem")  
 private int courseSem;  
  
 // 0 = allocation not yet started by admin  
 // 1 = allocation has been started and students can give prefs  
 @Column(name = "stud\_allocation\_flag")  
 private int studAllocFlag = 0;  
  
 @Column(name = "is\_theory")  
 private Integer isTheory;  
  
 @Column(name = "no\_of\_hours")  
 private Integer noOfHours;  
  
 public Set<StudentAllocCourse> getCourse\_Ids() {  
 return course\_Ids;  
 }  
  
 public void setCourse\_Ids(Set<StudentAllocCourse> course\_Ids) {  
 this.course\_Ids = course\_Ids;  
 }  
  
 public Integer getIsTheory() {  
 return isTheory;  
 }  
  
 public void setIsTheory(Integer isTheory) {  
 this.isTheory = isTheory;  
 }  
  
 public Integer getNoOfHours() {  
 return noOfHours;  
 }  
  
 public void setNoOfHours(Integer noOfHours) {  
 this.noOfHours = noOfHours;  
 }  
  
 public int getStudAllocFlag() {  
 return studAllocFlag;  
 }  
  
 public void setStudAllocFlag(int studAllocFlag) {  
 this.studAllocFlag = studAllocFlag;  
 }  
  
 public String getCourseId() {  
 return courseId;  
 }  
  
 public void setCourseId(String courseId) {  
 this.courseId = courseId;  
 }  
  
 public String getCourseName() {  
 return courseName;  
 }  
  
 public void setCourseName(String courseName) {  
 this.courseName = courseName;  
 }  
  
 public Integer getCourseCredits() {  
 return courseCredits;  
 }  
  
 public void setCourseCredits(Integer courseCredits) {  
 this.courseCredits = courseCredits;  
 }  
  
 public Department getDepartment() {  
 return department;  
 }  
  
 public void setDepartment(Department department) {  
 this.department = department;  
 }  
  
 public char getCourseType() {  
 return courseType;  
 }  
  
 public void setCourseType(char courseType) {  
 this.courseType = courseType;  
 }  
  
 public String getCourseYear() {  
 return courseYear;  
 }  
  
 public void setCourseYear(String courseYear) {  
 this.courseYear = courseYear;  
 }  
  
 public int getCourseSem() {  
 return courseSem;  
 }  
  
 public void setCourseSem(int courseSem) {  
 this.courseSem = courseSem;  
 }  
   
 public Set<Electives> getElectives() {  
 return electives;  
 }  
  
 public void setElectives(Set<Electives> electives) {  
 this.electives = electives;  
 }  
  
 public Set<CompanionCourse> getCompanionCourses() {  
 return companionCourses;  
 }  
  
 public void setCompanionCourses(Set<CompanionCourse> companionCourses) {  
 this.companionCourses = companionCourses;  
 }  
  
 public Set<CompanionCourse> getCourses() {  
 return courses;  
 }  
  
 public void setCourses(Set<CompanionCourse> courses) {  
 this.courses = courses;  
 }  
  
}

# spring\model\CourseList.java

package com.qrms.spring.model;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="course\_list\_faculty")  
public class CourseList {  
   
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name="id")  
 private int id;  
   
 @Column(name="course\_id")  
 private String courseId;  
   
 @Column(name="division\_id")  
 private String divisionId;  
   
 @Column(name="faculty\_id")  
 private String facultyId;  
   
 @Column(name="no\_of\_hours")  
 private int noOfHours;  
   
 public CourseList() {  
   
 }  
   
 public CourseList(String courseId, String divisionId, String facultyId,int noOfHours) {  
 super();  
 this.courseId = courseId;  
 this.divisionId = divisionId;  
 this.facultyId = facultyId;  
 this.noOfHours = noOfHours;  
 }  
  
 public int getNoOfHours() {  
 return noOfHours;  
 }  
  
 public void setNoOfHours(int noOfHours) {  
 this.noOfHours = noOfHours;  
 }  
  
 public String getDivisionId() {  
 return divisionId;  
 }  
  
 public void setDivisionId(String divisionId) {  
 this.divisionId = divisionId;  
 }  
  
 public String getCourseId() {  
 return courseId;  
 }  
  
 public void setCourseId(String courseId) {  
 this.courseId = courseId;  
 }  
   
 public String getFacultyId() {  
 return facultyId;  
 }  
  
 public void setFacultyId(String facultyId) {  
 this.facultyId = facultyId;  
 }  
  
   
}

# spring\model\CoursePrerequisites.java

package com.qrms.spring.model;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.Id;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="course\_prerequisites")  
public class CoursePrerequisites {  
  
 public CoursePrerequisites(String courseId, int isPrereq1Elective, int isPrereq2Elective, String prerequisiteNo1,  
 String prerequisiteNo2) {  
 super();  
 this.courseId = courseId;  
 this.isPrereq1Elective = isPrereq1Elective;  
 this.isPrereq2Elective = isPrereq2Elective;  
 this.prerequisiteNo1 = prerequisiteNo1;  
 this.prerequisiteNo2 = prerequisiteNo2;  
 }  
   
 public CoursePrerequisites() {  
   
 }  
   
 @Id  
 @Column(name="course\_id")  
 private String courseId;  
   
 @Column(name="is\_prereq\_1\_elective")  
 private int isPrereq1Elective;  
  
 @Column(name="is\_prereq\_2\_elective")  
 private int isPrereq2Elective;  
  
 @Column(name="prereq1")  
 private String prerequisiteNo1;  
  
 @Column(name="prereq2")  
 private String prerequisiteNo2;  
  
 public String getCourseId() {  
 return courseId;  
 }  
  
 public void setCourseId(String courseId) {  
 this.courseId = courseId;  
 }  
  
 public int getIsPrereq1Elective() {  
 return isPrereq1Elective;  
 }  
  
 public void setIsPrereq1Elective(int isPrereq1Elective) {  
 this.isPrereq1Elective = isPrereq1Elective;  
 }  
  
 public int getIsPrereq2Elective() {  
 return isPrereq2Elective;  
 }  
  
 public void setIsPrereq2Elective(int isPrereq2Elective) {  
 this.isPrereq2Elective = isPrereq2Elective;  
 }  
  
 public String getPrerequisiteNo1() {  
 return prerequisiteNo1;  
 }  
  
 public void setPrerequisiteNo1(String prerequisiteNo1) {  
 this.prerequisiteNo1 = prerequisiteNo1;  
 }  
  
 public String getPrerequisiteNo2() {  
 return prerequisiteNo2;  
 }  
  
 public void setPrerequisiteNo2(String prerequisiteNo2) {  
 this.prerequisiteNo2 = prerequisiteNo2;  
 }  
}

# spring\model\CurrentTimeSlots.java

package com.qrms.spring.model;  
  
import java.sql.Date;  
import java.sql.Time;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.ManyToOne;  
import javax.persistence.Table;  
  
  
@Entity  
@Table(name="current\_time\_slots")  
public class CurrentTimeSlots {  
  
 @Id  
 @Column(name="id")  
 private int id;  
   
 @ManyToOne(fetch=FetchType.LAZY)  
 @JoinColumn(name="resource\_id")  
 private Resource resourceId;  
   
 @Column(name="start\_time")  
 private Time startTime;  
   
 @Column(name="end\_time")  
 private Time endTime;  
   
 @Column(name="activity")  
 private String activity;  
   
 @ManyToOne(fetch=FetchType.LAZY)  
 @JoinColumn(name="activity\_incharge")  
 private Users activityIncharge;  
   
 @Column(name="date")  
 private Date date;  
   
 @Column(name="day")  
 private String day;  
  
 public Resource getResourceId() {  
 return resourceId;  
 }  
  
 public void setResourceId(Resource resourceId) {  
 this.resourceId = resourceId;  
 }  
  
 public Time getStartTime() {  
 return startTime;  
 }  
  
 public void setStartTime(Time startTime) {  
 this.startTime = startTime;  
 }  
  
 public Time getEndTime() {  
 return endTime;  
 }  
  
 public void setEndTime(Time endTime) {  
 this.endTime = endTime;  
 }  
  
 public String getActivity() {  
 return activity;  
 }  
  
 public void setActivity(String activity) {  
 this.activity = activity;  
 }  
  
 public Users getActivityIncharge() {  
 return activityIncharge;  
 }  
  
 public void setActivityIncharge(Users activityIncharge) {  
 this.activityIncharge = activityIncharge;  
 }  
  
 public Date getDate() {  
 return date;  
 }  
  
 public void setDate(Date date) {  
 this.date = date;  
 }  
  
 public String getDay() {  
 return day;  
 }  
  
 public void setDay(String day) {  
 this.day = day;  
 }  
  
 public CurrentTimeSlots(Resource resourceId, Time startTime, Time endTime, String activity, Users activityIncharge,  
 Date date, String day) {  
 super();  
 this.resourceId = resourceId;  
 this.startTime = startTime;  
 this.endTime = endTime;  
 this.activity = activity;  
 this.activityIncharge = activityIncharge;  
 this.date = date;  
 this.day = day;  
 }  
   
 public CurrentTimeSlots() {  
   
 }  
}

# spring\model\CustomUserDetails.java

package com.qrms.spring.model;  
  
import java.util.Collection;  
import java.util.stream.Collectors;  
  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
  
public class CustomUserDetails extends Users implements UserDetails {  
  
   
 /\*\*  
 \*   
 \*/  
 private static final long serialVersionUID = 1L;  
  
 public CustomUserDetails(final Users users) {  
 super(users);  
   
   
 }  
   
 @Override  
 public Collection<? extends GrantedAuthority> getAuthorities() {  
   
 return getRoles()  
 .stream()  
 .map(role -> new SimpleGrantedAuthority("ROLE\_" + role.getRole()))  
 .collect(Collectors.toList());  
   
 }  
  
 @Override  
 public String getPassword() {  
 return super.getPassword();  
 }  
  
 @Override  
 public String getUsername() {  
 return super.getFirstName();  
 }  
  
 @Override  
 public boolean isAccountNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isAccountNonLocked() {  
 return true;  
 }  
  
 @Override  
 public boolean isCredentialsNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isEnabled() {  
 return true;  
 }  
  
}

# spring\model\Department.java

package com.qrms.spring.model;  
  
import java.util.HashSet;  
import java.util.Set;  
  
import javax.persistence.CascadeType;  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.Id;  
import javax.persistence.OneToMany;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="department")  
public class Department {  
   
 @Id  
 @Column(name="dept\_id")  
 private String deptId;  
   
 @Column(name="dept\_name")  
 private String deptName;  
  
 //Parent of FK relation to Course -- Department update/deletion cascades to courses  
 @OneToMany(mappedBy="department",cascade=CascadeType.ALL)  
 Set<Course> courses = new HashSet<Course>();  
   
   
 public String getDeptId() {  
 return deptId;  
 }  
  
 public void setDeptId(String dept\_id) {  
 this.deptId = dept\_id;  
 }  
  
 public String getDeptName() {  
 return deptName;  
 }  
   
 public void setDeptName(String deptName) {  
 this.deptName = deptName;  
 }  
   
 @OneToMany(mappedBy = "department", cascade = CascadeType.ALL)  
 Set<Resource> resources = new HashSet<Resource>();  
   
 @OneToMany(mappedBy = "department", cascade = CascadeType.ALL)  
 Set<TimeTable> timetable = new HashSet<TimeTable>();  
   
}

# spring\model\DesignationToHours.java

package com.qrms.spring.model;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.Id;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="designation\_to\_hours")  
public class DesignationToHours {  
 @Id  
 @Column(name="designation")  
 private String designation;  
   
 @Column(name="min\_limit")  
 private int minLimit;  
  
 @Column(name="max\_limit")  
 private int maxLimit;  
  
 public String getDesignation() {  
 return designation;  
 }  
  
 public void setDesignation(String designation) {  
 this.designation = designation;  
 }  
  
 public int getMinLimit() {  
 return minLimit;  
 }  
  
 public void setMinLimit(int minLimit) {  
 this.minLimit = minLimit;  
 }  
  
 public int getMaxLimit() {  
 return maxLimit;  
 }  
  
 public void setMaxLimit(int maxLimit) {  
 this.maxLimit = maxLimit;  
 }  
}

# spring\model\Divisions.java

package com.qrms.spring.model;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.ManyToOne;  
import javax.persistence.Table;  
  
//year, dept, noOfbatches, divName, divId  
@Entity  
@Table(name="divisions")  
public class Divisions {  
   
 @Id  
 @Column(name="div\_id")  
 String divId;  
   
 @Column(name = "year")   
 String year;  
   
 // Child of FK relation to Department -- do not cascade on delete/update  
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "dept\_id")  
 private Department department;  
   
 @Column(name = "no\_of\_batches")  
 private Integer noOfBatches;  
   
 @Column(name = "div\_name")  
 private Character divName;  
  
 public Divisions() {  
   
 }  
   
 public String getYear() {  
 return year;  
 }  
  
 public String getDivId() {  
 return divId;  
 }  
  
 public void setDivId(String divId) {  
 this.divId = divId;  
 }  
  
 public void setDivName(char divName) {  
 this.divName = divName;  
 }  
  
 public void setYear(String year) {  
 this.year = year;  
 }  
  
 public Department getDepartment() {  
 return department;  
 }  
  
 public void setDepartment(Department department) {  
 this.department = department;  
 }  
  
 public Integer getNoOfBatches() {  
 return noOfBatches;  
 }  
  
 public void setNoOfBatches(Integer noOfBatches) {  
 this.noOfBatches = noOfBatches;  
 }  
  
 public Character getDivName() {  
 return divName;  
 }  
  
 public void setDivName(Character divName) {  
 this.divName = divName;  
 }  
  
}

# spring\model\ElectiveBatches.java

package com.qrms.spring.model;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.ManyToOne;  
import javax.persistence.Table;  
  
//Table to store active electives used for faculty allocation  
@Entity  
@Table(name="elective\_batches")  
public class ElectiveBatches {  
  
 @Id  
 @Column(name="batch\_id")  
 private String batchId;  
  
 @Column(name = "elective\_id")  
 String electiveId;  
   
 @Column(name = "year")  
 String year;  
   
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "dept\_id")  
 private Department department;  
  
 public String getBatchId() {  
 return batchId;  
 }  
  
 public void setBatchId(String batchId) {  
 this.batchId = batchId;  
 }  
  
 public String getElectiveId() {  
 return electiveId;  
 }  
  
 public void setElectiveId(String electiveId) {  
 this.electiveId = electiveId;  
 }  
  
 public String getYear() {  
 return year;  
 }  
  
 public void setYear(String year) {  
 this.year = year;  
 }  
  
 public Department getDepartment() {  
 return department;  
 }  
  
 public void setDepartment(Department department) {  
 this.department = department;  
 }  
  
 public ElectiveBatches() {  
 super();  
 }  
   
   
  
}

# spring\model\Electives.java

package com.qrms.spring.model;  
  
import java.util.HashSet;  
import java.util.Set;  
  
import javax.persistence.CascadeType;  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.ManyToOne;  
import javax.persistence.OneToMany;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="electives")  
public class Electives {  
   
 public Electives(String electiveCourseId, Course course, String electiveName) {  
 super();  
 this.electiveCourseId = electiveCourseId;  
 this.electiveName = electiveName;  
 this.course = course;  
 }  
   
 public Electives() {  
   
 }  
   
 @Id  
 @Column(name="elective\_course\_id")  
 private String electiveCourseId;  
  
 @Column(name="elective\_name")  
 private String electiveName;  
   
 //Child (owner) of FK relation to Course -- do not cascade on delete/update   
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "course\_id")  
 private Course course;  
   
 public Set<StudentAllocCourse> getStudentAllocs() {  
 return studentAllocs;  
 }  
  
 public void setStudentAllocs(Set<StudentAllocCourse> studentAllocs) {  
 this.studentAllocs = studentAllocs;  
 }  
  
 //Parent of FK relation to StudentPrefs -- on update/delete cascade  
 @OneToMany(mappedBy="elective",cascade=CascadeType.ALL)  
 Set<StudentPref> studentPrefs = new HashSet<StudentPref>();  
   
 //Parent of FK relation to StudentAllocs -- on update/delete cascade  
 @OneToMany(mappedBy="elective",cascade=CascadeType.ALL)  
 Set<StudentAllocCourse> studentAllocs = new HashSet<StudentAllocCourse>();  
   
 public Course getCourse() {  
 return course;  
 }  
  
 public void setCourse(Course course) {  
 this.course = course;  
 }  
  
 public void setElectiveName(String electiveName) {  
 this.electiveName = electiveName;  
 }  
  
 public String getElectiveCourseId() {  
 return electiveCourseId;  
 }  
  
 public void setElectiveCourseId(String electiveCourseId) {  
 this.electiveCourseId = electiveCourseId;  
 }  
  
 public String getElectiveName() {  
 return electiveName;  
 }  
  
 public void setElectiveCourseName(String electiveName) {  
 this.electiveName = electiveName;  
 }  
   
 public Set<StudentPref> getStudentPrefs() {  
 return studentPrefs;  
 }  
  
 public void setStudentPrefs(Set<StudentPref> studentPrefs) {  
 this.studentPrefs = studentPrefs;  
 }  
}

# spring\model\ElectiveVacancyPrefCounts.java

package com.qrms.spring.model;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.Id;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="student\_elective\_vacancy\_pref\_counts")  
public class ElectiveVacancyPrefCounts implements Comparable<ElectiveVacancyPrefCounts>{  
  
 public ElectiveVacancyPrefCounts(String courseId, String electiveId, int vacancyCount, int prefCount) {  
 super();  
 this.courseId = courseId;  
 this.electiveId = electiveId;  
 this.vacancyCount = vacancyCount;  
 this.prefCount = prefCount;  
 }  
  
 public ElectiveVacancyPrefCounts() {  
   
 }  
   
 @Override  
 public int compareTo(ElectiveVacancyPrefCounts e) {   
  
 return (this.getPrefCount() > e.getPrefCount() ? -1 :   
  
 (this.getPrefCount() == e.getPrefCount() ? 0 : 1));   
  
 }  
   
 @Id  
 @Column(name="elective\_id")  
 private String electiveId;  
   
 @Column(name="vacancy\_count")  
 private int vacancyCount;  
   
 @Column(name="pref\_count")  
 private int prefCount;   
   
 @Column(name="course\_id")  
 private String courseId;  
   
 public String getCourseId() {  
 return courseId;  
 }  
  
 public void setCourseId(String courseId) {  
 this.courseId = courseId;  
 }  
  
 public String getElectiveId() {  
 return electiveId;  
 }  
  
 public void setElectiveId(String electiveId) {  
 this.electiveId = electiveId;  
 }  
  
 public int getVacancyCount() {  
 return vacancyCount;  
 }  
  
 public void setVacancyCount(int vacancyCount) {  
 this.vacancyCount = vacancyCount;  
 }  
  
 public int getPrefCount() {  
 return prefCount;  
 }  
  
 public void setPrefCount(int prefCount) {  
 this.prefCount = prefCount;  
 }  
}

# spring\model\FacultyAcad.java

package com.qrms.spring.model;  
  
  
import java.util.HashSet;  
import java.util.Set;  
  
import javax.persistence.CascadeType;  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.ManyToOne;  
import javax.persistence.OneToMany;  
import javax.persistence.OneToOne;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="faculty\_acad")  
public class FacultyAcad {  
  
 public FacultyAcad(String userName, Department department, double yearsOfExperience, String qualification,  
 String designation) {  
 super();  
 this.userName = userName;  
 this.department = department;  
 this.yearsOfExperience = yearsOfExperience;  
 this.qualification = qualification;  
 this.designation = designation;  
 }  
  
 public FacultyAcad() {  
 }  
   
 @Id  
 @Column(name="user\_name")  
 private String userName;  
   
 @ManyToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)  
 @JoinColumn(name = "dept\_id")  
 private Department department;  
   
 @Column(name="years\_of\_experience")  
 private Double yearsOfExperience;  
   
 @Column(name="qualification")  
 private String qualification;  
   
 @Column(name="designation")  
 private String designation;  
  
 @OneToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)  
 @JoinColumn(name="user\_dets")  
 private Users userDets;  
   
 @OneToMany(mappedBy = "slotIncharge",cascade = CascadeType.ALL)  
 Set<TimeSlots> timeSlots = new HashSet<TimeSlots>();  
   
 @OneToMany(mappedBy = "slotIncharge",cascade = CascadeType.ALL)  
 Set<TimeTable> timeTableSlots = new HashSet<TimeTable>();  
   
 @OneToMany(mappedBy = "requestBy",cascade = CascadeType.ALL)  
 Set<ResourceRequests> requests = new HashSet<ResourceRequests>();  
   
 public Users getUserDets() {  
 return userDets;  
 }  
  
 public void setUserDets(Users userDets) {  
 this.userDets = userDets;  
 }  
   
 public String getUserName() {  
 return userName;  
 }  
  
 public void setUserName(String userName) {  
 this.userName = userName;  
 }  
  
 public Department getDepartment() {  
 return department;  
 }  
  
 public void setDepartment(Department department) {  
 this.department = department;  
 }  
  
 public Double getYearsOfExperience() {  
 return yearsOfExperience;  
 }  
  
 public void setYearsOfExperience(Double yearsOfExperience) {  
 this.yearsOfExperience = yearsOfExperience;  
 }  
  
 public String getQualification() {  
 return qualification;  
 }  
  
 public void setQualification(String qualification) {  
 this.qualification = qualification;  
 }  
  
 public String getDesignation() {  
 return designation;  
 }  
  
 public void setDesignation(String designation) {  
 this.designation = designation;  
 }  
   
   
   
}

# spring\model\FacultyAllocCourse.java

package com.qrms.spring.model;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="faculty\_alloc\_course")  
public class FacultyAllocCourse {  
  
 public FacultyAllocCourse() {  
   
 }  
   
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 int id;  
   
 @Column(name = "user\_name")  
 String userName;  
   
 @Column(name = "prefno")   
 int prefNo;  
   
 @Column(name = "course\_id")  
 String courseId;  
   
 @Column(name = "no\_of\_hrs")  
 int noOfHours;  
   
 @Column(name = "is\_elective")  
 int isElective;  
   
 @Column(name = "year")  
 String year;  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getUserName() {  
 return userName;  
 }  
  
 public void setUserName(String userName) {  
 this.userName = userName;  
 }  
  
 public int getPrefNo() {  
 return prefNo;  
 }  
  
 public void setPrefNo(int prefNo) {  
 this.prefNo = prefNo;  
 }  
  
 public String getCourseId() {  
 return courseId;  
 }  
  
 public void setCourseId(String courseId) {  
 this.courseId = courseId;  
 }  
  
 public int getNoOfHours() {  
 return noOfHours;  
 }  
  
 public void setNoOfHours(int noOfHours) {  
 this.noOfHours = noOfHours;  
 }  
  
 public int getIsElective() {  
 return isElective;  
 }  
  
 public void setIsElective(int isElective) {  
 this.isElective = isElective;  
 }  
  
 public String getYear() {  
 return year;  
 }  
  
 public void setYear(String year) {  
 this.year = year;  
 }  
  
 public FacultyAllocCourse(String userName, int prefNo, String courseId, int noOfHours, int isElective,  
 String year) {  
 super();  
 this.userName = userName;  
 this.prefNo = prefNo;  
 this.courseId = courseId;  
 this.noOfHours = noOfHours;  
 this.isElective = isElective;  
 this.year = year;  
 }  
   
   
   
   
}

# spring\model\FacultyAllotedHours.java

package com.qrms.spring.model;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.Id;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="faculty\_alloted\_hours")  
public class FacultyAllotedHours {  
   
 @Id  
 @Column(name="faculty\_id")  
 private String facultyId;  
   
 @Column(name="practical\_hours")  
 private int practicalHours;  
   
 @Column(name="theory\_hours")  
 private int theoryHours;  
   
 @Column(name="max\_hours")  
 private int maxHours;  
   
 @Column(name="alloted\_hours")  
 private int allotedHours;  
  
 public String getFacultyId() {  
 return facultyId;  
 }  
  
 public void setFacultyId(String faculty) {  
 this.facultyId = faculty;  
 }  
  
 public int getPracticalHours() {  
 return practicalHours;  
 }  
  
 public void setPracticalHours(int practicalHours) {  
 this.practicalHours = practicalHours;  
 }  
  
 public int getTheoryHours() {  
 return theoryHours;  
 }  
  
 public void setTheoryHours(int theoryHours) {  
 this.theoryHours = theoryHours;  
 }  
  
 public int getMaxHours() {  
 return maxHours;  
 }  
  
 public void setMaxHours(int maxHours) {  
 this.maxHours = maxHours;  
 }  
  
 public int getAllotedHours() {  
 return allotedHours;  
 }  
  
 public void setAllotedHours(int allotedHours) {  
 this.allotedHours = allotedHours;  
 }  
  
 public FacultyAllotedHours(String facultyId, int practicalHours, int theoryHours, int maxHours, int allotedHours) {  
 super();  
 this.facultyId = facultyId;  
 this.practicalHours = practicalHours;  
 this.theoryHours = theoryHours;  
 this.maxHours = maxHours;  
 this.allotedHours = allotedHours;  
 }  
   
 public FacultyAllotedHours() {  
   
 }  
   
   
}

# spring\model\FacultyCourseHistory.java

package com.qrms.spring.model;  
  
import javax.persistence.CascadeType;  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.ManyToOne;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="faculty\_course\_history")  
public class FacultyCourseHistory {  
   
 public FacultyCourseHistory(FacultyAcad facultyUserName, Course courseTaught,int courseExperience) {  
 super();  
 this.facultyUserName = facultyUserName;  
 this.courseTaught = courseTaught;  
 this.courseExperience = courseExperience;  
 }  
  
 public FacultyCourseHistory() {  
   
 }  
   
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name="id")  
 private int id;  
   
 @ManyToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)  
 @JoinColumn(name="user\_name")  
 private FacultyAcad facultyUserName;  
   
 @ManyToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)  
 @JoinColumn(name="course\_taught")  
 private Course courseTaught;  
   
 @Column(name="course\_experience")  
 private int courseExperience;  
   
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public int getCourseExperience() {  
 return courseExperience;  
 }  
  
 public void setCourseExperience(int courseExperience) {  
 this.courseExperience = courseExperience;  
 }  
  
 public FacultyAcad getFacultyUserName() {  
 return facultyUserName;  
 }  
  
 public void setFacultyUserName(FacultyAcad facultyUserName) {  
 this.facultyUserName = facultyUserName;  
 }  
  
 public Course getCourseTaught() {  
 return courseTaught;  
 }  
  
 public void setCourseTaught(Course courseTaught) {  
 this.courseTaught = courseTaught;  
 }  
   
}

# spring\model\FacultyPref.java

package com.qrms.spring.model;  
  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.Table;  
  
  
@Entity  
@Table(name="faculty\_pref")  
public class FacultyPref{  
   
 public FacultyPref(String userName, String courseId, String electiveId, int prefNo, Integer courseExp,  
 Integer prereq1Exp, Integer prereq2Exp, String year) {  
 super();  
 this.userName = userName;  
 this.courseId = courseId;  
 this.electiveId = electiveId;  
 this.prefNo = prefNo;  
 this.courseExp = courseExp;  
 this.prereq1Exp = prereq1Exp;  
 this.prereq2Exp = prereq2Exp;  
 this.year = year;  
 }  
   
 public FacultyPref() {  
   
 }  
   
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name="id")  
 private int id;  
   
 @Column(name="user\_name")  
 private String userName;  
   
 //course id  
 @Column(name="course\_id")  
 private String courseId;  
   
 @Column(name="elective\_id")  
 private String electiveId;  
   
 @Column(name="pref\_no")  
 private int prefNo;  
  
 //no of times taught  
 @Column(name="course\_exp")  
 private Integer courseExp;  
   
 //no of times prerequisite1 taught  
 @Column(name="prereq1\_exp")  
 private Integer prereq1Exp;  
   
 //no of times prerequisite2 taught  
 @Column(name="prereq2\_exp")  
 private Integer prereq2Exp;  
   
 //year  
 @Column(name="year")  
 private String year;  
   
 public String getYear() {  
 return year;  
 }  
  
 public void setYear(String year) {  
 this.year = year;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getUserName() {  
 return userName;  
 }  
  
 public void setUserName(String userName) {  
 this.userName = userName;  
 }  
  
 public String getCourseId() {  
 return courseId;  
 }  
  
 public void setCourseId(String courseId) {  
 this.courseId = courseId;  
 }  
  
 public String getElectiveId() {  
 return electiveId;  
 }  
  
 public void setElectiveId(String elective) {  
 this.electiveId = elective;  
 }  
  
 public int getPrefNo() {  
 return prefNo;  
 }  
  
 public void setPrefNo(int prefNo) {  
 this.prefNo = prefNo;  
 }  
  
 public Integer getCourseExp() {  
 return courseExp;  
 }  
  
 public void setCourseExp(Integer courseExp) {  
 this.courseExp = courseExp;  
 }  
  
 public Integer getPrereq1Exp() {  
 return prereq1Exp;  
 }  
  
 public void setPrereq1Exp(Integer prereq1Exp) {  
 this.prereq1Exp = prereq1Exp;  
 }  
  
 public Integer getPrereq2Exp() {  
 return prereq2Exp;  
 }  
  
 public void setPrereq2Exp(Integer prereq2Exp) {  
 this.prereq2Exp = prereq2Exp;  
 }  
  
// @Override  
// public int compareTo(FacultyPref s) {  
// // TODO Auto-generated method stub  
// return (this.getPrefNo() > s.getPrefNo() ? -1 :   
//  
// (this.getPrefNo() == s.getPrefNo() ? 0 : 1));  
// }  
  
   
   
}

# spring\model\OldTimeSlots.java

package com.qrms.spring.model;  
  
import java.sql.Date;  
import java.sql.Time;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.ManyToOne;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="old\_time\_slots")  
public class OldTimeSlots {  
   
 @Id  
 @Column(name="id")  
 private int id;  
   
 @ManyToOne(fetch=FetchType.LAZY)  
 @JoinColumn(name="resource\_id")  
 private Resource resourceId;  
   
 @Column(name="start\_time")  
 private Time startTime;  
   
 @Column(name="end\_time")  
 private Time endTime;  
   
 @Column(name="date")  
 private Date date;  
   
 @Column(name="day")  
 private String day;  
  
 @Column(name="activity")  
 private String activity;  
   
 @ManyToOne(fetch=FetchType.LAZY)  
 @JoinColumn(name="activity\_incharge")  
 private Users activityIncharge;  
   
 public Users getActivityIncharge() {  
 return activityIncharge;  
 }  
  
 public void setActivityIncharge(Users activityIncharge) {  
 this.activityIncharge = activityIncharge;  
 }  
  
 public String getActivity() {  
 return activity;  
 }  
  
 public void setActivity(String activity) {  
 this.activity = activity;  
 }  
  
 public Resource getResourceId() {  
 return resourceId;  
 }  
  
 public void setResourceId(Resource resourceId) {  
 this.resourceId = resourceId;  
 }  
  
 public Time getStartTime() {  
 return startTime;  
 }  
  
 public void setStartTime(Time startTime) {  
 this.startTime = startTime;  
 }  
  
 public Time getEndTime() {  
 return endTime;  
 }  
  
 public void setEndTime(Time endTime) {  
 this.endTime = endTime;  
 }  
  
 public Date getDate() {  
 return date;  
 }  
  
 public void setDate(Date date) {  
 this.date = date;  
 }  
  
 public String getDay() {  
 return day;  
 }  
  
 public void setDay(String day) {  
 this.day = day;  
 }  
  
 public OldTimeSlots(Resource resourceId, Time startTime, Time endTime, Date date, String day,String activity,Users activityIncharge) {  
 super();  
 this.resourceId = resourceId;  
 this.startTime = startTime;  
 this.endTime = endTime;  
 this.date = date;  
 this.day = day;  
 this.activity = activity;  
 this.activityIncharge = activityIncharge;  
 }  
   
 public OldTimeSlots(){  
   
 }  
}

# spring\model\OpenFacultyPrefs.java

package com.qrms.spring.model;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.Id;  
import javax.persistence.Table;  
  
@Entity  
@Table(name = "open\_faculty\_prefs")  
public class OpenFacultyPrefs {  
  
 @Id  
 @Column(name="dept\_id")  
 private String deptId;  
   
 //0 -- even 1 -- odd  
 @Column(name="sem\_type")  
 int semType;  
  
 @Column(name="status")  
 int status;  
   
 public String getDeptId() {  
 return deptId;  
 }  
  
 public void setDeptId(String deptId) {  
 this.deptId = deptId;  
 }  
  
 public int getSemType() {  
 return semType;  
 }  
  
 public void setSemType(int semType) {  
 this.semType = semType;  
 }  
  
 public int getStatus() {  
 return status;  
 }  
  
 public void setStatus(int status) {  
 this.status = status;  
 }  
  
   
 public OpenFacultyPrefs() {  
 super();  
 }  
  
   
}

# spring\model\PasswordResetToken.java

package com.qrms.spring.model;  
  
import java.util.Calendar;  
import java.util.Date;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.OneToOne;  
import javax.persistence.Temporal;  
import javax.persistence.TemporalType;  
  
@Entity  
public class PasswordResetToken {  
   
 public PasswordResetToken(PasswordResetToken passwordResetToken) {  
 this.expiryDate = passwordResetToken.expiryDate;  
 this.id = passwordResetToken.id;  
 this.token = passwordResetToken.token;  
 this.user = passwordResetToken.user;  
 }  
   
 public PasswordResetToken() {  
   
 }  
   
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 private Long id;  
   
 private String token;  
   
 @OneToOne(targetEntity = Users.class, fetch = FetchType.EAGER)  
 @JoinColumn(nullable = false, name = "user\_id")  
 private Users user;  
   
 @Temporal(TemporalType.TIMESTAMP)  
 private Date expiryDate;  
 public Long getId() {  
 return id;  
 }  
  
 public void setId(Long id) {  
 this.id = id;  
 }  
  
 public String getToken() {  
 return token;  
 }  
  
 public void setToken(String token) {  
 this.token = token;  
 }  
  
 public Users getUser() {  
 return user;  
 }  
  
 public void setUser(Users user) {  
 this.user = user;  
 }  
  
 public Date getExpiryDate() {  
 return expiryDate;  
 }  
  
 public void setExpiryDate(Date expiryDate) {  
 this.expiryDate = expiryDate;  
 }  
  
 public PasswordResetToken(String token, Users user) {  
 // TODO getDATE AND TIME both  
   
 this.user = user;  
 this.token = token;  
 Calendar c = Calendar.getInstance();  
 c.add(Calendar.DATE, 1);  
 Date date = new Date(c.getTime().getTime());  
 this.expiryDate = date;  
   
 }  
}

# spring\model\PracticalList.java

package com.qrms.spring.model;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="practical\_list\_faculty")  
public class PracticalList {  
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name="b\_id")  
 private int bId;  
   
 //eg: BECOB1, BECOB2  
 @Column(name="lab\_id")  
 private String labId;  
   
 //BECOB, BECOA..  
 @Column(name="div\_id")  
 private String divId;  
   
 public PracticalList(String labId, String practicalCourseId, String theoryCourseId, String facultyId,  
 int noOfHours) {  
 super();  
 this.labId = labId;  
 this.practicalCourseId = practicalCourseId;  
 this.theoryCourseId = theoryCourseId;  
 this.facultyId = facultyId;  
 this.noOfHours = noOfHours;  
 }  
  
 @Column(name="practical\_course\_id")  
 private String practicalCourseId;  
   
 @Column(name="theory\_course\_id")  
 private String theoryCourseId;  
   
 @Column(name="faculty\_id")  
 private String facultyId;  
  
 public int getbId() {  
 return bId;  
 }  
  
 public void setbId(int bId) {  
 this.bId = bId;  
 }  
  
 public String getDivId() {  
 return divId;  
 }  
  
 public void setDivId(String divId) {  
 this.divId = divId;  
 }  
  
 @Column(name="no\_of\_hours")  
 private int noOfHours;  
   
   
 public String getTheoryCourseId() {  
 return theoryCourseId;  
 }  
  
 public void setTheoryCourseId(String theoryCourseId) {  
 this.theoryCourseId = theoryCourseId;  
 }  
  
 public PracticalList() {  
 super();  
 }  
  
 public int getNoOfHours() {  
 return noOfHours;  
 }  
  
 public void setNoOfHours(int noOfHours) {  
 this.noOfHours = noOfHours;  
 }  
  
 public int getBId() {  
 return bId;  
 }  
  
 public void setBId(int bId) {  
 this.bId = bId;  
 }  
  
 public String getLabId() {  
 return labId;  
 }  
  
 public void setLabId(String labId) {  
 this.labId = labId;  
 }  
  
  
  
 public String getPracticalCourseId() {  
 return practicalCourseId;  
 }  
  
 public void setPracticalCourseId(String practicalCourseId) {  
 this.practicalCourseId = practicalCourseId;  
 }  
  
 public String getFacultyId() {  
 return facultyId;  
 }  
  
 public void setFacultyId(String facultyId) {  
 this.facultyId = facultyId;  
 }  
   
}

# spring\model\Resource.java

package com.qrms.spring.model;  
  
import java.util.HashSet;  
import java.util.Set;  
  
import javax.persistence.CascadeType;  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.ManyToOne;  
import javax.persistence.OneToMany;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="resource")  
public class Resource {  
 @Id  
 @Column(name="resource\_id")  
 private String resourceId;  
   
 @Column(name="resource\_name")  
 private String resourceName;  
   
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name="dept")  
 private Department department;  
   
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name="resource\_incharge")  
 private FacultyAcad resourceIncharge;  
   
 @Column(name="resource\_capacity")  
 private Integer resourceCapacity;  
   
 @Column(name="resource\_type")  
 private String resourceType;  
   
 @Column(name="resource\_info")  
 private String resourceInfo;  
   
 @OneToMany(mappedBy = "resourceId",cascade = CascadeType.ALL)  
 Set<ResourceRequests> requests = new HashSet<ResourceRequests>();  
   
 public Resource() {  
 // TODO Auto-generated constructor stub  
 }  
   
 public Resource(String resourceId, String resourceName, Department department, FacultyAcad resourceIncharge,  
 Integer resourceCapacity,String resourceType,String resourceInfo) {  
 super();  
 this.resourceId = resourceId;  
 this.resourceName = resourceName;  
 this.department = department;  
 this.resourceIncharge = resourceIncharge;  
 this.resourceCapacity = resourceCapacity;  
 this.resourceType = resourceType;  
 this.resourceInfo = resourceInfo;  
 }  
  
 public String getResourceInfo() {  
 return resourceInfo;  
 }  
  
 public void setResourceInfo(String resourceInfo) {  
 this.resourceInfo = resourceInfo;  
 }  
  
 public String getResourceType() {  
 return resourceType;  
 }  
  
 public void setResourceType(String resourceType) {  
 this.resourceType = resourceType;  
 }  
  
 public Integer getResourceCapacity() {  
 return resourceCapacity;  
 }  
  
 public void setResourceCapacity(Integer resourceCapacity) {  
 this.resourceCapacity = resourceCapacity;  
 }  
  
 @OneToMany(mappedBy = "resourceId", cascade = CascadeType.ALL)  
 Set<TimeSlots> timeSlots = new HashSet<TimeSlots>();  
   
 @OneToMany(mappedBy = "resourceId", cascade = CascadeType.ALL)  
 Set<OldTimeSlots> oldTimeSlots = new HashSet<OldTimeSlots>();  
  
 public String getResourceId() {  
 return resourceId;  
 }  
  
 public void setResourceId(String resourceId) {  
 this.resourceId = resourceId;  
 }  
  
 public String getResourceName() {  
 return resourceName;  
 }  
  
 public void setResourceName(String resourceName) {  
 this.resourceName = resourceName;  
 }  
  
 public Department getDepartment() {  
 return department;  
 }  
  
 public void setDepartment(Department department) {  
 this.department = department;  
 }  
  
 public FacultyAcad getResourceIncharge() {  
 return resourceIncharge;  
 }  
  
 public void setResourceIncharge(FacultyAcad resourceIncharge) {  
 this.resourceIncharge = resourceIncharge;  
 }  
   
}

# spring\model\ResourceRequests.java

package com.qrms.spring.model;  
  
import java.sql.Date;  
import java.sql.Time;  
  
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="resource\_requests")  
public class ResourceRequests {  
   
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name="request\_id")  
 private int requestId;  
   
 @ManyToOne  
 @JoinColumn(name="resource\_id")  
 private Resource resourceId;  
   
 @ManyToOne  
 @JoinColumn(name="request\_by")  
 private FacultyAcad requestBy;  
   
 @Column(name="slot\_date")  
 private Date slotDate;  
   
 @Column(name="slot\_start\_time")  
 private Time slotStartTime;  
   
 @Column(name="slot\_end\_time")  
 private Time slotEndTime;  
   
 @Column(name="slot\_activity\_name")  
 private String slotActivityName;   
   
 @Column(name="request\_date")  
 private Date requestedDate;  
   
 @Column(name="request\_time")  
 private Time requestTime;  
   
 @Column(name="slot\_day")  
 private String slotDay;  
   
 public int getRequestId() {  
 return requestId;  
 }  
  
 public void setRequestId(int requestId) {  
 this.requestId = requestId;  
 }  
  
 public Resource getResourceId() {  
 return resourceId;  
 }  
  
 public void setResourceId(Resource resourceId) {  
 this.resourceId = resourceId;  
 }  
  
 public FacultyAcad getRequestBy() {  
 return requestBy;  
 }  
  
 public void setRequestBy(FacultyAcad requestBy) {  
 this.requestBy = requestBy;  
 }  
  
 public Date getSlotDate() {  
 return slotDate;  
 }  
  
 public void setSlotDate(Date slotDate) {  
 this.slotDate = slotDate;  
 }  
  
 public Time getSlotStartTime() {  
 return slotStartTime;  
 }  
  
 public void setSlotStartTime(Time slotStartTime) {  
 this.slotStartTime = slotStartTime;  
 }  
  
 public Time getSlotEndTime() {  
 return slotEndTime;  
 }  
  
 public void setSlotEndTime(Time slotEndTime) {  
 this.slotEndTime = slotEndTime;  
 }  
  
 public String getSlotActivityName() {  
 return slotActivityName;  
 }  
  
 public void setSlotActivityName(String slotActivityName) {  
 this.slotActivityName = slotActivityName;  
 }  
  
 public Date getRequestedDate() {  
 return requestedDate;  
 }  
  
 public void setRequestedDate(Date requestedDate) {  
 this.requestedDate = requestedDate;  
 }  
  
 public Time getRequestTime() {  
 return requestTime;  
 }  
  
 public void setRequestTime(Time requestTime) {  
 this.requestTime = requestTime;  
 }  
  
 public String getSlotDay() {  
 return slotDay;  
 }  
  
 public void setSlotDay(String slotDay) {  
 this.slotDay = slotDay;  
 }  
  
 public ResourceRequests(int requestId,Resource resourceId, FacultyAcad requestBy, Date slotDate, Time slotStartTime,  
 Time slotEndTime, String slotActivityName, Date requestedDate, Time requestTime,String slotDay) {  
 super();  
 this.requestId = requestId;  
 this.resourceId = resourceId;  
 this.requestBy = requestBy;  
 this.slotDate = slotDate;  
 this.slotStartTime = slotStartTime;  
 this.slotEndTime = slotEndTime;  
 this.slotActivityName = slotActivityName;  
 this.requestedDate = requestedDate;  
 this.requestTime = requestTime;  
 this.slotDay = slotDay;  
   
 }  
   
 public ResourceRequests() {  
   
 }  
   
}

# spring\model\Role.java

package com.qrms.spring.model;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="role")  
public class Role {  
  
 public Role(int role\_id, String role) {  
 super();  
 this.role\_id = role\_id;  
 this.role = role;  
 }  
  
 public Role(){  
   
 }  
   
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name="role\_id")  
 private int role\_id;  
   
 @Column(name="role")  
 private String role;  
  
 public int getRole\_id() {  
 return role\_id;  
 }  
  
 public void setRole\_id(int role\_id) {  
 this.role\_id = role\_id;  
 }  
  
 public String getRole() {  
 return role;  
 }  
  
 public void setRole(String role) {  
 this.role = role;  
 }  
}

# spring\model\StudentAcad.java

package com.qrms.spring.model;  
  
import java.util.HashSet;  
import java.util.Set;  
  
import javax.persistence.CascadeType;  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.ManyToOne;  
import javax.persistence.OneToMany;  
import javax.persistence.OneToOne;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="student\_acad")  
public class StudentAcad implements Comparable <StudentAcad>{  
  
  
 public StudentAcad(){  
   
 }  
  
 @Override  
 public int compareTo(StudentAcad s) {   
  
 return (this.getAggMarks() > s.getAggMarks() ? -1 :   
  
 (this.getAggMarks() == s.getAggMarks() ? 0 : 1));   
  
 }  
  
 @Id  
 @Column(name="user\_name")  
 private String userName;  
   
 @OneToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)  
 @JoinColumn(name="user\_dets")  
 private Users userDets;  
  
 @ManyToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)  
 @JoinColumn(name = "dept\_id")  
 private Department department;  
   
 @Column(name="year")  
 private String year;  
   
 @Column(name="semester")  
 private int sem;  
   
 @ManyToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)  
 @JoinColumn(name = "div\_id")  
 private Divisions div;  
   
 @Column(name="shift")  
 private int shift;  
   
 @Column(name="rollno")  
 private String rollno;  
   
 @Column(name="agg\_marks")  
 private Float aggMarks;  
   
 //Parent of FK relation to StudentAlloc -- on update/delete cascade  
 @OneToMany(mappedBy="student",cascade=CascadeType.ALL)  
 Set<StudentAllocCourse> studentAllocs = new HashSet<StudentAllocCourse>();  
   
 public Users getUserDets() {  
 return userDets;  
 }  
  
 public void setUserDets(Users userDets) {  
 this.userDets = userDets;  
 }  
   
 public String getUserName() {  
 return userName;  
 }  
  
 public void setUserName(String userName) {  
 this.userName = userName;  
 }  
  
 public Department getDepartment() {  
 return department;  
 }  
  
 public void setDepartment(Department department) {  
 this.department = department;  
 }  
  
 public String getYear() {  
 return year;  
 }  
  
 public void setYear(String year) {  
 this.year = year;  
 }  
  
 public int getSem() {  
 return sem;  
 }  
  
 public void setSem(int sem) {  
 this.sem = sem;  
 }  
  
 public Divisions getDiv() {  
 return div;  
 }  
  
 public void setDiv(Divisions div) {  
 this.div = div;  
 }  
  
 public int getShift() {  
 return shift;  
 }  
  
 public void setShift(int shift) {  
 this.shift = shift;  
 }  
  
 public String getRollno() {  
 return rollno;  
 }  
  
 public void setRollno(String rollno) {  
 this.rollno = rollno;  
 }  
  
 public Float getAggMarks() {  
 return aggMarks;  
 }  
  
 public void setAggMarks(Float aggMarks) {  
 this.aggMarks = aggMarks;  
 }  
  
 public Set<StudentAllocCourse> getStudentAllocs() {  
 return studentAllocs;  
 }  
  
 public void setStudentAllocs(Set<StudentAllocCourse> studentAllocs) {  
 this.studentAllocs = studentAllocs;  
 }  
   
}

# spring\model\StudentAllocCourse.java

package com.qrms.spring.model;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.JoinColumn;  
import javax.persistence.ManyToOne;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="student\_alloc\_course")  
public class StudentAllocCourse{  
   
 public StudentAllocCourse(int id,String batchId, Course courseId,Electives elective,StudentAcad student,int prefNo ) {  
 super();  
 this.id = id;  
 this.elective = elective;  
 this.courseId = courseId;  
 this.prefNo = prefNo;  
 this.student = student;  
 this.batchId = batchId;  
 }  
 public StudentAllocCourse(Electives elective, Course courseId,StudentAcad student, int prefNo,String batchId) {  
 super();  
 this.elective = elective;  
 this.courseId = courseId;  
 this.prefNo = prefNo;  
 this.student = student;  
 this.batchId = batchId;  
 }  
   
 public StudentAllocCourse() {  
   
 }  
   
 @javax.persistence.Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name="id")  
 private int id;  
   
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "elective\_id")  
 private Electives elective;  
   
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name="course\_Id")  
 private Course courseId;  
   
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "student")  
 private StudentAcad student;  
  
 @Column(name="pref\_no")  
 private int prefNo;  
   
 @Column(name="batch\_id")  
 private String batchId;  
   
 public String getBatchId() {  
 return batchId;  
 }  
  
 public void setBatchId(String batchId) {  
 this.batchId = batchId;  
 }  
  
 public StudentAcad getStudent() {  
 return student;  
 }  
  
 public void setStudent(StudentAcad student) {  
 this.student = student;  
 }  
  
 public Electives getElective() {  
 return elective;  
 }  
  
 public void setElective(Electives elective) {  
 this.elective = elective;  
 }  
  
 public Course getCourseId() {  
 return courseId;  
 }  
  
 public void setCourseId(Course courseId) {  
 this.courseId = courseId;  
 }  
   
 public int getPrefNo() {  
 return prefNo;  
 }  
  
 public void setPrefNo(int prefNo) {  
 this.prefNo = prefNo;  
 }  
   
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
   
}

# spring\model\StudentPref.java

package com.qrms.spring.model;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.ManyToOne;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="student\_pref")  
public class StudentPref{  
  
 public StudentPref(String userName, String courseId, Electives elective, int prefNo) {  
 super();  
 this.userName = userName;  
 this.courseId = courseId;  
 this.elective = elective;  
 this.prefNo = prefNo;  
 }  
  
 public StudentPref() {  
   
 }  
   
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name="id")  
 private int id;  
   
 @Column(name="user\_name")  
 private String userName;  
   
 //course id  
 @Column(name="course\_id")  
 private String courseId;  
   
 @Column(name="pref\_no")  
 private int prefNo;  
  
 //Child (owner) of FK relation to Electives -- do not cascade on delete/update   
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "el\_pref")  
 private Electives elective;  
   
 public String getUserName() {  
 return userName;  
 }  
  
 public void setUserName(String userName) {  
 this.userName = userName;  
 }  
  
 public Electives getElective() {  
 return elective;  
 }  
  
 public void setElective(Electives elective) {  
 this.elective = elective;  
 }  
  
 public int getPrefNo() {  
 return prefNo;  
 }  
  
 public void setPrefNo(int prefNo) {  
 this.prefNo = prefNo;  
 }  
  
 public String getCourseId() {  
 return courseId;  
 }  
  
 public void setCourseId(String courseId) {  
 this.courseId = courseId;  
 }  
  
}

# spring\model\TimeSlots.java

package com.qrms.spring.model;  
  
import java.sql.Date;  
import java.sql.Time;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.ManyToOne;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="time\_slots")  
public class TimeSlots implements Comparable<TimeSlots>{  
   
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name="tid")  
 private int tid;  
   
 @Column(name="start\_time")  
 private Time startTime;  
   
 @Column(name="end\_time")  
 private Time endTime;  
   
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name="resource\_id")  
 private Resource resourceId;  
  
 @Column(name="date")  
 private Date date;  
   
 @Column(name="activity\_name")  
 private String activityName;  
   
 @ManyToOne(fetch=FetchType.LAZY)  
 @JoinColumn(name="slot\_incharge")  
 private FacultyAcad slotIncharge;  
   
 @Column(name="request\_id")  
 private int requestId;  
   
 public String getActivityName() {  
 return activityName;  
 }  
  
 public void setActivityName(String activityName) {  
 this.activityName = activityName;  
 }  
  
 public FacultyAcad getSlotIncharge() {  
 return slotIncharge;  
 }  
  
 public void setSlotIncharge(FacultyAcad slotIncharge) {  
 this.slotIncharge = slotIncharge;  
 }  
  
 public Date getDate() {  
 return date;  
 }  
  
 public void setDate(Date date) {  
 this.date = date;  
 }  
  
 public Time getStartTime() {  
 return startTime;  
 }  
  
 public void setStartTime(Time startTime) {  
 this.startTime = startTime;  
 }  
  
 public Time getEndTime() {  
 return endTime;  
 }  
  
 public void setEndTime(Time endTime) {  
 this.endTime = endTime;  
 }  
  
 public Resource getResourceId() {  
 return resourceId;  
 }  
  
 public void setResourceId(Resource resourceId) {  
 this.resourceId = resourceId;  
 }  
 public int getRequestId() {  
 return requestId;  
 }  
  
 public void setRequestId(int requestId) {  
 this.requestId = requestId;  
 }  
  
 public TimeSlots(Time startTime, Time endTime, Resource resourceId, Date date,FacultyAcad slotIncharge,String activityName,int requestId) {  
 super();  
 this.startTime = startTime;  
 this.endTime = endTime;  
 this.resourceId = resourceId;  
 this.date = date;  
 this.slotIncharge = slotIncharge;  
 this.activityName = activityName;  
 this.requestId = requestId;  
 }  
   
 public TimeSlots() {  
   
 }  
  
 @Override  
 public int compareTo(TimeSlots o) {  
 return (this.getStartTime().before(o.getStartTime()) ? -1 :   
  
 (this.getStartTime().equals(o.getStartTime()) ? 0 : 1));  
 }  
   
}

# spring\model\TimeTable.java

package com.qrms.spring.model;  
  
import java.sql.Time;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.ManyToOne;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="time\_table")  
public class TimeTable {  
   
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name="tid")  
 private int tid;  
   
 //startTime  
 //endTime  
 //roomNo/labNo  
 //seats  
   
 @Column(name="start\_time")  
 private Time startTime;  
   
 @Column(name="end\_time")  
 private Time endTime;  
   
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name="resource\_id")  
 private Resource resourceId;  
  
 @Column(name="day")  
 private String day;  
   
 @ManyToOne(fetch=FetchType.LAZY)  
 @JoinColumn(name="dept")  
 private Department department;  
  
 @Column(name="activity\_name")  
 private String activityName;  
   
 public String getActivityName() {  
 return activityName;  
 }  
  
 public void setActivityName(String activityName) {  
 this.activityName = activityName;  
 }  
  
 @ManyToOne(fetch=FetchType.LAZY)  
 @JoinColumn(name="slot\_incharge")  
 private FacultyAcad slotIncharge;  
   
 public FacultyAcad getSlotIncharge() {  
 return slotIncharge;  
 }  
  
 public void setSlotIncharge(FacultyAcad slotIncharge) {  
 this.slotIncharge = slotIncharge;  
 }  
   
 public Time getStartTime() {  
 return startTime;  
 }  
  
 public void setStartTime(Time startTime) {  
 this.startTime = startTime;  
 }  
  
 public Time getEndTime() {  
 return endTime;  
 }  
  
 public void setEndTime(Time endTime) {  
 this.endTime = endTime;  
 }  
  
 public Resource getResourceId() {  
 return resourceId;  
 }  
  
 public void setResourceId(Resource resourceId) {  
 this.resourceId = resourceId;  
 }  
   
 public String getDay() {  
 return day;  
 }  
  
 public void setDay(String day) {  
 this.day = day;  
 }  
  
 public Department getDepartment() {  
 return department;  
 }  
  
 public void setDepartment(Department department) {  
 this.department = department;  
 }  
   
 public TimeTable() {  
 // TODO Auto-generated constructor stub  
 }  
  
 public TimeTable(Time startTime, Time endTime, Resource resourceId, String day,  
 Department department,FacultyAcad slotIncharge,String activityName) {  
 super();  
 this.startTime = startTime;  
 this.endTime = endTime;  
 this.resourceId = resourceId;  
 this.day = day;  
 this.department = department;  
 this.slotIncharge = slotIncharge;  
 this.activityName = activityName;  
 }  
   
   
}

# spring\model\Users.java

package com.qrms.spring.model;  
  
import java.util.Set;  
  
import javax.persistence.CascadeType;  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.JoinTable;  
import javax.persistence.ManyToMany;  
import javax.persistence.Table;  
  
  
@Entity  
@Table(name="user")  
public class Users {  
   
 public Users(Users user) {  
 this.userName = user.getUserName();  
 this.email = user.getEmail();  
 this.firstName = user.getFirstName();  
 this.lastName = user.getLastName();  
 this.roles = user.getRoles();  
 this.active = user.getActive();  
 this.password = user.getPassword();  
 }  
   
 public Users() {  
   
 }  
   
 public Users(String userName, String firstName, String lastName) {  
 this.userName = userName;  
 this.firstName = firstName;  
 this.lastName = lastName;  
 }  
   
 @Id  
 @Column(name="user\_name")  
 private String userName;  
   
 @Column(name="email")  
 private String email;  
   
 @Column(name="active")  
 private int active;  
   
 @Column(name="first\_name")  
 private String firstName;  
   
 @Column(name="last\_name")  
 private String lastName;  
  
 @Column(name="password")  
 private String password;  
   
 @ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)  
 @JoinTable(name = "user\_role", joinColumns = @JoinColumn(name = "user\_name"))  
 private Set<Role> roles;  
  
 public String getUserName() {  
 return userName;  
 }  
  
 public void setUserName(String user\_id) {  
 this.userName = user\_id;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 public void setEmail(String email) {  
 this.email = email;  
 }  
  
 public int getActive() {  
 return active;  
 }  
  
 public void setActive(int active) {  
 this.active = active;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public void setFirstName(String firstName) {  
 this.firstName = firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public void setLastName(String lastName) {  
 this.lastName = lastName;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
  
 public Set<Role> getRoles() {  
 return roles;  
 }  
  
 public void setRoles(Set<Role> roles) {  
 this.roles = roles;  
 }  
   
 //Causes errors  
// @OneToMany(mappedBy = "resourceIncharge", cascade = CascadeType.ALL)  
// Set<Resource> resources = new HashSet<Resource>();  
//   
//   
// @OneToOne(mappedBy = "userDets", cascade = CascadeType.ALL)  
// StudentAcad student = new StudentAcad();  
//   
// @OneToOne(mappedBy = "userDets", cascade = CascadeType.ALL)  
// FacultyAcad faculty = new FacultyAcad();  
//   
}

# spring\queryBeans\CombinedCourseElective.java

package com.qrms.spring.queryBeans;  
  
public class CombinedCourseElective {  
 String id;  
 int noOfHours;  
 int isElective;  
 int isTheory;  
 String year;  
 int noOfBatches;  
   
 public CombinedCourseElective(String id,int isElective,int noOfHours,String year,int noOfBatches,int isTheory) {  
 this.id = id;  
 this.isElective = isElective;  
 this.year = year;  
 this.noOfHours = noOfHours;  
 this.noOfBatches = noOfBatches;  
 this.isTheory = isTheory;  
   
 }  
  
 public int getIsTheory() {  
 return isTheory;  
 }  
  
 public void setIsTheory(int isTheory) {  
 this.isTheory = isTheory;  
 }  
  
 public int getNoOfBatches() {  
 return noOfBatches;  
 }  
  
 public void setNoOfBatches(int noOfBatches) {  
 this.noOfBatches = noOfBatches;  
 }  
  
 public String getYear() {  
 return year;  
 }  
  
 public void setYear(String year) {  
 this.year = year;  
 }  
  
 public String getId() {  
 return id;  
 }  
  
 public void setId(String id) {  
 this.id = id;  
 }  
  
 public int getNoOfHours() {  
 return noOfHours;  
 }  
  
 public void setNoOfHours(int noOfHours) {  
 this.noOfHours = noOfHours;  
 }  
  
 public int getIsElective() {  
 return isElective;  
 }  
  
 public void setIsElective(int isElective) {  
 this.isElective = isElective;  
 }  
   
}

# spring\queryBeans\CourseAndElectives.java

package com.qrms.spring.queryBeans;  
  
import com.qrms.spring.model.Course;  
import com.qrms.spring.model.Electives;  
  
public class CourseAndElectives {  
   
 private Course course;  
 private Electives elective;   
 private String preReq1,preReq2;  
   
 public CourseAndElectives(Course c){  
 this.course = c;  
 }  
 public CourseAndElectives(Electives e){  
 this.elective=e;  
 }  
   
 public String getPreReq1() {  
 return preReq1;  
 }  
 public void setPreReq1(String preReq1) {  
 this.preReq1 = preReq1;  
 }  
 public String getPreReq2() {  
 return preReq2;  
 }  
 public void setPreReq2(String preReq2) {  
 this.preReq2 = preReq2;  
 }  
 public Course getCourse() {  
 return course;  
 }  
 public void setCourse(Course course) {  
 this.course = course;  
 }  
 public Electives getElective() {  
 return elective;  
 }  
 public void setElective(Electives elective) {  
 this.elective = elective;  
 }  
   
 public CourseAndElectives() {  
   
 }  
}

# spring\queryBeans\ElectiveBatchCount.java

package com.qrms.spring.queryBeans;  
  
public class ElectiveBatchCount {  
 private String electiveId;  
 private Integer noOfBatches;  
 public String getElectiveId() {  
 return electiveId;  
 }  
 public void setElectiveId(String electiveId) {  
 this.electiveId = electiveId;  
 }  
 public Integer getNoOfBatches() {  
 return noOfBatches;  
 }  
 public void setNoOfBatches(Integer noOfBatches) {  
 this.noOfBatches = noOfBatches;  
 }  
 public ElectiveBatchCount(String electiveId, Integer noOfBatches) {  
 super();  
 this.electiveId = electiveId;  
 this.noOfBatches = noOfBatches;  
 }  
   
   
}

# spring\queryBeans\ElectiveBatchCountList.java

package com.qrms.spring.queryBeans;  
  
import java.util.List;  
  
public class ElectiveBatchCountList {  
 private List<ElectiveBatchCount> electiveBatchCounts;  
 private String year,deptId;  
   
 public String getYear() {  
 return year;  
 }  
  
 public void setYear(String year) {  
 this.year = year;  
 }  
  
 public String getDeptId() {  
 return deptId;  
 }  
  
 public void setDeptId(String deptId) {  
 this.deptId = deptId;  
 }  
  
 public void setElectiveBatchCounts(List<ElectiveBatchCount> electiveBatchCounts) {  
 this.electiveBatchCounts = electiveBatchCounts;  
 }  
  
 public List<ElectiveBatchCount> getElectiveBatchCounts() {  
 return electiveBatchCounts;  
 }  
  
}

# spring\queryBeans\FacPrefCountInfo.java

package com.qrms.spring.queryBeans;  
  
public class FacPrefCountInfo {  
  
 String deptName;  
 String semType;  
 int submitCount,totalFacultyCount;  
   
 public String getDeptName() {  
 return deptName;  
 }  
 public void setDeptName(String deptName) {  
 this.deptName = deptName;  
 }  
 public String getSemType() {  
 return semType;  
 }  
 public void setSemType(String semType) {  
 this.semType = semType;  
 }  
 public int getSubmitCount() {  
 return submitCount;  
 }  
 public void setSubmitCount(int submitCount) {  
 this.submitCount = submitCount;  
 }  
 public int getTotalFacultyCount() {  
 return totalFacultyCount;  
 }  
 public void setTotalFacultyCount(int totalFacultyCount) {  
 this.totalFacultyCount = totalFacultyCount;  
 }  
 public FacPrefCountInfo() {  
 }  
}

# spring\queryBeans\FacPrefsList.java

package com.qrms.spring.queryBeans;  
  
import java.util.List;  
  
import com.qrms.spring.model.FacultyPref;  
  
public class FacPrefsList {  
 private List<FacultyPref> facultyPrefs;  
  
 public List<FacultyPref> getFacultyPrefs() {  
 return facultyPrefs;  
 }  
  
 public void setFacultyPrefs(List<FacultyPref> facultyPrefs) {  
 this.facultyPrefs = facultyPrefs;  
 }  
}

# spring\queryBeans\FacultyAllocations.java

package com.qrms.spring.queryBeans;  
  
import java.util.List;  
  
import com.qrms.spring.model.CourseList;  
import com.qrms.spring.model.PracticalList;  
  
public class FacultyAllocations {  
  
 String facultyId,name;  
   
 List<CourseList> courseAndDivs;  
 List<PracticalList> practicalsAndBatches;  
 List<String> courses;  
 List<String> practicals;  
   
 Integer theoryHours,practicalHours,allotedLoad,maxLoad;  
  
 public FacultyAllocations() {  
   
 }  
  
 public List<String> getCourses() {  
 return courses;  
 }  
  
 public void setCourses(List<String> courses) {  
 this.courses = courses;  
 }  
  
 public List<String> getPracticals() {  
 return practicals;  
 }  
  
 public void setPracticals(List<String> practicals) {  
 this.practicals = practicals;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getFacultyId() {  
 return facultyId;  
 }  
  
 public void setFacultyId(String facultyId) {  
 this.facultyId = facultyId;  
 }  
  
   
 public List<CourseList> getCourseAndDivs() {  
 return courseAndDivs;  
 }  
  
 public void setCourseAndDivs(List<CourseList> courseAndDivs) {  
 this.courseAndDivs = courseAndDivs;  
 }  
  
 public List<PracticalList> getPracticalsAndBatches() {  
 return practicalsAndBatches;  
 }  
  
 public void setPracticalsAndBatches( List<PracticalList> practicalsAndBatches) {  
 this.practicalsAndBatches = practicalsAndBatches;  
 }  
  
 public Integer getTheoryHours() {  
 return theoryHours;  
 }  
  
 public void setTheoryHours(Integer theoryHours) {  
 this.theoryHours = theoryHours;  
 }  
  
 public Integer getPracticalHours() {  
 return practicalHours;  
 }  
  
 public void setPracticalHours(Integer practicalHours) {  
 this.practicalHours = practicalHours;  
 }  
  
 public Integer getAllotedLoad() {  
 return allotedLoad;  
 }  
  
 public void setAllotedLoad(Integer allotedLoad) {  
 this.allotedLoad = allotedLoad;  
 }  
  
 public Integer getMaxLoad() {  
 return maxLoad;  
 }  
  
 public void setMaxLoad(Integer maxLoad) {  
 this.maxLoad = maxLoad;  
 }  
  
}

# spring\queryBeans\FacultyUsers.java

package com.qrms.spring.queryBeans;  
  
public class FacultyUsers {  
 private String name;  
 private String userName;  
 private String designation;  
 private String email;  
 private Double exp;  
 private String qualification;  
   
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name;  
 }  
 public String getUserName() {  
 return userName;  
 }  
 public void setUserName(String userName) {  
 this.userName = userName;  
 }  
 public String getDesignation() {  
 return designation;  
 }  
 public void setDesignation(String designation) {  
 this.designation = designation;  
 }  
 public String getEmail() {  
 return email;  
 }  
 public void setEmail(String email) {  
 this.email = email;  
 }  
 public Double getExp() {  
 return exp;  
 }  
 public void setExp(Double exp) {  
 this.exp = exp;  
 }  
 public String getQualification() {  
 return qualification;  
 }  
 public void setQualification(String qualification) {  
 this.qualification = qualification;  
 }  
}

# spring\queryBeans\PrefGroupByCourseStudent.java

package com.qrms.spring.queryBeans;  
  
public class PrefGroupByCourseStudent {  
   
 private long count; //stores the number of students who submitted the preference form for the course   
 private String courseId;  
   
 public PrefGroupByCourseStudent(long count, String courseId) {  
 this.courseId = courseId;  
 this.count = count;  
 }  
  
 public long getCount() {  
 return count;  
 }  
   
 public void setCount(long count) {  
 this.count = count;  
 }  
  
  
 public String getCourseId() {  
 return courseId;  
 }  
  
 public void setCourseId(String courseId) {  
 this.courseId = courseId;  
 }  
}

# spring\queryBeans\PrefNumCountPerElective.java

package com.qrms.spring.queryBeans;  
  
import com.qrms.spring.model.Electives;  
  
public class PrefNumCountPerElective {  
 private long count;  
 private int prefNo;  
 private String electiveId;  
 private Electives elective;  
   
 public long getCount() {  
 return count;  
 }  
 public Electives getElective() {  
 return elective;  
 }  
 public void setElective(Electives elective) {  
 this.elective = elective;  
 }  
 public void setCount(long count) {  
 this.count = count;  
 }  
 public int getPrefNo() {  
 return prefNo;  
 }  
 public void setPrefNo(int prefNo) {  
 this.prefNo = prefNo;  
 }  
   
 public PrefNumCountPerElective(long count, int prefNo, Electives elective) {  
 this.count = count;  
 this.prefNo = prefNo;  
 this.elective = elective;  
 }  
 public String getElectiveId() {  
 return electiveId;  
 }  
 public void setElectiveId(String electiveId) {  
 this.electiveId = electiveId;  
 }  
   
}

# spring\queryBeans\StudentCountByYearSem.java

package com.qrms.spring.queryBeans;  
  
public class StudentCountByYearSem {  
   
 private long count;  
 private int sem;  
 private String year;  
   
 public StudentCountByYearSem(long count,String year,int sem) {  
 this.count = count;  
 this.sem = sem;  
 this.year = year;  
 }  
  
 public long getCount() {  
 return count;  
 }  
  
 public void setCount(long count) {  
 this.count = count;  
 }  
  
 public int getSem() {  
 return sem;  
 }  
  
 public void setSem(int sem) {  
 this.sem = sem;  
 }  
  
 public String getYear() {  
 return year;  
 }  
  
 public void setYear(String year) {  
 this.year = year;  
 }  
}

# spring\queryBeans\StudentPrefCountInfo.java

package com.qrms.spring.queryBeans;  
  
public class StudentPrefCountInfo {  
  
 private long totalStudentCount;  
 private long submitCount;  
 private String deptId;  
 private String courseId;  
 private String courseName;  
 private String year;  
 private int sem;  
 private long count1,count2,count3,count4; //prefCounts for each priority value  
   
 public StudentPrefCountInfo(int c) {  
 this.count1 = c;  
 this.count2 = c;  
 this.count3 = c;  
 this.count4 = c;  
 }  
 public long getCount1() {  
 return count1;  
 }  
  
 public void setCount1(long count1) {  
 this.count1 = count1;  
 }  
  
 public long getCount2() {  
 return count2;  
 }  
  
 public void setCount2(long count2) {  
 this.count2 = count2;  
 }  
  
 public long getCount3() {  
 return count3;  
 }  
  
 public void setCount3(long count3) {  
 this.count3 = count3;  
 }  
  
 public long getCount4() {  
 return count4;  
 }  
  
 public void setCount4(long count4) {  
 this.count4 = count4;  
 }  
  
 public StudentPrefCountInfo() {  
   
 }  
   
 public long getTotalStudentCount() {  
 return totalStudentCount;  
 }  
 public void setTotalStudentCount(long totalStudentCount) {  
 this.totalStudentCount = totalStudentCount;  
 }  
 public long getSubmitCount() {  
 return submitCount;  
 }  
 public void setSubmitCount(long submitCount) {  
 this.submitCount = submitCount;  
 }  
 public String getDeptId() {  
 return deptId;  
 }  
 public void setDeptId(String deptId) {  
 this.deptId = deptId;  
 }  
 public String getCourseId() {  
 return courseId;  
 }  
 public void setCourseId(String courseId) {  
 this.courseId = courseId;  
 }  
 public String getCourseName() {  
 return courseName;  
 }  
 public void setCourseName(String courseName) {  
 this.courseName = courseName;  
 }  
 public String getYear() {  
 return year;  
 }  
 public void setYear(String year) {  
 this.year = year;  
 }  
 public int getSem() {  
 return sem;  
 }  
 public void setSem(int sem) {  
 this.sem = sem;  
 }  
}

# spring\queryBeans\StudentUsers.java

package com.qrms.spring.queryBeans;  
  
public class StudentUsers {  
 private int srNo;  
 private String rollNo;  
 private String name;  
 private String div;  
 private int shift;  
 private float aggrMarks;  
 private String userName;  
 private String email;  
   
 public String getUserName() {  
 return userName;  
 }  
 public void setUserName(String userName) {  
 this.userName = userName;  
 }  
 public StudentUsers() {  
   
 }  
 public int getSrNo() {  
 return srNo;  
 }  
 public void setSrNo(int srNo) {  
 this.srNo = srNo;  
 }  
 public String getRollNo() {  
 return rollNo;  
 }  
 public void setRollNo(String rollNo) {  
 this.rollNo = rollNo;  
 }  
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name;  
 }  
 public String getDiv() {  
 return div;  
 }  
 public void setDiv(String div) {  
 this.div = div;  
 }  
 public int getShift() {  
 return shift;  
 }  
 public void setShift(int shift) {  
 this.shift = shift;  
 }  
 public float getAggrMarks() {  
 return aggrMarks;  
 }  
 public void setAggrMarks(float aggrMarks) {  
 this.aggrMarks = aggrMarks;  
 }  
 public String getEmail() {  
 return email;  
 }  
 public void setEmail(String email) {  
 this.email = email;  
 }  
   
   
}

# spring\repository\CourseCompanionRespositoy.java

package com.qrms.spring.repository;  
  
import java.util.ArrayList;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
  
import com.qrms.spring.model.CompanionCourse;  
  
public interface CourseCompanionRespositoy extends JpaRepository<CompanionCourse, Integer> {  
  
 CompanionCourse findByCourse(String c);  
   
 //JPQL  
 @Query("SELECT "+  
 "new CompanionCourse(cc.course,cc.companionCourse) "+   
 "FROM CompanionCourse cc "+  
 "WHERE cc.companionCourse=?1 AND cc.course IN "+  
 "("  
 + "SELECT eb.electiveId FROM com.qrms.spring.model.ElectiveBatches eb"  
 + ")")  
 ArrayList<CompanionCourse> findByCompanionCourseAndCourseIdInElectiveBatches(String courseId);  
  
 CompanionCourse findByCompanionCourse(String courseId);  
  
}

# spring\repository\CourseListRepository.java

package com.qrms.spring.repository;  
  
import java.util.List;  
  
import javax.transaction.Transactional;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Modifying;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.stereotype.Repository;  
  
import com.qrms.spring.model.CourseList;  
import com.qrms.spring.model.Department;  
  
@Repository  
public interface CourseListRepository extends JpaRepository<CourseList, Integer>{  
   
 //JPQL  
 @Transactional  
 @Modifying  
 @Query("delete from CourseList c where c.facultyId in ?1")  
 void deleteByFacultyIdList(List<String> facIdList);  
  
 @Query("Select cl from CourseList cl where cl.facultyId in " +   
 "(SELECT fa.userName FROM com.qrms.spring.model.FacultyAcad fa where fa.department = ?1)")  
 List<CourseList> findByFacultyIdDepartment(Department dept);  
  
 List<CourseList> findByFacultyId(String userName);  
   
}

# spring\repository\CoursePrerequisitesRepository.java

package com.qrms.spring.repository;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import com.qrms.spring.model.CoursePrerequisites;  
  
public interface CoursePrerequisitesRepository extends JpaRepository<CoursePrerequisites, Integer> {  
 CoursePrerequisites findByCourseId(String c\_id);  
 void deleteByCourseId(String c\_id);  
}

# spring\repository\CourseRepository.java

package com.qrms.spring.repository;  
  
import java.util.ArrayList;  
import java.util.Optional;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.stereotype.Repository;  
  
import com.qrms.spring.model.Course;  
import com.qrms.spring.model.Department;  
  
@Repository  
public interface CourseRepository extends JpaRepository<Course, Integer> {  
  
 ArrayList<Course> findByCourseSemAndCourseYearAndCourseTypeAndDepartment(int sem, String year, char cType,Department d);  
 Course findByCourseId(String course\_id);  
 ArrayList<Course> findAll();  
 ArrayList<Course> findByStudAllocFlag(int flag);  
 ArrayList<Course> findByCourseSemAndCourseYear(int courseSem, String courseYear);  
 ArrayList<Course> findByCourseSemAndCourseYearAndCourseTypeAndDepartmentAndIsTheoryAndStudAllocFlag(int sem, String year, char cType,Department d,int isTheory,int stud\_allocation\_flag);  
 ArrayList<Course> findByCourseTypeNot(char courseType);  
  
 ArrayList<Course> findByCourseSemAndCourseYearAndCourseTypeNotAndDepartmentAndIsTheoryAndStudAllocFlag(int sem, String year, char cType,Department d,int isTheory,int stud\_allocation\_flag);  
 ArrayList<Course> findByDepartmentAndCourseSemAndCourseYearAndIsTheoryAndCourseTypeNotAndStudAllocFlag(int sem,String year, char cType,Department d,int isTheory,int stud\_allocation\_flag);  
 ArrayList<Course> findByCourseSemAndCourseYearAndCourseTypeAndIsTheoryAndStudAllocFlag(int sem, String year, char c,  
 int i, int j);  
 ArrayList<Course> findByCourseSemAndCourseYearAndCourseTypeNotAndDepartmentAndIsTheoryAndStudAllocFlagNot(  
 int courseSem, String courseYear, char c, Department department, int i, int j);  
 ArrayList<Course> findByStudAllocFlagNot(int i);  
 ArrayList<Course> findByDepartment(Department department);  
 Optional<Course> findByCourseIdAndDepartmentAndCourseYearAndCourseSemAndIsTheory(String companionTheory,  
 Department department, String courseYear, int courseSem, int i);  
 Optional<Course> findByCourseIdAndDepartmentAndIsTheory(String companionTheory, Department department, int i);  
 ArrayList<Course> findByDepartmentAndCourseType(Department department, char c);  
 ArrayList<Course> findByCourseYear(String year);  
 ArrayList<Course> findByCourseYearAndCourseTypeAndDepartment(String year, char c, Department department);  
  
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c "+  
 "WHERE c.courseSem%2=0 and c.courseType='R' and c.department=?1")  
 ArrayList<Course> findEvenSemCoursesAndCourseTypeRegAndDepartment(Department dept);  
  
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c "+  
 "WHERE c.courseSem%2=0 and c.courseType='R' and c.isTheory=0 and c.department=?1")  
 ArrayList<Course> findEvenSemCoursesAndCourseTypeRegAndIsTheoryNotAndDepartment(Department dept);  
  
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c "+  
 "WHERE c.courseSem%2<>0 and c.courseType='R' and c.isTheory=0 and c.department=?1")  
 ArrayList<Course> findOddSemCoursesAndCourseTypeRegAndIsTheoryNotAndDepartment(Department dept);  
  
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c "+  
 "WHERE c.courseSem%2=0 and c.courseType='R' and c.isTheory=1")  
 ArrayList<Course> findEvenSemCoursesAndCourseTypeRegAndIsTheoryAndDepartment(Department dept);  
  
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c "+  
 "WHERE c.courseSem%2<>0 and c.courseType='R' and c.isTheory=1 and c.department=?1")  
 ArrayList<Course> findOddSemCoursesAndCourseTypeRegAndIsTheoryAndDepartment(Department dept);  
  
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c "+  
 "WHERE c.courseSem%2<>0 and c.courseType='R' and c.department=?1")  
 ArrayList<Course> findOddSemCoursesAndCourseTypeRegAndDepartment(Department dept);  
  
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c "+  
 "WHERE c.courseSem%2<>0 and c.courseType<>'R' and c.department=?1")  
 ArrayList<Course> findOddSemCoursesAndCourseTypeNotRegAndDepartment(Department dept);  
  
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c "+  
 "WHERE c.courseSem%2=0 and c.courseType<>'R' and c.department=?1")  
 ArrayList<Course> findEvenSemCoursesAndCourseTypeNotRegAndDepartment(Department dept);  
  
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c "+  
 "WHERE c.courseSem%2=0 and c.courseType<>'R' and c.isTheory=0 and c.department=?1")  
 ArrayList<Course> findEvenSemCoursesAndCourseTypeNotRegAndIsTheoryNotAndDepartment(Department dept);  
  
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c "+  
 "WHERE c.courseSem%2<>0 and c.courseType<>'R' and c.isTheory=0 and c.department=?1")  
 ArrayList<Course> findOddSemCoursesAndCourseTypeNotRegAndIsTheoryNotAndDepartment(Department dept);  
  
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c "+  
 "WHERE c.courseSem%2=0 and c.courseType<>'R' and c.isTheory=1")  
 ArrayList<Course> findEvenSemCoursesAndCourseTypeNotRegAndIsTheoryAndDepartment(Department dept);  
  
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c "+  
 "WHERE c.courseSem%2<>0 and c.courseType<>'R' and c.isTheory=1 and c.department=?1")  
 ArrayList<Course> findOddSemCoursesAndCourseTypeNotRegAndIsTheoryAndDepartment(Department dept);  
  
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c WHERE c.courseSem%2<>0")  
 ArrayList<Course> findOddSemCourses();  
  
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c WHERE c.courseSem%2=0")  
 ArrayList<Course> findEvenSemCourses();  
  
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c WHERE c.courseSem%2=0 and c.department=?1")  
 ArrayList<Course> findByDepartmentAndEvenCourseSem(Department dept);  
  
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c WHERE c.courseSem%2<>0 and c.department=?1")  
 ArrayList<Course> findByDepartmentAndOddCourseSem(Department dept);  
  
   
  
 ArrayList<Course> findByCourseType(char c);  
 Optional<Course> findByCourseIdAndDepartmentAndIsTheoryAndCourseType(String prerequisiteNo1,  
 Department department, int i, char c);  
 Optional<Course> findByCourseIdAndDepartmentAndCourseYearAndCourseSemAndIsTheoryAndCourseType(  
 String companionTheory, Department department, String courseYear, int courseSem, int i, char c);  
   
 ArrayList<Course> findByCourseSem(Integer sem);  
 ArrayList<Course> findByCourseYearAndDepartment(String year, Department department);  
 ArrayList<Course> findByDepartmentAndCourseSem(Department department, Integer sem);  
 ArrayList<Course> findByDepartmentAndCourseSemAndCourseYear(Department department, Integer semester, String year);  
   
   
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c "+  
 "WHERE c.courseSem%2=0 and c.courseType='R' and c.isTheory=1 and c.department=?1 and c.courseYear=?2")  
 ArrayList<Course> findEvenSemCoursesAndCourseTypeRegAndIsTheoryAndDepartmentAndCourseYear(Department department,  
 String year);  
   
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c "+  
 "WHERE c.courseSem%2=0 and c.courseType<>'R' and c.isTheory=1 and c.department=?1 and c.courseYear=?2")  
 ArrayList<Course> findEvenSemCoursesAndCourseTypeNotRegAndIsTheoryAndDepartmentAndCourseYear(Department department,  
 String year);  
   
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c "+  
 "WHERE c.courseSem%2=1 and c.courseType='R' and c.isTheory=1 and c.department=?1 and c.courseYear=?2")  
 ArrayList<Course> findOddSemCoursesAndCourseTypeRegAndIsTheoryAndDepartmentAndCourseYear(Department department,  
 String year);  
   
 @Query("SELECT "+  
 "new com.qrms.spring.model.Course(c.courseId, c.courseName, c.courseCredits, c.department, c.courseType, c.courseYear, c.courseSem, c.studAllocFlag, c.isTheory, c.noOfHours) "+   
 "FROM Course c "+  
 "WHERE c.courseSem%2=1 and c.courseType<>'R' and c.isTheory=1 and c.department=?1 and c.courseYear=?2")  
 ArrayList<Course> findOddSemCoursesAndCourseTypeNotRegAndIsTheoryAndDepartmentAndCourseYear(Department department,  
 String year);  
 ArrayList<Course> findByCourseSemAndCourseYearAndCourseTypeNotAndDepartmentAndIsTheory(int courseSem,  
 String courseYear, char c, Department department, int i);  
  
}

# spring\repository\CurrTimeSlotsRepository.java

package com.qrms.spring.repository;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import com.qrms.spring.model.CurrentTimeSlots;  
  
public interface CurrTimeSlotsRepository extends JpaRepository<CurrentTimeSlots, Integer>{  
  
}

# spring\repository\DepartmentRepository.java

package com.qrms.spring.repository;  
  
import java.util.ArrayList;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
import com.qrms.spring.model.Department;  
  
@Repository  
public interface DepartmentRepository extends JpaRepository<Department, Integer> {  
   
 Department findByDeptId(String input\_dept);  
  
 Department findByDeptName(String input\_dept);  
   
 ArrayList<Department> findAll();  
}

# spring\repository\DesignationToHoursRepository.java

package com.qrms.spring.repository;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import com.qrms.spring.model.DesignationToHours;  
  
  
public interface DesignationToHoursRepository extends JpaRepository<DesignationToHours, Integer>{  
   
}

# spring\repository\DivisionsRepository.java

package com.qrms.spring.repository;  
  
import java.util.List;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import com.qrms.spring.model.Department;  
import com.qrms.spring.model.Divisions;  
  
public interface DivisionsRepository extends JpaRepository<Divisions,Integer>{  
  
 Divisions findByDivId(String divId);  
  
 List<Divisions> findByDepartment(Department dept);  
   
}

# spring\repository\ElectiveBatchesRepository.java

package com.qrms.spring.repository;  
  
import java.util.List;  
  
import javax.transaction.Transactional;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Modifying;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.stereotype.Repository;  
  
import com.qrms.spring.model.Course;  
import com.qrms.spring.model.Department;  
import com.qrms.spring.model.ElectiveBatches;  
  
@Repository  
public interface ElectiveBatchesRepository extends JpaRepository<ElectiveBatches, Integer> {  
  
 List<ElectiveBatches> findByDepartment(Department dept);  
   
   
 @Transactional  
 @Modifying  
 @Query("DELETE from ElectiveBatches eb where eb.electiveId in "  
 + "(SELECT el.electiveCourseId from com.qrms.spring.model.Electives el where el.course=?1)")  
 void deleteByCourseId(Course course);  
}

# spring\repository\ElectivesRepository.java

package com.qrms.spring.repository;  
  
import java.util.ArrayList;  
import java.util.Optional;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
  
import com.qrms.spring.model.Course;  
import com.qrms.spring.model.Electives;  
  
public interface ElectivesRepository extends JpaRepository<Electives, Integer> {  
 ArrayList <Electives> findByCourse(Course course);  
 Electives findByElectiveCourseId(String electiveId);  
 //Electives findByElectiveCourseIdAndIsTheory(String electiveId);  
 Optional<Electives> findByElectiveName(String electiveName);  
   
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.model.Electives(e.electiveCourseId, e.course, e.electiveName) "+   
 "FROM Electives e "+  
 "WHERE e.course.courseSem%2<>0")  
 ArrayList<Electives> findOddSemCoursesAndCourseTypeNotReg();  
   
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.model.Electives(e.electiveCourseId, e.course, e.electiveName) "+   
 "FROM Electives e "+  
 "WHERE e.course.courseSem%2=0")  
 ArrayList<Electives> findEvenSemCoursesAndCourseTypeNotReg();  
   
}

# spring\repository\ElectiveVacancyPrefCountsRepository.java

package com.qrms.spring.repository;  
  
import java.util.ArrayList;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import com.qrms.spring.model.ElectiveVacancyPrefCounts;  
  
public interface ElectiveVacancyPrefCountsRepository extends JpaRepository<ElectiveVacancyPrefCounts, Integer> {  
 ArrayList<ElectiveVacancyPrefCounts> findAll();  
 ArrayList<ElectiveVacancyPrefCounts> findByCourseId(String course\_id);  
 ElectiveVacancyPrefCounts findByElectiveId(String elective\_id);  
 void deleteByCourseId(String courseId);  
}

# spring\repository\FacultyAcadRepository.java

package com.qrms.spring.repository;  
  
import java.util.ArrayList;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
  
import com.qrms.spring.model.Department;  
import com.qrms.spring.model.FacultyAcad;  
  
public interface FacultyAcadRepository extends JpaRepository<FacultyAcad, Integer>{  
 FacultyAcad findByUserName(String userName);  
  
 ArrayList<FacultyAcad> findByDepartmentEquals(Department dept);  
  
 @Query("SELECT count(\*) from FacultyAcad fa where fa.designation=?1 and fa.department=?2")  
 int countFacultyByDesignationAndDepartment(String key,Department dept);  
   
 @Query("SELECT count(\*) from FacultyAcad fa where fa.department=?1")  
 int countFacultyByDepartment(Department dept);  
   
}

# spring\repository\FacultyAllocCourseRepository.java

package com.qrms.spring.repository;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import com.qrms.spring.model.FacultyAllocCourse;  
  
public interface FacultyAllocCourseRepository extends JpaRepository<FacultyAllocCourse,Integer> {  
  
}

# spring\repository\FacultyAllotedHoursRepository.java

package com.qrms.spring.repository;  
  
import java.util.List;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.stereotype.Repository;  
  
import com.qrms.spring.model.Department;  
import com.qrms.spring.model.FacultyAllotedHours;  
  
@Repository  
public interface FacultyAllotedHoursRepository extends JpaRepository<FacultyAllotedHours, Integer>{  
  
   
 @Query("Select fa from FacultyAllotedHours fa where fa.facultyId in " +   
 "(SELECT fa.userName FROM com.qrms.spring.model.FacultyAcad fa where fa.department = ?1)")  
 List<FacultyAllotedHours> findFacsByDepartment(Department dept);  
  
 FacultyAllotedHours findByFacultyId(String userName);  
   
   
  
}

# spring\repository\FacultyPrefRepository.java

package com.qrms.spring.repository;  
  
import java.util.ArrayList;  
  
import javax.transaction.Transactional;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Modifying;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.stereotype.Repository;  
  
import com.qrms.spring.model.Department;  
import com.qrms.spring.model.FacultyPref;  
  
@Repository  
public interface FacultyPrefRepository extends JpaRepository<FacultyPref, Integer>{  
 ArrayList<FacultyPref> findByUserName(String user\_name);  
 ArrayList<FacultyPref> findByUserNameAndYear(String userName, String year);  
 ArrayList<FacultyPref> findByElectiveId(String id);  
 ArrayList<FacultyPref> findByCourseId(String id);  
   
 @Query("SELECT "+  
 "count(DISTINCT fp.userName) "+   
 "FROM FacultyPref fp where fp.userName in "+  
 "("  
 + "SELECT fa.userName FROM com.qrms.spring.model.FacultyAcad fa where fa.department = ?1"  
 + ")")  
 int findFacultyPrefCountByDepartment(Department dept);  
  
 @Transactional  
 @Modifying  
 @Query("Delete from FacultyPref fp where fp.userName in "  
 + "(SELECT fa.userName FROM com.qrms.spring.model.FacultyAcad fa where fa.department = ?1)")  
 void deleteByDepartment(Department dept);  
   
}

# spring\repository\OldTimeSlotsRepository.java

package com.qrms.spring.repository;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import com.qrms.spring.model.OldTimeSlots;  
  
public interface OldTimeSlotsRepository extends JpaRepository<OldTimeSlots, Integer>{  
  
}

# spring\repository\OpenFacultyPrefsRepository.java

package com.qrms.spring.repository;  
  
  
import javax.transaction.Transactional;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Modifying;  
import org.springframework.stereotype.Repository;  
  
import com.qrms.spring.model.OpenFacultyPrefs;  
  
@Repository  
public interface OpenFacultyPrefsRepository extends JpaRepository<OpenFacultyPrefs, Integer> {  
  
 OpenFacultyPrefs findByDeptId(String deptId);  
  
 @Transactional  
 @Modifying  
 void deleteByDeptId(String deptId);  
  
}

# spring\repository\PasswordResetTokenRepository.java

package com.qrms.spring.repository;  
  
import java.util.Optional;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
import com.qrms.spring.model.PasswordResetToken;  
  
@Repository  
public interface PasswordResetTokenRepository extends JpaRepository<PasswordResetToken, Integer> {  
 Optional<PasswordResetToken> findByToken(String token);  
}

# spring\repository\PracticalListRepository.java

package com.qrms.spring.repository;  
  
import java.util.List;  
  
import javax.transaction.Transactional;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Modifying;  
import org.springframework.data.jpa.repository.Query;  
  
import com.qrms.spring.model.Department;  
import com.qrms.spring.model.PracticalList;  
  
public interface PracticalListRepository extends JpaRepository<PracticalList, Integer> {  
  
 @Transactional  
 @Modifying  
 @Query("delete from PracticalList p where p.facultyId in ?1")  
 void deleteByFacultyIdList(List<String> facIdList);  
  
 @Query("Select pl from PracticalList pl where pl.facultyId in " +   
 "(SELECT fa.userName FROM com.qrms.spring.model.FacultyAcad fa where fa.department = ?1)")  
 List<PracticalList> findByFacultyIdDepartment(Department dept);  
  
 List<PracticalList> findByFacultyId(String userName);  
  
}

# spring\repository\ResourceRepository.java

package com.qrms.spring.repository;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
  
import com.qrms.spring.model.Department;  
import com.qrms.spring.model.Resource;  
  
public interface ResourceRepository extends JpaRepository<Resource, Integer>{  
  
 List<Resource> findByDepartment(Department department);  
  
 Resource findByResourceId(String concat);  
   
 @Query("select r from Resource r where r.department = ?1 and r.resourceType = ?2 and r.resourceCapacity >= ?3")  
 ArrayList<Resource> findByDepartmentAndResourceTypeAndResourceCapacityGreaterThan(Department d, String rType,  
 int minSeats);  
  
}

# spring\repository\ResourceRequestsRepository.java

package com.qrms.spring.repository;  
  
import java.sql.Date;  
import java.sql.Time;  
import java.util.ArrayList;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Modifying;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.transaction.annotation.Transactional;  
  
import com.qrms.spring.model.Resource;  
import com.qrms.spring.model.ResourceRequests;  
  
public interface ResourceRequestsRepository extends JpaRepository<ResourceRequests, Integer>{  
 @Query("SELECT rr "+  
 //"new com.qrms.spring.model.ResourceRequests(rr.requestId,rr.resourceId,rr.requestBy,rr.slotDate,rr.slotStartTime,rr.slotEndTime,rr.slotActivityName,rr.requestedDate,rr.requestTime,rr.slotDay) "+   
 "FROM ResourceRequests rr " +  
 "WHERE rr.resourceId.resourceIncharge.userName=?1")  
 ArrayList<ResourceRequests> findByResourceIncharge(String userName);  
  
 ArrayList<ResourceRequests> findByResourceIdAndSlotDate(Resource resourceId, Date slotDate);  
  
 ResourceRequests findByRequestId(Integer getOverlapsFor);  
   
 @Transactional  
 @Modifying  
 @Query("DELETE FROM ResourceRequests rr WHERE (rr.slotDate < ?1) or (rr.slotDate = ?1 and rr.slotStartTime <= ?2)")  
 void deletePastRequests(Date sqlDate, Time t);  
   
 @Query("SELECT rr "+  
 //"new com.qrms.spring.model.ResourceRequests(rr.requestId,rr.resourceId,rr.requestBy,rr.slotDate,rr.slotStartTime,rr.slotEndTime,rr.slotActivityName,rr.requestedDate,rr.requestTime,rr.slotDay) "+   
 "FROM ResourceRequests rr " +  
 "WHERE rr.requestBy.userName=?1")  
 ArrayList<ResourceRequests> findByRequestBy(String userName);  
  
 void deleteByRequestId(Integer requestToDelete);  
  
}

# spring\repository\RoleRepository.java

package com.qrms.spring.repository;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
import com.qrms.spring.model.Role;  
  
@Repository  
public interface RoleRepository extends JpaRepository<Role, Integer> {  
 Role findByRole(String role);  
   
}

# spring\repository\StudentAcadRepository.java

package com.qrms.spring.repository;  
  
import java.util.ArrayList;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.stereotype.Repository;  
  
import com.qrms.spring.model.Department;  
import com.qrms.spring.model.StudentAcad;  
import com.qrms.spring.queryBeans.StudentCountByYearSem;  
  
@Repository  
public interface StudentAcadRepository extends JpaRepository<StudentAcad, Integer> {  
   
 StudentAcad findByUserName(String name);  
 ArrayList<StudentAcad> findBySemEqualsAndYearEquals(int semester,String year);  
   
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.queryBeans.StudentCountByYearSem(count(sa.userName),sa.year,sa.sem) "+   
 "FROM StudentAcad sa where "+  
 "sa.year=?1 and sa.sem=?2 and sa.department=?3 "+  
 "GROUP BY sa.sem,sa.year,sa.department")  
 StudentCountByYearSem findStudentCountByYearSemDept(String year, int sem, Department dept);  
   
 ArrayList<StudentAcad> findBySemEqualsAndYearEqualsAndDepartmentEquals(int semester, String year,  
 Department department);  
 ArrayList<StudentAcad> findByYearEqualsAndDepartmentEquals(String year, Department dept);  
}

# spring\repository\StudentAllocCourseRepository.java

package com.qrms.spring.repository;  
  
import java.util.ArrayList;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Modifying;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.transaction.annotation.Transactional;  
  
import com.qrms.spring.model.Course;  
import com.qrms.spring.model.Electives;  
import com.qrms.spring.model.StudentAcad;  
import com.qrms.spring.model.StudentAllocCourse;  
  
public interface StudentAllocCourseRepository extends JpaRepository<StudentAllocCourse, Integer> {  
 ArrayList<StudentAllocCourse> findByCourseId(Course course\_id);  
  
 void deleteByCourseId(Course course);  
  
 //JPQL  
 @Transactional  
 @Modifying  
 @Query("UPDATE StudentAllocCourse sac SET "  
 + "sac.batchId=?2 "  
 + "WHERE sac.elective = ?1")  
 void updateBatchIdByElectiveId(Electives elective,String batchId);  
  
 ArrayList <StudentAllocCourse> findByElective(Electives elective);  
// public StudentAllocCourse(int id,String batchId, Course courseId,Electives elective,StudentAcad student,int prefNo ) {  
  
 //select \* from student\_alloc\_course where student in (select user\_name as student from student\_acad order by div\_id);  
 @Query("SELECT new StudentAllocCourse(sac.id,sac.batchId,sac.courseId,sac.elective,sac.student,sac.prefNo) from "  
 + "StudentAllocCourse sac where sac.elective = ?1")  
// + " AND sac.student IN "  
// + "SELECT com.qrms.spring.model.StudentAcad(sa.userName) FROM com.qrms.spring.model.StudentAcad sa "  
// + "order by sa.divId")  
 ArrayList <StudentAllocCourse> findByElectiveIdSortedByDiv(Electives elective);  
  
 ArrayList<StudentAllocCourse> findByStudent(StudentAcad studentProfile);  
}

# spring\repository\StudentPrefRepository.java

package com.qrms.spring.repository;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.stereotype.Repository;  
  
import com.qrms.spring.model.Electives;  
import com.qrms.spring.model.StudentPref;  
import com.qrms.spring.queryBeans.PrefGroupByCourseStudent;  
import com.qrms.spring.queryBeans.PrefNumCountPerElective;  
  
@Repository  
public interface StudentPrefRepository extends JpaRepository<StudentPref, Integer>{  
   
   
 ArrayList<StudentPref> findByUserName(String user\_name);  
 ArrayList<StudentPref> findByCourseIdEquals(String electiveId);  
   
 //JPQL  
 @Query("SELECT "+  
 "new com.qrms.spring.queryBeans.PrefGroupByCourseStudent(count(DISTINCT sp.userName),sp.courseId) "+  
 "FROM StudentPref sp where sp.courseId = ?1 GROUP BY sp.courseId")  
 PrefGroupByCourseStudent findPrefsGroupByCourseStudent(String courseId);  
  
 //JPQL: Count priority-wise number of preferences for each elective  
 @Query("SELECT "+  
 "new com.qrms.spring.queryBeans.PrefNumCountPerElective(count(sp.prefNo),sp.prefNo,sp.elective) "+  
 "FROM StudentPref sp GROUP BY sp.prefNo,sp.elective")  
 List<PrefNumCountPerElective> findPrefNumCountPerElective();  
  
 void deleteByCourseId(String courseId);  
 ArrayList<StudentPref> findByUserNameAndCourseId(String userName,String course\_id);  
 ArrayList<StudentPref> findByElective(Electives e);  
  
}

# spring\repository\TimeSlotsRepository.java

package com.qrms.spring.repository;  
  
import java.sql.Date;  
import java.util.ArrayList;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
  
import com.qrms.spring.model.Resource;  
import com.qrms.spring.model.TimeSlots;  
  
public interface TimeSlotsRepository extends JpaRepository<TimeSlots, Integer>{  
  
 ArrayList<TimeSlots> findByResourceIdAndDate(Resource r, Date format);  
   
 @Query("SELECT ts "+  
 //"new com.qrms.spring.model.ResourceRequests(rr.requestId,rr.resourceId,rr.requestBy,rr.slotDate,rr.slotStartTime,rr.slotEndTime,rr.slotActivityName,rr.requestedDate,rr.requestTime,rr.slotDay) "+   
 "FROM TimeSlots ts " +  
 "WHERE ts.slotIncharge.userName=?1")  
 ArrayList<TimeSlots> findBySlotIncharge(String userName);  
   
}

# spring\repository\TimeTableRepository.java

package com.qrms.spring.repository;  
  
import java.util.ArrayList;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import com.qrms.spring.model.Department;  
import com.qrms.spring.model.Resource;  
import com.qrms.spring.model.TimeTable;  
  
public interface TimeTableRepository extends JpaRepository<TimeTable, Integer>{  
  
 ArrayList<TimeTable> findByResourceIdAndDay(Resource r, String day);  
  
 void deleteByDepartmentAndDay(Department department, String day);  
   
}

# spring\repository\UsersRepository.java

package com.qrms.spring.repository;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
import java.util.Set;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.stereotype.Repository;  
  
import com.qrms.spring.model.Role;  
import com.qrms.spring.model.Users;  
  
//repository to perform CRUD with Users table  
@Repository  
public interface UsersRepository extends JpaRepository<Users, Integer> {  
  
 Optional<Users> findByUserName(String user\_name);  
 Optional<Users> findByEmail(String email);  
 ArrayList<Users> findByRoles(Set<Role> roles);  
   
 @Query("select new Users(u.userName, u.firstName, u.lastName) from Users u where u.userName in ?1")  
 List<Users> findByUserNameList(List<String> facIds);  
   
  
}

# spring\resource\AdminController.java

package com.qrms.spring.resource;  
  
import java.io.File;  
import java.io.FileInputStream;  
import java.sql.Time;  
import java.text.DateFormat;  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.Collections;  
import java.util.HashMap;  
import java.util.HashSet;  
import java.util.Iterator;  
import java.util.LinkedHashSet;  
import java.util.List;  
import java.util.Map.Entry;  
import java.util.Optional;  
import java.util.Set;  
import java.util.StringTokenizer;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.transaction.Transactional;  
import javax.validation.Valid;  
  
import org.apache.poi.ss.usermodel.Cell;  
import org.apache.poi.ss.usermodel.Row;  
import org.apache.poi.xssf.usermodel.XSSFSheet;  
import org.apache.poi.xssf.usermodel.XSSFWorkbook;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.data.jpa.repository.Modifying;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestMethod;  
import org.springframework.web.bind.annotation.RequestParam;  
import org.springframework.web.bind.annotation.ResponseBody;  
import org.springframework.web.multipart.MultipartFile;  
import org.springframework.web.servlet.ModelAndView;  
  
import com.qrms.spring.model.Role;  
import com.qrms.spring.model.StudentAcad;  
import com.qrms.spring.model.StudentAllocCourse;  
import com.qrms.spring.model.StudentPref;  
import com.qrms.spring.model.TimeSlots;  
import com.qrms.spring.model.TimeTable;  
import com.qrms.spring.model.Users;  
import com.qrms.spring.queryBeans.FacultyUsers;  
import com.qrms.spring.queryBeans.FacultyAllocations;  
import com.qrms.spring.queryBeans.PrefNumCountPerElective;  
import com.qrms.spring.queryBeans.StudentPrefCountInfo;  
import com.qrms.spring.queryBeans.ElectiveBatchCount;  
import com.qrms.spring.queryBeans.ElectiveBatchCountList;  
import com.qrms.spring.queryBeans.FacPrefCountInfo;  
import com.qrms.spring.queryBeans.StudentUsers;  
import com.qrms.spring.model.Course;  
import com.qrms.spring.model.CourseList;  
import com.qrms.spring.comparators.DivisionsChainedComparator;  
import com.qrms.spring.comparators.DivisionsYearComparator;  
import com.qrms.spring.comparators.FacultyPrefChainedComparator;  
import com.qrms.spring.comparators.FacultyPrefCourseExpComparator;  
import com.qrms.spring.comparators.FacultyPrefNoComparator;  
import com.qrms.spring.comparators.FacultyPrefPrereqExp1Comparator;  
import com.qrms.spring.comparators.FacultyPrefPrereqExp2Comparator;  
import com.qrms.spring.model.CompanionCourse;  
import com.qrms.spring.model.CoursePrerequisites;  
import com.qrms.spring.model.Department;  
import com.qrms.spring.model.DesignationToHours;  
import com.qrms.spring.model.Divisions;  
import com.qrms.spring.model.ElectiveBatches;  
import com.qrms.spring.model.ElectiveVacancyPrefCounts;  
import com.qrms.spring.model.Electives;  
import com.qrms.spring.model.FacultyAcad;  
import com.qrms.spring.model.FacultyAllotedHours;  
import com.qrms.spring.model.FacultyPref;  
import com.qrms.spring.model.OpenFacultyPrefs;  
import com.qrms.spring.model.PracticalList;  
import com.qrms.spring.model.Resource;  
import com.qrms.spring.repository.CourseCompanionRespositoy;  
import com.qrms.spring.repository.CourseListRepository;  
import com.qrms.spring.repository.CoursePrerequisitesRepository;  
import com.qrms.spring.repository.CourseRepository;  
import com.qrms.spring.repository.DepartmentRepository;  
import com.qrms.spring.repository.DesignationToHoursRepository;  
import com.qrms.spring.repository.DivisionsRepository;  
import com.qrms.spring.repository.ElectiveBatchesRepository;  
import com.qrms.spring.repository.ElectiveVacancyPrefCountsRepository;  
import com.qrms.spring.repository.ElectivesRepository;  
import com.qrms.spring.repository.FacultyAcadRepository;  
import com.qrms.spring.repository.FacultyAllotedHoursRepository;  
import com.qrms.spring.repository.FacultyPrefRepository;  
import com.qrms.spring.repository.OpenFacultyPrefsRepository;  
import com.qrms.spring.repository.PracticalListRepository;  
import com.qrms.spring.repository.ResourceRepository;  
import com.qrms.spring.repository.RoleRepository;  
import com.qrms.spring.repository.StudentAcadRepository;  
import com.qrms.spring.repository.StudentAllocCourseRepository;  
import com.qrms.spring.repository.StudentPrefRepository;  
import com.qrms.spring.repository.TimeTableRepository;  
import com.qrms.spring.repository.UsersRepository;  
import com.qrms.spring.service.BookingsServiceImpl;  
import com.qrms.spring.service.CustomUserDetailsService;  
import com.qrms.spring.service.FacPrefServiceImpl;  
import com.qrms.spring.service.FacultyAcadService;  
import com.qrms.spring.service.StudentAcadServiceImpl;  
import com.qrms.spring.service.StudentPrefServiceImpl;  
  
@Controller  
@RequestMapping("/u/admin")  
public class AdminController {  
 @Autowired  
 private HttpServletRequest request;  
   
 @Autowired  
 private CustomUserDetailsService userDetails;  
   
 @Autowired  
 private StudentPrefServiceImpl studPrefService;  
   
 @Autowired  
 private FacPrefServiceImpl facPrefService;  
   
 @Autowired  
 private FacultyAcadService facAcadService;  
   
 @Autowired  
 private StudentAcadServiceImpl studAcadService;  
   
 @Autowired  
 private RoleRepository roleRepository;  
   
 @Autowired  
 private OpenFacultyPrefsRepository openFacultyPrefsRepository;  
   
 @Autowired  
 private StudentAcadRepository studentAcadRepository;  
   
 @Autowired  
 private DepartmentRepository departmentRepository;  
   
 @Autowired  
 private CourseRepository courseRepository;  
   
 @Autowired  
 private ElectiveVacancyPrefCountsRepository electiveVacancyPrefCountsRepository;  
   
 @Autowired  
 private StudentAllocCourseRepository studentAllocCourseRepository;  
   
 @Autowired  
 private StudentPrefRepository studentPrefRepository;  
   
 @Autowired  
 private FacultyAcadRepository facultyAcadRepository;  
   
 @Autowired  
 private ElectivesRepository electivesRepository;  
   
 @Autowired  
 private CourseCompanionRespositoy courseCompanionRepository;  
   
 @Autowired  
 private CoursePrerequisitesRepository coursePrerequisitesRepository;  
   
 @Autowired  
 private FacultyPrefRepository facultyPrefRepository;  
   
 @Autowired  
 private FacultyAllotedHoursRepository facultyAllotedHoursRepository;  
   
 @Autowired  
 private DesignationToHoursRepository designationToHoursRepository;  
   
 @Autowired  
 private DivisionsRepository divisionsRepository;  
   
 @Autowired  
 private ElectiveBatchesRepository electiveBatchesRepository;  
   
 @Autowired  
 private TimeTableRepository timeTableRepository;  
   
 @Autowired  
 private ResourceRepository resourceRepository;  
   
 @Autowired  
 private CourseListRepository courseListRepository;  
   
 @Autowired  
 private PracticalListRepository practicalListRepository;  
   
 @Autowired  
 private UsersRepository usersRepository;  
  
 @Autowired  
 private BookingsServiceImpl bookingsService;  
   
 private FacultyAcad faculty;  
   
 private List<Department> departments;  
   
 private List<Users> faculties;  
   
 private List<Role> roles;   
   
 private String g\_msg,g\_err\_msg;  
   
 private List<StudentPrefCountInfo> prefSummaryList;  
   
 private List<FacultyAllocations> rs;   
   
 private static final DateFormat TWELVE\_TF = new SimpleDateFormat("hh:mma");  
 // Replace with kk:mm if you want 1-24 interval  
 private static final DateFormat TWENTY\_FOUR\_TF = new SimpleDateFormat("HH:mm");  
  
 //show home page, without tables  
 @GetMapping("/home")  
 public ModelAndView adminHome() {  
 return getViewAdminHome(null,null);  
 }  
   
 //show home page  
 public ModelAndView getViewAdminHome(List<StudentPrefCountInfo> studCountInfo,List<FacPrefCountInfo> facPrefCountInfo) {  
 ModelAndView model = new ModelAndView();  
   
 if(studCountInfo!=null) {  
 if(studCountInfo.isEmpty())  
 model.addObject("err\_msg","There are no open student elective preference forms");  
 else {  
 model.addObject("studCountInfo",studCountInfo);  
 if(g\_msg!=null)  
 model.addObject("msg",g\_msg);  
 else  
 model.addObject("err\_msg",g\_err\_msg);  
 }  
 } else if(facPrefCountInfo!=null) {  
 if(facPrefCountInfo.isEmpty())  
 model.addObject("err\_msg","There are no open faculty course preference forms");  
 else {  
 model.addObject("facCountInfo",facPrefCountInfo);  
 if(g\_msg!=null)  
 model.addObject("msg",g\_msg);  
 else  
 model.addObject("err\_msg",g\_err\_msg);  
 }  
 }  
 model.setViewName("/admin/home");  
 return model;   
   
 }  
   
   
 @RequestMapping(value = "/getStudPrefDetailsTable\_async", method = RequestMethod.GET)  
 public String getStudPrefDetailsTable(Model model) {  
   
 List<StudentPrefCountInfo> studCountInfo = studPrefService.computeStudPrefTable();  
 if(studCountInfo.isEmpty()) {  
 model.addAttribute("err\_msg","There are no open student elective preference forms");  
 return "admin/home:: messageDiv";  
 }  
 else {  
 model.addAttribute("studCountInfo",studCountInfo);  
 return "admin/home:: studPrefTable";  
 }  
   
 }  
   
 @RequestMapping(value = "/getFacPrefDetailsTable\_async", method = RequestMethod.GET)  
 public String getFacPrefDetailsTable(Model model) {  
   
 List<FacPrefCountInfo> facCountInfo = facPrefService.computeFacPrefTable();  
 if(facCountInfo.isEmpty()) {  
 model.addAttribute("err\_msg","There are no open faculty course preference forms");  
 return "admin/home:: messageDiv";  
 }  
 else {  
 model.addAttribute("facCountInfo",facCountInfo);  
 return "admin/home:: facPrefTable";  
 }  
   
 }  
   
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Departments Page \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 @GetMapping("/getDepartmentsPage")  
 public ModelAndView getDepartmentsPage() {  
 ModelAndView model = new ModelAndView();  
 departments = departmentRepository.findAll();  
 model.addObject("departments", departments);  
 model.addObject("newDept", new Department());  
 model.setViewName("admin/departments");  
 return model;  
 }  
   
 @RequestMapping(value = "/addDept", method = RequestMethod.POST)  
 public ModelAndView addDept(Department newDept) {  
 departmentRepository.save(newDept);  
 return getDepartmentsPage();  
 }  
   
 @GetMapping("/manageDept")  
 public String getManageDept(Model model, String dept) {  
 Department department = departmentRepository.findByDeptId(dept);  
 ArrayList<FacultyAcad> fac = facultyAcadRepository.findByDepartmentEquals(department);  
 List<String> facIds = new ArrayList<>();  
 for(FacultyAcad f: fac) {  
 facIds.add(f.getUserName());  
 }  
 faculties = usersRepository.findByUserNameList(facIds);  
 model.addAttribute("manageDept",department);  
 model.addAttribute("div", new Divisions());  
 model.addAttribute("res", new Resource());  
 model.addAttribute("faculties", faculties);  
 return "admin/departments:: manageDeptFragment";  
 }  
   
 @RequestMapping(value = "/addDivision", method = RequestMethod.POST)  
 String addDivision(Model model, Divisions div, String dept) {  
 div.setDepartment(departmentRepository.findByDeptId(dept));  
 String divId = div.getYear() + div.getDepartment().getDeptId() + div.getDivName();  
 div.setDivId(divId);  
 divisionsRepository.save(div);  
 model.addAttribute("msg","Division added in "+div.getYear()+"-"+div.getDepartment().getDeptName());  
 return "admin/departments:: messageDiv";  
 }  
   
 @RequestMapping(value = "/addResource", method = RequestMethod.POST)  
 String addResource(Model model, Resource res, String dept, String incharge) {  
 res.setDepartment(departmentRepository.findByDeptId(dept));  
 res.setResourceIncharge(facultyAcadRepository.findByUserName(incharge));  
 resourceRepository.save(res);  
 model.addAttribute("msg","Resource added for "+res.getDepartment().getDeptName());  
 return "admin/departments:: messageDiv";  
 }  
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*end departments page\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
   
 @GetMapping("/getStudPrefDetailsTable")  
 public ModelAndView getStudPrefDetailsTable() {   
   
 return getViewAdminHome(studPrefService.computeStudPrefTable(),null);   
 }  
   
 @Transactional  
 @RequestMapping(value = "/performQuickAction-student", method = RequestMethod.POST)  
 public ModelAndView studentAllocQuickAction(String courseId, String selectAction, String courseName) {  
 String courseIds[] = courseId.split(",");  
 String actions[] = selectAction.split(",");  
 int i=0;  
 for(i=0; i<actions.length;i++) {  
 if(!actions[i].equals("none")) {  
 break;  
 }  
 }  
   
 if(actions[i].equals("clearPrefs")) {  
 clear\_preferences(courseIds[i]);   
   
 } else if(actions[i].equals("summary")) {  
 //give summary and perform allocation  
 prefSummaryList = new ArrayList<StudentPrefCountInfo>();  
 Course course = courseRepository.findByCourseId(courseIds[i]);  
 List<Electives> electives = electivesRepository.findByCourse(course);  
 List<PrefNumCountPerElective> countPerElectiveList = studentPrefRepository.findPrefNumCountPerElective();  
   
 for(Electives e: electives) {  
 StudentPrefCountInfo ps = new StudentPrefCountInfo(0); //initialing all prefCounts = 0  
 ps.setCourseId(e.getElectiveCourseId());  
 ps.setCourseName(e.getElectiveName());  
   
 for(PrefNumCountPerElective countPerElective : countPerElectiveList) {  
 if(countPerElective.getElective().getElectiveCourseId().equals(e.getElectiveCourseId())) {  
 switch(countPerElective.getPrefNo()) {  
 case 1: ps.setCount1(countPerElective.getCount()); break;  
 case 2: ps.setCount2(countPerElective.getCount()); break;  
 case 3: ps.setCount3(countPerElective.getCount()); break;  
 case 4: ps.setCount4(countPerElective.getCount()); break;   
 }  
 }  
 }   
 prefSummaryList.add(ps);  
 }  
   
 List<StudentPrefCountInfo> studentPrefInfo = studPrefService.computeStudPrefTable();  
 ModelAndView model = new ModelAndView();  
 model.addObject("studCountInfo",studentPrefInfo);  
 model.addObject("prefSummaryList",prefSummaryList);  
 model.setViewName("/admin/home");  
 return model;  
 }  
   
 else {  
 //action = close pref forms   
 Course course = courseRepository.findByCourseId(courseIds[i]);  
 if(course.getStudAllocFlag()!=2) {  
 course.setStudAllocFlag(2);  
 courseRepository.save(course);  
 g\_msg = "Preference forms for Course-id: "+courseIds[i]+" have been closed";  
 g\_err\_msg = null;   
 }  
 else  
 {  
 g\_err\_msg = "Preference forms for Course-id: "+courseIds[i]+" are already closed";  
 g\_msg = null;   
 }  
 }  
   
 return getStudPrefDetailsTable();  
 }  
   
 @GetMapping("/getFacPrefDetailsTable")  
 public ModelAndView getFacPrefDetailsTable() {   
   
 return getViewAdminHome(null,facPrefService.computeFacPrefTable());   
 }  
   
 @Transactional  
 @RequestMapping(value = "/performQuickAction-faculty", method = RequestMethod.POST)  
 public ModelAndView facultyAllocQuickAction(String deptName, String selectActionFac, String semType,   
 String totalFacultyCount, String submitCount) {  
  
 String actions[] = selectActionFac.split(",");  
 String deptNames[] = deptName.split(",");  
 String semTypes[] = semType.split(",");  
 String submitCounts[] = submitCount.split(",");  
 String totalFacultyCounts[] = totalFacultyCount.split(",");  
   
 int i;  
 for(i=0; i<actions.length;i++) {  
 if(!actions[i].equals("none")) {  
 break;  
 }  
 }  
   
 String action = actions[i];  
 String selectedDeptName = deptNames[i];  
 String selectedSemType = semTypes[i];  
 int selectedSubmitCount = Integer.parseInt(submitCounts[i]);  
 int selectedTotalFacultyCount = Integer.parseInt(totalFacultyCounts[i]);  
 Department dept = departmentRepository.findByDeptId(selectedDeptName);  
 OpenFacultyPrefs ofp = openFacultyPrefsRepository.findByDeptId(dept.getDeptId());  
  
 if(action.equals("clearPrefs")) {  
 if(ofp.getStatus()==0) {  
 facultyPrefRepository.deleteByDepartment(dept);  
 openFacultyPrefsRepository.deleteByDeptId(dept.getDeptId());  
 }  
 else {  
 g\_err\_msg = "Close preference forms before performing this action";  
 g\_msg = null;  
 }  
   
   
 } else if(action.equals("performAllocation")) {  
 if(ofp.getStatus()==0) {  
 if(selectedSemType.charAt(0)=='E')  
 allocFaculty(0, dept);  
 else  
 allocFaculty(1, dept);  
 g\_msg = "Faculty allocation has been completed for "+dept.getDeptName();  
 g\_err\_msg=null;   
 }  
 else {  
 g\_err\_msg = "Close preference forms before performing this action";  
 g\_msg = null;  
 }  
   
 }  
   
 else {  
 //action = close pref forms  
 if(ofp.getStatus()==0) {  
 g\_err\_msg = "Preference forms are already closed";  
 g\_msg = null;  
 }  
 else if(selectedSubmitCount!=selectedTotalFacultyCount) {  
   
 g\_err\_msg = "All faculties have not given preferences. Form has been closed but allocation cannot be performed.<br>"  
 + "Clear preferences to remove this entry";  
 g\_msg = null;  
 ofp.setStatus(0);  
 openFacultyPrefsRepository.save(ofp);  
 }  
 else {  
 g\_msg = "Preference forms have been closed";  
 g\_err\_msg = null;  
 ofp.setStatus(0);  
 openFacultyPrefsRepository.save(ofp);  
 }  
   
 }  
   
   
 return getFacPrefDetailsTable();  
 }  
   
   
   
 @RequestMapping(value="/view-courses",method=RequestMethod.GET)  
 public ModelAndView viewCourses() {  
 ModelAndView model = new ModelAndView();  
 departments = departmentRepository.findAll();  
 model.addObject("department",departments);  
 model.setViewName("admin/viewCourses");  
 return model;  
 }  
   
 @RequestMapping(value="/view-courses",method=RequestMethod.POST)  
 public String viewCoursesUsingRequirements(Model model,String dept,String year,char sem) {  
 ArrayList<Course> courseList;  
 System.out.println("dept "+dept);  
 System.out.println("year "+year);  
 System.out.println("sem "+sem);  
   
   
 if (dept.equals("none") && year.equals("none") && sem=='0') {  
 courseList = courseRepository.findAll();  
   
 System.out.println("all");  
 }else {  
 if (dept.equals("none") && year.equals("none") && sem!='0') {  
   
 if(sem == 'o') {  
 //odd  
 courseList = courseRepository.findOddSemCourses();  
 }else {  
 //even  
 courseList = courseRepository.findEvenSemCourses();  
 }  
   
 }else if(dept.equals("none") && !year.equals("none") && sem=='0'){  
 courseList = courseRepository.findByCourseYear(year);  
   
 }else if(!dept.equals("none") && year.equals("none") && sem=='0') {  
 Department department = departmentRepository.findByDeptId(dept);  
 courseList = courseRepository.findByDepartment(department);  
   
 }else if(dept.equals("none") && !year.equals("none") && sem!='0') {  
 Integer semester = Character.getNumericValue(sem);  
 courseList = courseRepository.findByCourseSemAndCourseYear(semester, year);  
   
 }else if(!dept.equals("none") && !year.equals("none") && sem=='0') {  
 Department department = departmentRepository.findByDeptId(dept);  
 courseList = courseRepository.findByCourseYearAndDepartment(year,department);  
   
 }else if(!dept.equals("none") && year.equals("none") && sem!='0'){  
 Department department = departmentRepository.findByDeptId(dept);  
   
 if(sem == 'o') {  
 //odd  
 courseList = courseRepository.findByDepartmentAndOddCourseSem(department);  
 }else {  
 //even  
 courseList = courseRepository.findByDepartmentAndEvenCourseSem(department);  
 }  
 }  
 else {  
 Department department = departmentRepository.findByDeptId(dept);  
 Integer semester = Character.getNumericValue(sem);  
 courseList = courseRepository.findByDepartmentAndCourseSemAndCourseYear(department,semester,year);  
   
 }  
 }  
 if (courseList.size()!=0){  
 model.addAttribute("courses", courseList);  
 return "admin/viewCourses:: courseTableDiv";  
 }else {  
 model.addAttribute("errmsg","No courses found!");  
 return "admin/viewCourses:: messageDiv";  
 }  
   
 }  
  
 @RequestMapping(value="/viewUsers", method = RequestMethod.GET)  
 public ModelAndView viewUsers() {  
 ModelAndView model = new ModelAndView();  
 departments = departmentRepository.findAll();  
 roles = roleRepository.findAll();  
 model.addObject("roles",roles);  
 model.addObject("department",departments);  
 model.setViewName("admin/viewUsers");  
 return model;  
   
 }  
   
   
 @RequestMapping(value="/viewStudents", method = RequestMethod.POST)  
 public String viewStudents(Model model, String year, String dept ) {  
   
 Department department = departmentRepository.findByDeptId(dept);  
 ArrayList<StudentUsers> studUsersList= studAcadService.getStudentList(department, year);  
 model.addAttribute("studUsersList",studUsersList);  
 return "admin/viewUsers:: studTable";  
 }  
   
 @RequestMapping(value="/viewAdmins", method = RequestMethod.GET)  
 public String viewAdmins(Model model) {  
   
 Set<Role> adminRole = new HashSet<Role>();  
 adminRole.add(new Role(1,"ADMIN"));  
 ArrayList<Users> adminUsers = userDetails.findByRole(adminRole);  
 model.addAttribute("adminUsersList",adminUsers);  
 return "admin/viewUsers:: adminsTable";  
   
 }  
   
  
 @RequestMapping(value="/viewFaculty", method = RequestMethod.POST)  
 public String viewFaculty(Model model,String dept) {  
   
 Department department = departmentRepository.findByDeptId(dept);  
 ArrayList<FacultyUsers> facUsersList= facAcadService.getFacultyList(department);  
   
 model.addAttribute("facultyUsersList",facUsersList);  
 return "admin/viewUsers:: facultyTable";  
   
 }  
   
 @Modifying  
 @Transactional  
 @RequestMapping(value = "/changeSeatsAndAllocate", method = RequestMethod.POST)  
 public ModelAndView changeSeatsAndAllocate(String courseIdList, String seatList) {  
 String electiveIds[] = courseIdList.split(",");  
 String seats[] = seatList.split(",");  
 String courseId = "";  
   
 //check whether no of students is more than seats available  
 //TODO  
   
 for(int i=0;i<electiveIds.length;i++) {  
 ElectiveVacancyPrefCounts ec = electiveVacancyPrefCountsRepository.findByElectiveId(electiveIds[i]);  
 ec.setVacancyCount(Integer.parseInt(seats[i]));  
 electiveVacancyPrefCountsRepository.save(ec);  
 courseId = ec.getCourseId();  
 }  
 set\_process\_student\_allocation(courseId);  
   
 ModelAndView model = new ModelAndView();  
  
 if(g\_err\_msg!=null) {  
 return getStudPrefDetailsTable();  
 }  
   
 // return getStudPrefDetailsTable();  
   
   
 Course c = courseRepository.findByCourseId(courseId);  
// System.out.println("course "+c.getCourseId());  
 ArrayList<StudentAllocCourse> allocs = studentAllocCourseRepository.findByCourseId(c);  
// System.out.println("allocs size "+allocs.size());  
 HashMap<String,Integer> electiveToCount = new HashMap<>();  
 for(StudentAllocCourse sac:allocs) {  
// System.out.println("sac "+sac.getCourseId());  
 String elective = sac.getElective().getElectiveCourseId();  
 if(electiveToCount.containsKey(elective)) {  
 electiveToCount.replace(elective, electiveToCount.get(elective)+1);  
 }else {  
 electiveToCount.put(elective, 1);  
 }  
 }  
   
 model.addObject("electiveToCount",electiveToCount);  
 model.addObject("deptId",c.getDepartment().getDeptId());  
 model.addObject("year",c.getCourseYear());  
 model.setViewName("admin/electiveBatches");  
   
 return model;  
   
 }  
  
 @Transactional  
 @Modifying  
 @ResponseBody  
 @RequestMapping(value="/set-batches",method=RequestMethod.POST)  
 public String setNoOfBatches(Model model,@RequestBody ElectiveBatchCountList electiveBatchCounts) {  
 String dept = electiveBatchCounts.getDeptId();  
 String year = electiveBatchCounts.getYear();  
 Course delCourse = electivesRepository.findByElectiveCourseId(electiveBatchCounts.getElectiveBatchCounts().get(0).getElectiveId()).getCourse();  
 System.out.println(delCourse.getCourseId());  
 electiveBatchesRepository.deleteByCourseId(delCourse);  
   
 for(ElectiveBatchCount ebc:electiveBatchCounts.getElectiveBatchCounts()) {  
 String batchId = "";  
 List<String> batchIdList = new ArrayList<String>();  
 for(int i=0;i<ebc.getNoOfBatches();i++) {  
 ElectiveBatches eb = new ElectiveBatches();  
 batchId = year+dept+"-"+ebc.getElectiveId()+"-"+(i+1);  
 eb.setBatchId(batchId);  
 eb.setYear(year);  
 eb.setDepartment(departmentRepository.findByDeptId(dept));  
 eb.setElectiveId(ebc.getElectiveId());  
 electiveBatchesRepository.save(eb);  
 batchIdList.add(batchId);  
 }  
 if(ebc.getNoOfBatches() == 1)  
 studentAllocCourseRepository.updateBatchIdByElectiveId(electivesRepository.findByElectiveCourseId(ebc.getElectiveId()), batchId);  
 else {  
 ArrayList<StudentAllocCourse> sacList = studentAllocCourseRepository.findByElectiveIdSortedByDiv(electivesRepository.findByElectiveCourseId(ebc.getElectiveId()));  
 int totalStudents = sacList.size();  
 int studentsPerBatch = Math.round(totalStudents/ebc.getNoOfBatches());  
 int i = 0,extra=0;  
   
 if(studentsPerBatch\*ebc.getNoOfBatches() < totalStudents)   
 extra = totalStudents - studentsPerBatch\*ebc.getNoOfBatches();  
 else   
 extra = studentsPerBatch\*ebc.getNoOfBatches() - totalStudents;  
   
 System.out.println("Extra: "+extra);  
 System.out.println("StudentsPerBatch: "+studentsPerBatch);  
 for(int j=0;j<ebc.getNoOfBatches();j++) {  
 int batchCount = 0;  
 if(extra-->0)  
 batchCount = studentsPerBatch+1;  
 else  
 batchCount = studentsPerBatch;  
   
 System.out.println(batchCount);  
 int cur = i;  
 for(;i<cur+batchCount;i++) {  
 StudentAllocCourse st = sacList.get(i);  
 st.setBatchId(batchIdList.get(j));  
 studentAllocCourseRepository.save(st);  
 }  
 }  
 }  
 }  
 return "success";  
 }  
   
   
 //display uploadTT page  
 @RequestMapping(value="/uploadTT", method = RequestMethod.GET)  
 public ModelAndView uploadTT() {  
 ModelAndView model = new ModelAndView();  
 departments = departmentRepository.findAll();   
 model.addObject("departments",departments);  
  
 model.setViewName("admin/uploadTT");  
 return model;  
 }  
   
 //Handle upload TT form  
 @Transactional  
 @RequestMapping(value = "/upload\_TT", method = RequestMethod.POST)  
 public ModelAndView upload\_TT(@RequestParam("timeTableFile") MultipartFile file, String dept, String day) {  
 ModelAndView model = new ModelAndView();   
   
 if(file.isEmpty()) {  
 model.addObject("err\_msg","Please select a file to upload");  
 }  
 else {  
 System.out.println(file.getOriginalFilename());  
   
 try {  
 String uploadsDir = "/uploads/";  
 String realPathToUploads = request.getServletContext().getRealPath(uploadsDir);  
 System.out.println(realPathToUploads);  
 if(! new File(realPathToUploads).exists())  
 {  
 new File(realPathToUploads).mkdir();  
 }  
   
 String orgName = file.getOriginalFilename();  
 String filePath = realPathToUploads + orgName;  
 System.out.println(filePath);  
 File dest = new File(filePath);  
 file.transferTo(dest);  
   
 String msg = readTT(filePath,dept,day);  
   
 if(msg.equals("Time Table has been uploaded successfully.")) {  
 model.addObject("msg",msg);  
 }else {  
 model.addObject("err\_msg",msg);  
 }  
   
   
 } catch (Exception e) {  
 e.printStackTrace();  
 }   
 }  
   
   
 departments = departmentRepository.findAll();  
 model.addObject("departments", departments);   
 model.setViewName("admin/uploadTT");  
 return model;  
   
 }  
   
 //Display register user form  
 @RequestMapping(value = "/register", method = RequestMethod.GET)  
 public ModelAndView registerUsers(String msg, String err\_msg) {  
 ModelAndView model = new ModelAndView();  
 Users user = new Users();  
 roles = roleRepository.findAll();  
 departments = departmentRepository.findAll();  
 model.addObject("user",user);  
 model.addObject("student",new StudentAcad());  
 model.addObject("roles",roles);  
 model.addObject("department", departments);  
 model.setViewName("admin/registerUsers");  
   
 if(msg == null && err\_msg!=null) {  
 model.addObject("errmsg",err\_msg);  
 }  
 else if(msg !=null && err\_msg==null) {  
 model.addObject("msg",msg);  
 }  
 return model;  
 }  
   
 //Handle register user form  
 @RequestMapping(value = "/register\_users", method = RequestMethod.POST)  
 public ModelAndView createUser(Users user, String role, StudentAcad student,String dept,  
 String dept1, Double facExp, String facDesignation, String facQualification, String divName  
 ) {  
 ModelAndView model = new ModelAndView();   
 Role userRole = roleRepository.findByRole(role);  
 user.setRoles(new HashSet<Role>(Arrays.asList(userRole)));  
 //check unique email  
 String email = user.getEmail();  
 if(!userDetails.isUniqueEmail(email)) {  
 String errmsg = "A user is already registered with the given email";  
 return registerUsers(null,errmsg);  
 }  
   
 if(role.equals("STUDENT")) {  
 System.out.println("Adding user to studAcad");  
 Department department = departmentRepository.findByDeptId(dept);  
 student.setDepartment(department);  
 if(studAcadService.validateAndSetStudDiv(student,divName)) {  
 userDetails.saveUser(user);  
 student.setUserName(user.getUserName());  
 student.setUserDets(user);  
 studentAcadRepository.save(student);   
 }  
 else   
 return registerUsers(null,"Invalid division");  
 }  
 else if(role.equals("FACULTY")) {  
   
 userDetails.saveUser(user);  
   
 faculty = new FacultyAcad();  
 System.out.println("Adding user to facultyAcad");  
 faculty.setUserName(user.getUserName());  
 Department department = departmentRepository.findByDeptId(dept1);  
   
 faculty.setDepartment(department);  
 faculty.setDesignation(facDesignation);  
 faculty.setQualification(facQualification);  
 faculty.setYearsOfExperience(facExp);  
 faculty.setUserDets(user);  
 facultyAcadRepository.save(faculty);  
   
 }  
 else {  
 userDetails.saveUser(user);  
 }  
   
 departments = departmentRepository.findAll();  
 model.addObject("msg","User has been successfully registered");  
 model.addObject("user",new Users());  
 model.addObject("roles",roles);  
 model.addObject("department", departments);  
 model.addObject("student",new StudentAcad());   
 model.setViewName("admin/registerUsers");  
 return model;  
   
 }  
   
 //Display add course page  
 @RequestMapping(value = "/add-course", method = RequestMethod.GET)  
 public ModelAndView addCourses() {  
 ModelAndView model = new ModelAndView();  
 Course course = new Course();  
 departments = departmentRepository.findAll();  
 model.addObject("course",course);   
 model.addObject("departments",departments);  
 model.setViewName("admin/addCourses");  
 return model;  
 }  
   
 //Handle add course form  
 @RequestMapping(value = "/add\_courses", method = RequestMethod.POST)  
 public ModelAndView addCourse(@Valid Course course, String dept,String companionTheory,String prerequisiteNo1,String prerequisiteNo2) {  
 ModelAndView model = new ModelAndView();  
   
 Department deptObj = departmentRepository.findByDeptName(dept);  
 course.setDepartment(deptObj);  
 course.setStudAllocFlag(0);  
   
 System.out.println(companionTheory);  
 System.out.println(prerequisiteNo1+" "+prerequisiteNo2);  
   
 if(companionTheory.equals("") && prerequisiteNo1.equals("") && prerequisiteNo2.equals("")) {  
 model.addObject("msg","Course has been added successfully");  
 model.addObject("course",new Course());  
 courseRepository.save(course);  
 }   
 else if(!companionTheory.equals("")) {  
 Optional<Course> thCourse = courseRepository.findByCourseIdAndDepartmentAndCourseYearAndCourseSemAndIsTheoryAndCourseType(companionTheory,course.getDepartment(),course.getCourseYear(),course.getCourseSem(),1,'R');  
   
 if(thCourse.isPresent()) {  
 //saving course  
 model.addObject("msg","Course has been added successfully");  
 model.addObject("course",new Course());  
 courseRepository.save(course);  
   
 CompanionCourse cc = new CompanionCourse(thCourse.get().getCourseId(), course.getCourseId());  
 courseCompanionRepository.save(cc);  
   
 CompanionCourse ncc = new CompanionCourse(course.getCourseId(), thCourse.get().getCourseId());  
 courseCompanionRepository.save(ncc);  
   
   
 }  
 else {  
 model.addObject("err\_msg","Companion theory course doesn't exist!");  
 model.addObject("course",new Course());  
 }  
 }else if(!prerequisiteNo1.equals("") && !prerequisiteNo2.equals("")) {  
 Optional<Course> pr1Course = courseRepository.findByCourseIdAndDepartmentAndIsTheoryAndCourseType(prerequisiteNo1,course.getDepartment(),1,'R');  
 Optional<Course> pr2Course = courseRepository.findByCourseIdAndDepartmentAndIsTheoryAndCourseType(prerequisiteNo2,course.getDepartment(),1,'R');  
   
 if(pr1Course.isPresent() && pr2Course.isPresent()) {  
 CoursePrerequisites cr = new CoursePrerequisites(course.getCourseId(),0,0,prerequisiteNo1,prerequisiteNo2);  
 courseRepository.save(course);  
 coursePrerequisitesRepository.save(cr);  
 model.addObject("msg","Course has been added successfully");  
 model.addObject("course",new Course());  
   
 } else if(!pr1Course.isPresent() && !pr2Course.isPresent()){  
 model.addObject("err\_msg","Prerequisite 1 and 2 theory courses don't exist!");  
 model.addObject("course",new Course());   
   
 }else if(!pr1Course.isPresent()){  
 model.addObject("err\_msg","Prerequisite 1 theory course doesn't exist!");  
 model.addObject("course",new Course());   
   
 }else if(!pr2Course.isPresent()){  
 model.addObject("err\_msg","Prerequisite 2 theory course doesn't exist!");  
 model.addObject("course",new Course());  
   
 }  
 } else if(!prerequisiteNo1.equals("") && prerequisiteNo2.equals("")) {  
 Optional<Course> pr1Course = courseRepository.findByCourseIdAndDepartmentAndIsTheoryAndCourseType(prerequisiteNo1,course.getDepartment(),1,'R');  
   
 if(pr1Course.isPresent()) {  
 CoursePrerequisites cr = new CoursePrerequisites(course.getCourseId(),0,-1,prerequisiteNo1,"NA");  
 courseRepository.save(course);  
 coursePrerequisitesRepository.save(cr);  
 model.addObject("msg","Course has been added successfully");  
 model.addObject("course",new Course());  
   
 } else if(!pr1Course.isPresent()){  
 model.addObject("err\_msg","Prerequisite 1 theory course doesn't exist!");  
 model.addObject("course",new Course());   
   
 }  
 } else {  
 model.addObject("err\_msg","Cannot add course, something went wrong!");  
 model.addObject("course",new Course());  
 }  
   
   
 model.addObject("departments",departments);  
 model.setViewName("admin/addCourses");  
 return model;  
 }  
   
 //Display add elective page  
 @RequestMapping(value="/add-elective",method=RequestMethod.GET)  
 public ModelAndView get\_all\_elective() {  
   
 ModelAndView model = new ModelAndView();  
 Electives elective = new Electives();  
   
 ArrayList<Course> electivesList = courseRepository.findByCourseTypeNot('R');  
   
 model.addObject("electivesList",electivesList);  
 model.addObject("elective",elective);  
 model.setViewName("/admin/addElective");  
 return model;  
   
 }  
   
 //Handle add elective form  
 @RequestMapping(value="/add\_elective",method=RequestMethod.POST)  
 public ModelAndView set\_all\_elective(String suffix, Electives elective,String courseId, String companionTheory,String prerequisiteNo1,String prerequisiteNo2) {  
   
 ModelAndView model = new ModelAndView();  
 System.out.println("checkL-"+courseId);  
 System.out.println(companionTheory);  
 System.out.println(prerequisiteNo1);  
 System.out.println(prerequisiteNo2);  
 elective.setElectiveCourseId(courseId.concat(suffix));  
 Course course = courseRepository.findByCourseId(courseId);  
 elective.setCourse(course);  
   
 if(electivesRepository.findByElectiveName(elective.getElectiveName()).isPresent() || electivesRepository.findByElectiveCourseId(elective.getElectiveCourseId())!=null) {  
 System.out.println("Elective with same name or ID already exists");  
 model.addObject("err\_msg","Elective with same name or ID already exists");  
 }  
 else {  
 if(companionTheory.equals("") && prerequisiteNo1.equals("") && prerequisiteNo2.equals("")) {  
 model.addObject("msg","Elective has been added successfully");  
 electivesRepository.save(elective);  
 }   
 else if(!companionTheory.equals("")) {  
 Electives compElective = electivesRepository.findByElectiveCourseId(companionTheory);  
   
 if(compElective!=null) {  
 electivesRepository.save(elective);  
 CompanionCourse cc = new CompanionCourse(compElective.getElectiveCourseId(), elective.getElectiveCourseId());  
 courseCompanionRepository.save(cc);  
   
 CompanionCourse ncc = new CompanionCourse(elective.getElectiveCourseId(), compElective.getElectiveCourseId());  
 courseCompanionRepository.save(ncc);  
 model.addObject("msg","Elective has been added successfully");  
 }  
 else {  
 model.addObject("err\_msg","Companion theory course doesn't exist!");  
 model.addObject("course",new Course());  
 }  
 }else if(!prerequisiteNo1.equals("") && !prerequisiteNo2.equals("")) {  
 Optional<Course> pr1Course = courseRepository.findByCourseIdAndDepartmentAndIsTheoryAndCourseType(prerequisiteNo1,course.getDepartment(),1,'R');  
 Optional<Course> pr2Course = courseRepository.findByCourseIdAndDepartmentAndIsTheoryAndCourseType(prerequisiteNo2,course.getDepartment(),1,'R');  
 Electives pr1Elective = electivesRepository.findByElectiveCourseId(prerequisiteNo1);  
 Electives pr2Elective = electivesRepository.findByElectiveCourseId(prerequisiteNo2);  
   
 //both preq are regular  
 if(pr1Course.isPresent() && pr2Course.isPresent() && pr1Elective ==null && pr2Elective ==null) {  
 electivesRepository.save(elective);  
 CoursePrerequisites cr = new CoursePrerequisites(elective.getElectiveCourseId(),0,0,prerequisiteNo1,prerequisiteNo2);  
 coursePrerequisitesRepository.save(cr);  
 model.addObject("msg","Elective has been added successfully");  
   
 //both preq are electives  
 } else if(!pr1Course.isPresent() && !pr2Course.isPresent() && pr1Elective !=null && pr2Elective !=null){  
 electivesRepository.save(elective);  
 CoursePrerequisites cr = new CoursePrerequisites(elective.getElectiveCourseId(),1,1,prerequisiteNo1,prerequisiteNo2);  
 coursePrerequisitesRepository.save(cr);  
 model.addObject("msg","Elective has been added successfully");  
   
 //first is regular and second is elective   
 } else if(pr1Course.isPresent() && !pr2Course.isPresent() && pr1Elective ==null && pr2Elective !=null){  
 electivesRepository.save(elective);  
 CoursePrerequisites cr = new CoursePrerequisites(elective.getElectiveCourseId(),0,1,prerequisiteNo1,prerequisiteNo2);  
 coursePrerequisitesRepository.save(cr);  
 model.addObject("msg","Elective has been added successfully");  
   
 //first is elective and second is regular  
 } else if(!pr1Course.isPresent() && pr2Course.isPresent() && pr1Elective !=null && pr2Elective ==null){  
 electivesRepository.save(elective);  
 CoursePrerequisites cr = new CoursePrerequisites(elective.getElectiveCourseId(),1,0,prerequisiteNo1,prerequisiteNo2);  
 coursePrerequisitesRepository.save(cr);  
 model.addObject("msg","Elective has been added successfully");  
   
 } else if(pr1Course.isPresent() && pr2Course.isPresent() && pr1Elective !=null && pr2Elective !=null){  
 model.addObject("err\_msg", "Ambiguous Ids entered.");  
   
 } else if(!pr1Course.isPresent() && !pr2Course.isPresent() && pr1Elective ==null && pr2Elective ==null){  
 model.addObject("err\_msg","Prerequisite 1 and 2 theory courses don't exist!");  
   
 }else if(!pr1Course.isPresent() && pr1Elective==null){  
 model.addObject("err\_msg","Prerequisite 1 theory course doesn't exist!");  
   
 }else if(!pr2Course.isPresent() && pr2Elective==null){  
 model.addObject("err\_msg","Prerequisite 2 theory course doesn't exist!");  
   
 }  
 } else if(!prerequisiteNo1.equals("") && prerequisiteNo2.equals("")) {  
 Optional<Course> pr1Course = courseRepository.findByCourseIdAndDepartmentAndIsTheoryAndCourseType(prerequisiteNo1,course.getDepartment(),1,'R');  
 Electives pr1Elective = electivesRepository.findByElectiveCourseId(prerequisiteNo1);  
  
 if(pr1Course.isPresent() && pr1Elective==null) {  
 CoursePrerequisites cr = new CoursePrerequisites(elective.getElectiveCourseId(),0,-1,prerequisiteNo1,"NA");  
 electivesRepository.save(elective);  
 coursePrerequisitesRepository.save(cr);  
 model.addObject("msg","Elective has been added successfully");  
   
 } else if(!pr1Course.isPresent() && pr1Elective!=null) {  
 CoursePrerequisites cr = new CoursePrerequisites(elective.getElectiveCourseId(),1,-1,prerequisiteNo1,"NA");  
 electivesRepository.save(elective);  
 coursePrerequisitesRepository.save(cr);  
 model.addObject("msg","Elective has been added successfully");  
   
 } else if(!pr1Course.isPresent() && pr1Elective==null){  
 model.addObject("err\_msg","Prerequisite 1 theory course doesn't exist!");  
   
 } else if(pr1Course.isPresent() && pr1Elective!=null) {  
 model.addObject("err\_msg","Ambigious Ids entered");  
 }  
 }  
 else {  
 model.addObject("err\_msg","Cannot add course, something went wrong!");  
 }  
 //model.addObject("msg","Elective added successfully");   
 }  
   
   
 ArrayList<Course> electivesList = courseRepository.findByCourseTypeNot('R');  
 model.addObject("electivesList",electivesList);  
 model.addObject("elective",new Electives());  
 model.setViewName("/admin/addElective");  
 return model;  
   
 }  
   
 //Get studCourseAllocation HTML page for opening course allocation forms  
 @RequestMapping(value="/openCourseAllocation",method=RequestMethod.GET)  
 public ModelAndView openCourseAllocation(ArrayList<Course> elective\_ids,String err\_msg) {  
 ModelAndView model = new ModelAndView();  
   
 ArrayList<Department> departments = departmentRepository.findAll();  
 model.addObject("departments",departments);  
 model.setViewName("/admin/studCourseAllocation");  
 model.addObject("course",new Course());  
   
 if(elective\_ids!=null && !elective\_ids.isEmpty())  
 model.addObject("elective\_ids",elective\_ids);  
   
 if(err\_msg!=null)  
 model.addObject("err\_msg",err\_msg);  
   
 return model;  
 }  
   
 // CHANGE NEEDED AFTER HANDLING OPEN ELECTIVES   
 //Retrieve all electives for specified dept, year and sem   
 @RequestMapping(value="/findElective",method=RequestMethod.POST)  
 public ModelAndView findElective(@Valid Course course, String dept) {  
   
 Department department = departmentRepository.findByDeptId(dept);  
   
 System.out.println(course.getCourseSem()+" "+course.getCourseYear());  
 System.out.println(department.getDeptId());  
   
 ArrayList<Course> courses = courseRepository.findByCourseSemAndCourseYearAndCourseTypeNotAndDepartmentAndIsTheory(course.getCourseSem(),course.getCourseYear(),'R',department,1);  
 if(courses.size()==0) {  
 String msg = "No Elective Courses found!";  
 return openCourseAllocation(null,msg);  
 }  
   
 ArrayList<Course> elective\_ids= courseRepository.findByCourseSemAndCourseYearAndCourseTypeNotAndDepartmentAndIsTheoryAndStudAllocFlag(course.getCourseSem(),course.getCourseYear(),'R',department,1,0);  
  
 if(elective\_ids.size()==0) {  
 String msg = "Student elective preference forms have already been released for the specified department, year and semester.";  
 return openCourseAllocation(elective\_ids,msg);  
 }  
   
 return openCourseAllocation(elective\_ids,null);   
 }  
   
  
 //open course allocation forms for specified electives, (set studAllocFlag = 1)  
 @RequestMapping(value="/open\_student\_allocation",method=RequestMethod.POST)  
 public ModelAndView open\_student\_allocation(String electiveIdOption) {  
   
 ModelAndView model = new ModelAndView();  
   
 Course startCourse = courseRepository.findByCourseId(electiveIdOption);   
   
 if(startCourse== null)  
 model.addObject("err\_msg","Specified elective does not exist for the given Year and Semester");  
 else {  
 ArrayList<Electives> all\_electives = electivesRepository.findByCourse(startCourse);  
 if(all\_electives.size()!=0) {  
 for (Electives electives : all\_electives) {  
 ElectiveVacancyPrefCounts electiveVacancyPrefCounts = new ElectiveVacancyPrefCounts();  
 System.out.println(startCourse.getCourseId());  
 electiveVacancyPrefCounts.setCourseId(startCourse.getCourseId());  
 electiveVacancyPrefCounts.setPrefCount(0);  
 electiveVacancyPrefCounts.setVacancyCount(2); //total seats available for an elective course  
 electiveVacancyPrefCounts.setElectiveId(electives.getElectiveCourseId());  
 electiveVacancyPrefCountsRepository.save(electiveVacancyPrefCounts);  
   
 startCourse.setStudAllocFlag(1);  
 courseRepository.save(startCourse);   
   
 model.addObject("msg","Student preference forms are released for: "  
 +startCourse.getCourseYear()+" Semester: "+startCourse.getCourseSem()  
 +" "+startCourse.getCourseName());  
 }  
 }else {  
 model.addObject("err\_msg","No electives found for the Course Id: ".concat(electiveIdOption));  
 }  
   
 }  
 ArrayList<Department> departments = departmentRepository.findAll();  
  
 model.addObject("departments",departments);   
 model.addObject("course",new Course());  
 model.setViewName("/admin/studCourseAllocation");   
 return model;  
 }  
   
 //showing allocations - getShowAllocations  
 @RequestMapping(value="/getShowAllocations",method=RequestMethod.GET)  
 public ModelAndView getShowAllocations(ArrayList<Course> elective\_ids,String msg,String err\_msg)   
 {  
 ModelAndView model = new ModelAndView();  
   
 ArrayList<Department> departments = departmentRepository.findAll();  
   
 model.addObject("course", new Course());  
 model.addObject("departments", departments);  
 model.setViewName("/admin/showAllocations");  
 if(elective\_ids.size()!=0)  
 model.addObject("elective\_ids",elective\_ids);  
 if(msg!=null)  
 model.addObject("msg", msg);  
 if(err\_msg!=null)  
 model.addObject("err\_msg",err\_msg);  
 return model;  
 }  
   
   
 //Get facCourseAllocation HTML page for opening course allocation forms  
 @RequestMapping(value="/openFacCourseAllocation",method=RequestMethod.GET)  
 public ModelAndView openFacCourseAllocation() {  
 ModelAndView model = new ModelAndView();  
   
 ArrayList<Department> departments = departmentRepository.findAll();  
 model.addObject("departments",departments);  
 model.setViewName("/admin/openFacPrefs");  
   
 return model;  
 }  
   
 @Transactional  
 @Modifying  
 @RequestMapping(value="/open\_faculty\_allocation",method=RequestMethod.POST)  
 public ModelAndView open\_faculty\_allocation(String deptId, String semType) {  
   
 ModelAndView model = new ModelAndView();  
   
 OpenFacultyPrefs openFacPrefs = new OpenFacultyPrefs();  
 openFacPrefs.setDeptId(deptId);  
 openFacPrefs.setSemType(Integer.parseInt(semType));  
 openFacPrefs.setStatus(1);  
   
 openFacultyPrefsRepository.save(openFacPrefs);  
   
 Department dept = departmentRepository.findByDeptId(deptId);  
   
 facultyPrefRepository.deleteByDepartment(dept);  
   
 ArrayList<Department> departments = departmentRepository.findAll();  
 model.addObject("departments",departments);   
 model.addObject("msg", "Faculty preference forms have been opened successfully");  
  
 model.setViewName("/admin/openFacPrefs");   
 return model;  
 }  
   
   
 //@RequestMapping(value="/process\_student\_allocation\_post",method=RequestMethod.POST)  
 @Transactional   
 private void set\_process\_student\_allocation(String electiveIdOption) {  
  
 //check whether course is open for preferences or not  
 Course course = courseRepository.findByCourseId(electiveIdOption);  
 System.out.println(course.getCourseId()+" "+course.getStudAllocFlag());  
 if(course.getStudAllocFlag()==2) {  
 ArrayList<Electives> all\_electives = electivesRepository.findByCourse(course);  
   
 if(all\_electives.size()!=0) {  
   
 int alloc = allocation\_of\_students\_to\_elective\_course(course.getCourseId(),course.getCourseYear(),course.getCourseSem());   
 if (alloc==1) {  
 g\_msg = "The allocation has been completed for Course-Id: "+course.getCourseId();  
 g\_err\_msg = null;  
 }  
 else if(alloc==0) {  
 g\_err\_msg = "No preferences are recorded!";  
 g\_msg = null;  
 }  
 else if(alloc==-1) {  
 g\_err\_msg = "Allocation has already been performed for Course-id: "+electiveIdOption;  
 g\_msg = null;  
 }  
 else if(alloc==-2) {  
 g\_err\_msg = "Total number of seats is less than number of students.";  
 g\_msg = null;  
 }  
   
 }else {  
 g\_err\_msg = "No electives found for the course Id: ".concat(electiveIdOption);  
 g\_msg = null;  
 }  
 }else {  
 g\_err\_msg = "Preference forms should be closed before performing allocation.";  
 g\_msg = null;  
 }  
   
 // return process\_student\_allocation(null,g\_msg,g\_err\_msg);  
 }  
   
 // CHANGE NEEDED AFTER HANDLING OPEN ELECTIVES   
 private int allocation\_of\_students\_to\_elective\_course(String course\_id,String year,int semester) {  
   
 ArrayList<StudentPref> studentPrefs = studentPrefRepository.findByCourseIdEquals(course\_id);  
  
 Course ElCourse = courseRepository.findByCourseId(course\_id);  
 ArrayList<StudentAllocCourse> studentAllocs = studentAllocCourseRepository.findByCourseId(ElCourse);  
   
 if(studentAllocs.size()!=0) {  
 studentAllocCourseRepository.deleteByCourseId(ElCourse);  
 }  
 if(studentPrefs.size()!=0) {  
   
 ArrayList<StudentAcad> studentAcads = studentAcadRepository.findBySemEqualsAndYearEqualsAndDepartmentEquals(semester,year,ElCourse.getDepartment());  
  
 Collections.sort(studentAcads);  
   
 //Store all vacancy counts for all courses  
   
 ArrayList <ElectiveVacancyPrefCounts> allElectiveCounts = electiveVacancyPrefCountsRepository.findByCourseId(course\_id);  
  
 HashMap<String,Integer> eVHM = new HashMap<String,Integer>();  
   
 for (ElectiveVacancyPrefCounts e : allElectiveCounts) {  
 eVHM.put(e.getElectiveId(), e.getVacancyCount());   
 }  
   
 ArrayList<Electives> popularElectives = calculatePrefCounts(course\_id,year,semester,allElectiveCounts);  
   
   
 HashMap<StudentAcad,StudentAllocCourse> studAllocs = new HashMap<StudentAcad,StudentAllocCourse>();  
   
 //for each student in reverse   
 for (StudentAcad studentAcad : studentAcads) {  
 ArrayList<StudentPref> stud = studentPrefRepository.findByUserNameAndCourseId(studentAcad.getUserName(),course\_id);  
   
 if(stud.size()!=0) {  
   
 Electives prefs[] = new Electives[stud.size()];  
 int i = 0;  
 for (StudentPref studPref : stud) {  
 prefs[i]= studPref.getElective();  
 i+=1;  
 }  
   
 int prefNo = 1;  
 int flag = 0;  
   
 for (Electives pref : prefs) {  
 int vCount = eVHM.get(pref.getElectiveCourseId());  
 if (vCount>0) {  
 eVHM.replace(pref.getElectiveCourseId(), vCount-1);  
 StudentAllocCourse s = new StudentAllocCourse(pref, pref.getCourse(), studentAcad, prefNo,"");  
 studAllocs.put(studentAcad, s);  
 flag = 1;  
 break;  
 }  
 prefNo+=1;  
 }  
 if(flag==0) {  
 //assign popular course  
 System.out.println("No preference left!");  
 System.out.println("Assigning course according to popularity!");  
 for (Electives e : popularElectives) {  
 int vCount = eVHM.get(e.getElectiveCourseId());  
 if (vCount>0) {  
 eVHM.replace(e.getElectiveCourseId(), vCount-1);  
 StudentAllocCourse s = new StudentAllocCourse(e,e.getCourse(),studentAcad,-1,"");  
 studAllocs.put(studentAcad, s);  
 flag = 1;  
 break;  
 }  
 }  
 if(flag==0) {  
 System.out.println("NO OPTION LEFT!!!! NEED TO INCREASE CAPACITY!!!");  
 return -2;  
 }  
 }  
   
 }else {  
 //assign popular course  
// System.out.println("Hasn't given preference");  
// System.out.println("Assigning course according to popularity!");  
 int flag=0;  
 for (Electives e : popularElectives) {  
 int vCount = eVHM.get(e.getElectiveCourseId());  
 if (vCount>0) {  
 eVHM.replace(e.getElectiveCourseId(), vCount-1);  
 StudentAllocCourse s = new StudentAllocCourse(e,e.getCourse(),studentAcad,-1,"");  
 studAllocs.put(studentAcad, s);  
 flag = 1;  
 break;  
 }  
 }  
 if(flag==0) {  
 System.out.println("NO OPTION LEFT!!!! NEED TO INCREASE CAPACITY!!!");  
 return -2;  
 }  
  
 }  
   
 }  
 for (StudentAcad s : studAllocs.keySet()) {  
 studentAllocCourseRepository.save(studAllocs.get(s));  
 }  
   
 return 1;  
 }else {  
 return 0;  
 }  
  
   
 }  
  
 private ArrayList<Electives> calculatePrefCounts(String course\_id,String year,int semester,ArrayList <ElectiveVacancyPrefCounts> electiveVacancyPrefCounts){  
  
 Collections.sort(electiveVacancyPrefCounts);  
   
 ArrayList<Electives> electivesList = new ArrayList<>();  
   
 for (ElectiveVacancyPrefCounts e : electiveVacancyPrefCounts) {  
 electivesList.add(electivesRepository.findByElectiveCourseId(e.getElectiveId()));  
 }  
 return electivesList;  
 }  
   
  
 @Transactional  
 private void clear\_preferences(String electiveIdOption) {  
   
 int status = courseRepository.findByCourseId(electiveIdOption).getStudAllocFlag();  
 if(status!=2) {  
 g\_msg = null;  
 if(status == 1)  
 g\_err\_msg = "Cannot clear preferences for open forms. Close forms before clearing preferences";  
 else  
 g\_err\_msg = "Preferences have already been cleared";  
 }  
 else {  
 studentPrefRepository.deleteByCourseId(electiveIdOption);  
 electiveVacancyPrefCountsRepository.deleteByCourseId(electiveIdOption);  
 Course c = courseRepository.findByCourseId(electiveIdOption);  
 c.setStudAllocFlag(0);  
 courseRepository.save(c);  
   
 g\_msg = "Cleared preferences for Course Id: "+electiveIdOption;  
 g\_err\_msg = null;  
   
 }  
 }   
   
  
 //findElectivesToShow  
 //@ResponseBody  
 @RequestMapping(value="/findElectivesToShow",method=RequestMethod.POST)  
 public ModelAndView findElectivesToShow(Course course,String dept) {  
   
 String year = course.getCourseYear();  
 Integer sem = course.getCourseSem();  
 Department department = departmentRepository.findByDeptId(dept);  
   
 ArrayList<Course> elective\_ids= courseRepository.findByCourseSemAndCourseYearAndCourseTypeNotAndDepartmentAndIsTheoryAndStudAllocFlag(sem,year,'R',department,1,2);  
   
 if(elective\_ids.size()==0) {  
 String err\_msg = "No electives are opened for preference forms.";  
 return getShowAllocations(elective\_ids,null,err\_msg);  
  
 }  
 return getShowAllocations(elective\_ids, null,null);  
   
 }  
   
 //showAllocations  
 @RequestMapping(value="/showAllocations",method=RequestMethod.POST)   
 public ModelAndView showAllocations(String electiveIdOption) {  
 ModelAndView model = new ModelAndView();  
  
 ArrayList<Department> departments = departmentRepository.findAll();  
   
 model.addObject("course", new Course());  
 model.addObject("departments", departments);  
 model.setViewName("/admin/showAllocations");  
   
 Course c = courseRepository.findByCourseId(electiveIdOption);  
 ArrayList<StudentAllocCourse> studAllocations = studentAllocCourseRepository.findByCourseId(c);  
 if(studAllocations.size()!=0)  
 model.addObject("studAllocations", studAllocations);  
 else  
 model.addObject("err\_msg", "No allocations done yet for Course Id: "+electiveIdOption);  
 return model;  
 }  
   
 //delete course or elective  
 @Transactional  
 @RequestMapping(value="/delete-course-elective",method=RequestMethod.GET)  
 public ModelAndView getDelCourseOrElective(String msg,String err\_msg) {  
 ModelAndView model = new ModelAndView();  
 model.setViewName("/admin/deleteCourseOrElective");  
 if(msg!=null)  
 model.addObject("msg",msg);  
 else if(err\_msg!=null)  
 model.addObject("err\_msg",err\_msg);  
 return model;  
 }  
   
 @Transactional  
 @RequestMapping(value="/delete-course-elective",method=RequestMethod.POST)  
 public ModelAndView setDelCourseOrElective(String c\_id,String e\_id) {  
 String msg = null;  
 String err\_msg = null;  
 if(!c\_id.equals("")) {  
 System.out.println("course");  
 Course c = courseRepository.findByCourseId(c\_id);  
 CoursePrerequisites cprereq = coursePrerequisitesRepository.findByCourseId(c\_id);  
 ArrayList<StudentPref> cstudPref = studentPrefRepository.findByCourseIdEquals(c\_id);  
 if(c!=null) {  
 courseRepository.delete(c);  
   
 if(cprereq!=null)  
 {  
 coursePrerequisitesRepository.deleteByCourseId(c\_id);  
 //msg.concat("Corresponding prerequisites mapping are deleted.\n");  
 }  
 if(cstudPref.size()!=0)  
 {  
 studentPrefRepository.deleteByCourseId(c\_id);  
 //msg.concat("Corresponding prerequisites mapping are deleted.\n");  
 }  
 msg = "Course has been deleted successfully.";  
 }else {  
 err\_msg = "Specified Course does not exist. Please check again.";  
 }  
 }else if(!e\_id.equals("")) {  
 System.out.println("elective");  
 Electives e = electivesRepository.findByElectiveCourseId(e\_id);  
   
 ArrayList<StudentPref> estudPref = studentPrefRepository.findByElective(e);  
   
 if(e!=null) {  
 if(e.getCourse().getStudAllocFlag()!=0) {  
 err\_msg = "Cannot delete the elective as its status is Open or Closed.";  
   
 }else {  
   
 electivesRepository.delete(e);  
 if(estudPref.size()!=0) {  
 studentPrefRepository.deleteByCourseId(e\_id);  
 //msg.concat("Corresponding prerequisites mapping are deleted.\n");  
 }  
 msg = "Elective has been deleted successfully.";  
 }  
 }else {  
 err\_msg = "Specified Elective does not exist. Please check again.";  
 }  
 }else if(c\_id.equals("") && e\_id.equals("")) {  
 System.out.println("error");  
 err\_msg = "Something went wrong.";  
 }  
   
 return getDelCourseOrElective(msg,err\_msg);  
 }  
  
  
// @RequestMapping(value="/allocFaculty",method=RequestMethod.POST)  
// public ModelAndView  
 // 0 -- odd, 1 -- even  
 public void allocFaculty(int isSemOdd,Department dept) {  
   
 //find all the courses  
 //get data of designation and faculty for that department  
   
 //designationToHours objects list  
 List<DesignationToHours> desigList = designationToHoursRepository.findAll();  
 //all faculty list for that department  
 List<FacultyAcad> allFacs = facultyAcadRepository.findByDepartmentEquals(dept);  
   
 //clear courseList and practicalList tables  
 List<String> facIdList = new ArrayList<String>();  
 for(FacultyAcad f:allFacs) {  
 facIdList.add(f.getUserName());  
 }  
   
 courseListRepository.deleteByFacultyIdList(facIdList);  
 practicalListRepository.deleteByFacultyIdList(facIdList);  
   
 //designation to min max hours hashmap  
 HashMap<String, int[]> desigHours = new HashMap<>();  
   
 for(DesignationToHours d:desigList) {  
 desigHours.put(d.getDesignation(), new int[] {d.getMinLimit(),d.getMaxLimit()});   
 }  
   
 //faculty to min hours  
 //max hours hashmap, faculty to alloted hours hashmap  
 //key: facultyID, value: current\_alloted\_hrs  
 HashMap <String,Integer> facAllotedHours = new HashMap<>();  
   
 //key: facultyID, value: <minLimit, maxLimit>  
 HashMap <String,int[]> facLimits = new HashMap<>();  
   
 //key: facultyID, value: loadLeft (maxLimit - current\_alloted\_hrs)  
 HashMap <String,Integer> facLoadLeft = new HashMap<>();  
 HashMap<String,Integer> facTheoryHours = new HashMap<String,Integer>();  
 HashMap<String,Integer> facPracticalHours = new HashMap<String,Integer>();  
 int totalCourseHours = 0, totalFacLoad = 0;  
   
 for(FacultyAcad f:allFacs) {  
 totalFacLoad+= desigHours.get(f.getDesignation())[1];  
 }  
 //divisions for each year sem  
 List<Divisions> divisionNeeds = divisionsRepository.findByDepartment(dept);  
  
 List<ElectiveBatches> electiveNeeds = electiveBatchesRepository.findByDepartment(dept);  
   
 Collections.sort(divisionNeeds,new DivisionsChainedComparator(new DivisionsYearComparator()));  
   
 //theory courses for that sem  
 ArrayList<CourseList> courseList = new ArrayList<>();  
   
 ArrayList<PracticalList> practicalList = new ArrayList<>();  
 //maps all course ids with their practical courses in prac List   
 HashMap<String,List<Integer>> practicalListPointer = new HashMap<String,List<Integer>>();  
 HashMap<String,List<Integer>> practicalListPointerPerDiv = new HashMap<String,List<Integer>>();  
   
 ArrayList<Course> allTheoryCourses;  
 ArrayList<Course> allTheoryElectiveCourses;  
 ArrayList<Course> allElectivePracticals;  
   
 if(isSemOdd==1) { //odd  
 allTheoryCourses = courseRepository.findOddSemCoursesAndCourseTypeRegAndIsTheoryAndDepartment(dept);  
 allTheoryElectiveCourses = courseRepository.findOddSemCoursesAndCourseTypeNotRegAndIsTheoryAndDepartment(dept);  
 allElectivePracticals = courseRepository.findOddSemCoursesAndCourseTypeNotRegAndIsTheoryNotAndDepartment(dept);  
 }  
 else { //even  
 allTheoryCourses = courseRepository.findEvenSemCoursesAndCourseTypeRegAndIsTheoryAndDepartment(dept);  
 allTheoryElectiveCourses = courseRepository.findEvenSemCoursesAndCourseTypeNotRegAndIsTheoryAndDepartment(dept);  
 allElectivePracticals = courseRepository.findEvenSemCoursesAndCourseTypeNotRegAndIsTheoryNotAndDepartment(dept);  
 }  
   
 int pl = 0;  
   
 // add regular theory courses to courseList and practical courses to practicalList  
 for(Course c:allTheoryCourses)  
 {  
 for(Divisions d:divisionNeeds) {  
   
 if(d.getDepartment().equals(c.getDepartment()) && d.getYear().equals(c.getCourseYear())) {  
 courseList.add(new CourseList(c.getCourseId(),d.getDivId(),"",c.getNoOfHours()));  
 totalCourseHours+=c.getNoOfHours();  
 CompanionCourse cc = courseCompanionRepository.findByCourse(c.getCourseId());  
 if(cc!=null) {  
 //lab  
 for(int i = 1;i<=d.getNoOfBatches();i++) {  
 PracticalList p = new PracticalList();  
 p.setFacultyId("");  
 System.out.println(d.getDivId());  
 p.setDivId(d.getDivId());  
 p.setLabId(d.getDivId().concat(Integer.toString(i)));  
 p.setNoOfHours(courseRepository.findByCourseId(cc.getCompanionCourse()).getNoOfHours());  
 p.setPracticalCourseId(cc.getCompanionCourse());  
 p.setTheoryCourseId(cc.getCourse());  
 if(practicalListPointer.get(cc.getCourse()) == null) {  
 List<Integer> li = new ArrayList<Integer>();  
 li.add(pl);  
 practicalListPointer.put(cc.getCourse(),li);  
 } else {  
 practicalListPointer.get(cc.getCourse()).add(pl);  
 }  
 if(practicalListPointerPerDiv.get(cc.getCourse()+d.getDivId()) == null) {  
 List<Integer> li = new ArrayList<Integer>();  
 li.add(pl);  
 practicalListPointerPerDiv.put(cc.getCourse()+d.getDivId(),li);  
 } else {  
 practicalListPointerPerDiv.get(cc.getCourse()+d.getDivId()).add(pl);  
 }  
 totalCourseHours+=p.getNoOfHours();  
 pl++;  
 practicalList.add(p);  
 }  
 }  
 }  
 }  
 }  
   
 //add elective theory courses to course list   
 for(Course c:allTheoryElectiveCourses) {  
 ArrayList<Electives> allTheoryElectives = electivesRepository.findByCourse(c);  
   
 for(Electives e:allTheoryElectives)  
 {  
 for(ElectiveBatches eb:electiveNeeds) {  
   
 if(e.getElectiveCourseId().equals(eb.getElectiveId()) && eb.getDepartment().equals(c.getDepartment()) && eb.getYear().equals(c.getCourseYear()))  
 {  
 courseList.add(new CourseList(e.getElectiveCourseId(),eb.getBatchId(),"",c.getNoOfHours()));  
 totalCourseHours+=c.getNoOfHours();  
 }  
 }  
   
 }   
 }  
   
 //add elective practical courses to practical list  
 for(Course c: allElectivePracticals) {  
 System.out.println(c.getCourseId());  
// ArrayList<CompanionCourse> ccs = courseCompanionRepository.findByCompanionCourseAndCourseIdInElectiveBatches(c.getCourseId());  
// System.out.println("ccs size "+ccs.size());  
 CompanionCourse cc = courseCompanionRepository.findByCompanionCourse(c.getCourseId());  
 ArrayList<Electives> courseElectives = electivesRepository.findByCourse(courseRepository.findByCourseId(cc.getCourse()));  
// for(CompanionCourse cc:ccs) {  
 for(Divisions d:divisionNeeds) {  
 if(d.getDepartment().equals(c.getDepartment()) && d.getYear().equals(c.getCourseYear())) {  
 for(int i = 1;i<=d.getNoOfBatches();i++) {  
 PracticalList p = new PracticalList();  
 p.setFacultyId("");  
 p.setDivId(d.getDivId());  
 p.setLabId(d.getDivId().concat(Integer.toString(i)));  
 p.setNoOfHours(c.getNoOfHours());  
 p.setPracticalCourseId(c.getCourseId());  
 p.setTheoryCourseId(cc.getCourse());  
 for(Electives e: courseElectives) {  
 if(practicalListPointer.get(e.getElectiveCourseId()) == null) {  
 List<Integer> li = new ArrayList<Integer>();  
 li.add(pl);  
 practicalListPointer.put(e.getElectiveCourseId(),li);  
 }   
 else {  
 practicalListPointer.get(e.getElectiveCourseId()).add(pl);  
 }  
 for(ElectiveBatches eb:electiveNeeds) {  
 if(eb.getElectiveId().equals(e.getElectiveCourseId())) {  
 if(practicalListPointerPerDiv.get(e.getElectiveCourseId()+d.getDivId()) == null) {  
 List<Integer> li = new ArrayList<Integer>();  
 li.add(pl);  
 practicalListPointerPerDiv.put(e.getElectiveCourseId()+eb.getBatchId(),li);  
 } else {  
 practicalListPointerPerDiv.get(e.getElectiveCourseId()+eb.getBatchId()).add(pl);  
 }   
 }  
 }  
   
 }  
 totalCourseHours+=p.getNoOfHours();  
 pl++;  
 practicalList.add(p);  
 }  
 }  
 }  
// }  
 }  
// for(practicalList)  
   
 //more fac hrs than courses  
 if(totalCourseHours < totalFacLoad) {  
 HashMap<String,Integer> facsPerDesig = new HashMap<String,Integer>();  
 for(Entry<String,int[]> entry: desigHours.entrySet()) {  
 int count = facultyAcadRepository.countFacultyByDesignationAndDepartment(entry.getKey(),dept);  
 facsPerDesig.put(entry.getKey(),count);  
 }  
 int cur = 0;  
 cur = totalFacLoad;  
 while(cur>=totalCourseHours) {  
 cur = 0;  
 for(Entry<String,int[]> entry: desigHours.entrySet()) {  
 cur+= facsPerDesig.get(entry.getKey())\*(entry.getValue()[1]-1);  
 desigHours.put(entry.getKey(),new int[]{entry.getValue()[0],entry.getValue()[1]-1});  
 }  
 }  
 for(Entry<String,int[]> entry: desigHours.entrySet()) {  
 desigHours.put(entry.getKey(),new int[]{entry.getValue()[0],entry.getValue()[1]+1});  
 }  
 }  
 else {  
 //less fac hrs than courses  
 HashMap<String,Integer> facsPerDesig = new HashMap<String,Integer>();  
   
 for(Entry<String,int[]> entry: desigHours.entrySet()) {  
 int count = facultyAcadRepository.countFacultyByDesignationAndDepartment(entry.getKey(),dept);  
 facsPerDesig.put(entry.getKey(),count);  
 }  
 int cur = 0;  
 cur = totalFacLoad;  
 while(cur<totalCourseHours) {  
 cur = 0;  
 for(Entry<String,int[]> entry: desigHours.entrySet()) {  
 cur+= facsPerDesig.get(entry.getKey())\*(entry.getValue()[1]+1);  
 desigHours.put(entry.getKey(),new int[]{entry.getValue()[0],entry.getValue()[1]+1});  
 }  
 }  
 }  
   
 for(FacultyAcad f:allFacs) {  
 facAllotedHours.put(f.getUserName(), 0);  
 facLimits.put(f.getUserName(), desigHours.get(f.getDesignation()));  
 facLoadLeft.put(f.getUserName(), desigHours.get(f.getDesignation())[1]);  
 facTheoryHours.put(f.getUserName(), 0);  
 facPracticalHours.put(f.getUserName(), 0);  
 totalFacLoad+= desigHours.get(f.getDesignation())[1];  
 }  
   
 int courseIndex = -1;  
 ArrayList<Integer> nonPreferredCourseIndices = new ArrayList<>();  
 String prevCourse = "";  
 HashMap <String,LinkedHashSet<String>> courseFacs = new HashMap<>();  
   
 for(CourseList c:courseList) {  
   
 courseIndex++;  
 ArrayList<FacultyPref> fpref=new ArrayList<>();  
  
 //allocate unalloted practicals for prevIterated course  
 if(!c.getCourseId().equals(prevCourse) && !prevCourse.equals("")) {  
 List<Integer> pracListptrs = practicalListPointer.get(prevCourse);  
 LinkedHashSet<String> facs = courseFacs.get(prevCourse);  
   
 if(facs!=null && pracListptrs !=null)  
 for(int i: pracListptrs) {  
   
 PracticalList p = practicalList.get(i);  
   
 if(p.getFacultyId().equals("")) {  
 for(String fId: facs) {  
 if(facAllotedHours.get(fId) + p.getNoOfHours() <= facLimits.get(fId)[1]) {  
 p.setFacultyId(fId);  
 //System.out.println(p.getFacultyId());  
 facAllotedHours.replace(fId, facAllotedHours.get(fId) + p.getNoOfHours());  
 facLoadLeft.replace(fId, facLoadLeft.get(fId) - p.getNoOfHours());  
 facPracticalHours.replace(fId, facPracticalHours.get(fId)+p.getNoOfHours());  
 }  
 }  
   
 }  
 }   
 }  
 prevCourse = c.getCourseId();  
 fpref = facultyPrefRepository.findByElectiveId(c.getCourseId());  
 if(fpref.size()==0) {  
 fpref = facultyPrefRepository.findByCourseId(c.getCourseId());  
 }  
   
 if(fpref.size()==0) {  
 //No faculty gave pref for this course  
 nonPreferredCourseIndices.add(courseIndex);  
 continue;  
 }  
   
 Collections.sort(fpref,new FacultyPrefChainedComparator(new FacultyPrefNoComparator(),new FacultyPrefCourseExpComparator(),new FacultyPrefPrereqExp1Comparator(),new FacultyPrefPrereqExp2Comparator()));  
   
// List<Integer> pracListptrs1 = practicalListPointer.get(c.getCourseId());  
 List<Integer> pracListptrs = practicalListPointerPerDiv.get(c.getCourseId()+c.getDivisionId());  
 int pracAllotCounter=0;  
 for (FacultyPref fp : fpref) {  
 if(facAllotedHours.get(fp.getUserName()) >= facLimits.get(fp.getUserName())[1])  
 continue;  
   
 if(c.getFacultyId().equals("")) {  
   
 if(facAllotedHours.get(fp.getUserName()) + c.getNoOfHours() <= facLimits.get(fp.getUserName())[1]) {  
// System.out.println("Alloting course "+c.getCourseId()+" "+c.getDivisionId()+" to "+fp.getUserName());  
 c.setFacultyId(fp.getUserName());  
 facAllotedHours.replace(fp.getUserName(), facAllotedHours.get(fp.getUserName()) + c.getNoOfHours());  
 facLoadLeft.replace(fp.getUserName(), facLoadLeft.get(fp.getUserName()) - c.getNoOfHours());  
 if(courseFacs.get(c.getCourseId())==null) {  
 LinkedHashSet<String> temp = new LinkedHashSet<>();  
 temp.add(fp.getUserName());  
 courseFacs.put(c.getCourseId(), temp);  
 }else {  
 courseFacs.get(c.getCourseId()).add(fp.getUserName());  
 }  
 facTheoryHours.replace(fp.getUserName(), facTheoryHours.get(fp.getUserName())+c.getNoOfHours());  
 }   
 }  
   
 if(facAllotedHours.get(fp.getUserName()) >= facLimits.get(fp.getUserName())[1])  
 continue;  
 if(pracListptrs !=null && pracAllotCounter<pracListptrs.size())  
 for(int i: pracListptrs) {  
   
 PracticalList p = practicalList.get(i);  
// System.out.println("Checking for prac "+p.getTheoryCourseId()+" "+p.getPracticalCourseId()+" "+p.getLabId());  
 if(p.getFacultyId().equals("")) {  
   
 if(facAllotedHours.get(fp.getUserName()) + p.getNoOfHours() <= facLimits.get(fp.getUserName())[1]) {  
 System.out.println("Alloting practical "+p.getPracticalCourseId()+" "+p.getLabId()+" to "+fp.getUserName());  
 pracAllotCounter++;  
 p.setFacultyId(fp.getUserName());  
 facAllotedHours.replace(fp.getUserName(), facAllotedHours.get(fp.getUserName()) + p.getNoOfHours());  
 facLoadLeft.replace(fp.getUserName(), facLoadLeft.get(fp.getUserName()) - p.getNoOfHours());  
 facPracticalHours.replace(fp.getUserName(), facPracticalHours.get(fp.getUserName())+p.getNoOfHours());  
   
 }  
 }  
 }  
 }  
 }  
   
 //allocate remaining pracs for last courseID  
 if(!prevCourse.equals("")) {  
 List<Integer> pracListptrs = practicalListPointer.get(prevCourse);  
 LinkedHashSet<String> facs = courseFacs.get(prevCourse);  
 if(pracListptrs !=null && facs!=null)  
 for(int i: pracListptrs) {  
   
 PracticalList p = practicalList.get(i);  
   
 if(p.getFacultyId().equals("")) {  
 for(String fId: facs) {  
 if(facAllotedHours.get(fId) + p.getNoOfHours() <= facLimits.get(fId)[1]) {  
 p.setFacultyId(fId);  
 //System.out.println(p.getFacultyId());  
 facAllotedHours.replace(fId, facAllotedHours.get(fId) + p.getNoOfHours());  
 facLoadLeft.replace(fId, facLoadLeft.get(fId) - p.getNoOfHours());  
 facPracticalHours.replace(fId, facPracticalHours.get(fId)+p.getNoOfHours());  
 }  
 }  
 }  
 }   
 }  
   
 ArrayList<Integer> temp = new ArrayList<>();  
 int facLimitReached = 0;  
   
 //for each unpreferred course, find fac with lowest load and allocate  
 for(int i: nonPreferredCourseIndices) {  
 //find fac with highest load left and allocate  
 int ll = Integer.MIN\_VALUE;  
 String bestFac = null;  
   
 for (Entry<String, Integer> fac : facLoadLeft.entrySet()) {  
 if(fac.getValue()>ll) {  
 ll=fac.getValue();  
 bestFac = fac.getKey();  
 }  
 }  
 if(bestFac==null) {  
 //No faculty has load left  
 facLimitReached = 1;  
 break;  
 }  
   
 CourseList c = courseList.get(i);  
 if(facLoadLeft.get(bestFac) >= c.getNoOfHours()) {  
 //allocate  
 c.setFacultyId(bestFac);  
 facAllotedHours.replace(bestFac, facAllotedHours.get(bestFac) + c.getNoOfHours());  
 facLoadLeft.replace(bestFac, facLoadLeft.get(bestFac) - c.getNoOfHours());  
 temp.add(i);  
 facTheoryHours.replace(bestFac, facTheoryHours.get(bestFac)+c.getNoOfHours());  
 }  
 List<Integer> pracListptrs = practicalListPointer.get(c.getCourseId());  
 if(pracListptrs !=null)  
 for(int pIndex: pracListptrs) {  
 PracticalList p = practicalList.get(pIndex);  
 if(p.getFacultyId().equals("")) {  
 if(facLoadLeft.get(bestFac) >= p.getNoOfHours()) {  
 p.setFacultyId(bestFac);  
 //System.out.println(p.getFacultyId());  
 facAllotedHours.replace(bestFac, facAllotedHours.get(bestFac) + p.getNoOfHours());  
 facLoadLeft.replace(bestFac, facLoadLeft.get(bestFac) - p.getNoOfHours());  
 facPracticalHours.replace(bestFac, facPracticalHours.get(bestFac)+p.getNoOfHours());  
 }  
 }  
 }   
 }  
   
 //remove alloted non-preferredcourses from list  
 for(int i:temp) {  
 nonPreferredCourseIndices.remove(new Integer(i));  
 }  
   
 //allot to all remaining pracs  
 for(PracticalList p :practicalList) {  
   
 if(p.getFacultyId().equals("")) {  
 String bestFac = null;  
 int ll = Integer.MIN\_VALUE;  
 System.out.println("Finding fac for "+p.getPracticalCourseId());  
 //find fac with highest load left and allocate  
 for (Entry<String, Integer> fac : facLoadLeft.entrySet()) {  
 if(fac.getValue()>ll) {  
 ll=fac.getValue();  
 bestFac = fac.getKey();  
 }  
 }  
 if(bestFac==null) {  
 facLimitReached=1;  
 break;  
 }  
 if(facLoadLeft.get(bestFac) >= p.getNoOfHours()) {  
 p.setFacultyId(bestFac);  
 facAllotedHours.replace(bestFac, facAllotedHours.get(bestFac) + p.getNoOfHours());  
 facLoadLeft.replace(bestFac, facLoadLeft.get(bestFac) - p.getNoOfHours());  
 facPracticalHours.replace(bestFac, facPracticalHours.get(bestFac)+p.getNoOfHours());  
 }  
 else {  
 //increment bestFac max limit  
 System.out.println("Incrementing for "+bestFac);  
 p.setFacultyId(bestFac);  
 int newHrs = facAllotedHours.get(bestFac) + p.getNoOfHours();  
 facAllotedHours.replace(bestFac, newHrs);  
 facLoadLeft.replace(bestFac, 0);  
 facPracticalHours.replace(bestFac, facPracticalHours.get(bestFac)+p.getNoOfHours());  
 facLimits.put(bestFac, new int[] {facLimits.get(bestFac)[0],newHrs});  
 System.out.println("Hours "+newHrs);  
 }  
 }  
 }  
 if(facLimitReached==1) {  
 System.out.println("All facs have reached limit");  
 }  
   
 System.out.println("FacGroups");  
 for (Entry<String, LinkedHashSet<String>> fac : courseFacs.entrySet()) {  
 System.out.println(fac.getKey() + " "+ fac.getValue().toString() );  
 }  
 System.out.println("course list");  
 System.out.println("courseID divisionID facID noOfHrs");  
 for(CourseList c:courseList) {  
 courseListRepository.save(c);  
 if(c.getFacultyId().equals("")) {  
 System.out.println(c.getCourseId()+" "+c.getDivisionId()+" NA "+c.getNoOfHours());  
 }  
 else {  
 System.out.println(c.getCourseId()+" "+c.getDivisionId()+" "+c.getFacultyId()+" "+c.getNoOfHours());  
 }  
 }  
   
 System.out.println("practical list");  
 System.out.println("courseID labID practicalID facID noOfHrs");  
   
 for(PracticalList p:practicalList) {  
 practicalListRepository.save(p);  
 if(p.getFacultyId().equals(""))  
 System.out.println(p.getTheoryCourseId()+" "+p.getLabId()+" "+p.getPracticalCourseId()+" NA "+p.getNoOfHours());  
 else  
 System.out.println(p.getTheoryCourseId()+" "+p.getLabId()+" "+p.getPracticalCourseId()+" "+p.getFacultyId()+" "+p.getNoOfHours());  
 }  
   
 FacultyAllotedHours fh = new FacultyAllotedHours();  
 System.out.println("\n\nFac Details");  
 System.out.println("FacID facMax facAlloted facLoadLeft");  
 int totalLoadLeft=0,totalLoadAlloted=0,maxLoad=0;  
 for (Entry<String, int[]> fac : facLimits.entrySet()) {  
 System.out.println(fac.getKey() + " "+ fac.getValue()[1] + " "+ facAllotedHours.get(fac.getKey()) + " " + facLoadLeft.get(fac.getKey()) );  
 totalLoadLeft+=facLoadLeft.get(fac.getKey());  
 maxLoad+=fac.getValue()[1];  
 totalLoadAlloted+=facAllotedHours.get(fac.getKey());  
 fh.setFacultyId(fac.getKey());  
 fh.setMaxHours(fac.getValue()[1]);  
 fh.setAllotedHours(facAllotedHours.get(fac.getKey()));  
 fh.setTheoryHours(facTheoryHours.get(fac.getKey()));  
 fh.setPracticalHours(facPracticalHours.get(fac.getKey()));  
 facultyAllotedHoursRepository.save(fh);  
 }  
   
 System.out.println("Total hrs to be alloted: "+totalCourseHours);  
 System.out.println("MaxLoad: "+maxLoad+" TotalLoadAlloted: "+totalLoadAlloted+" TotalLoadLeft: "+totalLoadLeft);  
   
 }  
   
 @GetMapping("/getfacultyAllocationPage")  
 public ModelAndView getFacAllocationPage() {  
 ModelAndView model = new ModelAndView();  
 model.setViewName("admin/facultyAllocation");  
 departments = departmentRepository.findAll();  
 model.addObject("departments",departments);  
 return model;  
 }  
   
 @RequestMapping(value="/showFacultyAllocation", method=RequestMethod.POST)  
 public ModelAndView showFacultyAllocation(String dept) {  
   
 ModelAndView model = new ModelAndView();  
 model.setViewName("admin/facultyAllocation");  
 departments = departmentRepository.findAll();  
 model.addObject("departments",departments);  
   
 Department d = departmentRepository.findByDeptId(dept);  
 List<FacultyAllotedHours> facs = facultyAllotedHoursRepository.findFacsByDepartment(d);  
 if(facs.isEmpty()) {  
 model.addObject("err\_msg","Allocation has not been performed for "+d.getDeptName());  
 }  
 else {  
 model.addObject("facAllotmentList",generateFacultyAllotedList(d));  
 }  
 return model;  
 }   
   
 public List<FacultyAllocations> generateFacultyAllotedList(Department dept) {  
 List<FacultyAllotedHours> facs = facultyAllotedHoursRepository.findFacsByDepartment(dept);  
 List<CourseList> courses = courseListRepository.findByFacultyIdDepartment(dept);  
 List<PracticalList> practicals = practicalListRepository.findByFacultyIdDepartment(dept);  
 rs = new ArrayList<FacultyAllocations>();  
   
 //{key: facID, value:{key:courseId,value:courseList} }  
 HashMap<String,List<CourseList>> facCourses = new HashMap<String,List<CourseList>>();  
 HashMap<String,List<PracticalList>> facPracticalCourses = new HashMap<String,List<PracticalList>>();  
 HashMap<String,Integer> facTheoryHours = new HashMap<String,Integer>();  
 HashMap<String,Integer> facPracticalHours = new HashMap<String,Integer>();  
   
 //init all hashmaps  
 for(FacultyAllotedHours f: facs) {  
 List<CourseList> cList = new ArrayList<CourseList>();  
 facCourses.put(f.getFacultyId(),cList);  
 facTheoryHours.put(f.getFacultyId(),0);  
  
 List<PracticalList> pList = new ArrayList<PracticalList>();  
 facPracticalCourses.put(f.getFacultyId(), pList);  
 facPracticalHours.put(f.getFacultyId(),0);  
 }  
   
 for(CourseList c:courses) {  
   
 facCourses.get(c.getFacultyId()).add(c);  
 facTheoryHours.replace(c.getFacultyId(),facTheoryHours.get(c.getFacultyId())+c.getNoOfHours());  
 }  
   
 for(PracticalList p: practicals) {  
   
 facPracticalCourses.get(p.getFacultyId()).add(p);  
 facPracticalHours.replace(p.getFacultyId(),facPracticalHours.get(p.getFacultyId())+p.getNoOfHours());  
   
 }  
   
 for(FacultyAllotedHours f: facs) {  
 Users faculty = userDetails.findByUserName(f.getFacultyId());  
 FacultyAllocations fa = new FacultyAllocations();  
 fa.setName(faculty.getFirstName() + " "+ faculty.getLastName());  
 fa.setAllotedLoad(f.getAllotedHours());  
 fa.setFacultyId(f.getFacultyId());  
 fa.setMaxLoad(f.getMaxHours());  
 fa.setCourseAndDivs(facCourses.get(f.getFacultyId()));  
 fa.setPracticalsAndBatches(facPracticalCourses.get(f.getFacultyId()));  
 fa.setPracticalHours(f.getPracticalHours());  
 fa.setTheoryHours(f.getTheoryHours());  
 ArrayList<String> courseNames = new ArrayList<>();  
 ArrayList<String> pracNames = new ArrayList<>();  
   
 for(CourseList c:facCourses.get(f.getFacultyId())) {  
 Course tempc = courseRepository.findByCourseId(c.getCourseId());  
 if(tempc!=null) {  
 courseNames.add(tempc.getCourseName());  
 }  
 else{  
 Electives tempe = electivesRepository.findByElectiveCourseId(c.getCourseId());  
 courseNames.add(tempe.getElectiveName());  
 }  
 }  
   
 for(PracticalList p:facPracticalCourses.get(f.getFacultyId())) {  
 Course tempc = courseRepository.findByCourseId(p.getPracticalCourseId());  
 System.out.println(p.getPracticalCourseId()+" "+p.getTheoryCourseId());  
 if(tempc!=null) {  
 pracNames.add(tempc.getCourseName());  
 }  
 }  
   
 fa.setCourses(courseNames);  
 fa.setPracticals(pracNames);  
 rs.add(fa);  
 }  
 return rs;  
 }  
   
 @RequestMapping(value="/getFacultyAllocationByIndex", method=RequestMethod.GET)  
 public String getFacultyAllocationByIndex(Model model,Integer i) {  
 model.addAttribute("facultyAllocation",rs.get(i-1));  
 return "admin/facultyAllocation :: viewDetailsDiv";  
 }  
   
 public static String convertTo24HoursFormat(String twelveHourTime)  
 throws ParseException {  
 return TWENTY\_FOUR\_TF.format(  
 TWELVE\_TF.parse(twelveHourTime));  
 }  
   
  
 @Transactional  
 String readTT(String path, String dept,String day) {  
   
 Department department = departmentRepository.findByDeptId(dept);  
   
 HashMap<Integer,Time[]> timeSlots = new HashMap<>();  
   
 File myFile = new File(path);  
 FileInputStream fis;  
  
 List<TimeTable> timetable = new ArrayList<>();  
   
 timeTableRepository.deleteByDepartmentAndDay(department,day);  
  
 String msg = "Time Table has been uploaded successfully.";  
   
 try {  
 fis = new FileInputStream(myFile);  
 // Finds the workbook instance for XLSX file  
 XSSFWorkbook myWorkBook;  
 myWorkBook = new XSSFWorkbook (fis);  
   
 // Return first sheet from the XLSX workbook  
 XSSFSheet mySheet = myWorkBook.getSheetAt(0);  
   
   
 Row row0 = mySheet.getRow(0);  
   
 Iterator<Cell> c = row0.cellIterator();  
 c.next();  
 while(c.hasNext()) {  
 Cell cNext = c.next();  
   
 String arr[] = cNext.getStringCellValue().trim().split(" to ");  
 String startTime = convertTo24HoursFormat(arr[0]);  
 String endTime = convertTo24HoursFormat(arr[1]);  
 DateFormat formatter = new SimpleDateFormat("HH:mm");  
 timeSlots.put(cNext.getColumnIndex(),new Time[] {new Time(formatter.parse(startTime).getTime()),new Time(formatter.parse(endTime).getTime())});  
 }  
   
 // Get iterator to all the rows in current sheet  
 Iterator<Row> rowIterator = mySheet.iterator();  
   
 rowIterator.next();  
   
 // Traversing over each row of XLSX file  
 while (rowIterator.hasNext()) {  
 Row row = rowIterator.next();  
   
 Cell cprev = row.cellIterator().next();  
 String div = cprev.getStringCellValue().trim();  
   
 if(cprev.getStringCellValue().equals("END") || cprev.getStringCellValue().equals("") || cprev.getCellType()==Cell.CELL\_TYPE\_BLANK) {  
 break;  
 }  
   
 for(int i:timeSlots.keySet()) {  
   
 Time[] slot = timeSlots.get(i);  
 if(row.getCell(i)==null) {  
   
 }  
 else if(row.getCell(i).getStringCellValue().equals("") || row.getCell(i).getCellType()==Cell.CELL\_TYPE\_BLANK) {  
   
 }else {  
 String delim = " \n\t";  
 StringTokenizer st= new StringTokenizer(row.getCell(i).getStringCellValue().trim(),delim);  
 int j=1;  
 while(st.hasMoreTokens()) {  
 String activityName = "Time Table";  
 if(j==1) {  
 activityName = st.nextToken().trim();  
 activityName = activityName.concat(" ("+div+")");  
 j++;  
 }  
 if(j==2) {  
   
 String str = st.nextToken().trim();  
 if(str.contains(",")) {  
 String[] temp = str.split(",");  
 for(String temps:temp) {  
 Resource r = resourceRepository.findByResourceId(dept.concat(temps));  
 timetable.add(new TimeTable(slot[0], slot[1], r, day, department,r.getResourceIncharge(),activityName));  
 }  
 }else {  
 Resource r = resourceRepository.findByResourceId(dept.concat(str));  
 System.out.println("resource-"+r.getResourceId());  
 timetable.add(new TimeTable(slot[0], slot[1], r, day, department,r.getResourceIncharge(),activityName));  
 }  
 j=0;  
 }else {  
 if(st.hasMoreTokens()) {st.nextToken();}  
 j++;  
 }  
 }  
 }  
 }  
 }  
   
 for(TimeTable tt:timetable) {  
 timeTableRepository.save(tt);  
 }  
   
 myWorkBook.close();  
   
 } catch (Exception e) {  
 e.printStackTrace();  
 msg = "Uploaded timetable is in incorrect format.";  
 myFile.delete();  
 return msg;  
 }  
 myFile.delete();  
 return msg;  
 }   
   
  
 @RequestMapping(value = "/viewSchedule",method=RequestMethod.GET)  
 public ModelAndView getViewSchedule() {  
 ModelAndView model = new ModelAndView();  
 ArrayList<Department> depts = bookingsService.listDepartments();  
 model.addObject("departments", depts);  
   
 model.setViewName("/admin/viewSchedule");  
 return model;  
 }  
   
 @RequestMapping(value="/getScheduleForResource", method=RequestMethod.GET)  
 public String getScheduleForResource(Model model,String getTT,String cur\_date){  
   
 List<TimeSlots> list = bookingsService.getTimeSlotsForDate(cur\_date, getTT);  
   
 if(list.size()==0) {  
 model.addAttribute("msg","All slots are empty!");   
 return "admin/viewSchedule:: messageDiv";  
 }else {  
 model.addAttribute("ttForResource",list);  
 return "admin/viewSchedule:: resourceTT";  
 }  
 }  
   
 @RequestMapping(value="/getTTForResourceForDate",method=RequestMethod.POST)  
 public String getTTForResourceForDate(Model model,String booking\_date,String getTT){  
 System.out.println("hello :)"+booking\_date+getTT);  
   
 List<TimeSlots> list = bookingsService.getTimeSlotsForDate(booking\_date, getTT);  
   
 if(list.isEmpty()) {  
 model.addAttribute("msg","All slots are empty!");  
 return "admin/viewSchedule:: messageDiv";  
 }else {  
 model.addAttribute("ttForResource",list);  
 return "admin/viewSchedule:: resourceTT";  
 }  
 }  
   
 @RequestMapping(value="/getViewOptions",method=RequestMethod.POST)  
 public String getViewOptions(Model model,String dept,String rType,Integer minSeats) {  
   
 if(minSeats==null) {  
 minSeats=0;  
 }  
 ArrayList<Resource> options = bookingsService.listResourcesByDepartmentAndRTypeAndMinSeats(dept, rType, minSeats);  
   
 if(options.isEmpty()) {  
 model.addAttribute("err\_msg","No suitable rooms/halls/classrooms were found.");  
 return "admin/viewSchedule:: messageDiv";  
 }else {  
 model.addAttribute("options",options);  
 return "admin/viewSchedule:: resourceOptionsTable";  
 }  
 }  
  
   
}

# spring\resource\FacultyController.java

package com.qrms.spring.resource;  
  
import java.sql.Date;  
import java.sql.Time;  
import java.time.LocalDate;  
import java.time.LocalDateTime;  
import java.time.format.DateTimeFormatter;  
import java.util.ArrayList;  
import java.util.List;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.data.jpa.repository.Modifying;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.stereotype.Controller;  
import org.springframework.transaction.annotation.Transactional;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestMethod;  
import org.springframework.web.bind.annotation.ResponseBody;  
import org.springframework.web.servlet.ModelAndView;  
  
import com.qrms.spring.model.FacultyPref;  
import com.qrms.spring.model.OpenFacultyPrefs;  
import com.qrms.spring.model.PracticalList;  
import com.qrms.spring.model.Resource;  
import com.qrms.spring.model.ResourceRequests;  
import com.qrms.spring.model.TimeSlots;  
import com.qrms.spring.model.TimeTable;  
import com.qrms.spring.model.Course;  
import com.qrms.spring.model.CourseList;  
import com.qrms.spring.model.CoursePrerequisites;  
import com.qrms.spring.model.Department;  
import com.qrms.spring.model.Electives;  
import com.qrms.spring.model.FacultyAcad;  
import com.qrms.spring.model.FacultyAllotedHours;  
import com.qrms.spring.model.Users;  
import com.qrms.spring.queryBeans.CourseAndElectives;  
import com.qrms.spring.queryBeans.FacPrefsList;  
import com.qrms.spring.queryBeans.FacultyAllocations;  
import com.qrms.spring.repository.CourseListRepository;  
import com.qrms.spring.repository.CoursePrerequisitesRepository;  
import com.qrms.spring.repository.CourseRepository;  
import com.qrms.spring.repository.ElectivesRepository;  
import com.qrms.spring.repository.FacultyPrefRepository;  
import com.qrms.spring.repository.OpenFacultyPrefsRepository;  
import com.qrms.spring.repository.PracticalListRepository;  
import com.qrms.spring.repository.ResourceRepository;  
import com.qrms.spring.repository.ResourceRequestsRepository;  
import com.qrms.spring.repository.TimeSlotsRepository;  
import com.qrms.spring.repository.TimeTableRepository;  
import com.qrms.spring.service.BookingsServiceImpl;  
import com.qrms.spring.service.EmailServiceImpl;  
import com.qrms.spring.repository.FacultyAcadRepository;  
import com.qrms.spring.repository.FacultyAllotedHoursRepository;  
  
  
@Controller  
@RequestMapping("/u/faculty")  
public class FacultyController {  
   
 @Autowired  
 private FacultyPrefRepository facultyPrefRepository;  
   
 @Autowired  
 private FacultyAcadRepository facultyAcadRepository;  
   
 @Autowired  
 private ElectivesRepository electivesRepository;  
   
 @Autowired  
 private CourseRepository courseRepository;  
   
 @Autowired  
 private CoursePrerequisitesRepository coursePrerequisitesRepository;  
   
 @Autowired  
 private BookingsServiceImpl bookingsService;  
   
 @Autowired  
 private OpenFacultyPrefsRepository openFacultyPrefsRepository;  
  
 @Autowired  
 private EmailServiceImpl emailServiceImpl;  
   
 @Autowired  
 private ResourceRepository resourceRepository;  
   
 @Autowired  
 private ResourceRequestsRepository resourceRequestsRepository;  
   
 @Autowired  
 private TimeSlotsRepository timeSlotsRepository;  
   
 @Autowired  
 private TimeTableRepository timeTableRepository;  
   
 @Autowired  
 private CourseListRepository courseListRepository;  
   
 @Autowired  
 private PracticalListRepository practicalListRepository;  
   
 @Autowired  
 private FacultyAllotedHoursRepository facultyAllotedHoursRepository;  
  
 @Value("${spring.mail.username}")  
 private String qrmsEmailId;  
   
 @GetMapping("/home")  
 public ModelAndView facultyHome() {  
 ModelAndView model = new ModelAndView();  
 Users user = (Users)SecurityContextHolder.getContext().getAuthentication().getPrincipal();  
 String userName = user.getUserName();  
   
 FacultyAcad facultyProfile = facultyAcadRepository.findByUserName(userName);  
 if(facultyProfile!=null)  
   
 {  
 model.addObject("facultyProfile", facultyProfile);  
   
 FacultyAllocations fa = new FacultyAllocations();  
   
 FacultyAllotedHours fac = facultyAllotedHoursRepository.findByFacultyId(userName);  
   
 if(fac!=null) {  
 List<CourseList> courses = courseListRepository.findByFacultyId(userName);  
 List<PracticalList> practicals = practicalListRepository.findByFacultyId(userName);  
   
 List<String> courseNames = new ArrayList<>();  
 List<String> pracNames = new ArrayList<>();  
   
 int facTheoryHours = 0;  
 for(CourseList c:courses) {  
 facTheoryHours += c.getNoOfHours();  
 Course tempc = courseRepository.findByCourseId(c.getCourseId());  
 if(tempc!=null) {  
 courseNames.add(tempc.getCourseName());  
 }  
 else{  
 Electives tempe = electivesRepository.findByElectiveCourseId(c.getCourseId());  
 courseNames.add(tempe.getElectiveName());  
 }  
 }  
   
 int practicalTheoryHours = 0;  
 for(PracticalList p:practicals) {  
 practicalTheoryHours += p.getNoOfHours();  
 Course tempc = courseRepository.findByCourseId(p.getPracticalCourseId());  
 System.out.println(p.getPracticalCourseId()+" "+p.getTheoryCourseId());  
 if(tempc!=null) {  
 pracNames.add(tempc.getCourseName());  
 }  
 }  
   
 fa.setAllotedLoad(fac.getAllotedHours());  
 fa.setFacultyId(fac.getFacultyId());  
 fa.setMaxLoad(fac.getMaxHours());  
 fa.setCourseAndDivs(courses);  
 fa.setPracticalsAndBatches(practicals);  
 fa.setPracticalHours(practicalTheoryHours);  
 fa.setTheoryHours(facTheoryHours);  
 fa.setPracticals(pracNames);  
 fa.setCourses(courseNames);  
 model.addObject("facultyAllocation", fa);  
 }  
   
 }  
   
 model.setViewName("/faculty/home");  
   
 return model;  
 }  
   
 @RequestMapping(value = "/givePreferenceChoice", method = RequestMethod.GET)  
 public ModelAndView facultyPref(String msg) {  
 ModelAndView model = new ModelAndView();  
   
 Users user = (Users)SecurityContextHolder.getContext().getAuthentication().getPrincipal();  
 String userName = user.getUserName();  
   
 FacultyAcad currUserAcad = facultyAcadRepository.findByUserName(userName);  
   
 OpenFacultyPrefs ofp = openFacultyPrefsRepository.findByDeptId(currUserAcad.getDepartment().getDeptId());  
  
 if(ofp==null) {  
 model.addObject("err\_msg\_1","Preference forms are closed for "+currUserAcad.getDepartment().getDeptName());  
 model.setViewName("/faculty/facultyPref");  
 return model;  
 }  
 ArrayList <FacultyPref> facultyPrefs = facultyPrefRepository.findByUserName(currUserAcad.getUserName());  
 if(facultyPrefs.size()!=0)   
 {  
 model.addObject("err\_msg\_1","You have already submitted your preferences. Please wait until allocation process takes place.");  
 }  
 model.addObject("semType",ofp.getSemType());  
 model.setViewName("/faculty/facultyPref");  
 return model;  
 }  
   
 @RequestMapping(value = "/getFacPrefForm", method = RequestMethod.GET)  
 public String getFacPrefForm(Model model, String year) {  
 Users user = (Users)SecurityContextHolder.getContext().getAuthentication().getPrincipal();  
 String userName = user.getUserName();  
   
 FacultyAcad currUserAcad = facultyAcadRepository.findByUserName(userName);  
   
 //ArrayList <FacultyPref> facultyPrefs = facultyPrefRepository.findByUserNameAndYear(currUserAcad.getUserName(),year);  
 OpenFacultyPrefs ofp = openFacultyPrefsRepository.findByDeptId(currUserAcad.getDepartment().getDeptId());  
 System.out.println(ofp.getSemType());  
 ArrayList<CourseAndElectives> resultSet = new ArrayList<CourseAndElectives>() ;  
 ArrayList<Course> regCourses,elCourses;  
 if(ofp.getSemType() == 0) {  
 regCourses = courseRepository.findEvenSemCoursesAndCourseTypeRegAndIsTheoryAndDepartmentAndCourseYear(currUserAcad.getDepartment(),year);  
 elCourses = courseRepository.findEvenSemCoursesAndCourseTypeNotRegAndIsTheoryAndDepartmentAndCourseYear(currUserAcad.getDepartment(),year);  
   
 }else {  
 regCourses = courseRepository.findOddSemCoursesAndCourseTypeRegAndIsTheoryAndDepartmentAndCourseYear(currUserAcad.getDepartment(),year);  
 elCourses = courseRepository.findOddSemCoursesAndCourseTypeNotRegAndIsTheoryAndDepartmentAndCourseYear(currUserAcad.getDepartment(),year);  
   
 }  
 //change later when admin gives current Sem input  
   
 if(regCourses.isEmpty() && elCourses.isEmpty())  
 {  
 model.addAttribute("err\_msg","No courses found for the selected year");  
 return "faculty/facultyPref :: messageDiv";  
 }  
 else {  
 for(Course elCourse: elCourses) {  
 ArrayList<Electives> electives = electivesRepository.findByCourse(elCourse);  
   
 for(Electives el: electives) {  
 CoursePrerequisites cp = coursePrerequisitesRepository.findByCourseId(el.getElectiveCourseId());  
 CourseAndElectives ce = new CourseAndElectives();  
  
 if(cp == null) {  
 ce.setPreReq1("NA");  
 ce.setPreReq2("NA");  
 }  
 else {  
 String prereq1,prereq2;  
   
 if(cp.getIsPrereq1Elective() == 1)  
 prereq1 = electivesRepository.findByElectiveCourseId(cp.getPrerequisiteNo1()).getElectiveName();   
 else if(cp.getIsPrereq1Elective() == 0)  
 prereq1 = courseRepository.findByCourseId(cp.getPrerequisiteNo1()).getCourseName();  
 else  
 prereq1 = "NA";   
   
 if(cp.getIsPrereq2Elective() == 1)  
 prereq2 = electivesRepository.findByElectiveCourseId(cp.getPrerequisiteNo2()).getElectiveName();   
 else if(cp.getIsPrereq2Elective() == 0)  
 prereq2 = courseRepository.findByCourseId(cp.getPrerequisiteNo2()).getCourseName();  
 else  
 prereq2="NA";  
   
 ce.setPreReq1(prereq1);  
 ce.setPreReq2(prereq2);   
 }  
   
 ce.setCourse(elCourse);  
 ce.setElective(el);  
   
 resultSet.add(ce);  
   
 }   
 }  
 for(Course regCourse: regCourses) {  
 CourseAndElectives ce = new CourseAndElectives();  
 CoursePrerequisites cp = coursePrerequisitesRepository.findByCourseId(regCourse.getCourseId());  
 if(cp == null) {  
 ce.setPreReq1("NA");  
 ce.setPreReq2("NA");  
 }  
 else {  
 String prereq1,prereq2;  
   
 if(cp.getIsPrereq1Elective() == 1)  
 prereq1 = electivesRepository.findByElectiveCourseId(cp.getPrerequisiteNo1()).getElectiveName();   
 else if(cp.getIsPrereq1Elective() == 0)  
 prereq1 = courseRepository.findByCourseId(cp.getPrerequisiteNo1()).getCourseName();  
 else  
 prereq1="NA";  
 if(cp.getIsPrereq2Elective() == 1)  
 prereq2 = electivesRepository.findByElectiveCourseId(cp.getPrerequisiteNo2()).getElectiveName();   
 else if(cp.getIsPrereq2Elective() == 0)  
 prereq2 = courseRepository.findByCourseId(cp.getPrerequisiteNo2()).getCourseName();   
 else  
 prereq2="NA";  
   
 ce.setPreReq1(prereq1);  
 ce.setPreReq2(prereq2);   
 }  
 ce.setCourse(regCourse);   
 resultSet.add(ce);   
 }  
 }  
 model.addAttribute("resultSet",resultSet);  
 return "faculty/facultyPref :: selectPreferenceFragment";  
 }  
   
 @ResponseBody  
 @RequestMapping(value = "/givePreference", method = RequestMethod.GET)  
 public String givePreference(Model model, String selectPref, int courseExp, int prereq1Exp, int prereq2Exp) {  
 return selectPref;  
 }  
   
 //--------------------------------------------------------------------------------------  
 //MODIFY THIS  
 //--------------------------------------------------------------------------------------  
 @ResponseBody  
 @RequestMapping(value = "/setFacPrefs", method = RequestMethod.POST)  
 public String setFacPreferences(Model model, @RequestBody FacPrefsList facultyPrefs ) {  
 Users user = (Users)SecurityContextHolder.getContext().getAuthentication().getPrincipal();  
 String userName = user.getUserName();  
   
 List<FacultyPref> prefArray = facultyPrefs.getFacultyPrefs();  
   
 for(FacultyPref fp : prefArray) {  
 fp.setUserName(userName);  
   
 Course c = courseRepository.findByCourseId(fp.getCourseId());  
 if(c==null)  
 {  
 fp.setElectiveId(fp.getCourseId());  
 fp.setCourseId(null);  
 }  
 else  
 {  
 fp.setElectiveId(null);  
 }  
 facultyPrefRepository.save(fp);  
 }  
   
 return "success";  
   
 }  
   
 /\*  
 @RequestMapping(value = "/getFacultyPrefs", method = RequestMethod.GET)  
 public ModelAndView studentPref(String year) {  
 ModelAndView model = new ModelAndView();  
   
 Users user = (Users)SecurityContextHolder.getContext().getAuthentication().getPrincipal();  
 String userName = user.getUserName();  
 System.out.println(userName);  
   
 FacultyAcad currUserAcad = facultyAcadRepository.findByUserName(userName);  
   
 ArrayList <StudentPref> facultyPrefs = facultyPrefRepository.findByUserNameAndYear(currUserAcad.getUserName(),year);  
 if(facultyPrefs.size()!=0)   
 return facultyPref("Your preferences for chosen elective have been recorded already!");  
 else {  
 ArrayList<Course> chosen\_year\_courses = courseRepository.findByCourseYear(year);  
// ArrayList<Electives> electiveList = electivesRepository.findByCourse(chosen\_year);   
// if(electiveList.size()==0) {  
// System.out.println("No courses exist");  
// String msg = "Please choose other elective-id!";  
// return getElectiveId(msg);  
// }  
   
// ArrayList<Course> elective\_ids = courseRepository.findByCourseSemAndCourseYearAndCourseTypeNotAndDepartmentAndIsTheoryAndStudAllocFlag(currUserAcad.getSem(),currUserAcad.getYear(),'R',currUserAcad.getDepartment(),1,1);  
// model.addObject("chosen\_course\_name",chosen\_course.getCourseName());  
// model.addObject("chosen\_course\_id",chosen\_course.getCourseId());  
// model.addObject("elective\_ids",elective\_ids);  
// model.addObject("studentPref",new StudentPref());  
// model.addObject("courseList", electiveList);  
// model.setViewName("student/studentPref");  
 return model;  
 }  
   
   
 }\*/  
   
   
 @RequestMapping(value="/bookings",method=RequestMethod.GET)  
 public ModelAndView getRequirements() {  
 ModelAndView model = new ModelAndView();  
 model.setViewName("/faculty/bookings");  
 ArrayList<Department> depts = bookingsService.listDepartments();  
 model.addObject("departments", depts);  
 return model;  
 }  
   
 @RequestMapping(value="/getOptions",method=RequestMethod.POST)  
 public String setRequirements(Model model,String dept,String rType,Integer minSeats) {  
   
 if(minSeats==null) {  
 minSeats=0;  
 }  
 ArrayList<Resource> options = bookingsService.listResourcesByDepartmentAndRTypeAndMinSeats(dept, rType, minSeats);  
   
 if(options.isEmpty()) {  
 model.addAttribute("err\_msg","No suitable rooms/halls/classrooms were found.");  
 return "faculty/bookings:: messageDiv";  
 }  
 else {  
 model.addAttribute("options",options);  
 return "faculty/bookings:: resourceOptionsTable";  
 }  
 }  
   
 @RequestMapping(value="/getTTForResource", method=RequestMethod.GET)  
 public String getTTForResource(Model model,String getTT,String cur\_date){  
   
 List<TimeSlots> list = bookingsService.getTimeSlotsForDate(cur\_date, getTT);  
   
 if(list.size()==0) {  
 model.addAttribute("msg","All slots are empty!");  
 return "faculty/bookings:: messageDiv";  
 }else {  
 model.addAttribute("ttForResource",list);  
 return "faculty/bookings:: resourceTT";  
 }  
 }  
   
 @RequestMapping(value="/getTTForResourceForDate",method=RequestMethod.POST)  
 public String getTTForResourceForDate(Model model,String booking\_date,String getTT){  
 List<TimeSlots> list = bookingsService.getTimeSlotsForDate(booking\_date, getTT);  
   
 if(list.isEmpty()) {  
 model.addAttribute("msg","All slots are empty!");  
 return "faculty/bookings:: messageDiv";  
 }else {  
 model.addAttribute("ttForResource",list);  
 return "faculty/bookings:: resourceTT";  
 }  
   
 }  
   
   
 @RequestMapping(value="/sendBookingRequest",method=RequestMethod.POST)  
 public String sendBookingRequest(Model model,String booking\_date,String resource,String startTime,String endTime,String activityName) {  
   
 LocalDateTime reqTime = LocalDateTime.now();  
 String reqGenTime = reqTime.getYear()+"/"+reqTime.getMonthValue()+"/"+reqTime.getDayOfMonth()+" "+reqTime.getHour()+":"+reqTime.getMinute()+":"+reqTime.getSecond();  
   
 Users user = (Users)SecurityContextHolder.getContext().getAuthentication().getPrincipal();  
 String userName = user.getUserName();  
   
 FacultyAcad requestingFaculty = facultyAcadRepository.findByUserName(userName);  
   
 Resource resourceObj = resourceRepository.findByResourceId(resource);  
 String resourceIncharge = resourceObj.getResourceIncharge().getUserDets().getEmail();  
   
 String body;  
 body = "Request BY: "+requestingFaculty.getUserDets().getFirstName()+" "+requestingFaculty.getUserDets().getLastName()+"\n";  
 body = body.concat("Resource: "+resource+"\n");  
 body = body.concat("Slot time: "+startTime+" - "+endTime+" on "+booking\_date+" for the activity \""+activityName+"\""+"\n");  
 body = body.concat("Request generated on - "+reqGenTime+"\n\n\n");  
 body = body.concat("Please login to your QRMS account to manage the requests!\n\nRegards,\nQRMS Team.");  
  
 DateTimeFormatter df = DateTimeFormatter.ofPattern("yyyy-MM-dd");  
 LocalDate slotDate = LocalDate.parse(booking\_date, df);  
 Date slotSqlDate = java.sql.Date.valueOf(slotDate.toString());  
 String day = slotDate.getDayOfWeek().name();  
   
 LocalDate requestDate = LocalDate.parse(reqTime.getYear()+"-"+String.format("%02d", reqTime.getMonthValue())+"-"+String.format("%02d", reqTime.getDayOfMonth()), df);  
 Date requestedSqlDate = java.sql.Date.valueOf(requestDate.toString());  
   
 ResourceRequests resourceRequest = new ResourceRequests();  
 resourceRequest.setSlotActivityName(activityName);  
 resourceRequest.setSlotDate(slotSqlDate);  
 resourceRequest.setSlotDay(day);  
 resourceRequest.setSlotStartTime(Time.valueOf(startTime+":00"));  
 resourceRequest.setSlotEndTime(Time.valueOf(endTime+":00"));  
 resourceRequest.setRequestBy(requestingFaculty);  
 resourceRequest.setResourceId(resourceObj);  
 resourceRequest.setRequestedDate(requestedSqlDate);  
 resourceRequest.setRequestTime(Time.valueOf(reqTime.getHour()+":"+reqTime.getMinute()+":"+reqTime.getSecond()));  
   
 resourceRequestsRepository.save(resourceRequest);  
   
 try {  
 emailServiceImpl.send(qrmsEmailId, "bmk15897@gmail.com", "QRMS: Request to book resource "+resource, body);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }   
   
 model.addAttribute("msg","Sent request to the resource incharge, the updates will be mailed to you soon!");  
 return "faculty/bookings:: messageDiv";  
 }  
   
 @Transactional  
 @Modifying  
 @RequestMapping(value="/resourceRequests",method=RequestMethod.GET)  
 public ModelAndView getResourceRequests() {  
   
 ModelAndView model = new ModelAndView();  
   
 LocalDate localDate = LocalDate.now();  
 Date sqlDate = Date.valueOf(localDate.toString());  
   
 long curTime = new java.util.Date().getTime();  
   
 Time t = new Time(curTime);  
   
 resourceRequestsRepository.deletePastRequests(sqlDate,t);  
   
 Users user = (Users)SecurityContextHolder.getContext().getAuthentication().getPrincipal();  
 String userName = user.getUserName();  
 ArrayList<ResourceRequests> requests = resourceRequestsRepository.findByResourceIncharge(userName);  
   
 if(requests.isEmpty()) {  
 model.addObject("msg","No requests!");  
 }else {  
 model.addObject("requests",requests);  
 }  
 model.setViewName("/faculty/resourceRequests");  
 return model;  
 }  
   
 @Transactional  
 @Modifying  
 @RequestMapping(value="/getOverlappingSlots",method=RequestMethod.POST)  
 public String getOverlappingSlots(Model model,Integer getOverlapsFor) {  
 LocalDate localDate = LocalDate.now();  
 Date sqlDate = Date.valueOf(localDate.toString());  
   
 long curTime = new java.util.Date().getTime();  
   
 Time t = new Time(curTime);  
 resourceRequestsRepository.deletePastRequests(sqlDate,t);  
   
 ResourceRequests obj = resourceRequestsRepository.findByRequestId(getOverlapsFor);  
   
 ArrayList<ResourceRequests> allRequests = resourceRequestsRepository.findByResourceIdAndSlotDate(obj.getResourceId(),obj.getSlotDate());  
 ArrayList<ResourceRequests> overlappingRequests = new ArrayList<>();  
   
 for(ResourceRequests rr:allRequests) {  
 if(obj.getRequestId() != rr.getRequestId()) {  
 Long st = rr.getSlotStartTime().getTime();  
 Long et = rr.getSlotEndTime().getTime();  
 Long gost = obj.getSlotStartTime().getTime();  
 Long goet = obj.getSlotEndTime().getTime();  
 if((st>=gost && et<=goet) || (st>=gost && st<goet) || (et>gost && et<=goet) || (st<=gost && et>=goet) || (st<=gost && et>gost && et<=goet)) {  
 overlappingRequests.add(rr);  
 }  
   
 }  
 }  
   
 ArrayList<TimeSlots> allTimeSlots = timeSlotsRepository.findByResourceIdAndDate(obj.getResourceId(), obj.getSlotDate());  
 ArrayList<TimeSlots> overlappingTimeSlots = new ArrayList<>();  
   
 for(TimeSlots ts:allTimeSlots) {  
 Long gost = ts.getStartTime().getTime();  
 Long goet = ts.getEndTime().getTime();  
 Long st = obj.getSlotStartTime().getTime();  
 Long et = obj.getSlotEndTime().getTime();  
 if((st>=gost && et<=goet) || (st>=gost && st<goet) || (et>gost && et<=goet) || (st<=gost && et>=goet) || (st<=gost && et>gost && et<=goet)) {  
 overlappingTimeSlots.add(ts);  
 }  
   
 }  
   
 String day = obj.getSlotDay();  
   
 ArrayList<TimeTable> allTimeTableSlots = timeTableRepository.findByResourceIdAndDay(obj.getResourceId(), day);  
   
 ArrayList<TimeTable> overlappingTimeTableSlots = new ArrayList<>();  
   
 for(TimeTable tt:allTimeTableSlots) {  
 Long st = tt.getStartTime().getTime();  
 Long et = tt.getEndTime().getTime();  
 Long gost = obj.getSlotStartTime().getTime();  
 Long goet = obj.getSlotEndTime().getTime();  
 if((st>=gost && et<=goet) || (st>=gost && st<goet) || (et>gost && et<=goet) || (st<=gost && et>=goet) || (st<=gost && et>gost && et<=goet)) {  
 overlappingTimeTableSlots.add(tt);  
 }  
   
 }  
   
 if(!overlappingRequests.isEmpty() || !overlappingTimeSlots.isEmpty() || !overlappingTimeTableSlots.isEmpty()) {  
   
 if(!overlappingRequests.isEmpty()) {  
 model.addAttribute("overlappingRequests",overlappingRequests);  
 }  
 if(!overlappingTimeSlots.isEmpty()) {  
 model.addAttribute("overlappingTimeSlots",overlappingTimeSlots);  
 }  
 if(!overlappingTimeTableSlots.isEmpty()) {  
 model.addAttribute("overlappingTimeTableSlots",overlappingTimeTableSlots);  
 }  
   
 return "faculty/resourceRequests:: overlapDiv";  
 }else {  
 model.addAttribute("msg","No overlapping requests!");  
 return "faculty/resourceRequests:: messageDiv";   
 }  
   
 }  
   
 @RequestMapping(value="/deleteResourceRequest",method=RequestMethod.POST)  
 public String deleteResourceRequest(Model model,Integer deleteRequestFor) {  
   
 ResourceRequests obj = resourceRequestsRepository.findByRequestId(deleteRequestFor);  
   
 resourceRequestsRepository.delete(obj);  
 Users user = (Users)SecurityContextHolder.getContext().getAuthentication().getPrincipal();  
 String userName = user.getUserName();  
 ArrayList<ResourceRequests> requests = resourceRequestsRepository.findByResourceIncharge(userName);  
   
 if(requests.isEmpty()) {  
 model.addAttribute("msg","Deleted successfully! No requests!");  
 return "faculty/resourceRequests:: messageDiv";  
 }else {  
 model.addAttribute("requests",requests);  
 return "faculty/resourceRequests:: requestsDiv";  
 }  
   
 }  
   
 @RequestMapping(value="/finalAcceptResourceRequest",method=RequestMethod.POST)  
 public String finalAcceptResourceRequest(Model model,Integer getOverlapsFor) {  
  
 ResourceRequests obj = resourceRequestsRepository.findByRequestId(getOverlapsFor);  
   
 ArrayList<ResourceRequests> allRequests = resourceRequestsRepository.findByResourceIdAndSlotDate(obj.getResourceId(),obj.getSlotDate());  
   
 String body;  
   
   
 ArrayList<ResourceRequests> overlappingRequests = new ArrayList<>();  
 ArrayList<TimeSlots> overlappingTimeSlots = new ArrayList<>();  
  
 for(ResourceRequests rr:allRequests) {  
 Long gost = rr.getSlotStartTime().getTime();  
 Long goet = rr.getSlotEndTime().getTime();  
 Long st = obj.getSlotStartTime().getTime();  
 Long et = obj.getSlotEndTime().getTime();  
 if((st>=gost && et<=goet) || (st>=gost && st<goet) || (et>gost && et<=goet) || (st<=gost && et>=goet)) {  
 overlappingRequests.add(rr);  
 resourceRequestsRepository.delete(rr);  
 }  
   
 }  
   
 ArrayList<TimeSlots> allTimeSlots = timeSlotsRepository.findByResourceIdAndDate(obj.getResourceId(), obj.getSlotDate());  
   
 for(TimeSlots ts:allTimeSlots) {  
 Long gost = ts.getStartTime().getTime();  
 Long goet = ts.getEndTime().getTime();  
 Long st = obj.getSlotStartTime().getTime();  
 Long et = obj.getSlotEndTime().getTime();  
 if((st>=gost && et<=goet) || (st>=gost && st<goet) || (et>gost && et<=goet) || (st<=gost && et>=goet)) {  
 overlappingTimeSlots.add(ts);  
 timeSlotsRepository.delete(ts);  
 }  
   
 }  
 body = "Your request to book the resource "+obj.getResourceId().getResourceId()+" with Request No: "+obj.getRequestId()+" has been Accepted!\n";  
 body = body.concat("Resource: "+obj.getResourceId().getResourceId()+"\n");  
 body = body.concat("Activity Name: "+obj.getSlotActivityName()+"\n");  
 body = body.concat("Slot: "+obj.getSlotDate()+" "+obj.getSlotStartTime()+" - "+obj.getSlotEndTime()+"\n");  
 body = body.concat("Please login to your QRMS account to book another slot!\n\nRegards,\nQRMS Team.");  
 TimeSlots timeslot = new TimeSlots(obj.getSlotStartTime(), obj.getSlotEndTime(), obj.getResourceId(), obj.getSlotDate(), obj.getRequestBy(), obj.getSlotActivityName(), obj.getRequestId());  
 timeSlotsRepository.save(timeslot);  
   
 try {  
 emailServiceImpl.send(qrmsEmailId, "bmk15897@gmail.com", "QRMS: Request to book resource "+obj.getResourceId().getResourceId()+" Accepted!", body);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
   
 for(ResourceRequests rr:overlappingRequests) {  
 body = "Your request to book the resource "+rr.getResourceId().getResourceId()+" with Request No: "+rr.getRequestId()+" has been rejected!\n";  
 body = body.concat("Resource: "+rr.getResourceId().getResourceId()+"\n");  
 body = body.concat("Activity Name: "+rr.getSlotActivityName()+"\n");  
 body = body.concat("Slot: "+rr.getSlotDate()+" "+rr.getSlotStartTime()+" - "+rr.getSlotEndTime()+"\n");  
 body = body.concat("Please login to your QRMS account to book another slot!\n\nRegards,\nQRMS Team.");  
  
 try {  
 emailServiceImpl.send(qrmsEmailId, "bmk15897@gmail.com", "QRMS: Request to book resource "+rr.getResourceId().getResourceId()+" Rejected", body);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
   
 for(TimeSlots ts:overlappingTimeSlots) {  
 body = "Your booked slot the resource "+ts.getResourceId().getResourceId()+" with Request No: "+ts.getRequestId()+" has been overridden by another request!\n";  
 body = body.concat("Resource: "+ts.getResourceId().getResourceId()+"\n");  
 body = body.concat("Activity Name: "+ts.getActivityName()+"\n");  
 body = body.concat("Slot: "+ts.getDate()+" "+ts.getStartTime()+" - "+ts.getEndTime()+"\n");  
 body = body.concat("Please login to your QRMS account to book another slot!\n\nRegards,\nQRMS Team.");  
 try {  
 emailServiceImpl.send(qrmsEmailId, "bmk15897@gmail.com", "QRMS: Request to book resource "+ts.getResourceId().getResourceId()+" Overridden", body);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
   
   
 model.addAttribute("msg", "Request Accepted!");  
 return "faculty/resourceRequests:: messageDiv";  
 }  
   
 @RequestMapping(value = "/viewSchedule",method=RequestMethod.GET)  
 public ModelAndView getViewSchedule() {  
 ModelAndView model = new ModelAndView();  
 ArrayList<Department> depts = bookingsService.listDepartments();  
 model.addObject("departments", depts);  
   
 model.setViewName("/faculty/viewSchedule");  
 return model;  
   
 }  
   
 @RequestMapping(value="/getScheduleForResource", method=RequestMethod.GET)  
 public String getScheduleForResource(Model model,String getTT,String cur\_date){  
   
 List<TimeSlots> list = bookingsService.getTimeSlotsForDate(cur\_date, getTT);  
   
 if(list.size()==0) {  
 model.addAttribute("msg","All slots are empty!");  
 return "faculty/viewSchedule:: messageDiv";  
 }else {  
 model.addAttribute("ttForResource",list);  
 return "faculty/viewSchedule:: resourceTT";  
 }  
 }  
   
 @RequestMapping(value="/getViewOptions",method=RequestMethod.POST)  
 public String getViewOptions(Model model,String dept,String rType,Integer minSeats) {  
   
 if(minSeats==null) {  
 minSeats=0;  
 }  
 ArrayList<Resource> options = bookingsService.listResourcesByDepartmentAndRTypeAndMinSeats(dept, rType, minSeats);  
  
 if(options.isEmpty()) {  
 model.addAttribute("err\_msg","No suitable rooms/halls/classrooms were found.");  
 return "faculty/viewSchedule:: messageDiv";  
 }  
 else {  
 model.addAttribute("options",options);  
 return "faculty/viewSchedule:: resourceOptionsTable";  
 }  
 }  
   
 @Transactional  
 @Modifying  
 @RequestMapping(value="/viewBookingHistory",method=RequestMethod.GET)  
 public ModelAndView getPreviousHistory() {  
 ModelAndView model = new ModelAndView();  
   
 Users user = (Users)SecurityContextHolder.getContext().getAuthentication().getPrincipal();  
 String userName = user.getUserName();  
   
 LocalDate localDate = LocalDate.now();  
 Date sqlDate = Date.valueOf(localDate.toString());  
   
 long curTime = new java.util.Date().getTime();  
   
 Time t = new Time(curTime);  
   
 resourceRequestsRepository.deletePastRequests(sqlDate,t);  
   
 ArrayList <ResourceRequests> historyRequests = resourceRequestsRepository.findByRequestBy(userName);  
 if(!historyRequests.isEmpty()) {  
 model.addObject("historyRequests",historyRequests);  
 }  
   
 ArrayList <TimeSlots> historyAccepted = timeSlotsRepository.findBySlotIncharge(userName);  
 if(!historyAccepted.isEmpty()) {  
 model.addObject("historyAccepted",historyAccepted);  
 }  
   
 if(historyRequests.isEmpty() && historyAccepted.isEmpty()) {  
 model.addObject("msg","No History found!");  
 }else if(historyRequests.isEmpty()) {  
 model.addObject("msg", "No pending requests!");  
 }else if(historyAccepted.isEmpty()) {  
 model.addObject("msg","No accepted requests!");  
 }  
   
 model.setViewName("/faculty/viewBookingHistory");  
 return model;  
 }  
   
 @Transactional  
 @Modifying  
 @RequestMapping(value="/deleteRequest",method=RequestMethod.POST)  
 public String deleteRequest(Model model,Integer requestToDelete) {  
   
 resourceRequestsRepository.deleteByRequestId(requestToDelete);  
   
 return "/faculty/viewBookingHistory:: messageDiv";  
 }  
}

# spring\resource\LoginController.java

package com.qrms.spring.resource;  
  
import java.util.UUID;  
  
import javax.servlet.http.HttpServletRequest;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestMethod;  
import org.springframework.web.bind.annotation.RequestParam;  
import org.springframework.web.servlet.ModelAndView;  
import com.qrms.spring.model.Users;  
import com.qrms.spring.service.EmailServiceImpl;  
import com.qrms.spring.service.UserService;  
  
  
@Controller  
public class LoginController {  
   
 @Autowired  
 private EmailServiceImpl emailObj;  
   
 @Autowired  
 private UserService userService;  
  
 @GetMapping("/login")  
 public String userLogin() {  
 return "login";  
 }  
   
 @GetMapping("/forgotPassword")  
 public String forgotPass() {  
 return "forgotPass";  
 }  
   
 //handles forgot password form (user requests token and posts email through form)  
 @RequestMapping(value = "/forgotPassword", method = RequestMethod.POST)  
 public ModelAndView resetToken(String email,HttpServletRequest request) {  
   
 ModelAndView model = new ModelAndView();  
 String token = UUID.randomUUID().toString();   
 Users user = userService.findUserByEmail(email);  
   
 if(user == null) {  
 model.addObject("message","Invalid email-id");  
 model.setViewName("forgotPass");  
 return model;  
 }  
 userService.createPasswordTokenForUser(user, token);   
 String emailBody = constructEmail(request,user,token);  
   
 try {  
 emailObj.send("qrmsmail@gmail.com", user.getEmail(), "QRMS- Password Reset", emailBody);  
 model.addObject("message","Password reset email has been sent");  
   
 }catch(Exception e) {  
 System.out.println("Error Sending Password Token Email: " + e.getMessage());  
 model.addObject("message","Error Sending Password Token Email: " + e.getMessage());  
 }  
   
 model.setViewName("login");  
 return model;  
 }  
   
 //redirects to this URL when email reset link is clicked & show reset Password form after validating token  
 @RequestMapping(value = "/validateToken", method = RequestMethod.GET)  
 public ModelAndView showChangePasswordPage( @RequestParam("id") String username, @RequestParam("token") String token) {  
 ModelAndView model = new ModelAndView();  
   
 String result = userService.validatePasswordResetToken(username, token);  
   
 if (result != null) {  
 model.addObject("errmsg",result);  
 model.setViewName("login");  
 return model;  
 }  
 model.addObject("username",username);  
 model.setViewName("resetPassword");  
 return model;  
 }  
  
 //POST URL to set new password  
 @RequestMapping(value = "/updatePassword", method = RequestMethod.POST)  
 public ModelAndView updatePass(String password1, String h\_username) {  
   
 Users user = userService.findByUserName(h\_username);  
 userService.savePassword(user, password1);  
 ModelAndView model = new ModelAndView();  
 model.addObject("message","Password has been reset successfully. Login with your new password");  
 model.setViewName("login");  
 return model;  
  
 }  
   
 //added this function for readability  
 private String constructEmail(HttpServletRequest request,Users user, String token) {  
 String url = "http://"+request.getLocalName()+":"+request.getLocalPort()+"/validateToken?id=" + user.getUserName() + "&token=" + token;  
 String body = "Hi "+user.getFirstName()+",\nTo initiate the password reset process for your QRMS Account, click the link below:\n"+url+"\nIf clicking the link above doesn't work, please copy and paste the URL in a new browser window instead.\nSincerely, QRMS Team.";  
 return body;  
 }  
   
  
}

# spring\resource\StudentController.java

package com.qrms.spring.resource;  
  
  
import java.util.ArrayList;  
import java.util.List;  
  
import org.springframework.beans.factory.annotation.Autowired;  
  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestMethod;  
import org.springframework.web.servlet.ModelAndView;  
  
import com.qrms.spring.model.Department;  
import com.qrms.spring.model.Resource;  
import com.qrms.spring.model.TimeSlots;  
import com.qrms.spring.model.Users;  
import com.qrms.spring.service.BookingsServiceImpl;  
import com.qrms.spring.service.StudentHomeServiceImpl;  
import com.qrms.spring.service.StudentPrefServiceImpl;  
  
@Controller  
@RequestMapping("/u/student")  
public class StudentController {  
   
 @Autowired  
 private StudentHomeServiceImpl studHomeService;  
   
 @Autowired  
 private BookingsServiceImpl bookingsService;  
   
 @Autowired  
 private StudentPrefServiceImpl studPrefService;  
   
 @GetMapping("/home")  
 public ModelAndView studentHome() {  
 Users user = (Users)SecurityContextHolder.getContext().getAuthentication().getPrincipal();  
 String userName = user.getUserName();  
 return studHomeService.getStudentHome(userName);  
 }  
   
 @RequestMapping(value="/getElectiveId",method=RequestMethod.GET)  
 public ModelAndView getElectiveId(String msg) {  
 Users user = (Users)SecurityContextHolder.getContext().getAuthentication().getPrincipal();  
 String userName = user.getUserName();  
 return studPrefService.getElectiveId(msg,userName);  
 }  
   
 @RequestMapping(value = "/getStudentPrefs", method = RequestMethod.GET)  
 public ModelAndView studentPref(String elective\_id) {  
 Users user = (Users)SecurityContextHolder.getContext().getAuthentication().getPrincipal();  
 String userName = user.getUserName();  
 return studPrefService.getStudentPrefs(elective\_id,userName);  
 }  
   
 //Handle student preference form  
 @RequestMapping(value = "/setStudentPrefs", method = RequestMethod.POST)  
 public ModelAndView addPreferences(String course1,String course2,String course3,String course4) {  
 Users user = (Users)SecurityContextHolder.getContext().getAuthentication().getPrincipal();  
 String userName = user.getUserName();  
 studPrefService.setStudentPrefs(course1, course2, course3, course4,userName);  
 return getElectiveId("Your preferences for electives have been recorded!");   
 }  
   
  
 @RequestMapping(value = "/viewSchedule",method=RequestMethod.GET)  
 public ModelAndView getViewSchedule() {  
 ModelAndView model = new ModelAndView();  
 ArrayList<Department> depts = bookingsService.listDepartments();  
 model.addObject("departments", depts);  
 model.setViewName("/student/viewSchedule");  
 return model;  
 }  
   
 @RequestMapping(value="/getScheduleForResource", method=RequestMethod.GET)  
 public String getScheduleForResource(Model model,String getTT,String cur\_date){  
 List<TimeSlots> list = bookingsService.getTimeSlotsForDate(cur\_date, getTT);  
 if(list.size()==0) {  
 model.addAttribute("msg","All slots are empty!");   
 return "student/viewSchedule:: messageDiv";  
 }else {  
 model.addAttribute("ttForResource",list);  
 return "student/viewSchedule:: resourceTT";  
 }  
 }  
   
 @RequestMapping(value="/getTTForResourceForDate",method=RequestMethod.POST)  
 public String getTTForResourceForDate(Model model,String booking\_date,String getTT){  
 List<TimeSlots> list = bookingsService.getTimeSlotsForDate(booking\_date, getTT);  
 if(list.isEmpty()) {  
 model.addAttribute("msg","All slots are empty!");  
 return "student/viewSchedule:: messageDiv";  
 }else {  
 model.addAttribute("ttForResource",list);  
 return "student/viewSchedule:: resourceTT";  
 }  
   
 }  
   
 @RequestMapping(value="/getViewOptions",method=RequestMethod.POST)  
 public String getViewOptions(Model model,String dept,String rType,Integer minSeats) {  
 if(minSeats==null) {  
 minSeats=0;  
 }  
 ArrayList<Resource> options = bookingsService.listResourcesByDepartmentAndRTypeAndMinSeats(dept, rType, minSeats);  
 if(options.isEmpty()) {  
 model.addAttribute("err\_msg","No suitable rooms/halls/classrooms were found.");  
 return "student/viewSchedule:: messageDiv";  
 }else {  
 model.addAttribute("options",options);  
 return "student/viewSchedule:: resourceOptionsTable";  
 }  
 }  
  
}

# spring\service\BookingsService.java

package com.qrms.spring.service;  
  
import java.sql.Date;  
import java.util.ArrayList;  
import java.util.List;  
  
import com.qrms.spring.model.Department;  
import com.qrms.spring.model.Resource;  
import com.qrms.spring.model.TimeSlots;  
  
public interface BookingsService {  
 public ArrayList<Department> listDepartments();  
 public Resource findByResourceId(String getTT);  
 public ArrayList<Resource> listResourcesByDepartmentAndRTypeAndMinSeats(String dept, String rType, Integer minSeats);  
 public ArrayList<TimeSlots> findTimeSlotsByResourceForDate(String getTT,String day,Date sqlDate);  
 public List<TimeSlots> getTimeSlotsForDate(String booking\_date,String getTT);  
}

# spring\service\BookingsServiceImpl.java

package com.qrms.spring.service;  
  
import java.sql.Date;  
import java.time.LocalDate;  
import java.time.format.DateTimeFormatter;  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.List;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import com.qrms.spring.model.Department;  
import com.qrms.spring.model.Resource;  
import com.qrms.spring.model.TimeSlots;  
import com.qrms.spring.model.TimeTable;  
import com.qrms.spring.repository.DepartmentRepository;  
import com.qrms.spring.repository.ResourceRepository;  
import com.qrms.spring.repository.TimeSlotsRepository;  
import com.qrms.spring.repository.TimeTableRepository;  
  
@Service  
public class BookingsServiceImpl implements BookingsService{  
  
 @Autowired  
 private DepartmentRepository departmentRepository;  
 @Autowired  
 private ResourceRepository resourceRepository;  
 @Autowired  
 private TimeSlotsRepository timeSlotsRepository;  
 @Autowired  
 private TimeTableRepository timeTableRepository;  
   
 @Override  
 public ArrayList<Department> listDepartments() {  
 return departmentRepository.findAll();  
 }  
  
 @Override  
 public ArrayList<Resource> listResourcesByDepartmentAndRTypeAndMinSeats(String dept, String rType, Integer minSeats) {  
 Department d = departmentRepository.findByDeptId(dept);  
 return resourceRepository.findByDepartmentAndResourceTypeAndResourceCapacityGreaterThan(d,rType,minSeats);  
 }  
  
 public Resource findByResourceId(String getTT) {  
 return resourceRepository.findByResourceId(getTT);  
 }  
  
 public ArrayList<TimeSlots> findTimeSlotsByResourceForDate(String getTT,String day,Date sqlDate) {  
  
 Resource r = resourceRepository.findByResourceId(getTT);  
 ArrayList<TimeTable> tt = timeTableRepository.findByResourceIdAndDay(r,day);  
 ArrayList<TimeSlots> ts = timeSlotsRepository.findByResourceIdAndDate(r,sqlDate);  
  
 ArrayList<TimeSlots> finalTS = new ArrayList<>();  
   
 for(TimeSlots tss:ts) {  
 finalTS.add(tss);  
 }  
 for(TimeTable t:tt) {  
 Long st = t.getStartTime().getTime();  
 Long et = t.getEndTime().getTime();  
 int i = 0;  
 for(TimeSlots tss:ts) {  
 Long gost = tss.getStartTime().getTime();  
 Long goet = tss.getEndTime().getTime();  
   
 if((st>=gost && et<=goet) || (st>=gost && st<goet) || (et>gost && et<=goet) || (st<=gost && et>=goet) || (st<=gost && et>gost && et<=goet)) {  
 i+=1;  
 }  
   
 }  
 if(i==0) {  
 finalTS.add(new TimeSlots(t.getStartTime(), t.getEndTime(), t.getResourceId(),sqlDate,t.getSlotIncharge(),t.getActivityName(),0));  
 }  
 }  
   
 return finalTS;  
 }  
  
  
 public List<TimeSlots> getTimeSlotsForDate(String booking\_date,String getTT){  
 DateTimeFormatter df = DateTimeFormatter.ofPattern("yyyy-MM-dd");  
 LocalDate date = LocalDate.parse(booking\_date, df);  
 Date sqlDate = java.sql.Date.valueOf(date.toString());  
 String day = date.getDayOfWeek().name();  
  
 ArrayList<TimeSlots> list = findTimeSlotsByResourceForDate(getTT,day,sqlDate);  
   
 Collections.sort(list);  
 return list;  
 }  
}

# spring\service\CustomUserDetailsService.java

package com.qrms.spring.service;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.Calendar;  
import java.util.Optional;  
import java.util.Set;  
  
import org.passay.CharacterRule;  
import org.passay.EnglishCharacterData;  
import org.passay.PasswordGenerator;  
import org.passay.CharacterData;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.context.annotation.Bean;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.stereotype.Service;  
  
import com.qrms.spring.model.CustomUserDetails;  
import com.qrms.spring.model.PasswordResetToken;  
import com.qrms.spring.model.Role;  
import com.qrms.spring.model.Users;  
import com.qrms.spring.repository.PasswordResetTokenRepository;  
import com.qrms.spring.repository.UsersRepository;  
  
@Service  
public class CustomUserDetailsService implements UserService,UserDetailsService {  
  
 @Autowired  
 private UsersRepository usersRepository;  
   
 @Autowired  
 private PasswordResetTokenRepository passwordResetTokenRepository;  
   
 @Autowired  
 private EmailServiceImpl email;  
   
 @Value("${spring.mail.username}")  
 private String qrmsEmailId;  
   
 @Override  
 public UserDetails loadUserByUsername(String userName) throws UsernameNotFoundException {  
 Optional<Users> optionalUsers = usersRepository.findByUserName(userName);  
 optionalUsers.orElseThrow(() -> new UsernameNotFoundException("Username not found"));  
 return optionalUsers.map(CustomUserDetails::new).get();  
 }  
   
 @Override  
 public Users findByUserName(String username) {  
 Optional<Users> optionalUsers = usersRepository.findByUserName(username);  
 optionalUsers.orElseThrow(() -> new UsernameNotFoundException("Username not found"));  
 return optionalUsers.map(Users::new).get();  
 }  
   
 @Override  
 public Users findUserByEmail(String email) {  
 Optional<Users> optionalUsers = usersRepository.findByEmail(email);  
 if(!optionalUsers.isPresent()) {  
 return null;  
 }  
 return optionalUsers.map(Users::new).get();  
  
 }  
   
 @Bean  
 public BCryptPasswordEncoder bCryptPasswordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
   
 public String generatePassayPassword() {  
 PasswordGenerator gen = new PasswordGenerator();  
 CharacterData lowerCaseChars = EnglishCharacterData.LowerCase;  
 CharacterRule lowerCaseRule = new CharacterRule(lowerCaseChars);  
 lowerCaseRule.setNumberOfCharacters(2);  
   
 CharacterData upperCaseChars = EnglishCharacterData.UpperCase;  
 CharacterRule upperCaseRule = new CharacterRule(upperCaseChars);  
 upperCaseRule.setNumberOfCharacters(2);  
   
 CharacterData digitChars = EnglishCharacterData.Digit;  
 CharacterRule digitRule = new CharacterRule(digitChars);  
 digitRule.setNumberOfCharacters(2);  
   
 CharacterData specialChars = new CharacterData() {  
 public String getErrorCode() {  
 return "Special char password error";  
 }  
 public String getCharacters() {  
 return "!@#$%^&\*()\_+";  
 }  
 };  
 CharacterRule splCharRule = new CharacterRule(specialChars);  
 splCharRule.setNumberOfCharacters(2);  
   
 String password = gen.generatePassword(10, splCharRule, lowerCaseRule, upperCaseRule, digitRule);  
 return password;  
 }  
   
 public boolean isUniqueEmail(String email) {  
 Users tempUser = findUserByEmail(email);  
 if(tempUser!=null) {  
 return false;  
 }  
 return true;  
 }  
  
 public void saveUser(Users user) {  
 String username = user.getFirstName().toLowerCase() + user.getLastName().toLowerCase().charAt(0);  
 String tempUsername = username;  
 System.out.println(tempUsername);  
 int i = 0;  
   
 //Check if email is unique  
   
 while(true) {  
 try {  
 Optional<Users> userexists = usersRepository.findByUserName(tempUsername);  
 userexists.map(CustomUserDetails::new).get().getUserName();  
 i++;  
 tempUsername = username + Integer.toString(i);  
   
 }  
 catch(Exception e) {  
 user.setUserName(tempUsername);   
 break;  
 }   
 }  
   
 String password= generatePassayPassword();  
  
 String body = "Hi "+user.getFirstName()+",\nYour QRMS Account has been successfully created.\n"  
 + "Use the following credentials to login:\n"  
 + "Username: "+tempUsername+"\n"+ "Password: "+password  
 + "\n\nDo not share your credentials with anyone.\n"  
 + "Regards,\nQRMS Team.";  
   
 try {  
 email.send(qrmsEmailId, user.getEmail(), "Login Credentials for QRMS", body);  
 }catch(Exception e) {  
 System.out.println("Error Sending Email: " + e.getMessage());  
 }  
 user.setPassword(bCryptPasswordEncoder().encode(password));  
 user.setActive(1);  
   
 usersRepository.save(user);  
   
 }  
  
 @Override  
 public void createPasswordTokenForUser(Users user, String token) {  
 PasswordResetToken myToken = new PasswordResetToken(token, user);  
 passwordResetTokenRepository.save(myToken);  
   
 }  
   
 public String validatePasswordResetToken(String username, String token) {  
 Optional<PasswordResetToken> optionalPassToken = passwordResetTokenRepository.findByToken(token);  
 PasswordResetToken passToken;  
   
 //check if token is valid  
 if (optionalPassToken.isPresent()) {  
 passToken = optionalPassToken.map(PasswordResetToken::new).get();  
   
 //check if username is valid  
 if(!(passToken.getUser().getUserName().equals(username)))  
 return "Invalid Token";  
 else {  
 //check if token has expired  
 Calendar cal = Calendar.getInstance();  
 if ((passToken.getExpiryDate().getTime() - cal.getTime().getTime()) <= 0) {  
 passwordResetTokenRepository.delete(passToken);  
 return "Token has Expired";  
 }  
   
 Users user = passToken.getUser();  
 Authentication auth = new UsernamePasswordAuthenticationToken(  
 user, null, Arrays.asList(new SimpleGrantedAuthority("CHANGE\_PASSWORD\_PRIVILEGE")));  
   
 SecurityContextHolder.getContext().setAuthentication(auth);  
   
 //Delete the token since it is now used  
 passwordResetTokenRepository.delete(passToken);  
   
 //token and username are valid, return null  
 return null;  
 }  
 }  
 else  
 return "Invalid Token"; //if token String is not valid   
 }  
  
 @Override  
 public void savePassword(Users user, String password) {  
 user.setPassword(bCryptPasswordEncoder().encode(password));  
 usersRepository.save(user);  
 }  
  
 @Override  
 public ArrayList<Users> findByRole(Set<Role> role) {  
 ArrayList<Users> users = usersRepository.findByRoles(role);  
 return users;  
 }  
  
   
}

# spring\service\EmailService.java

package com.qrms.spring.service;  
  
  
public interface EmailService {  
 void send(String from,String to, String title, String body);  
}

# spring\service\EmailServiceImpl.java

package com.qrms.spring.service;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.mail.MailException;  
import org.springframework.mail.SimpleMailMessage;  
import org.springframework.mail.javamail.JavaMailSender;  
import org.springframework.scheduling.annotation.Async;  
import org.springframework.stereotype.Component;  
  
@Component  
public class EmailServiceImpl {  
   
 @Autowired  
 public JavaMailSender emailSender;  
   
 @Async  
 public void send(String from, String to, String title, String body) throws MailException, InterruptedException{  
   
 SimpleMailMessage message = new SimpleMailMessage();   
 message.setTo(to);   
 message.setSubject(title);   
 message.setText(body);  
 emailSender.send(message);  
   
 }  
}

# spring\service\FacPrefService.java

package com.qrms.spring.service;  
  
import java.util.List;  
  
import com.qrms.spring.queryBeans.FacPrefCountInfo;  
  
public interface FacPrefService {  
 List<FacPrefCountInfo> computeFacPrefTable();  
  
}

# spring\service\FacPrefServiceImpl.java

package com.qrms.spring.service;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import com.qrms.spring.model.Department;  
import com.qrms.spring.model.OpenFacultyPrefs;  
import com.qrms.spring.queryBeans.FacPrefCountInfo;  
import com.qrms.spring.repository.DepartmentRepository;  
import com.qrms.spring.repository.FacultyAcadRepository;  
import com.qrms.spring.repository.FacultyPrefRepository;  
import com.qrms.spring.repository.OpenFacultyPrefsRepository;  
  
@Service  
public class FacPrefServiceImpl implements FacPrefService {  
  
 @Autowired  
 OpenFacultyPrefsRepository openFacultyPrefsRepository;  
   
 @Autowired  
 FacultyAcadRepository facultyAcadRepository;  
   
 @Autowired  
 FacultyPrefRepository facultyPrefRepository;  
   
 @Autowired  
 DepartmentRepository departmentRepository;  
   
 @Override  
 public List<FacPrefCountInfo> computeFacPrefTable() {  
 List<FacPrefCountInfo> fpList = new ArrayList<FacPrefCountInfo>();  
 List<OpenFacultyPrefs> openFacPrefs = openFacultyPrefsRepository.findAll();  
   
 for(OpenFacultyPrefs op : openFacPrefs) {  
 Department dept = departmentRepository.findByDeptId(op.getDeptId());  
 FacPrefCountInfo fp = new FacPrefCountInfo();  
 fp.setDeptName(dept.getDeptId());  
 if(op.getSemType() == 0)  
 fp.setSemType("Even semesters");  
 else  
 fp.setSemType("Odd semesters");  
 fp.setSubmitCount(facultyPrefRepository.findFacultyPrefCountByDepartment(dept));  
 fp.setTotalFacultyCount(facultyAcadRepository.countFacultyByDepartment(dept));  
 fpList.add(fp);  
 }  
 return fpList;  
   
 }  
   
}

# spring\service\FacultyAcadService.java

package com.qrms.spring.service;  
  
import java.util.ArrayList;  
  
import com.qrms.spring.model.Department;  
import com.qrms.spring.queryBeans.FacultyUsers;  
  
public interface FacultyAcadService {  
  
 public ArrayList<FacultyUsers> getFacultyList(Department dept);  
}

# spring\service\FacultyAcadServiceImp.java

package com.qrms.spring.service;  
  
import java.util.ArrayList;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import com.qrms.spring.model.Department;  
import com.qrms.spring.model.FacultyAcad;  
import com.qrms.spring.model.Users;  
import com.qrms.spring.queryBeans.FacultyUsers;  
import com.qrms.spring.repository.FacultyAcadRepository;  
import com.qrms.spring.repository.UsersRepository;  
  
@Service  
public class FacultyAcadServiceImp implements FacultyAcadService{  
  
 @Autowired  
 private UsersRepository userRepository;  
   
 @Autowired  
 private FacultyAcadRepository facultyAcadRepository;  
   
 @Override  
 public ArrayList<FacultyUsers> getFacultyList(Department dept) {  
   
 FacultyUsers facUser;  
 ArrayList <FacultyUsers> facUserList = new ArrayList<FacultyUsers>();  
 ArrayList <FacultyAcad> faculties = facultyAcadRepository.findByDepartmentEquals(dept);  
 for(FacultyAcad faculty: faculties) {  
 facUser = new FacultyUsers();  
 Users user = userRepository.findByUserName(faculty.getUserName()).get();  
 facUser.setDesignation(faculty.getDesignation());  
 facUser.setEmail(user.getEmail());  
 facUser.setExp(faculty.getYearsOfExperience());  
 facUser.setName(user.getFirstName()+" "+user.getLastName());  
 facUser.setQualification(faculty.getQualification());  
 facUser.setUserName(user.getUserName());  
 facUserList.add(facUser);  
 }  
 return facUserList;  
 }  
   
}

# spring\service\StudentAcadService.java

package com.qrms.spring.service;  
  
import java.util.ArrayList;  
  
import com.qrms.spring.model.Department;  
import com.qrms.spring.model.StudentAcad;  
import com.qrms.spring.queryBeans.StudentUsers;  
  
public interface StudentAcadService {  
 public void saveStudentAcad(StudentAcad student, String username);  
 public ArrayList<StudentUsers> getStudentList(Department dept, String year);  
 boolean validateAndSetStudDiv(StudentAcad student, String divName);  
}

# spring\service\StudentAcadServiceImpl.java

package com.qrms.spring.service;  
  
import java.util.ArrayList;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
  
import com.qrms.spring.model.Department;  
import com.qrms.spring.model.Divisions;  
import com.qrms.spring.model.StudentAcad;  
import com.qrms.spring.model.Users;  
import com.qrms.spring.queryBeans.StudentUsers;  
import com.qrms.spring.repository.DivisionsRepository;  
import com.qrms.spring.repository.StudentAcadRepository;  
import com.qrms.spring.repository.UsersRepository;  
  
@Component  
public class StudentAcadServiceImpl implements StudentAcadService {  
  
 @Autowired  
 private StudentAcadRepository studentAcadRepository;  
  
 @Autowired  
 private UsersRepository userRepository;  
   
 @Autowired  
 private DivisionsRepository divisionsRepository;  
   
 @Override  
 public void saveStudentAcad(StudentAcad student, String username) {  
 student.setUserName(username);  
 studentAcadRepository.save(student);  
   
 }  
  
 @Override  
 public ArrayList<StudentUsers> getStudentList(Department department, String year) {  
 StudentUsers studUser;  
 ArrayList <StudentUsers> studUserList = new ArrayList<StudentUsers>();  
 ArrayList <StudentAcad> students = studentAcadRepository.findByYearEqualsAndDepartmentEquals(year, department);  
 int i = 0;  
 for(StudentAcad student: students) {  
 i++;  
 studUser = new StudentUsers();  
 Users user = userRepository.findByUserName(student.getUserName()).get();  
 studUser.setSrNo(i);  
 studUser.setAggrMarks(student.getAggMarks());  
 studUser.setUserName(student.getUserName());  
 studUser.setDiv(student.getDiv().getDivName().toString());  
 studUser.setEmail(user.getEmail());  
 studUser.setName(user.getFirstName().concat(" ".concat(user.getLastName())));  
 studUser.setRollNo(student.getRollno());  
 studUser.setShift(student.getShift());  
 studUserList.add(studUser);  
 }  
 return studUserList;  
 }  
   
 @Override  
 public boolean validateAndSetStudDiv(StudentAcad s, String divName) {  
 String divId = s.getYear() + s.getDepartment().getDeptId() + divName;  
 Divisions div = divisionsRepository.findByDivId(divId);  
 if(div!=null) {  
 s.setDiv(div);  
 s.setRollno(divId + s.getRollno());  
 return true;  
 }  
 return false;  
 }  
   
}

# spring\service\StudentHomeService.java

package com.qrms.spring.service;  
  
import org.springframework.web.servlet.ModelAndView;  
  
public interface StudentHomeService {  
 public ModelAndView getStudentHome(String userName);  
}

# spring\service\StudentHomeServiceImpl.java

package com.qrms.spring.service;  
  
import java.util.ArrayList;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.web.servlet.ModelAndView;  
  
import com.qrms.spring.model.StudentAcad;  
import com.qrms.spring.model.StudentAllocCourse;  
import com.qrms.spring.repository.StudentAcadRepository;  
import com.qrms.spring.repository.StudentAllocCourseRepository;  
  
@Service  
public class StudentHomeServiceImpl implements StudentHomeService{  
  
 @Autowired  
 private StudentAcadRepository studentAcadRepository;  
   
 @Autowired  
 private StudentAllocCourseRepository studentAllocCourseRepository;   
   
 @Override  
 public ModelAndView getStudentHome(String userName) {  
 ModelAndView model = new ModelAndView();  
 StudentAcad studentProfile = studentAcadRepository.findByUserName(userName);  
 if(studentProfile!=null)  
 model.addObject("studentProfile",studentProfile);  
 ArrayList<StudentAllocCourse> studentAllocations = studentAllocCourseRepository.findByStudent(studentProfile);  
 if(!studentAllocations.isEmpty())  
 model.addObject("studentAllocations", studentAllocations);  
 model.setViewName("/student/home");  
 return model;  
 }  
  
}

# spring\service\StudentPrefService.java

package com.qrms.spring.service;  
  
import java.util.List;  
  
import org.springframework.web.servlet.ModelAndView;  
  
import com.qrms.spring.model.StudentPref;  
import com.qrms.spring.queryBeans.StudentPrefCountInfo;  
  
public interface StudentPrefService {  
 public void saveStudentPref(StudentPref studentPref, String username);  
 public List<StudentPrefCountInfo> computeStudPrefTable();  
 public void setStudentPrefs(String course1,String course2,String course3,String course4,String userName);  
 public ModelAndView getStudentPrefs(String elective\_id,String userName);  
 public ModelAndView getElectiveId(String msg,String userName);  
}

# spring\service\StudentPrefServiceImpl.java

package com.qrms.spring.service;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.web.servlet.ModelAndView;  
  
import com.qrms.spring.model.Course;  
import com.qrms.spring.model.ElectiveVacancyPrefCounts;  
import com.qrms.spring.model.Electives;  
import com.qrms.spring.model.StudentAcad;  
import com.qrms.spring.model.StudentPref;  
import com.qrms.spring.queryBeans.PrefGroupByCourseStudent;  
import com.qrms.spring.queryBeans.StudentCountByYearSem;  
import com.qrms.spring.queryBeans.StudentPrefCountInfo;  
import com.qrms.spring.repository.CourseRepository;  
import com.qrms.spring.repository.ElectiveVacancyPrefCountsRepository;  
import com.qrms.spring.repository.ElectivesRepository;  
import com.qrms.spring.repository.StudentAcadRepository;  
import com.qrms.spring.repository.StudentPrefRepository;  
  
@Service  
public class StudentPrefServiceImpl implements StudentPrefService {  
   
 @Autowired  
 private StudentPrefRepository studentPrefRepository;  
  
 @Autowired  
 private StudentAcadRepository studentAcadRepository;  
   
 @Autowired  
 private CourseRepository courseRepository;  
   
 @Autowired  
 private ElectivesRepository electivesRepository;  
   
 @Autowired  
 private ElectiveVacancyPrefCountsRepository electiveVacancyPrefCountsRepository;  
   
   
 @Override  
 public void saveStudentPref(StudentPref student, String username) {  
 student.setUserName(username);  
 studentPrefRepository.save(student);  
   
 }  
  
 @Override  
 public List<StudentPrefCountInfo> computeStudPrefTable() {  
 StudentCountByYearSem totalStudentCount;  
   
 PrefGroupByCourseStudent p;  
 List<StudentPrefCountInfo> studCountInfo = new ArrayList<StudentPrefCountInfo>();  
   
// totalStudentCount = studentAcadRepository.findStudentCountByYearSemDept();  
// prefsPerElective = studentPrefRepository.findPrefsGroupByCourseStudent();  
 Course c;  
   
 List<Course> openCourses = courseRepository.findByStudAllocFlagNot(0);  
   
// for(PrefGroupByCourseStudent p: prefsPerElective) {  
// c = courseRepository.findByCourseId(p.getCourseId());  
 for(Course openCourse: openCourses) {  
 c = openCourse;  
 totalStudentCount = studentAcadRepository.findStudentCountByYearSemDept(c.getCourseYear(),c.getCourseSem(),c.getDepartment());  
 p = studentPrefRepository.findPrefsGroupByCourseStudent(c.getCourseId());  
 StudentPrefCountInfo si = new StudentPrefCountInfo();  
 si.setCourseId(c.getCourseId());  
 si.setCourseName(c.getCourseName());  
 si.setDeptId(c.getDepartment().getDeptId());  
 si.setSem(c.getCourseSem());  
 if(p!=null) {  
 si.setSubmitCount(p.getCount());  
 }else {  
 si.setSubmitCount(0);  
 }  
   
 si.setTotalStudentCount(totalStudentCount.getCount());  
 si.setYear(c.getCourseYear());  
 studCountInfo.add(si);  
// openCourses.remove(c);  
 }  
   
   
// for(Course openCourse: openCourses) {  
// StudentPrefCountInfo si = new StudentPrefCountInfo();  
// si.setCourseId(openCourse.getCourseId());  
// si.setCourseName(openCourse.getCourseName());  
// si.setDeptId(openCourse.getDepartment().getDeptId());  
// si.setSem(openCourse.getCourseSem());  
// si.setSubmitCount(0);  
// for(StudentCountByYearSem s: totalStudentCount) {  
// if(s.getSem() == openCourse.getCourseSem() && s.getYear().equals(openCourse.getCourseYear())) {  
// si.setTotalStudentCount(s.getCount());  
// break;  
// }  
// }  
// si.setYear(openCourse.getCourseYear());  
// studCountInfo.add(si);  
// }  
 return studCountInfo;  
 }  
  
 @Override  
 public void setStudentPrefs(String course1,String course2,String course3,String course4,String userName) {  
 Electives electives[] = {electivesRepository.findByElectiveCourseId(course1),electivesRepository.findByElectiveCourseId(course2),electivesRepository.findByElectiveCourseId(course3),electivesRepository.findByElectiveCourseId(course4)};  
   
 for(int i = 0;i<4;i++) {  
 System.out.println(electives[i].getElectiveCourseId());  
 StudentPref studentPref = new StudentPref(userName,electives[i].getCourse().getCourseId(),electives[i],i+1);   
 studentPrefRepository.save(studentPref);  
 }  
  
 ElectiveVacancyPrefCounts electiveVacancyPrefCounts = electiveVacancyPrefCountsRepository.findByElectiveId(electivesRepository.findByElectiveCourseId(course1).getElectiveCourseId());  
   
 int prefCount = electiveVacancyPrefCounts.getPrefCount();  
   
 electiveVacancyPrefCounts.setPrefCount(++prefCount);  
  
 electiveVacancyPrefCountsRepository.save(electiveVacancyPrefCounts);  
   
 }  
  
 @Override  
 public ModelAndView getStudentPrefs(String elective\_id, String userName) {  
  
 ModelAndView model = new ModelAndView();  
   
 StudentAcad currUserAcad = studentAcadRepository.findByUserName(userName);  
   
 ArrayList <StudentPref> studentPrefs = studentPrefRepository.findByUserNameAndCourseId(currUserAcad.getUserName(),elective\_id);  
 if(studentPrefs.size()!=0)   
 return getElectiveId("Your preferences for chosen elective have been recorded already!",userName);  
 else {  
 Course chosen\_course = courseRepository.findByCourseId(elective\_id);  
 ArrayList<Electives> electiveList = electivesRepository.findByCourse(chosen\_course);   
 if(electiveList.size()==0) {  
 String msg = "Please choose other elective-id!";  
 return getElectiveId(msg,userName);  
 }  
   
 ArrayList<Course> elective\_ids = courseRepository.findByCourseSemAndCourseYearAndCourseTypeNotAndDepartmentAndIsTheoryAndStudAllocFlag(currUserAcad.getSem(),currUserAcad.getYear(),'R',currUserAcad.getDepartment(),1,1);  
 model.addObject("chosen\_course\_name",chosen\_course.getCourseName());  
 model.addObject("chosen\_course\_id",chosen\_course.getCourseId());  
 model.addObject("elective\_ids",elective\_ids);  
 model.addObject("studentPref",new StudentPref());  
 model.addObject("courseList", electiveList);  
 model.setViewName("student/studentPref");  
 return model;  
 }  
 }  
  
 @Override  
 public ModelAndView getElectiveId(String msg, String userName) {  
 ModelAndView model = new ModelAndView();  
 StudentAcad currUserAcad = studentAcadRepository.findByUserName(userName);  
  
 ArrayList<Course> elective\_ids = courseRepository.findByCourseSemAndCourseYearAndCourseTypeAndDepartmentAndIsTheoryAndStudAllocFlag(currUserAcad.getSem(),currUserAcad.getYear(),'E',currUserAcad.getDepartment(),1,1);  
 ArrayList<Course> open\_elective\_ids = courseRepository.findByCourseSemAndCourseYearAndCourseTypeAndIsTheoryAndStudAllocFlag(currUserAcad.getSem(),currUserAcad.getYear(),'O',1,1);  
   
 elective\_ids.addAll(open\_elective\_ids);  
   
 model.addObject("elective\_ids",elective\_ids);  
 model.setViewName("/student/studentPref");  
  
 if(msg!=null)  
 model.addObject("msg",msg);  
 return model;  
 }  
   
   
}

# spring\service\TimeSlotsService.java

package com.qrms.spring.service;  
  
public interface TimeSlotsService {  
 public void updateTimeSlots();  
}

# spring\service\TimeSlotsServiceImpl.java

package com.qrms.spring.service;  
  
import java.util.List;  
  
import org.springframework.beans.factory.annotation.Autowired;  
  
import com.qrms.spring.model.CurrentTimeSlots;  
import com.qrms.spring.model.OldTimeSlots;  
import com.qrms.spring.repository.CurrTimeSlotsRepository;  
import com.qrms.spring.repository.OldTimeSlotsRepository;  
  
public class TimeSlotsServiceImpl implements TimeSlotsService{  
   
 @Autowired  
 private CurrTimeSlotsRepository currTimeSlotsRepository;  
   
 @Autowired  
 private OldTimeSlotsRepository oldTimeSlotsRepository;  
   
 @Override  
 public void updateTimeSlots() {  
 List<CurrentTimeSlots> currTimeSlots = currTimeSlotsRepository.findAll();  
 for(CurrentTimeSlots cts:currTimeSlots) {  
 OldTimeSlots ots = new OldTimeSlots();  
 ots.setActivity(cts.getActivity());  
 ots.setActivityIncharge(cts.getActivityIncharge());  
 ots.setDate(cts.getDate());  
 ots.setDay(cts.getDay());  
 ots.setStartTime(cts.getStartTime());  
 ots.setEndTime(cts.getEndTime());  
 ots.setResourceId(cts.getResourceId());  
 oldTimeSlotsRepository.save(ots);  
 }  
 }  
  
}

# spring\service\UserService.java

package com.qrms.spring.service;  
import java.util.ArrayList;  
import java.util.Set;  
  
import com.qrms.spring.model.Role;  
import com.qrms.spring.model.Users;  
  
public interface UserService {  
  
 public void saveUser(Users user);  
 public Users findUserByEmail(String email);  
 public Users findByUserName(String username);  
 public void createPasswordTokenForUser(Users user, String token);  
 public String validatePasswordResetToken(String username,String token);  
 public void savePassword(Users user, String password);  
 public ArrayList<Users> findByRole(Set<Role> role);  
}