**CHAPTER 5**

**CODING**

**5.1 Introduction:**

**5.1.1: Domain classes:**

The domain classes are the Plain Old Java Objects and they are the replica of the database table’s columns. The domain classes consists of variables, getters and setters and constructors and annotations like @Entity, @Id, @Column etc.., The annotations are present in “javax.persistence.\*” library.

**5.1.1.1: Data Domain class:**

@Entity

**public** **class** Data {

@Id

@GeneratedValue(strategy=GenerationType.***AUTO***)

**int** id;

@Column

String code;

@Column

String institutename;

@Column

String coursecode;

@Column

String region;

@Column

String district;

@Column

String place;

@Column

Integer ocb;

@Column

Integer ocg;

@Column

Integer scb;

@Column

Integer scg;

@Column

Integer stb;

@Column

Integer stg;

@Column

Integer bcab;

@Column

Integer bcag;

@Column

Integer bcbb;

@Column

Integer bcbg;

@Column

Integer bccb;

@Column

Integer bccg;

@Column

Integer bcdb;

@Column

Integer bcdg;

@Column

Integer bceb;

@Column

Integer bceg;

@Column

String type;

@Column

**int** year;

//Constructors, getters and setters  
}

**5.1.1.2: Role Domain class:**

@Entity

**public** **class** Role {

@Id

**int** roleid;

@Column

String rolename;

//Constructors, getters and setters  
 }

**5.1.1.3: User Domain class:**

@Entity

**public** **class** User {

@Id

@GeneratedValue(strategy=GenerationType.***AUTO***)

**int** id;

@Column

String username;

@Column

String password;

@Column

String email;

@Column

String contact;

@ManyToOne  
Role role;

//Constructors, getters and setters  
}

**5.1.2: User Model classes:**

The model classes are the also the Plain Old Java Objects, but these are used to take the data from the view layer (User Interface). The members of these classes are same as the members of the domain classes but the difference between domain classes and model classes arises at the annotations. The model classes consists of @Max, @Min, @Email, @NotEmpty etc.., The annotations are present in ” org.hibernate.validator.constraints” library.

**5.1.2.1: User Model Class:**

**public** **class** UserModel {

**int** id;

@Range(min=5,max=20)

String username;

@NotEmpty

String password;

@NotEmpty

String confirmPassword;

@Email

String email;

@NotEmpty

String contact;

**int** role;  
 String rolename;

//Constructors, getters and setters  
 }

**5.2 Code to save/update a record:**

**5.2.1: View:**

The data from the user is taken from the view and it is set to the model object. On submitting the form, the request mapping is done and the control goes to the controller.

<form method=*"post"* th:action=*"@{/submit}"* th:object=*"${userPojo}"* action=*"#"*>

<label for=*"username"* class=*"col-sm-2 control-label text-right"*>UserName:</label>

<input type=*"text"* class=*"form-control"* th:field=*"\*{username}"* id=*"username"* minlength=*"7"* name=*"username"* placeholder=*"username"* required=*"required"* />

<label for=*"password"* class=*"col-sm-2 control-label text-right"*>Password:</label>

<input type=*"password"* class=*"form-control"* th:field=*"\*{password}"* id=*"password"* minlength=*"7"* name=*"password"* placeholder=*"password"* required=*"required"*/>

<label for=*"confirmPassword"* class=*"col-sm-2 control-label text-right"*>Confirm Password:</label>

<input type=*"password"* class=*"form-control"* th:field=*"\*{confirmPassword}"* id=*"confirmPassword"* minlength=*"7"* name=*"confirmPassword"* placeholder=*"confirm password"* required=*"required"* oninput=*"check(this)"*/>

<label for=*"email"* class=*"col-sm-2 control-label text-right"*>E-mail:</label>

<input type=*"email"* class=*"form-control"* th:field=*"\*{email}"* id=*"email"* name=*"email"* placeholder=*"email"* required=*"required"*/>

<label for=*"contact"* class=*"col-sm-2 control-label text-right"*>Contact:</label>

<input type=*"tel"* class=*"form-control"* th:field=*"\*{contact}"* id=*"contact"* name=*"contact"* placeholder=*"contact"* required=*"required"* />

<label for=*"role"* class=*"col-sm-2 control-label text-right"*>Role:</label>

<select th:field=*"\*{role}"* id=*"role"* name=*"role"* class=*"form-control"*>

<option value=*"2"* name=*"Admin"*>Admin</option>

</select>

<button type=*"submit"* class=*"btn btn-default"*>Add</button>

<input type=*"reset"* class=*"btn btn-default*

</form>

**5.2.2: Controller:**

@RequestMapping(value="/submit", method=RequestMethod.***POST***)

**public** String edit(@ModelAttribute UserModel userPojo, BindingResult result,Map<String, Object> map, Model model){

**if**(result.hasErrors()){

**return** Constants.***VIEW\_ADDRUPDATE***;

}

userService.edit(userPojo);

List<UserModel> userPojoList=userService.getAll();

map.put(Constants.***CONST\_USER\_LIST***,userPojoList );

model.addAttribute(Constants.***CONST\_USER\_COUNT***,userPojoList.size());

**return** Constants.***VIEW\_ADMINHOME***;

}

**5.2.3: Service:**

@Transactional

**public** **void** edit(UserModel userPojo) {

User user= **new** User();

user.setUsername(userPojo.getUsername());

user.setPassword(userPojo.getPassword());

user.setContact(userPojo.getContact());

user.setEmail(userPojo.getEmail());

user.setId(userPojo.getId());

**int** roleid=userPojo.getRole();

Role role= **new** Role();

role=roleDao.getUserRole(roleid);

user.setRole(role);

userDao.edit(user);

}

**5.2.4: Data Access Object (DAO):**

**public** **void** edit(User user) {

userRepo.saveAndFlush(user);

}

**5.2.5: Repository:**

**public** **interface** UserRepo **extends** JpaRepository<User, Integer> {

}

**5.3: Retrieve data:**

**5.3.1: View:**

<a th:href=*"@{/list}"*>List Users</a>

**5.3.2: Controller:**

@RequestMapping(value="/list", method=RequestMethod.***GET***)

**public** String getall(@ModelAttribute UserModel userPojo,Map<String, Object> map, Model model){

List<UserModel> userPojoList=userService.getAll();

map.put(Constants.***CONST\_USER\_LIST***,userPojoList );

model.addAttribute(Constants.***CONST\_USER\_COUNT***,userPojoList.size());

**return** Constants.***VIEW\_ADMINHOME***;

}

**5.3.3: Service:**

@Transactional

**public** List<UserModel> getAll() {

List<User> userList= **new** ArrayList<User>();

List<UserModel> userPojoList= **new** ArrayList<UserModel>();

userList= userDao.getAll();

**for**(User user:userList){

UserModel userPojo= **new** UserModel();

userPojo.setContact(user.getContact());

userPojo.setEmail(user.getEmail());

userPojo.setId(user.getId());

userPojo.setUsername(user.getUsername());

String rolename=user.getRole().getRollname();

userPojo.setRolename(rolename);

userPojoList.add(userPojo);

}

**return** userPojoList;

}

**5.3.4: Data Access Object (DAO):**

**public** List<User> getAll() {

**return** userRepo.findAll();

}

**5.3.5: Repository:**

**public** **interface** UserRepo **extends** JpaRepository<User, Integer> {

}

**5.4: Excel data upload to database table:**

**5.4.1: View:**

<form method=*"post"* th:action=*"@{/uploaddata}"* action=*"#"* enctype=*"multipart/form-data"* th:object=*"${dataPojo}"*>

<input type=*"file"* name=*"file"* value=*"file"* required=*"required"*/><br/>

<select th:field=*"\*{type}"*>

<option value=*"All"* name=*"All"* >All</option>

<option value=*"Engineering"* name=*"Engineering"* >Engineering</option> <option value=*"Medical"* name=*"Medical"*>Medical</option>

<option value=*"Agricultural"* name=*"Agricultural"*>Agricultural</option>

</select>

<select th:field=*"\*{year}"*>

<option value=*"2014"* name=*"2014"* >2014</option>

<option value=*"2015"* name=*"2015"*>2015</option>

<option value=*"2016"* name=*"2016"*>2016</option>

<option value=*"2017"* name=*"2017"*>2017</option>

<option value=*"2018"* name=*"2018"*>2018</option>

<option value=*"2019"* name=*"2019"*>2019</option>

<option value=*"2020"* name=*"2020"*>2020</option>

</select>

<input type=*"Submit"* value=*"Upload"*/>

</form>

**5.4.2: Controller:**

@RequestMapping(value="/uploaddata",headers = "content-type=multipart/\*", method=RequestMethod.***POST***)

**public** String handleFormUpload(@RequestParam("file") MultipartFile file, @ModelAttribute DataModel dataPojo, Model model,Map<String, Object> map)**throws** IOException {

logger.log(Level.***INFO***,"Controller");

String status = **null**;

**try**{

**if** (!file.isEmpty()) {

String check=ValidateUpload.*validateOfficeData*(file);

**if**(check.equals(Constants.***CONST\_DATA\_VALID***)){

**try**{

status=dataServiceSave.saveContents(file, dataPojo);

}**catch**(InvalidException invalidException){

**return** Constants.***VIEW\_DATAMANAGE***;

}

model.addAttribute(Constants.***CONST\_DATA\_MESSAGE***, status);

map.put(Constants.***CONST\_DATA\_POJO***, dataPojo);

}

**else**{

model.addAttribute(Constants.***CONST\_DATA\_MESSAGE***, check);

}

}

}**catch**(MultipartException multipartException){

logger.log(Level.***SEVERE***,multipartException.getMessage(), multipartException);

}

**return** Constants.***VIEW\_DATAMANAGE***;

}

**5.4.3: Validation of Excel file:**

**public** **static** String validateOfficeData(MultipartFile file){

/\*

\* Check whether file is of excel type

\*/

**if**(!file.getContentType().equals(Constants.***CONST\_EXCEL\_SHEET\_TYPE***)){

**return** Constants.***CONST\_ERROR\_IN\_FILE\_TYPE***;

}

/\*

\* Check whether file size is less than 10 MB

\*/

**if**(file.getSize() > 10\*1024\*1024) {

**return** Constants.***CONST\_ERROR\_IN\_FILE\_SIZE***;

}

**return** Constants.***CONST\_DATA\_VALID***;

}

**5.4.4: Service:**

@Transactional

**public** String saveContents(MultipartFile file, DataModel dataPojo) **throws** InvalidException {

XSSFWorkbook offices;

**int** rowValue=1;

**try** {

offices = **new** XSSFWorkbook(file.getInputStream());

} **catch** (IOException e) {

**throw** **new** RuntimeException(e);

}

XSSFSheet worksheet=offices.getSheetAt(0);

Iterator<Row> rite= worksheet.rowIterator();

**if**(rite.hasNext()){

XSSFRow row=(XSSFRow)rite.next();

**if**(!ValidateUpload.*validateHeader*(row)){

**return** Constants.***CONST\_ERROR\_IN\_EXCEL\_FORMAT***;

}

}

Iterable<Data> dataList= **new** ArrayList<Data>();

**while**(rite.hasNext()){

XSSFRow dataRow=(XSSFRow)rite.next();

ArrayList<Integer> intList= **new** ArrayList<Integer>();

ArrayList<String> strList=**new** ArrayList<String>();

Iterator<Cell> cite = dataRow.cellIterator();

**while**(cite.hasNext()){

Cell cell = cite.next();

**switch**(cell.getCellType()){

**case** Cell.***CELL\_TYPE\_STRING***: strList.add(cell.getStringCellValue().toString()); **break**;

**case** Cell.***CELL\_TYPE\_NUMERIC***: intList.add((**int**) cell.getNumericCellValue()); **break**;

**case** Cell.***CELL\_TYPE\_BLANK***: **throw** **new** InvalidException(Constants.***CONST\_ERROR\_BLANK\_DATA\_CELLS***);

**default** : **throw** **new** InvalidException(Constants.***CONST\_ERROR\_IN\_EXCEL\_CELL***);

}

}

rowValue++;

Data data=**new** Data();

**if**((intList.size()<17)||(strList.size()<7)){

**return** Constants.***CONST\_ERROR\_INVALID\_DATA***+rowValue;

}

data.setCode(strList.get(0));

data.setInstitutename(strList.get(1));

data.setCoursecode(strList.get(2));

data.setRegion(strList.get(3));

data.setdistrict(strList.get(4));

data.setPlace(strList.get(5));

data.setType(strList.get(6));

data.setOcb(intList.get(0));

data.setOcg(intList.get(1));

data.setScb(intList.get(2));

data.setScg(intList.get(3));

data.setStb(intList.get(4));

data.setStg(intList.get(5));

data.setBcab(intList.get(6));

data.setBcag(intList.get(7));

data.setBcbb(intList.get(8));

data.setBcbg(intList.get(9));

data.setBccb(intList.get(10));

data.setBccg(intList.get(11));

data.setBcdb(intList.get(12));

data.setBcdg(intList.get(13));

data.setBceb(intList.get(14));

data.setBceg(intList.get(15));

data.setYear(dataPojo.getYear());

((ArrayList<Data>)dataList).add(data);

intList.clear();

strList.clear();

}

String type=dataPojo.getType();

**int** year = dataPojo.getYear();

**if**(dataPojo.getType().equalsIgnoreCase("All")){

dataDao.deleteAll(year);

}

**else**{

dataDao.delete(type, year);

}

dataDao.saveAll(dataList);

**return** Constants.***CONST\_SUCCESSFULL***;

}

**5.4.5: Data Access Object (DAO):**

**public** **void** saveAll(Iterable<Data> dataList) {

dataRepo.save(dataList);

}

**5.4.6: Repository:**

**public** **interface** DataRepo **extends** JpaRepository<Data, Long> {

}

**5.5: Download data from database table in the form of excel file:**

**5.5.1: View:**

<form method=*"post"* th:action=*"@{/downloadData}"* action=*"#"* enctype=*"multipart/form-data"* th:object=*"${dataPojo}"*>

<select th:field=*"\*{type}"*>

<option value=*"All"* name=*"All"* >All</option>

<option value=*"Engineering"* name=*"Engineering"* >Engineering</option> <option value=*"Medical"* name=*"Medical"*>Medical</option>

<option value=*"Agricultural"* name=*"Agricultural"*>Agricultural</option>

</select>

<select th:field=*"\*{year}"*>

<option value=*"2014"* name=*"2014"* >2014</option>

<option value=*"2015"* name=*"2015"*>2015</option>

<option value=*"2016"* name=*"2016"*>2016</option>

<option value=*"2017"* name=*"2017"*>2017</option>

<option value=*"2018"* name=*"2018"*>2018</option>

<option value=*"2019"* name=*"2019"*>2019</option>

<option value=*"2020"* name=*"2020"*>2020</option>

</select>

<input type=*"submit"* value=*"download"*/>

</form>

**5.5.2: Controller:**

@RequestMapping(value = "/downloadData", method = RequestMethod.***POST***)

**public** String generateExcel(HttpServletRequest request, HttpServletResponse response, @ModelAttribute DataModel dataPojo, Model model, Map<String, Object> map) **throws** Exception {

List<Data> list = dataServiceSave.download(dataPojo);

**if**(list.size()==0){

. . model.addAttribute(Constants.***CONST\_DATA\_MESSAGE***,Constants.***CONST\_DATA\_NOT\_AVAILABLE***);

map.put(Constants.***CONST\_DATA\_POJO***, dataPojo);

}**else**{

response.setHeader(Constants.***CONST\_EXCEL\_CONTENT***, Constants.***CONST\_EXCEL\_ATTACHEMNT***);

response.setHeader(Constants.***CONST\_EXCEL\_CONTENT\_TYPE***, Constants.***CONST\_EXCEL\_SHEET\_TYPE***);

XSSFWorkbook workbook=ExcelDownload.*create*(list);

OutputStream outputStream=response.getOutputStream();

workbook.write(outputStream);

outputStream.flush();

outputStream.close();

}

**return** Constants.***VIEW\_DOWNLOAD***;

}

**5.5.3: Service:**

**public** List<Data> download(DataModel dataModel) **throws** FileNotFoundException {

List<Data> dataList= **new** ArrayList<Data>();

**if**(dataModel.getType().equalsIgnoreCase("All")){

dataList= dataDao.retrieve(dataModel.getYear());

}**else**{

dataList =dataDao.retrieve(dataModel.getType(), dataModel.getYear());

}

**return** dataList;

}

**5.5.4: Data Access Objects (DAO):**

**public** java.util.List<Data> retrieve(String type, Integer year) {

**return** dataRepo.findByTypeAndYear(type, year);

}

@Override

**public** List<Data> retrieve(**int** year) {

**return** dataRepo.findByYear(year);

}

**5.5.5: Repository:**

**public** **interface** DataRepo **extends** JpaRepository<Data, Long> {

List<Data> findByTypeAndYear(String type, Integer year);

List<Data> findByYear(**int** year);

}

**5.5.6: Writing data to excel workbook:**

**public** **static** XSSFWorkbook create(List<Data> data1){

XSSFWorkbook workbook = **null**;

**try**{

workbook = **new** XSSFWorkbook();

XSSFSheet spreadsheet = workbook.createSheet(Constants.***CONST\_DOWNLOAD\_FILE\_NAME***);

**int** rowIndex=0,columnIndex=0;

XSSFRow row =spreadsheet.createRow(rowIndex);

**for**(ExcelDataHeaders dataHeaders:ExcelDataHeaders.*values*()){

XSSFCell cell=row.createCell(columnIndex);

cell.setCellValue(dataHeaders.toString());

columnIndex++;

}

**for**(Data data:data1){

row=spreadsheet.createRow(++rowIndex);

XSSFCell id1=row.createCell(0);

id1.setCellValue(data.getCode());

XSSFCell code1=row.createCell(1);

code1.setCellValue(data.getInstitutename());

XSSFCell name1=row.createCell(2);

name1.setCellValue(data.getCoursecode());

XSSFCell region1=row.createCell(3);

region1.setCellValue(data.getRegion());

XSSFCell district1=row.createCell(4);

district1.setCellValue(data.getDistrict());

XSSFCell place1=row.createCell(5);

place1.setCellValue(data.getPlace());

XSSFCell ocb1=row.createCell(6);

ocb1.setCellValue(data.getOcb());

XSSFCell ocg1=row.createCell(7);

ocg1.setCellValue(data.getOcg());

XSSFCell scb1=row.createCell(8);

scb1.setCellValue(data.getScb());

XSSFCell scg1=row.createCell(9);

scg1.setCellValue(data.getScg());

XSSFCell stb1=row.createCell(10);

stb1.setCellValue(data.getStb());

XSSFCell stg1=row.createCell(11);

stg1.setCellValue(data.getStg());

XSSFCell bcab1=row.createCell(12);

bcab1.setCellValue(data.getBcab());

XSSFCell bcag1=row.createCell(13);

bcag1.setCellValue(data.getBcag());

XSSFCell bcbb1=row.createCell(14);

bcbb1.setCellValue(data.getBcbb());

XSSFCell bcbg1=row.createCell(15);

bcbg1.setCellValue(data.getBcbg());

XSSFCell bccb1=row.createCell(16);

bccb1.setCellValue(data.getBccb());

XSSFCell bccg1=row.createCell(17);

bccg1.setCellValue(data.getBccg());

XSSFCell bcdb1=row.createCell(18);

bcdb1.setCellValue(data.getBcdb());

XSSFCell bcdg1=row.createCell(19);

bcdg1.setCellValue(data.getBcdg());

XSSFCell bceb1=row.createCell(20);

bceb1.setCellValue(data.getBceb());

XSSFCell bceg1=row.createCell(21);

bceg1.setCellValue(data.getBceg());

XSSFCell type1=row.createCell(22);

type1.setCellValue(data.getType());

XSSFCell year1=row.createCell(23);

year1.setCellValue(data.getYear());

}

}**catch**(Exception exception){

***logger***.log(Level.***SEVERE***,exception.getMessage(), exception);

}

**return** workbook;

}