Project Report

by Vidushi Garg

Submission date: 24-Apr-2019 10:02PM (UTC-0400)
Submission ID: 1118694460
File name: ProjectReport.pdf (1.5M)
Word count: 7478
Character count: 56795

Vidushi Garg
Section - 6

Reader's Nest

This is a web-based application wherein a user can buy books from as per his/her interest.

Following are the ***roles*** that are currently considered for application access:

1. Admin: There will be only one admin who will take care of the staff working at the store. He shall have the authority to decide whether an applicant is a potential librarian and hence, will approve the rights given to him.
2. Librarian: Once he is approved by the admin, he will be responsible for adding and maintaining the database of books.
3. Customer: If a reader needs a book from this store, he needs to be a member of the Customer Club. So, he will need to register before making any purchase. Henceforth, he can surf through the list of all the books available in the store and buy as many books he wants.

Following ***functionalities*** that are the point of focus throughout the application:

1. A librarian can add, update and delete the books as per his will.
2. A customer can buy only one copy of a book at a time. However, in another order he can get the second.
3. The number of books that a librarian adds, will be visible to him only. There will be no cross-exchange of data visibility of data in between the staff members.
4. A customer will make a purchase unaware of the fact stated in 3.
5. Purchases made for the books under a librarian will be shown, unshared by other staff members.
6. User can sort the results based on price
7. Pagination is applied
8. Librarian cannot add a book if it already exists in the database
9. User will be created only if the credentials (email + phoneNumber) that he is trying with, don't exist already
10. All the validations for incorrect data/empty fields have been implemented

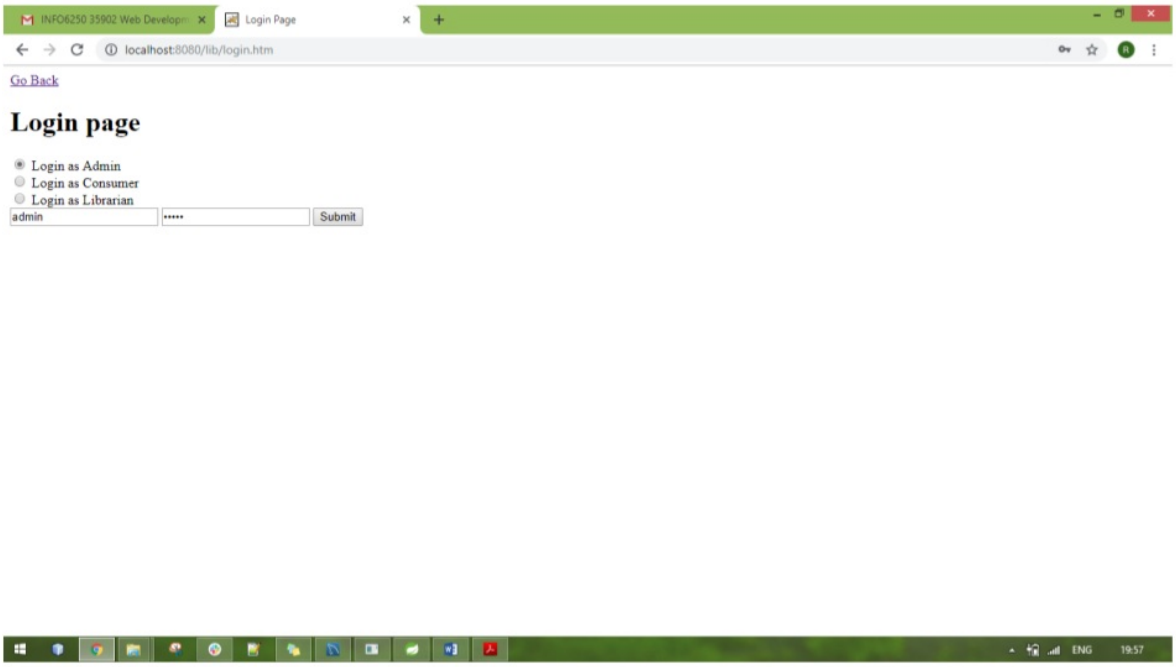
Technologies used to perform crucial operations are as follows:

1. *AJAX* is used along with jQuery for:
    - Search functionality
    - Activate/Deactivate the librarian (the status is displayed without the whole page-reload)
    - Sort the books based on price
    - The list of books will be updated on deleting an item (without the whole page reload)
2. *Interceptors* (preHandle()) is used to handle the incoming requests from the users.
3. *SpringMVC* has been used to control the webFlow
4. *Hibernate* is used to persist the data using *MySQL*
5. *Maven*

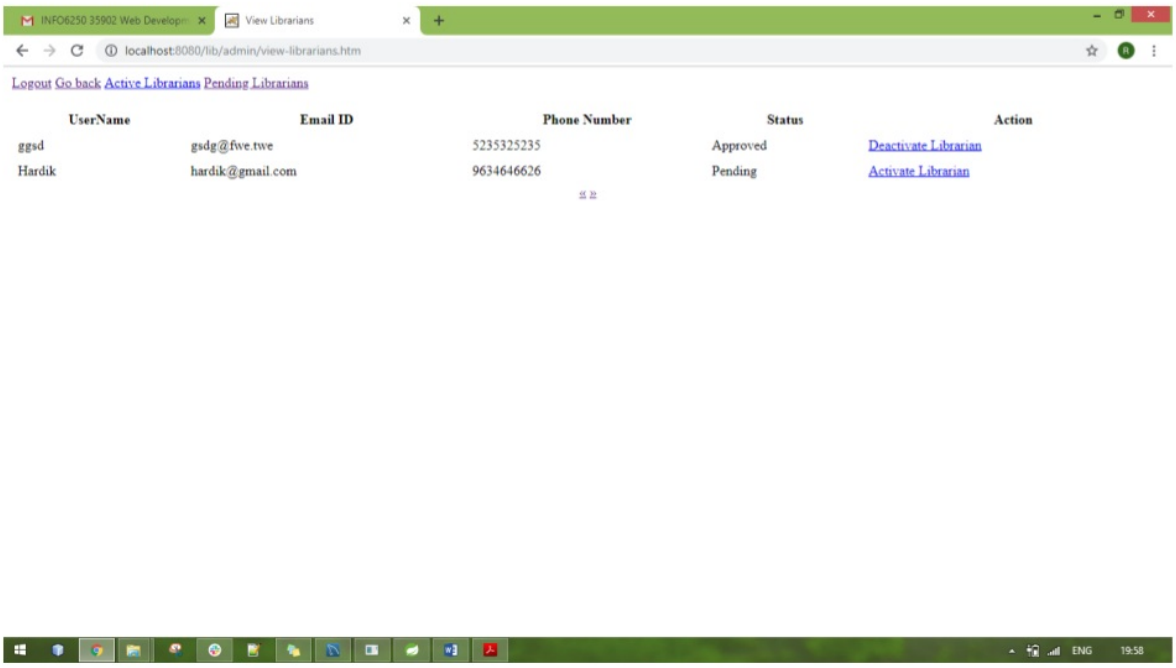
Vidushi Garg
Section - 6

SCREENSHOTS

Login Page:

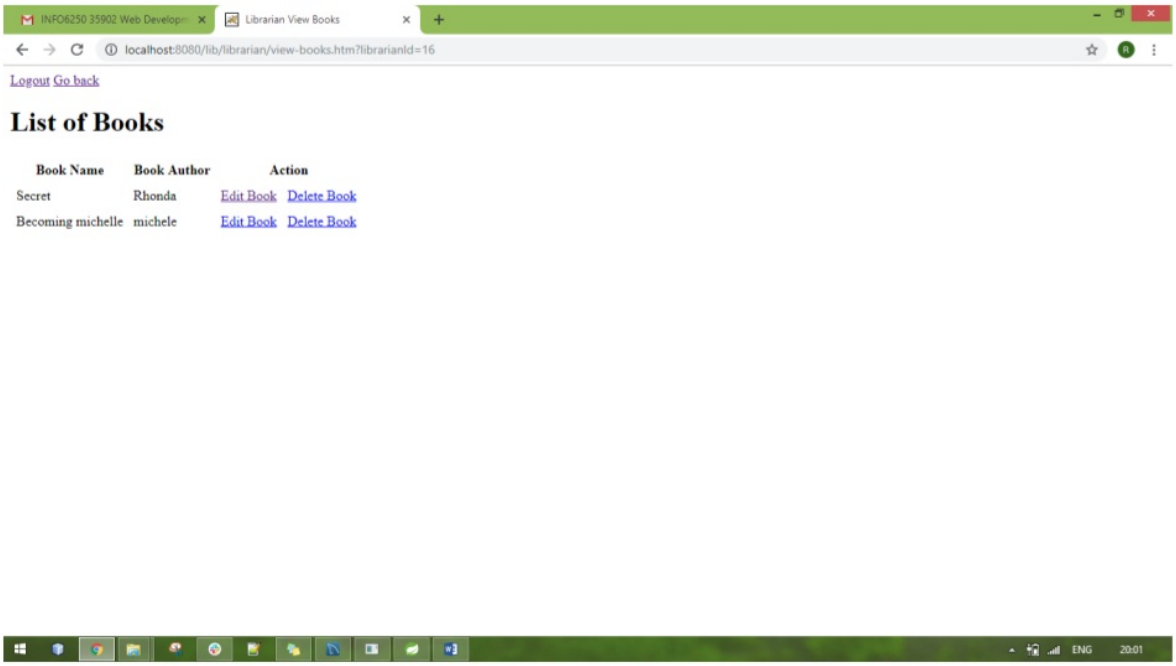


Admin dashboard to activate/deactivate the store staff:

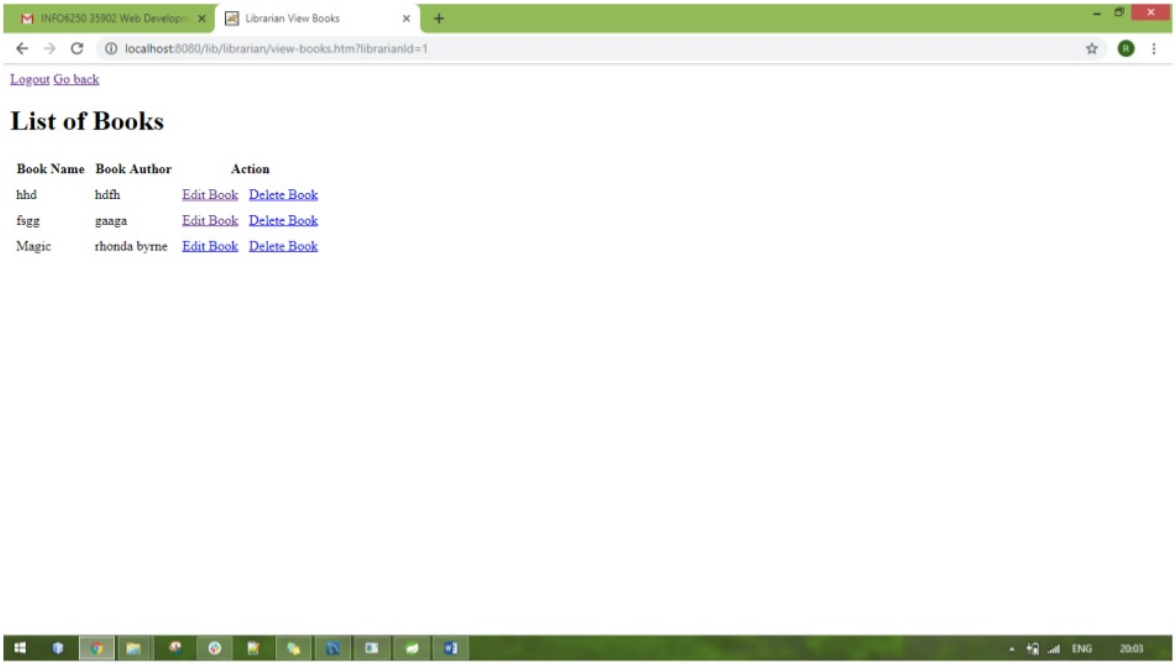


Vidushi Garg
Section - 6

The dashboard of a librarian:

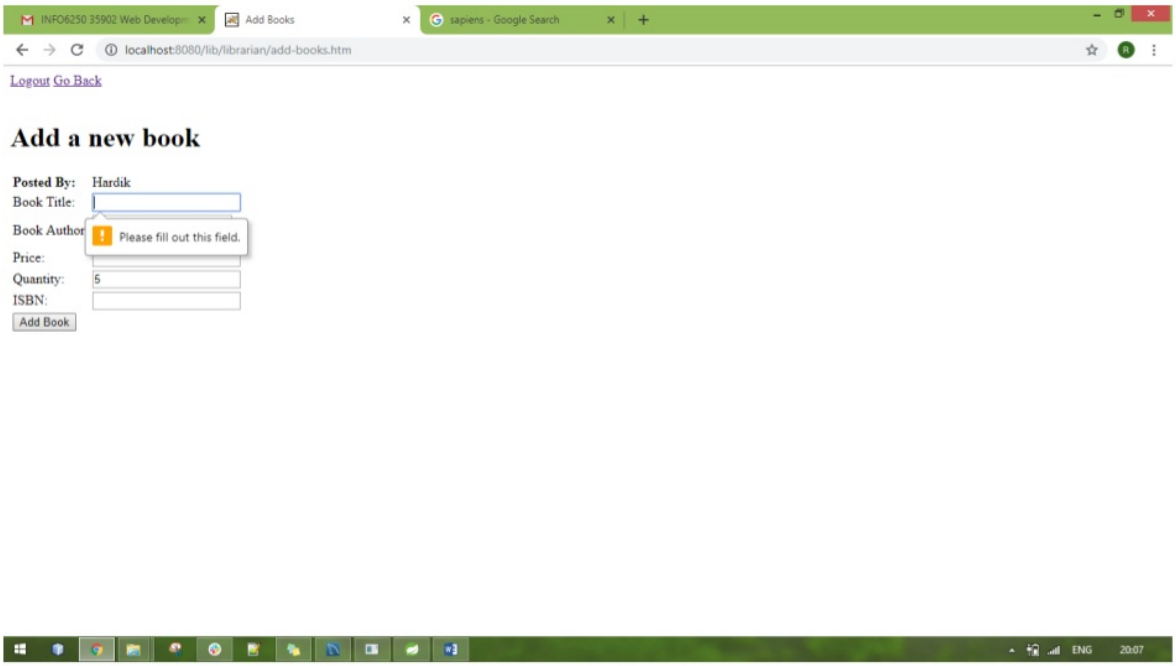


The dashboards of different librarians will contain the books added by himself only:

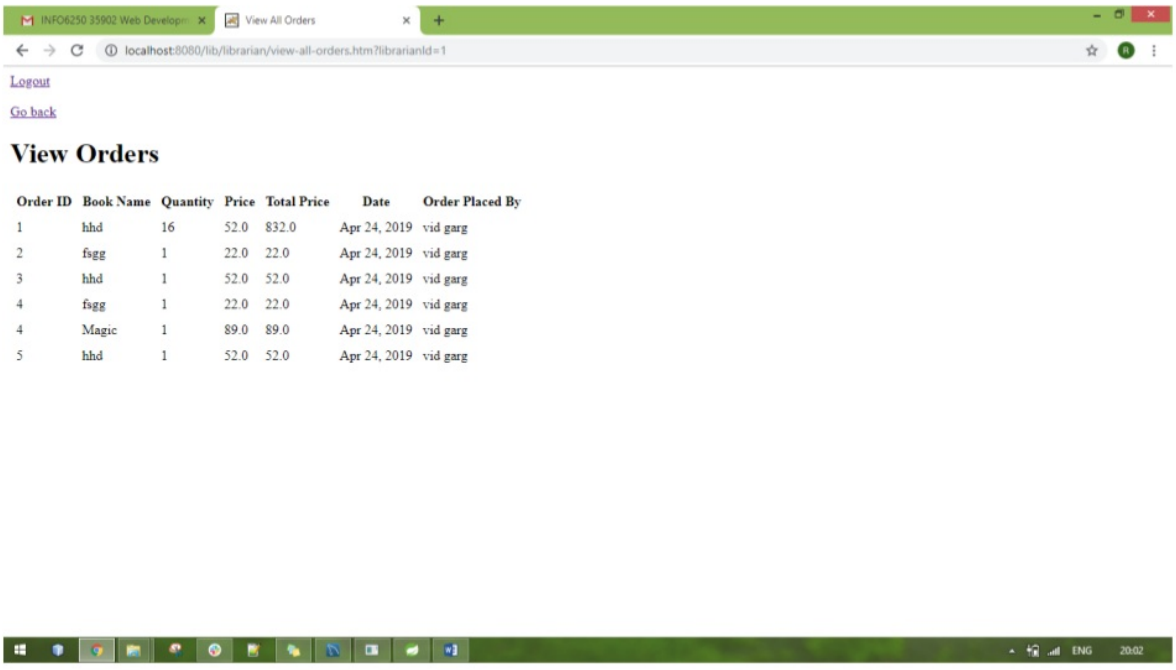


Vidushi Garg
Section - 6

Validations on update book details:

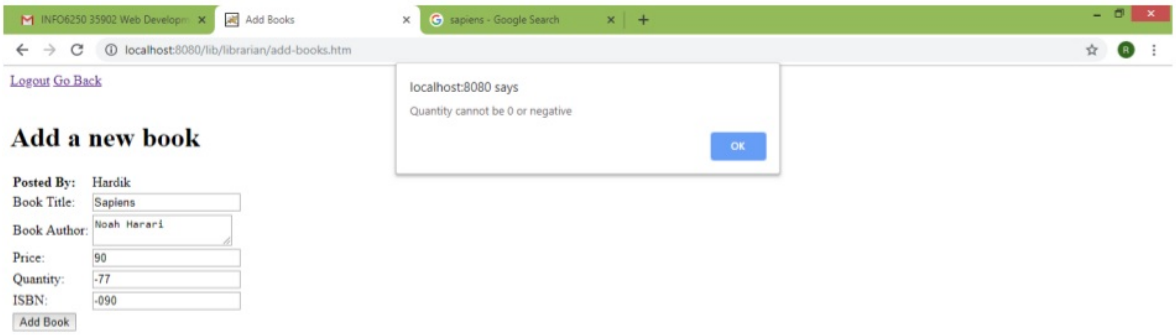


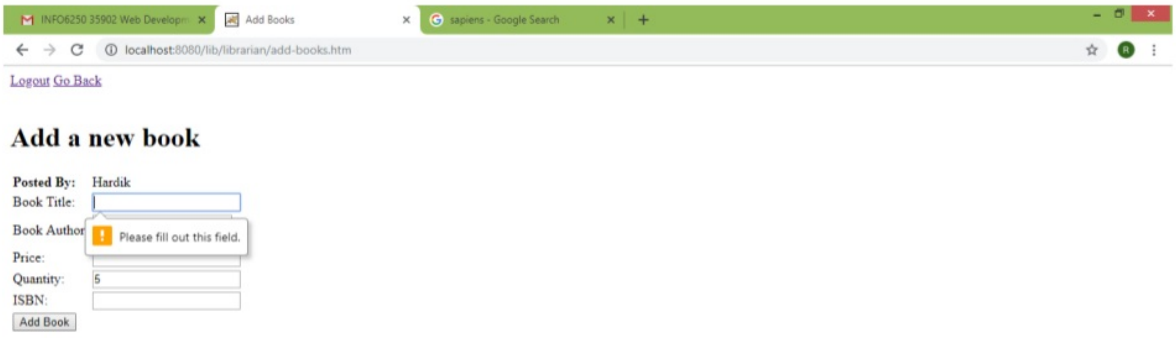
The order history page for the books added by librarian:



Vidushi Garg
Section - 6

Validations on the add-book:

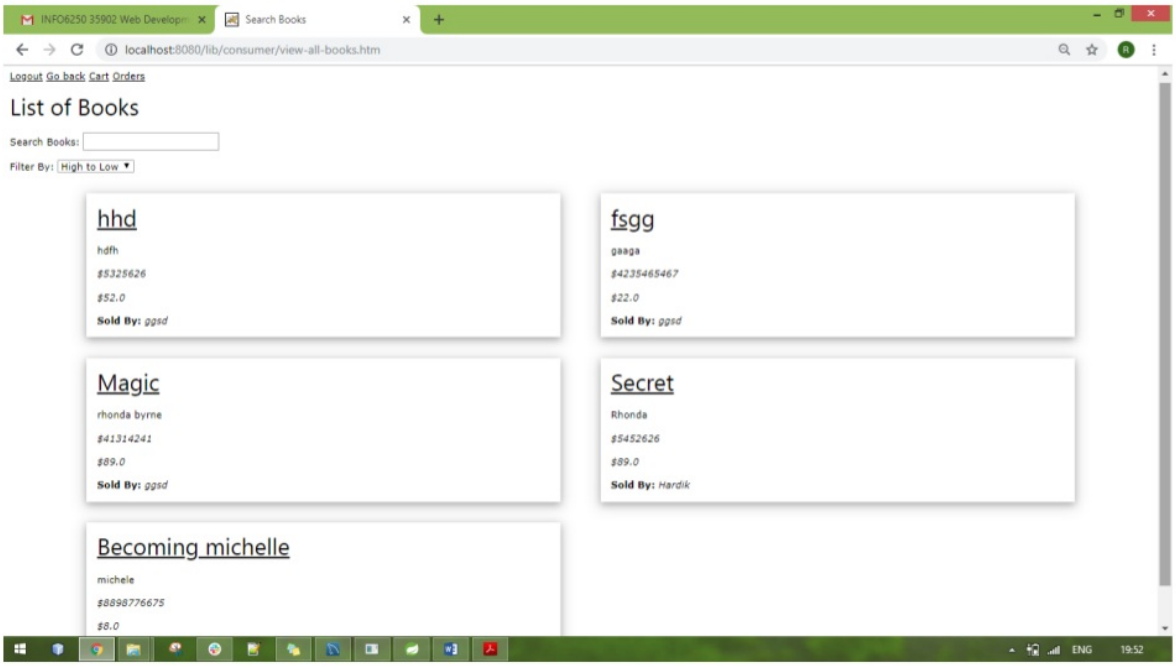




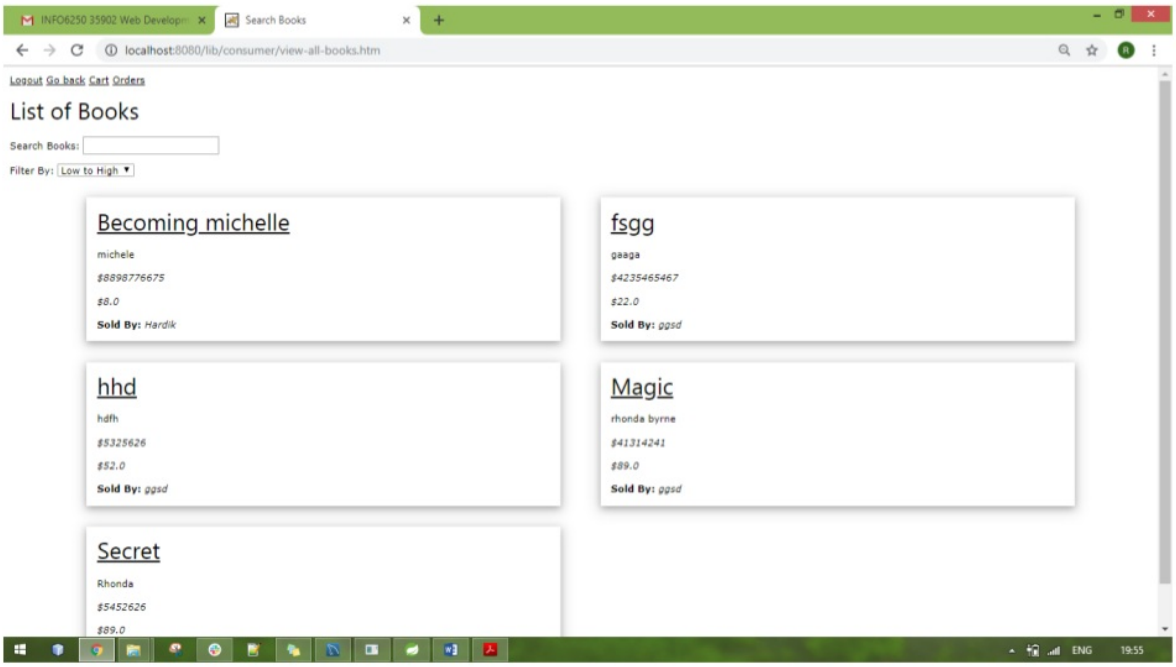
Vidushi Garg
Section - 6

The search page visible to customer:

Based on the sort order mentioned as high-to-low -

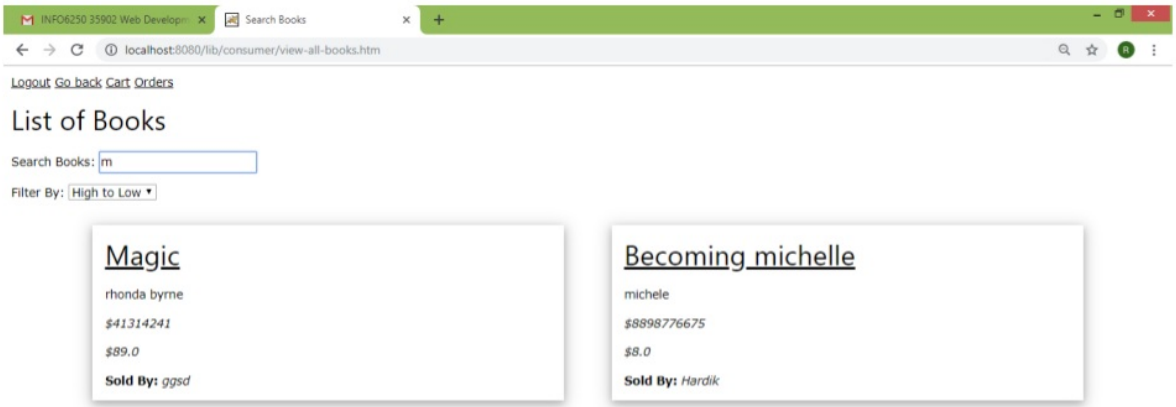


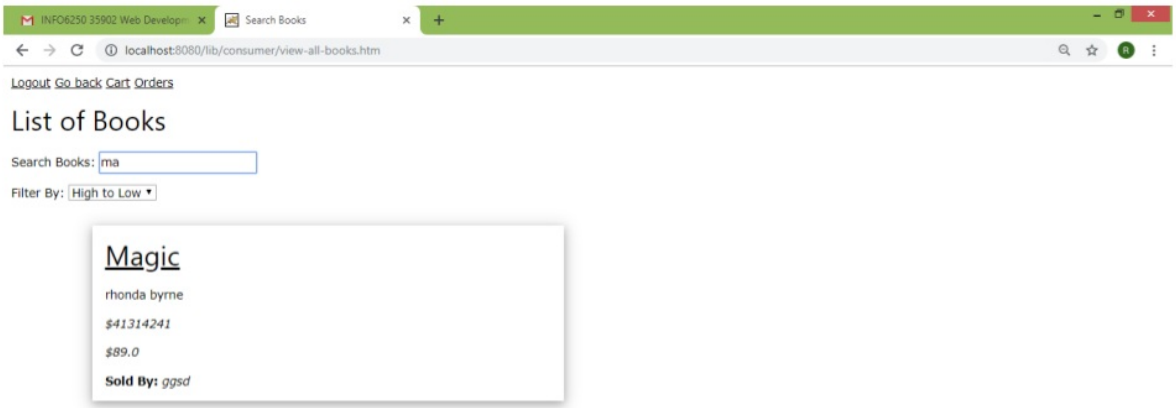
Based on the sort order mentioned as low-to-high -



Vidushi Garg
Section - 6

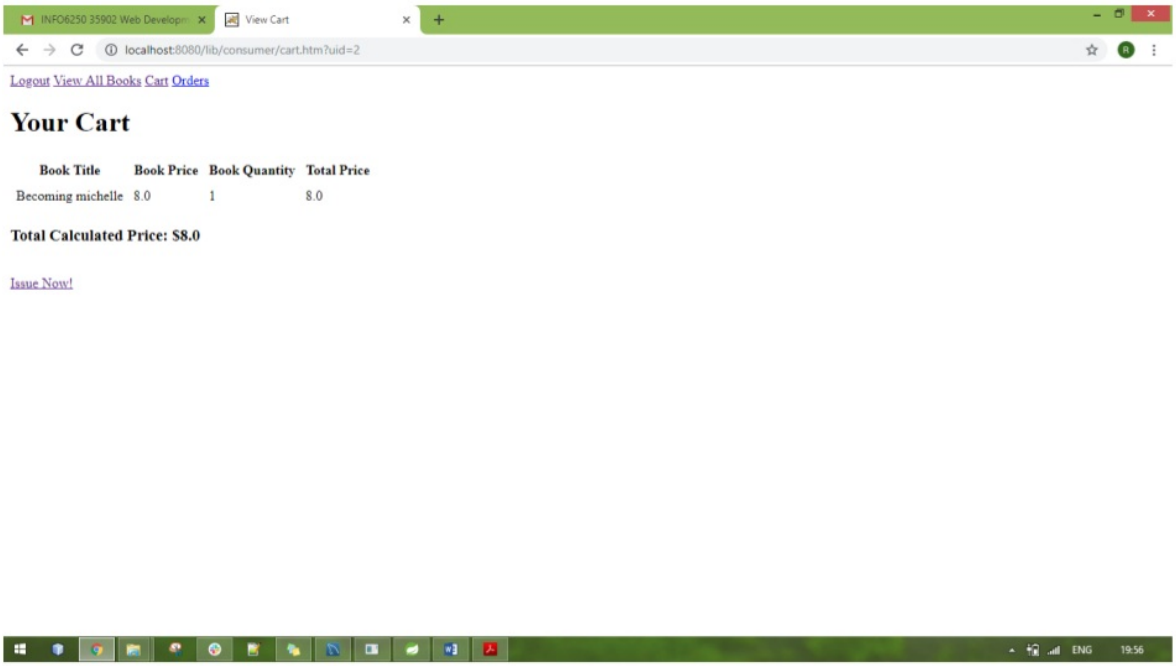
Use of AJAX for search filter:



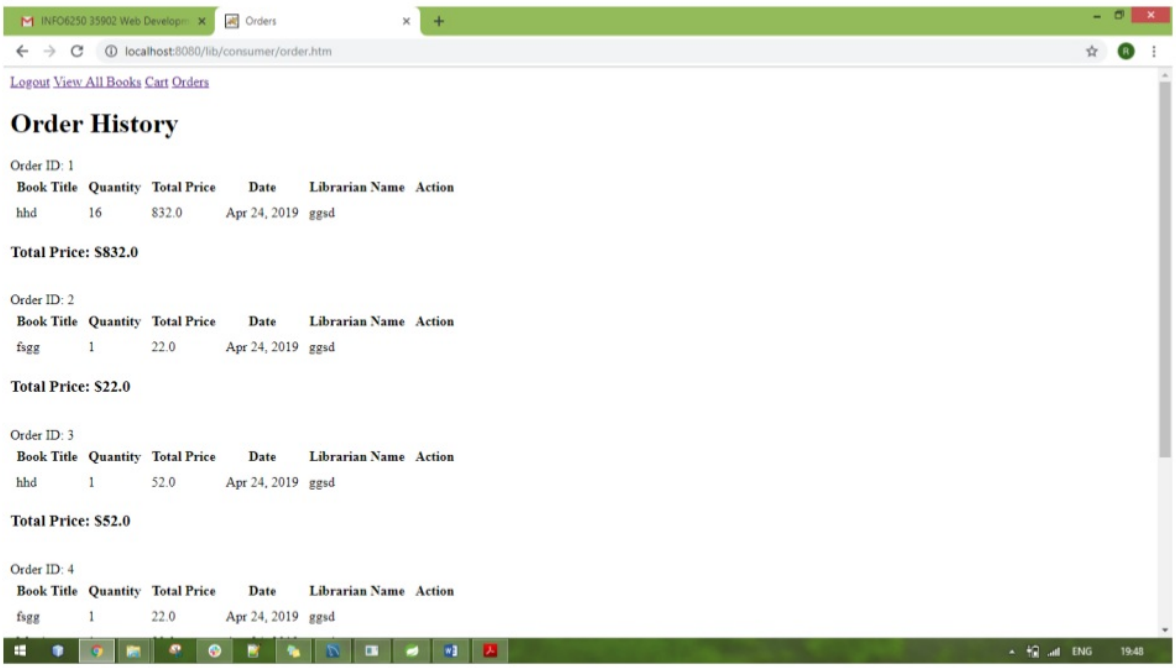


Vidushi Garg
Section - 6

Cart-page:



Order-History Page of the customer:



Vidushi Garg
Section - 6

APPENDIX

POJOs

Address.java

```
package com.me.lib.pojo;
import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.OneToOne;
import javax.persistence.Table;

@Entity
@Table(name = "address_table")
public class Address implements Serializable{
      public Address() {}
      public Address(String street, String city, String state, String zip, String
country) {  this.street = street;
            this.city = city;
            this.state = state;
            this.zip = zip;
            this.country = country;
      }
      @Id
      @GeneratedValue(strategy=GenerationType.IDENTITY)
      @Column(name = "addressID", unique=true, nullable = false)
      private long addressID;
      @Column(name="street")
      private String street;
      @Column(name="city")
      private String city;
      @Column(name="state")
      private String state;
      @Column(name="zip")
      private String zip;
      @Column(name="country")
      private String country;
      @Column(name="addressType")
      private String addressType;
      @ManyToOne
    private User user;
      @OneToOne
    private Librarian librarian;
      public long getAddressID() {return addressID;}
      public void setAddressID(long addressID) {this.addressID = addressID;}
      public String getStreet() {    return street;}
      public void setStreet(String street) {this.street = street;}
      public String getCity() {return city;}
      public void setCity(String city) {this.city = city;}
      public String getState() {return state;}
      public void setState(String state) {this.state = state;}
      public String getZip()  {return zip;}
      public void setZip(String zip) {this.zip = zip;}
      public String getCountry() {return country;}
      public void setCountry(String country) {this.country = country;}
      public String getAddressType() {return addressType;}
      public void setAddressType(String addressType) {this.addressType = addressType;}
      public User getUser() {  return user;}
```

```
	public void setUser(User user) {this.user = user;}
	public Librarian getLibrarian() {return librarian;}
	public void setLibrarian(Librarian librarian) { this.librarian = librarian;}
}
```

Admin.java

```
Package com.me.lib.pojo;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Admin {
	public Admin() {  }
	public Admin(String username, String password) {
		this.username = username;
		this.password = password;	}
	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	@Column(name = "adminId", nullable = false, unique=true)
	private long adminId;
	@Column(name="username")
	private String username;
	@Column(name="password")
	private String password;
	public long getAdminId()	{return adminId;}
	public void setAdminId(long adminId) {this.adminId = adminId;}
	public String getUsername() {return username;	}
	public void setUsername(String username) {this.username = username;}
	public String getPassword()  {return password;  }
	public void setPassword(String password) {this.password = password;}}
```

Book.java

```
package com.me.lib.pojo;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.Table;
@Entity
@Table(name = "book_table")
public class Book {
	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	@Column(name = "bookId", unique=true, nullable = false)
	private long bookId;
	@Column(name="title")
	private String title;
	@Column(name="isbn")
	private String isbn;
	@Column(name="author")
	private String author;
	@Column(name="charge")
	private Double charge;
	@Column(name="qty")
```

```
    private long quantity;
      @ManyToOne
    private Librarian librarian;
      @OneToMany(mappedBy = "book")
    private List<Cart> cart = new ArrayList<Cart>();
      @OneToMany(mappedBy = "books")
    private List<Order> order = new ArrayList<Order>();
      public Book() {}
      public long getBookId() {return bookId;}
      public void setBookId(long bookId) {this.bookId = bookId;}
      public String getTitle() {return title;}
      public void setTitle(String title) {this.title = title;}
      public String getIsbn() {return isbn;}
      public void setIsbn(String isbn) {this.isbn = isbn;}
      public String getAuthor() {   return author;}
      public void setAuthor(String author) {    this.author = author;}
      public Double getCharge() {   return charge;}
      public void setCharge(Double charge) {this.charge = charge;}
      public long getQuantity() {   return quantity;}
      public void setQuantity(long quantity) {this.quantity = quantity;}
      public Librarian getLibrarian() {return librarian;}
      public void setLibrarian(Librarian librarian) { this.librarian = librarian;}
      public List<Cart> getCart() {return cart;}
      public void setCart(List<Cart> cart) {this.cart = cart;}
      public List<Order> getOrder() {     return order;}
      public void setOrder(List<Order> order) {this.order = order;}}
```

Cart.java

```
package com.me.lib.pojo;
import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
import com.me.lib.pojo.User;

@Entity
@Table(name="cart_table")
public class Cart implements Serializable{
      public Cart()    {}
      @Id
    @ManyToOne
    @JoinColumn(name="userID")
    private User user;
      @Id
    @ManyToOne
    @JoinColumn(name="bookID")
    private Book book;
      @Column(name="quantity")
    private int quantity;
      public User getUser() {return user;}
      public void setUser(User user) {this.user = user;}
      public Book getBook() { return book;}
      public void setBook(Book book) {this.book = book;}
      public int getQuantity() {    return quantity;}
      public void setQuantity(int quantity) {   this.quantity = quantity;}
}
```

Vidushi Garg
Section - 6

Consumer.java

```
package com.me.lib.pojo;
import java.io.Serializable;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.Table;

@Entity
@Table(name="consumer_table")
public class Consumer implements Serializable{
	public Consumer() {}
	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	@Column(name = "consumerId", unique=true, nullable = false)
	private long consumerId;
	@Column(name="firstName")
	private String firstName;
	@Column(name="lastName")
	private String lastName;
	@Column(name="emailId")
	private String emailId;
	@OneToOne(cascade = CascadeType.ALL)
	private User user;
	public Consumer(String firstName, String lastName) {
		this.firstName = firstName;
		this.lastName = lastName;
	}
	public long getConsumerId() { return consumerId;}
	public void setConsumerId(long consumerId) {	this.consumerId = consumerId;}
	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 User getUser() {	return user;}
	public void setUser(User user) {	this.user = user;}
	public String getEmailId() {  return emailId;}
	public void setEmailId(String emailId) {	this.emailId = emailId;}}
```

Librarian.java

```
package com.me.lib.pojo;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.OneToMany;
import javax.persistence.OneToOne;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;
@Entity
@Table(name="librarian_table")
@PrimaryKeyJoinColumn(name = "personId")
public class Librarian extends Person{
	public Librarian() {}
	public Librarian(String username) { this.username = username;}
	@Column(name="username")
	private String username;
	@Column(name="email")
```

```
	private String email;
	public String getEmail() {	return email;}
	public void setEmail(String email) {this.email = email;}
	@Column(name="password")
	private String password;
	@Column(name="status")
	private Boolean status;
	@OneToOne(mappedBy = "librarian" , cascade = CascadeType.ALL)
	private Phone phone;
	@OneToOne(mappedBy = "librarian" , cascade = CascadeType.ALL)
	private Address address;
	@OneToMany(mappedBy = "librarian", cascade = CascadeType.ALL)
	private List<Book> books = new ArrayList<Book>();
	public Librarian(String username, String password) {
		this.username=username;
		this.password=password;
	}
	public String getUsername() {return username;}
	public void setUsername(String username) {this.username = username;}
	public String getPassword() { return password;}
	public void setPassword(String password) {	this.password = password;}
	public Boolean getStatus() {  return status;}
	public void setStatus(Boolean status) {   this.status = status;}
	public Phone getPhone() {	return phone;}
	public void setPhone(Phone phone) {	this.phone = phone;}
	public Address getAddress() { return address;}
	public void setAddress(Address address) { this.address = address;}
	public List<Book> getBooks() {	return books;}
	public void setBooks(List<Book> books) {  this.books = books;}}
```

CONTROLLERs

AjaxController.java

```
package com.me.lib.controller;
import java.util.ArrayList;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import org.hibernate.HibernateException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.http.MediaType;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import com.me.lib.dao.BookDAO;
import com.me.lib.dao.ConsumerDAO;
import com.me.lib.dao.LibrarianDAO;
import com.me.lib.exception.BookException;
import com.me.lib.exception.ConsumerException;
import com.me.lib.exception.LibrarianException;
import com.me.lib.pojo.Book;
import com.me.lib.pojo.Cart;
import com.me.lib.pojo.User;

@Controller
public class AjaxController {
	@Autowired
    @Qualifier("librarianDao")
	  LibrarianDAO librarianDao;

	  @Autowired
```

```
	@Qualifier("bookDao")
	  BookDAO bookDao;

	  @Autowired
	@Qualifier("consumerDao")
	  ConsumerDAO consumerDao;

	  @RequestMapping(value = "/admin/activateLibrarian.htm", method =
RequestMethod.GET, produces = MediaType.TEXT_PLAIN_VALUE)
	  public @ResponseBody
	  String activateLibrarian(HttpServletRequest request) throws LibrarianException {
		String output = "";
		try {
			int aid = Integer.parseInt(request.getParameter("aid"));
			System.out.println("ID is: "+aid);
			librarianDao.activateLibrarian(aid);
			output = String.valueOf(aid);
		} catch(HibernateException e) {
			System.out.println("Exception: " + e.getMessage());
		}
		return output;
	  }

	  @RequestMapping(value = "/consumer/addToCart.htm", method = RequestMethod.GET,
produces = MediaType.TEXT_PLAIN_VALUE)
	  public @ResponseBody
	  String addToCart(HttpServletRequest request) throws BookException,
ConsumerException {
		String output = "";

		try {
			int qty_cart = Integer.parseInt(request.getParameter("qty_cart"));
			int quantity = qty_cart;
			int bookId = Integer.parseInt(request.getParameter("bookID"));
			Book book = bookDao.get(bookId);
			User user = (User)request.getSession().getAttribute("user");
			Long pId = book.getBookId();
			Long userId = user.getPersonID();
			boolean b1 = consumerDao.checkCart(pId, userId);
			System.out.println(b1);
			if(b1) {
				System.out.println("Inserting...");
				consumerDao.insertIntoCart(qty_cart, book, user);
			} else {
				System.out.println("Updating...");
				Cart cart = consumerDao.getQuantity(pId, userId);
				quantity = cart.getQuantity();
				quantity = quantity + qty_cart;
				consumerDao.updateCart(quantity, pId, userId);

			}
			Book pQty = bookDao.get(bookId);
			Long oldQty = pQty.getQuantity();
			Long newQty = oldQty - qty_cart;
			bookDao.updateQty(newQty, pId);
			output = String.valueOf(qty_cart);
		} catch(HibernateException e) {
			System.out.println("Exception: " + e.getMessage());
		}
		return output;
	  }

	  @RequestMapping(value = "/admin/deactivateLibrarian.htm", method =
RequestMethod.GET, produces = MediaType.TEXT_PLAIN_VALUE)
	  public @ResponseBody
```

```
	String deactivateLibrarian(HttpServletRequest request) throws LibrarianException
{
		String output = "";
		try {
			int aid = Integer.parseInt(request.getParameter("aid"));
			librarianDao.deactivateLibrarian(aid);
			output = String.valueOf(aid);
		} catch(HibernateException e) {
			System.out.println("Exception: " + e.getMessage());
		}
		return output;
	}

	@RequestMapping(value = "/consumer/searchBooks.htm", method = RequestMethod.GET)
	public @ResponseBody
	String searchBooks(HttpServletRequest request) throws BookException {
		String output = "";
		try {
			HttpSession session = (HttpSession) request.getSession();
			String searchValue = request.getParameter("searchValue");
			System.out.println("Search for: "+searchValue);
			List<Book> searchList = consumerDao.searchBooks(searchValue);
			System.out.println("Size List: "+searchList.size());
			session.setAttribute("books", searchList);
			output = String.valueOf(searchList.size());
			return output;
		} catch(HibernateException e) {
			System.out.println("Exception: " + e.getMessage());
		}	return output;
	}

	@RequestMapping(value = "/consumer/filterBooks.htm", method = RequestMethod.GET)
	public @ResponseBody
	String filterBooks(HttpServletRequest request) throws BookException {
		String output = "";
		try {
			HttpSession session = (HttpSession) request.getSession();
			String filter = request.getParameter("filter");
			System.out.println("Search for: "+filter);
			List<Book> filterList = new ArrayList<Book>();
			if(filter.equals("h2l")) filterList = bookDao.filterH2l();
			else if(filter.equals("l2h")) filterList = bookDao.filterL2h();
			System.out.println("Size List: "+filterList.size());
			session.setAttribute("books", filterList);
			output = "list retured";
			return output;
		} catch(HibernateException e) {
			System.out.println("Exception: " + e.getMessage());
		}	return output;
	}
	@RequestMapping(value = "/librarian/deleteBooks.htm", method = RequestMethod.GET,
produces = MediaType.TEXT_PLAIN_VALUE)
	public @ResponseBody
	String deleteBook(HttpServletRequest request) throws BookException {
		String output = "";
		try {
			int pid = Integer.parseInt(request.getParameter("pid"));
			System.out.println("ID is: "+pid);
			bookDao.deleteBook(pid);
			output = String.valueOf(pid);
			System.out.println("output is: "+output);
		} catch(HibernateException e) {
			System.out.println("Exception: " + e.getMessage());
		}	return output;}}
```

Vidushi Garg
Section - 6

NavigatoryController.java

```
package com.me.lib.controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import org.hibernate.HibernateException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
import com.me.lib.dao.AdminDAO;
import com.me.lib.dao.LibrarianDAO;
import com.me.lib.dao.UserDAO;
import com.me.lib.pojo.Admin;
import com.me.lib.pojo.Librarian;
import com.me.lib.pojo.User;

@Controller
public class NavigatoryController {
	@Autowired
	@Qualifier("userDao")
	UserDAO userDao;
	@Autowired
	@Qualifier("librarianDao")
	LibrarianDAO librarianDao;
	@Autowired
	@Qualifier("adminDao")
	AdminDAO adminDao;
	@RequestMapping(value = "/home.htm", method = RequestMethod.GET)
	public ModelAndView redirectIndex(HttpServletRequest request) {	return new
ModelAndView("home");}
	@RequestMapping(value = "/login.htm", method = RequestMethod.GET)
	public ModelAndView redirectLogin(HttpServletRequest request) {	return new
ModelAndView("login");}

	@RequestMapping(value = "/signup.htm", method = RequestMethod.GET)
	public ModelAndView redirectSignup(HttpServletRequest request) {return new
ModelAndView("signup");}

	@RequestMapping(value = "/logout.htm", method = RequestMethod.GET)
	public ModelAndView redirectLogout(HttpServletRequest request) {HttpSession
session = request.getSession();
		session.invalidate();
		return new ModelAndView("home");}

	@RequestMapping(value = "/redirectsignup.htm", method = RequestMethod.POST)
	public ModelAndView redirectSignupBy(HttpServletRequest request) {	String
value = request.getParameter("signup");
		if (value.equalsIgnoreCase("consumer")) {
			return new ModelAndView("consumer-signup", "user", new User());
		} else if (value.equalsIgnoreCase("librarian")) {
			return new ModelAndView("librarian-signup", "librarian", new
Librarian());
		}
		return new ModelAndView("signup","results",null);}

	@RequestMapping(value = "/redirectlogin.htm", method = RequestMethod.GET)
	public ModelAndView getLoginUser(HttpServletRequest request) {
		System.out.println("Get Login User");
		return new ModelAndView("login");
	}

	@RequestMapping(value = "/404")
```

```
	public ModelAndView error404(HttpServletRequest request) {
		System.out.println("custom error handler");
		return new ModelAndView("error404");
	}

	@RequestMapping(value = "/redirectlogin.htm", method = RequestMethod.POST)
	protected ModelAndView loginUser(HttpServletRequest request) throws Exception {

		HttpSession session = (HttpSession) request.getSession();
		String loginType = request.getParameter("login");
		System.out.println(loginType);
		if(loginType.equalsIgnoreCase("consumer")) {
			try {
				User u = userDao.get(request.getParameter("username"),
request.getParameter("password"));
				if (u == null) {
					System.out.println("UserName/Password does not exist");
					session.setAttribute("errorMessage", "UserName/Password
does not exist");
					return new ModelAndView("error");
				}
				session.setAttribute("user", u);
				session.setAttribute("role", "consumer");
				return new ModelAndView("consumer-home");
			} catch(HibernateException e) {
				System.out.println("Exception: " + e.getMessage());
				session.setAttribute("errorMessage", "error while login");
				return new ModelAndView("error");
			}
		} else if(loginType.equalsIgnoreCase("librarian")) {
			try {

				Librarian s =
librarianDao.get(request.getParameter("username"), request.getParameter("password"));

				if(s == null){
					System.out.println("UserName/Password does not exist");
					session.setAttribute("errorMessage", "UserName/Password
does not exist");
					return new ModelAndView("error");
				}

				boolean sell = s.getStatus();
				if(sell == false) {
					return new ModelAndView("librarian-nologin");
				}

				session.setAttribute("librarian", s);
				session.setAttribute("role", "librarian");
				return new ModelAndView("librarian-home");

			} catch (HibernateException e) {
				System.out.println("Exception: " + e.getMessage());
				session.setAttribute("errorMessage", "error while login");
				return new ModelAndView("error");
			}
		} else if(loginType.equalsIgnoreCase("admin")) {
			try {

				Admin a = adminDao.get(request.getParameter("username"),
request.getParameter("password"));

				if(a == null){
					System.out.println("UserName/Password does not exist");
```

```
                                session.setAttribute("errorMessage", "UserName/Password
does not exist");
                                return new ModelAndView("error");
                          }

                          session.setAttribute("startpage", 0);
                          session.setAttribute("admin", a);
                          session.setAttribute("role", "admin");
                          return new ModelAndView("admin-home");

                    } catch (HibernateException e) {
                          System.out.println("Exception: " + e.getMessage());
                          session.setAttribute("errorMessage", "error while login");
                          return new ModelAndView("error");
                    }
              }
              return null;
        }
}
```

BookController.java

```
package com.me.lib.controller;
import java.io.IOException;

import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

import com.me.lib.dao.BookDAO;
import com.me.lib.dao.LibrarianDAO;
import com.me.lib.exception.BookException;
import com.me.lib.pojo.Book;
import com.me.lib.pojo.Librarian;
import com.me.lib.validator.BookValidator;

@Controller
public class BookController {
      @Autowired
      @Qualifier("bookDao")
      BookDAO bookDao;
      @Autowired
      @Qualifier("librarianDao")
      LibrarianDAO librarianDao;
      @Autowired
      @Qualifier("bookValidator")
      BookValidator validator;
      @Autowired
      ServletContext servletContext;

      @InitBinder
      private void initBinder(WebDataBinder binder) {
            binder.setValidator(validator);
      }
```

```
	@RequestMapping(value = "/librarian/add-books.htm", method = RequestMethod.GET)
	protected ModelAndView addProd() throws Exception {
		ModelAndView mv = new ModelAndView();
		mv.addObject("book", new Book());
		mv.setViewName("add-books");
		return mv;
	}

	@RequestMapping(value = "/librarian/view-books.htm", method = RequestMethod.GET)
	protected ModelAndView viewBooks(HttpServletRequest request) throws Exception {
		ModelAndView mv = new ModelAndView();
		int librarianId = Integer.parseInt(request.getParameter("librarianId"));
		System.out.println(librarianId);
		mv.addObject("books", bookDao.list(librarianId));
		mv.setViewName("view-books");
		return mv;
	}

	@RequestMapping(value = "/librarian/edit-books.htm", method = RequestMethod.GET)
	protected ModelAndView editBooks(HttpServletRequest request) throws Exception {
		ModelAndView mv = new ModelAndView();
		int bookId = Integer.parseInt(request.getParameter("id"));
		System.out.println(bookId);
		mv.addObject("book", bookDao.get(bookId));
		mv.setViewName("edit-books");
		return mv;
	}

	@RequestMapping(value = "/librarian/addBooks.htm", method = RequestMethod.POST)
	protected ModelAndView addBooks(HttpServletRequest request,
@ModelAttribute("book") Book book, BindingResult result) throws Exception {
		try {
			System.out.println(book.getTitle());
			if(book.getTitle().trim() != "" || book.getTitle() != null) {
				int librarianId =
Integer.parseInt(request.getParameter("librarian-name"));
				for(Book b: bookDao.list(librarianId)) {
					if(b.getTitle().equalsIgnoreCase(book.getTitle()) &&
b.getAuthor().equalsIgnoreCase(book.getAuthor()))
						return new ModelAndView("book-available");}
				Librarian librarian = librarianDao.get(librarianId);
				book.setLibrarian(librarian);

				Book p = bookDao.addBook(book);
				return new ModelAndView("book-success");
			}
		}	catch (IllegalStateException e) {
			System.out.println("*** IllegalStateException: " + e.getMessage());
		}	catch (BookException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		return null;
	}

	@RequestMapping(value = "/librarian/updateBooks.htm", method =
RequestMethod.POST)
	protected ModelAndView updateBooks(HttpServletRequest request,
@ModelAttribute("book") Book book, BindingResult result) throws Exception {
		try {
			long bookId = Long.parseLong(request.getParameter("pid"));
			System.out.println(book.getAuthor());
			if(book.getTitle().trim() != "" || book.getTitle() != null) {
			for(Book b: bookDao.list()) {
```

```
                        if(b.getTitle().equalsIgnoreCase(book.getTitle()) &&
b.getAuthor().equalsIgnoreCase(book.getAuthor()))
                                return new ModelAndView("book-avail");}
                        bookDao.updateBook(book);

                        return new ModelAndView("update-success");

                        }

                  }     catch (IllegalStateException e) {
                        System.out.println("*** IllegalStateException: " + e.getMessage());
                  }     catch (BookException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                  }
                  return null;
            }

            @RequestMapping(value = "/consumer/view-all-books.htm", method =
RequestMethod.GET)
            protected ModelAndView viewAllBooks(HttpServletRequest request) throws Exception
{
                  ModelAndView mv = new ModelAndView();
                  HttpSession session = (HttpSession) request.getSession();
                  if(session.getAttribute("books") != null) {
                        mv.setViewName("view-all-books");
                        return mv;
                  }

                  session.setAttribute("books", bookDao.list());
                  mv.setViewName("view-all-books");
                  return mv;
            }

            @RequestMapping(value = "/consumer/single-book.htm", method = RequestMethod.GET)
            protected ModelAndView viewSingleBook(HttpServletRequest request) throws
Exception {
                  ModelAndView mv = new ModelAndView();
                  int bookid = Integer.parseInt(request.getParameter("id"));
                  mv.addObject("book", bookDao.get(bookid));
                  mv.setViewName("single-book");
                  return mv;
            }
}
```

LibrarianController.java

```
package com.me.lib.controller;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
import com.me.lib.dao.LibrarianDAO;
import com.me.lib.exception.LibrarianException;
import com.me.lib.pojo.Librarian;
import com.me.lib.validator.LibrarianValidator;
```

Vidushi Garg
Section - 6

```
@Controller
@RequestMapping("/librarian/*")
public class LibrarianController {
	@Autowired
    @Qualifier("librarianDao")
	LibrarianDAO librarianDao;
	@Autowired
	@Qualifier("librarianValidator")
	LibrarianValidator validator;
	@InitBinder
	private void initBinder(WebDataBinder binder) {
		binder.setValidator(validator);
	}
	@RequestMapping(value = "/librarian/librarian-home.htm", method =
RequestMethod.GET)
	protected ModelAndView librarianHome() throws Exception {
		return new ModelAndView("librarian-home");
	}
	@RequestMapping(value = "/librarian/librariansignup.htm", method =
RequestMethod.GET)
	protected ModelAndView registerLibrarian() throws Exception {
		return new ModelAndView("librarian-signup", "librarian", new Librarian());}

	@RequestMapping(value = "/librarian/librariansignup.htm", method =
RequestMethod.POST)
	protected ModelAndView registerLibrarian(HttpServletRequest request,
@ModelAttribute("librarian") Librarian librarian, BindingResult result) throws
Exception {
		try {
			Boolean b1 =
librarianDao.checkIfUsernameExists(request.getParameter("username"));
			Boolean b3 =
librarianDao.checkIfPhoneNoExists(request.getParameter("phone.phoneNo"));
			if(b1 && b3) {
				Librarian s = librarianDao.register(librarian);
				return new ModelAndView("signup-success");
			}
			return new ModelAndView("librarian-error");
		} catch (LibrarianException e) {
	System.out.println("Exception: " + e.getMessage());
			return new ModelAndView("error", "errorMessage", "error while
login");
		}}

	@RequestMapping(value = "/librarian/view-all-orders.htm", method =
RequestMethod.GET)
	protected ModelAndView viewAllBooks(HttpServletRequest request) throws Exception
{
		ModelAndView mv = new ModelAndView();
		HttpSession session = (HttpSession) request.getSession();
		long id = Long.parseLong(request.getParameter("librarianId"));
		mv.addObject("librarianID", id);
		mv.addObject("orders", librarianDao.orderlist());
		mv.setViewName("view-all-orders");
		return mv;	}}
```

Vidushi Garg
Section - 6

ConsumerController.java

```
package com.me.lib.controller;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import org.hibernate.HibernateException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.http.MediaType;
import org.springframework.stereotype.Controller;
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.me.lib.dao.BookDAO;
import com.me.lib.dao.ConsumerDAO;
import com.me.lib.exception.BookException;
import com.me.lib.exception.ConsumerException;
import com.me.lib.pojo.Book;
import com.me.lib.pojo.Cart;
import com.me.lib.pojo.User;
import com.me.lib.pojo.Order;

@Controller
public class ConsumerController {
	@Autowired
    @Qualifier("consumerDao")
	ConsumerDAO consumerDao;
	@Autowired
    @Qualifier("bookDao")
	BookDAO bookDao;
	@RequestMapping(value = "/consumer/cart.htm", method = RequestMethod.GET)
	protected ModelAndView getCart(HttpServletRequest request) throws Exception {
		ModelAndView mv = new ModelAndView();
		int uid = Integer.parseInt(request.getParameter("uid"));
		mv.addObject("books", consumerDao.list(uid));
		mv.setViewName("view-cart");
		return mv;
	}
	@RequestMapping(value = "/consumer/order.htm", method = RequestMethod.GET)
	protected ModelAndView getOrders(HttpServletRequest request) throws Exception {
		User user = (User)request.getSession().getAttribute("user");
		Long uid = user.getPersonID();
		List<Order> orders = consumerDao.orderlist(uid);
		HashMap <Integer, ArrayList<Order>> hashmap = new HashMap<Integer,
ArrayList<Order>>();
		for(Order o: orders) {
			ArrayList<Order> orderList = hashmap.get(o.getOrderID());
			if(orderList == null) {
				orderList = new ArrayList<Order>();
				orderList.add(o);
				hashmap.put(o.getOrderID(), orderList);
			} else {
				orderList.add(o);
			}}
		return new ModelAndView("view-orders", "hashmap", hashmap);}
	@RequestMapping(value = "/consumer/updateValueInCart.htm", method =
RequestMethod.GET, produces = MediaType.TEXT_PLAIN_VALUE)
	public @ResponseBody
```

```
	String deleteCart(HttpServletRequest request) throws BookException,
ConsumerException {
		String output = "";
		try {
			int bookID = Integer.parseInt(request.getParameter("bookID"));
			Long pId = Long.parseLong(request.getParameter("bookID"));
			User user = (User)request.getSession().getAttribute("user");
			Long userId = user.getPersonID();
			consumerDao.deleteBook(pId, userId);
			Long prd_qty = Long.parseLong(request.getParameter("bk_quantity"));
			System.out.println(prd_qty);
			Book b = bookDao.get(bookID);
			Long oldQty = b.getQuantity();
			Long newQty = oldQty + prd_qty;
			bookDao.updateQty(newQty, pId);
			output = String.valueOf(bookID);
		} catch(HibernateException e) {
			System.out.println("Exception: " + e.getMessage());}
		return output;
	}
	@RequestMapping(value = "/consumer/issue-now.htm", method = RequestMethod.GET)
	protected ModelAndView insertOrder(HttpServletRequest request) throws Exception {
		User user = (User)request.getSession().getAttribute("user");
		int uid = (int) user.getPersonID();
		Long userid = user.getPersonID();
		List<Cart> list = consumerDao.cartlist(uid);
		int max_value = consumerDao.getmax();
		if (max_value == 0)	max_value = 1;
	  else max_value = max_value + 1;
		for(Cart c: list) {
			Order o = new Order();
			DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd");
			Date date = new Date();
			o.setDate(date);
			o.setBooks(c.getBook());
			Cart cq = consumerDao.getQuantity(c.getBook().getBookId(), userid);
			o.setQuantity(cq.getQuantity());
			o.setUser(user);
			o.setOrderID(max_value);
			consumerDao.insertOrder(o);
			consumerDao.deleteBook(c.getBook().getBookId(), userid);
		}
		return new ModelAndView("issue-success");}}
```

DAOs

AdminDAO.java

```
package com.me.lib.dao;
import org.hibernate.HibernateException;
import org.hibernate.Query;
import com.me.lib.exception.AdminException;
import com.me.lib.pojo.Admin;

public class AdminDAO extends DAO {
	public AdminDAO() {}

	public Admin get(String username, String password) throws AdminException {
		try {
			begin();
			Query q = getSession().createQuery("from Admin where username =
:username and password = :password");
			q.setString("username", username);
			q.setString("password", password);
```

```
                Admin admin = (Admin) q.uniqueResult();
                commit();
                close();
                return admin;
            } catch (HibernateException e) {
                rollback();
                throw new AdminException("Could not get admin " + username, e);
            }}}
```

BookDAO.java

```
package com.me.lib.dao;

import java.util.List;
import org.hibernate.Criteria;
import org.hibernate.HibernateException;
import org.hibernate.Query;
import org.hibernate.criterion.Order;
import com.me.lib.exception.BookException;
import com.me.lib.exception.ConsumerException;
import com.me.lib.pojo.Book;
public class BookDAO extends DAO {
      public BookDAO() {      }
      public List<Book> list(int librarianId) throws BookException {
            try {
                begin();
                Query q = getSession().createQuery("from Book where librarian =
:librarianId");
                q.setInteger("librarianId", librarianId);
                List<Book> list = q.list();
                commit();
                return list;
            } catch (HibernateException e) {
                rollback();
                throw new BookException("Could not list the books", e);
            }}
      public List<Book> list() throws BookException {
            try {
                begin();
                Query q = getSession().createQuery("from Book");
                List<Book> list = (List<Book>)q.list();
                commit();
                return list;
            } catch (HibernateException e) {
                rollback();
                throw new BookException("Could not list the books", e);
            }
      }

      public List<Book> filterH2l() throws BookException {
        try {
            System.out.println("In search books dao");
            begin();
            Criteria crit = getSession().createCriteria(Book.class);
            crit.addOrder(Order.desc("charge"));
            List<Book> list = (List<Book>)crit.list();
            System.out.println("List size: "+list.size());
            System.out.println(list);
            commit();
            return list;
        } catch (HibernateException e) {
            rollback();
            throw new BookException("Could not list the books", e);
        }
    }
```

```
	public List<Book> filterL2h() throws BookException {
	  try {
	      System.out.println("In search books dao");
	      begin();
	      Criteria crit = getSession().createCriteria(Book.class);
	      crit.addOrder(Order.asc("charge"));
	      List<Book> list = (List<Book>)crit.list();
	      System.out.println("List size: "+list.size());
	      System.out.println(list);
	      commit();
	      return list;
	  } catch (HibernateException e) {
	      rollback();
	      throw new BookException("Could not list the books", e);
	  }
    }

	public Book get(int bookid) throws BookException {
		try {
			begin();
			Query q = getSession().createQuery("from Book where bookId=
:bookId");
			q.setInteger("bookId", bookid);
			Book book = (Book) q.uniqueResult();
			commit();
			return book;
		} catch (HibernateException e) {
			rollback();
			throw new BookException("Could not get book " + bookid, e);
		}
	}

	public Book addBook(Book p)
			throws BookException {
		try {
			begin();
			System.out.println("inside Book DAO");

			getSession().save(p);
			commit();
			return p;

		} catch (HibernateException e) {
			rollback();
			throw new BookException("Exception while creating book: " +
e.getMessage());
		}
	}
	public void updateBook(Book p)
			throws BookException {
		try {
			begin();
			System.out.println("inside update Book DAO");

			Query q = getSession().createQuery("Update Book set author=:author ,
title = :title, isbn=:isbn, charge = :charge, quantity = :quantity where bookId =
:bookid");
			q.setLong("bookid", p.getBookId());
			q.setString("title", p.getTitle());
			q.setDouble("charge", p.getCharge());
			q.setLong("quantity", p.getQuantity());
			q.setString("isbn", p.getIsbn());
			q.setString("author", p.getAuthor());
			System.out.println(p.getBookId());
			System.out.println(p.getTitle());
```

```
                System.out.println(p.getCharge());
                System.out.println(p.getQuantity());
                System.out.println(p.getIsbn());
                System.out.println(p.getAuthor());
                int r = q.executeUpdate();
                System.out.print(r);
                commit();
                close();
            } catch (HibernateException e) {
                rollback();
                throw new BookException("Exception while creating book: " +
e.getMessage());
            }
      }

      public void updateQty(Long qty_cart, Long bookid) throws ConsumerException {
            try {
                System.out.println("updateCart");
                begin();
                Query query = getSession().createQuery("Update Book set quantity
=:qty_cart where bookid =:bookid");
                query.setLong("bookid", bookid);
                query.setLong("qty_cart", qty_cart);
                int res = query.executeUpdate();
                commit();
                close();
            } catch (HibernateException e) {
                rollback();
                throw new ConsumerException("Error while updating cart: " +
e.getMessage());
            }
      }

      public void deleteBook(int bookid) throws BookException {
            try {
                begin();
                Query q = getSession().createQuery("delete Book where bookId =
:bookid");
                q.setInteger("bookid", bookid);
                int res = q.executeUpdate();
                commit();
                close();
            } catch(HibernateException e) {
                rollback();
                throw new BookException("Could not delete the book", e);
            }}}
```

ConsumerDAO.java

```
package com.me.lib.dao;

import java.util.List;
import org.hibernate.Criteria;
import org.hibernate.HibernateException;
import org.hibernate.Query;
import org.hibernate.criterion.Projections;
import com.me.lib.exception.BookException;
import com.me.lib.exception.ConsumerException;
import com.me.lib.pojo.*;

public class ConsumerDAO extends DAO {
      public ConsumerDAO() {}
      public List<Book> list(int userId) throws BookException {
        try {
            begin();
            Query q = getSession().createQuery("from Cart where userID = :userId");
```

```
            q.setInteger("userId", userId);
            List<Book> list = q.list();
            commit();
            return list;
        } catch (HibernateException e) {
            rollback();
            throw new BookException("Couldn't list the books", e);
        }  }

    public List<Book> searchBooks(String keyword) throws BookException {
        try {
            System.out.println("In search books dao");
            begin();
            Query query = getSession().createQuery("from Book where title like
:keyword");
            query.setParameter("keyword", "%"+keyword+"%");
            List<Book> list = (List<Book>)query.list();
            System.out.println("List size: "+list.size());
            System.out.println(list);
            commit();
            return list;
        } catch (HibernateException e) {
            rollback();
            throw new BookException("Could not list the books", e);
        }   }

    public List<Order> orderlist(Long userId) throws BookException {
        try {
            begin();
            Query q = getSession().createQuery("from Order where userID = :userId");
            q.setLong("userId", userId);
            List<Order> list = q.list();
            commit();
            return list;
        } catch (HibernateException e) {
            rollback();
            throw new BookException("Could not list the orders", e);
        }   }

    public List<Order> orderlistpdf(Long orderId) throws BookException {
        try {
            begin();
            Query q = getSession().createQuery("from Order where orderID = :orderId");
            q.setLong("orderId", orderId);
            List<Order> list = q.list();
            commit();
            return list;
        } catch (HibernateException e) {
            rollback();
            throw new BookException("Could not list the orders", e);
        }  }

    public List<Cart> cartlist(int userId) throws BookException {
        try {
            begin();
            Query q = getSession().createQuery("from Cart where userID = :userId");
            q.setInteger("userId", userId);
            List<Cart> list = q.list();
            commit();
            return list;
        } catch (HibernateException e) {
            rollback();
            throw new BookException("Could not list the books", e);
        }  }
```

```
	public int getmax() {
	  begin();
	  Criteria crit = getSession().createCriteria(Order.class);
	  crit.setProjection(Projections.max("orderID"));
	  int maxid=0;
	  if(crit.uniqueResult()==null){
	      maxid=0;
	  }else{
	      maxid = (Integer)crit.uniqueResult();
	  }
	  commit();
	  close();
	  return maxid;
    }

	public Boolean checkCart(Long bookID, Long userId) throws ConsumerException {
		try {
			System.out.println("checkCart");
			begin();
			Query query = getSession().createQuery("from Cart where userID
=:userId AND bookID =:bookID");
			query.setLong("userId", userId);
			query.setLong("bookID", bookID);
			List res = query.list();
			System.out.println(res);
			if (res.size() == 0) {
				return true;
			} else {
				return false;
			}
		} catch (HibernateException e) {
			System.out.println("Exception it is"+e);
			rollback();
			throw new ConsumerException("Error while checking cart: " +
e.getMessage());
		}
	}

	public Cart getQuantity(Long bookID, Long userId) throws ConsumerException {
		try {
			System.out.println("getQuantity");
			begin();
			Query query = getSession().createQuery("from Cart where userID
=:userId AND bookID =:bookID");
			query.setLong("userId", userId);
			query.setLong("bookID", bookID);
			Cart cart = (Cart)query.uniqueResult();
			System.out.println("Cart: "+cart);
			close();
			return cart;
		} catch (HibernateException e) {
			System.out.println("Quantity Exception it is: "+e);
			rollback();
			throw new ConsumerException("Error while getting quantity: " +
e.getMessage());
		}
	}

	public void updateCart(int qty_cart, Long bookID, Long userId) throws
ConsumerException {
		try {
			System.out.println("updateCart");
			begin();
			Query query = getSession().createQuery("Update Cart set quantity
=:qty_cart where userID =:userId AND bookID =:bookID");
```

```
                query.setLong("userId", userId);
                query.setLong("bookID", bookID);
                query.setInteger("qty_cart", qty_cart);
                int res = query.executeUpdate();
                commit();
                close();
            } catch (HibernateException e) {
                rollback();
                throw new ConsumerException("Error while updating cart: " +
e.getMessage());
            } }

      public void insertIntoCart(int qty_cart, Book book, User userId) throws
ConsumerException {
            try {
                begin();
                Cart cart = new Cart();
                cart.setQuantity(qty_cart);
                cart.setBook(book);
                cart.setUser(userId);
                getSession().save(cart);
                commit();
            } catch (HibernateException e) {
                rollback();
                throw new ConsumerException("Error while adding cart: " +
e.getMessage());
            }}

      public void deleteBook(Long bookid, Long userId) throws ConsumerException {
            try {
                begin();
                Query q = getSession().createQuery("delete Cart where bookID =
:bookid AND userID = :userId");
                q.setLong("bookid", bookid);
                q.setLong("userId", userId);
                int res = q.executeUpdate();
                commit();
                close();
            } catch(HibernateException e) {
                rollback();
            throw new ConsumerException("Could not delete the book", e);
            }}

      public void insertOrder(Order order) throws ConsumerException {
            try {
            begin();
            getSession().save(order);
            commit();
            close();
        } catch (HibernateException e) {
            rollback();
            throw new ConsumerException("Insert order " + e);
        }}}
```

DAO.java

```
package com.me.lib.dao;

import java.util.logging.Level;
import java.util.logging.Logger;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class DAO {
      private static final Logger log = Logger.getAnonymousLogger();
```

```
    private static final ThreadLocal sessionThread = new ThreadLocal();
    private static final SessionFactory sessionFactory = new
Configuration().configure("hibernate.cfg.xml").buildSessionFactory();
    protected DAO() {   }

    public static Session getSession()
    {
        Session session = (Session) DAO.sessionThread.get();
        if (session == null)
        {
            session = sessionFactory.openSession();
            DAO.sessionThread.set(session);
        }
        return session;
    }

    protected void begin() {
        getSession().beginTransaction();
    }

    protected void commit() {
        getSession().getTransaction().commit();
    }

    protected void rollback() {
        try {
            getSession().getTransaction().rollback();
        } catch (HibernateException e) {
            log.log(Level.WARNING, "Cannot rollback", e);
        }
        try {
            getSession().close();
        } catch (HibernateException e) {
            log.log(Level.WARNING, "Cannot close", e);
        }
        DAO.sessionThread.set(null);
    }

    public static void close() {
        getSession().close();
        DAO.sessionThread.set(null);
    }
}
```

LibrarianDAO.java

```
package com.me.lib.dao;

import java.util.List;
import org.hibernate.Criteria;
import org.hibernate.HibernateException;
import org.hibernate.Query;
import com.me.lib.exception.LibrarianException;
import com.me.lib.pojo.Address;
import com.me.lib.pojo.Librarian;
import com.me.lib.pojo.Order;
import com.me.lib.pojo.Phone;

public class LibrarianDAO extends DAO {
      public LibrarianDAO() { }
      public List<Order> orderlist() throws LibrarianException {
        try {
            begin();
            Query q = getSession().createQuery("from Order");
            List<Order> list = q.list();
            commit();
            return list;
```

```
        } catch (HibernateException e) {
            rollback();
            throw new LibrarianException("Could not list the orders", e);
        }    }

      public int getTotalCount() throws LibrarianException {
            try {
                  begin();
                  int count = ((Long)getSession().createQuery("select count(*) from
Librarian").uniqueResult()).intValue();
                  commit();
                  return count;
            } catch (HibernateException e) {
                  rollback();
                  throw new LibrarianException("Could not get librarian " + e);
            }     }

      public List<Librarian> paginateList(int firstResult, int maxResults) {
            try {
                  begin();
                  Criteria criteria = getSession().createCriteria(Librarian.class);
                  criteria.setFirstResult(firstResult);
                  criteria.setMaxResults(maxResults);
                  List<Librarian> librarians = (List<Librarian>)criteria.list();
                  commit();
                  close();
                  return librarians;

            } catch (HibernateException e) {
                  e.printStackTrace();
            }
            return null;
      }

      public Librarian get(String username, String password) throws LibrarianException
{
            try {
                  begin();
                  Query q = getSession().createQuery("from Librarian where username =
:username and password = :password");
                  q.setString("username", username);
                  q.setString("password", password);
                  Librarian librarian = (Librarian) q.uniqueResult();
                  commit();
                  close();
                  return librarian;
            } catch (HibernateException e) {
                  rollback();
                  throw new LibrarianException("Could not get librarian " + username,
e);
            }
      }

      public List<Librarian> list() throws LibrarianException {
        try {
            begin();
            Query q = getSession().createQuery("from Librarian");
            List<Librarian> list = q.list();
            commit();
            return list;
        } catch (HibernateException e) {
            rollback();
            throw new LibrarianException("Could not list the librarians", e);
        }
    }
```

```
      public List<Librarian> activelist() throws LibrarianException {
        try {
            begin();
            Query q = getSession().createQuery("from Librarian where status =
:status");
            q.setBoolean("status", true);
            List<Librarian> list = q.list();
            commit();
            return list;
        } catch (HibernateException e) {
            rollback();
            throw new LibrarianException("Could not list the librarians", e);
        }
    }

      public List<Librarian> deactivelist() throws LibrarianException {
        try {
            begin();
            Query q = getSession().createQuery("from Librarian where status =
:status");
            q.setBoolean("status", false);
            List<Librarian> list = q.list();
            commit();
            return list;
        } catch (HibernateException e) {
            rollback();
            throw new LibrarianException("Could not list the librarians", e);
        }
    }

      public void activateLibrarian(int librarianId) throws LibrarianException {
            try {
                  begin();
                  Query q = getSession().createQuery("Update Librarian set status =
:status where personID = :personID");
                  q.setBoolean("status", true);
                  q.setInteger("personID", librarianId);
                  int res = q.executeUpdate();
                  commit();
                  close();
            } catch(HibernateException e) {
                  rollback();
            throw new LibrarianException("Could not activate the librarian", e);
            }
      }

      public void deactivateLibrarian(int librarianId) throws LibrarianException {
            try {
                  begin();
                  Query q = getSession().createQuery("Update Librarian set status =
:status where personID = :personID");
                  q.setBoolean("status", false);
                  q.setInteger("personID", librarianId);
                  int res = q.executeUpdate();
                  commit();
                  close();
            } catch(HibernateException e) {
                  rollback();
            throw new LibrarianException("Could not activate the librarian", e);
            }
      }

      public Librarian get(int librarianId) throws LibrarianException {
            try {
```

```
                begin();
                Query q = getSession().createQuery("from Librarian where personID=
:personID");
                q.setInteger("personID", librarianId);
                Librarian librarian = (Librarian) q.uniqueResult();
                commit();
                return librarian;
            } catch (HibernateException e) {
                rollback();
                throw new LibrarianException("Could not get librarian " +
librarianId, e);
            }
      }
      public boolean checkIfUsernameExists(String username) throws LibrarianException {
            try {
                begin();
                Query q = getSession().createQuery("from Librarian where username =
:username");
                q.setString("username", username);
                System.out.print(q);
                Librarian librarian = (Librarian) q.uniqueResult();
                System.out.print(librarian);
                if(librarian == null) {
                      return true;
                } else {
                      return false;
                }
            } catch(HibernateException e) {
                System.out.println("Username Already Exists!");
            }
            return false;
      }
      public boolean checkIfPhoneNoExists(String phone) throws LibrarianException {
            try {
                begin();
                Query q = getSession().createQuery("from Phone where phoneNo =
:phoneNo");
                q.setString("phoneNo", phone);
                System.out.print(q);
                Phone phoneNo = (Phone) q.uniqueResult();
                System.out.print(phoneNo);
                if(phoneNo == null) {
                      return true;
                } else {
                      return false;
                }
            } catch(HibernateException e) {
                System.out.println("Phone Number Already Exists!");
            }
            return false;
      }
      public Librarian register(Librarian s)
                throws LibrarianException {
            try {
                begin();
                System.out.println("inside Librarian DAO");
                Phone phone = new Phone(s.getPhone().getPhoneNo());
                Address address = new Address(s.getAddress().getStreet(),
s.getAddress().getCity(), s.getAddress().getState(), s.getAddress().getZip(),
s.getAddress().getCountry());
                Librarian librarian = new Librarian(s.getUsername(),
s.getPassword());
                librarian.setFirstName(s.getFirstName());
                librarian.setLastName(s.getLastName());
                librarian.setStatus(false);
```

```
                librarian.setPhone(phone);
                librarian.setEmail(s.getEmail());
                String addressType = "Office";
                address.setAddressType(addressType);
                librarian.setAddress(address);
                phone.setLibrarian(librarian);
                address.setLibrarian(librarian);
                getSession().save(librarian);
                commit();
                return librarian;
          } catch (HibernateException e) {
                rollback();
                throw new LibrarianException("Exception while creating librarian: " +
e.getMessage());
          }     }
      public void delete(Librarian librarian) throws LibrarianException {
          try {
                begin();
                getSession().delete(librarian);
                commit();
          } catch (HibernateException e) {
                rollback();
                throw new LibrarianException("Could not delete librarian " +
librarian.getUsername(), e);}}}
```

UserDAO.java

```
package com.me.lib.dao;

import java.util.Iterator;
import java.util.List;
import org.hibernate.HibernateException;
import org.hibernate.Query;
import com.me.lib.exception.UserException;
import com.me.lib.pojo.Address;
import com.me.lib.pojo.Phone;
import com.me.lib.pojo.User;

public class UserDAO extends DAO {
      public UserDAO() {      }

      public User get(String username, String password) throws UserException {
          try {
                begin();
                Query q = getSession().createQuery("from User where username =
:username and password = :password");
                q.setString("username", username);
                q.setString("password", password);
                User user = (User) q.uniqueResult();
                commit();
                close();
                return user;
          } catch (HibernateException e) {
                rollback();
                throw new UserException("Could not get user " + username, e);
          }
      }

      public User get(int userId) throws UserException {
          try {
                begin();
                Query q = getSession().createQuery("from User where personID=
:personID");
                q.setInteger("personID", userId);
                User user = (User) q.uniqueResult();
```

```
			commit();
			return user;
		} catch (HibernateException e) {
			rollback();
			throw new UserException("Could not get user " + userId, e);
		}
	}

	public boolean checkIfUsernameExists(String username) throws UserException {
		try {
			begin();
			Query q = getSession().createQuery("from User where username =
:username");
			q.setString("username", username);
			System.out.print(q);
			User user = (User) q.uniqueResult();
			System.out.print(user);
			if(user == null) {
				return true;
			} else {
				return false;
			}
		} catch(HibernateException e) {
			System.out.println("Username Already Exists!");
		}
		return false;
	}

	public boolean checkIfPhoneNoExists(String phone) throws UserException {
		try {
			begin();
			Query q = getSession().createQuery("from Phone where phoneNo =
:phoneNo");
			q.setString("phoneNo", phone);
			System.out.print(q);
			Phone phoneNo = (Phone) q.uniqueResult();
			System.out.print(phoneNo);
			if(phoneNo == null) {
				return true;
			} else {
				return false;
			}
		} catch(HibernateException e) {
			System.out.println("Phone Number Already Exists!");
		}
		return false;
	}

	public User register(User u)
			throws UserException {
		try {
			begin();
			System.out.println("inside DAO");
			Phone phone = new Phone(u.getPhone().getPhoneNo());
			User user = new User(u.getUsername(), u.getPassword());
			List<Address> address = u.getAddress();
			String at1 = "Billing";
			String at2 = "Shipping";
			int i = 0;
			Iterator<Address> itr = address.iterator();
			while(itr.hasNext()) {
				Address add = new Address();
				add = itr.next();
				if (i == 0) {
					add.setAddressType(at1);
```

```
                        } else {
                              add.setAddressType(at2);
                        }
                        add.setUser(user);
                        i++;
                  }
                  user.setFirstName(u.getFirstName());
                  user.setLastName(u.getLastName());
                  user.setEmail(u.getEmail());
                  user.setPhone(phone);
                  phone.setUser(user);
                  user.setAddress(address);
                  getSession().save(user);
                  commit();
                  return user;

            } catch (HibernateException e) {
                  rollback();
                  throw new UserException("Exception while creating user: " +
e.getMessage());
            }
      }

      public void delete(User user) throws UserException {
            try {
                  begin();
                  getSession().delete(user);
                  commit();
            } catch (HibernateException e) {
                  rollback();
                  throw new UserException("Could not delete user " +
user.getUsername(), e);
            }     }}
```

EXCEPTIONs

AdminException.java

```
package com.me.lib.exception;
public class AdminException extends Exception {
      public AdminException(String message)
      {super("AdminException-"+message);}
      public AdminException(String message, Throwable cause)
      {super("AdminException-"+message,cause);}
}
```

BookException.java

```
package com.me.lib.exception;
public class BookException extends Exception {
      public BookException(String message)
      {super("BookException-"+message);}
      public BookException(String message, Throwable cause)
      {           super("BookException-"+message,cause);}
}
```

ConsumerException.java

```
package com.me.lib.exception;
public class ConsumerException extends Exception {
      public ConsumerException(String message)
      {           super("ConsumerException-"+message);}
      public ConsumerException(String message, Throwable cause)
      {           super("ConsumerException-"+message,cause);}
}
```

Vidushi Garg
Section - 6

LibrarianException.java

```
package com.me.lib.exception;
public class LibrarianException extends Exception {
      public LibrarianException(String message)
      {     super("LibrarianException-"+message);}
      public LibrarianException(String message, Throwable cause)
      {super("LibrarianException-"+message,cause);}
}
```

UserException.java

```
package com.me.lib.exception;
public class UserException extends Exception {
      public UserException(String message)
      {super("UserException-"+message);}
      public UserException(String message, Throwable cause)
      {     super("UserException-"+message,cause);}
}
```

INTERCEPTOR

```
package com.me.lib.interceptor;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;

public class Interceptor extends HandlerInterceptorAdapter {
      String errorPage;
    public String getErrorPage() {      return errorPage;}
    public void setErrorPage(String errorPage) {      this.errorPage = errorPage;}
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
Object handler)
            throws Exception {
        HttpSession session = (HttpSession) request.getSession();
        if(request.getParameter("consumersignup")!=null){
            if(request.getParameter("consumersignup").equals("Register Consumer")){
                session.setAttribute("role", "consumer");
            }
        }
        if(request.getParameter("librariansignup")!=null){
            if(request.getParameter("librariansignup").equals("Register Librarian")){
                session.setAttribute("role", "librarian");
            }
        }

        if(session.getAttribute("role") == null){
            if((request.getRequestURI().contains("librarian/"))||
     (request.getRequestURI().contains("consumer/"))||(request.getRequestURI().contains
("admin/")))
            {
                response.sendRedirect(request.getContextPath());
                return false;
            }
            return true;
        }

        if(session.getAttribute("role") != null) {
if((request.getRequestURI().contains("admin") &&
session.getAttribute("role").equals("admin")) ||
(request.getRequestURI().contains("librarian") &&
session.getAttribute("role").equals("librarian"))||
                        (request.getRequestURI().contains("consumer") &&
session.getAttribute("role").equals("consumer")))
```

```
            return true;
            else if((!request.getRequestURI().contains("admin"))&&
                    (!request.getRequestURI().contains("librarian"))&&
                    (!request.getRequestURI().contains("consumer")))
                return true;
        }
        response.sendRedirect(request.getContextPath());
        return  false;}}
```

VALIDATORs

BookValidator.java

```
package com.me.lib.validator;

import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;
import com.me.lib.pojo.Book;
public class BookValidator implements Validator {
      @Override
      public boolean supports(Class a) {return a.equals(Book.class);}

      @Override
      public void validate(Object obj, Errors errors) {
            // TODO Auto-generated method stub
            ValidationUtils.rejectIfEmptyOrWhitespace(errors, "author",
"error.invalid.author", "Author Name Required");
            ValidationUtils.rejectIfEmptyOrWhitespace(errors, "title",
"error.invalid.title", "Book Title Required");
            ValidationUtils.rejectIfEmptyOrWhitespace(errors, "charge",
"error.invalid.charge", "Book Price Required");
            ValidationUtils.rejectIfEmptyOrWhitespace(errors, "quantity",
"error.invalid.quantity", "Book Quantity Required");
            ValidationUtils.rejectIfEmptyOrWhitespace(errors, "isbn",
"error.invalid.isbn", "ISBN Required");
      }}
```

LibrarianValidator.java

```
package com.me.lib.validator;

import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;
import com.me.lib.pojo.Book;
public class BookValidator implements Validator {
      @Override
      public boolean supports(Class a) {return a.equals(Book.class);}

      @Override
      public void validate(Object obj, Errors errors) {
            // TODO Auto-generated method stub
            ValidationUtils.rejectIfEmptyOrWhitespace(errors, "author",
"error.invalid.author", "Author Name Required");
            ValidationUtils.rejectIfEmptyOrWhitespace(errors, "title",
"error.invalid.title", "Book Title Required");
            ValidationUtils.rejectIfEmptyOrWhitespace(errors, "charge",
"error.invalid.charge", "Book Price Required");
            ValidationUtils.rejectIfEmptyOrWhitespace(errors, "quantity",
"error.invalid.quantity", "Book Quantity Required");
            ValidationUtils.rejectIfEmptyOrWhitespace(errors, "isbn",
"error.invalid.isbn", "ISBN Required");
      }}
```

Vidushi Garg
Section - 6

UserValidator.java

```
package com.me.lib.validator;
import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;
import com.me.lib.pojo.Book;
public class BookValidator implements Validator {
	@Override
	public boolean supports(Class a) {return a.equals(Book.class);}

	@Override
	public void validate(Object obj, Errors errors) {
		// TODO Auto-generated method stub
		ValidationUtils.rejectIfEmptyOrWhitespace(errors, "author",
"error.invalid.author", "Author Name Required");
		ValidationUtils.rejectIfEmptyOrWhitespace(errors, "title",
"error.invalid.title", "Book Title Required");
		ValidationUtils.rejectIfEmptyOrWhitespace(errors, "charge",
"error.invalid.charge", "Book Price Required");
		ValidationUtils.rejectIfEmptyOrWhitespace(errors, "quantity",
"error.invalid.quantity", "Book Quantity Required");
		ValidationUtils.rejectIfEmptyOrWhitespace(errors, "isbn",
"error.invalid.isbn", "ISBN Required");
	}}
```

Project Report

GRADEMARK REPORT

FINAL GRADE

/100

GENERAL COMMENTS

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39