1 ***INTRODUCTION TO THE PROJECT***

RAGA – ONLINE MUSIC STORE will be a platform which can handle small scale purchasing orders for music records, instruments or accessories.

The project covers the following implementations.

1.1 For User

An online product catalog that can be browsed: The work starts with adding many new product catalog features which includes displaying categories, products, and product details.

Handling Customer Accounts: Customers can log in via a login page or dialog box to get access to secured areas of the web site. Once logged in, the Web Application remembers the customer until the customer logs out (either manually via a Log Out button or automatically, if the session times out or a server error occurs).
All secure pages in a Web Application need to check whether a customer is logged in before allowing access.

Searching the Catalog: For the visual part, a text box is used in which the visitor can enter one or more words to search through the product catalog. In Music CD Shop, the words entered by the visitor are searched for in the products' names and descriptions. Also, the user can search for a particular song by entering the title, artist, style, format and the price.

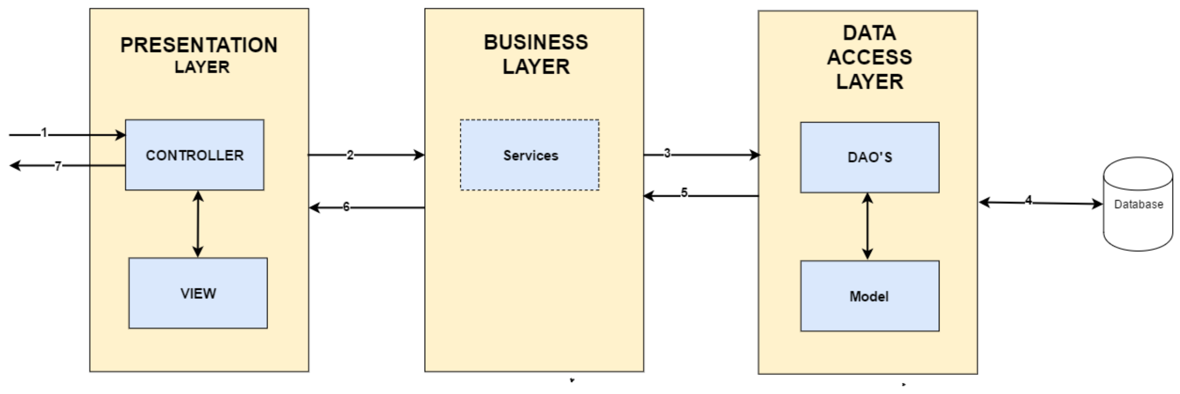
A Custom Shopping Cart and checkout: A custom shopping basket is implemented, which stores its data into the local database.

1.2 For Administrator

Catalog Administration: This administrative interface is implemented for easy management of the web store data. The catalog administration page allows the administrator to:

- Add, remove or edit records, music accessories etc.
- View and manage the products that belong to a category
- Manage the list of products in a specific category, and edit product details
- Manage orders by updating their status.
- Manage the shopping carts by removing those which haven't been updated by the customer in certain amount of time.
- The administration page also needs to ask for a username and password so that only the website administrator can perform administrative tasks.

2. ***TECHNOLOGIES USED***



Presentation Layer

Views

- HTML (Hypertext Markup Language)
- CSS (Cascading Style Sheet)
- JavaScript
- Angluar JS
- Bootstrap
- JSP
- Spring Webflow framework
- Spring Security framework

Controllers

- Core Java Technologies
  - Exception Handling
  - Collections Framework
- Spring MVC framework

Business Layer

Services

- Core Java Technologies

    - Exception Handling
    - Collections Framework

Data Access Layer

Data Access Objects

- Core Java Technologies
    - Exception Handling
    - Collections Framework
    - Maven
    - JDBC (Java Database Connectivity)
- Hibernate framework
- Spring MVC framework

Models

- Core Java Technologies
    - Exception Handling
    - Collections Framework
- Hibernate framework
- Spring MVC framework

Database Layer

- MySQL

Server Used

- Tomcat Server

3 *MODULAR DESCRIPTION OF THE PROJECT*

The project can be divided into two main modules: User and Admin.

The User and Admin modules can be further divided into submodules as following.

3.1 USER

REGISTRATION

- Allows user to register with a username and password.
- Also required is the shipping address, billing address, valid email id, valid phone number,

LOGIN and LOGOUT

- Allows user to login with the registered username and password.
- The user can also logout.
- Both these use Spring Security

PRODUCT CATALOG

- Allows user to search a variety of products.
- User can order the products using the order now button.

CHECKOUT

- Uses Spring-Webflow for checkout purposes only after the order has been accepted.

ORDER MANAGEMENT

- Allows user to check if the order has been accepted, shipped or delivered.

3.2 ADMIN

CUSTOMER MANAGEMENT

- See all the users that have registered.

PRODUCT INVENTORY

- The admin can add, edit or remove a product from the product inventory.

ORDER MANAGEMENT

- The admin can accept an order.
- After the user checks out, the admin can further ship and deliver the order.

4 *DETAILED ANALYSIS OF MODULES*

4.1 USER MODULES

4.1.1 Module 1: Registration

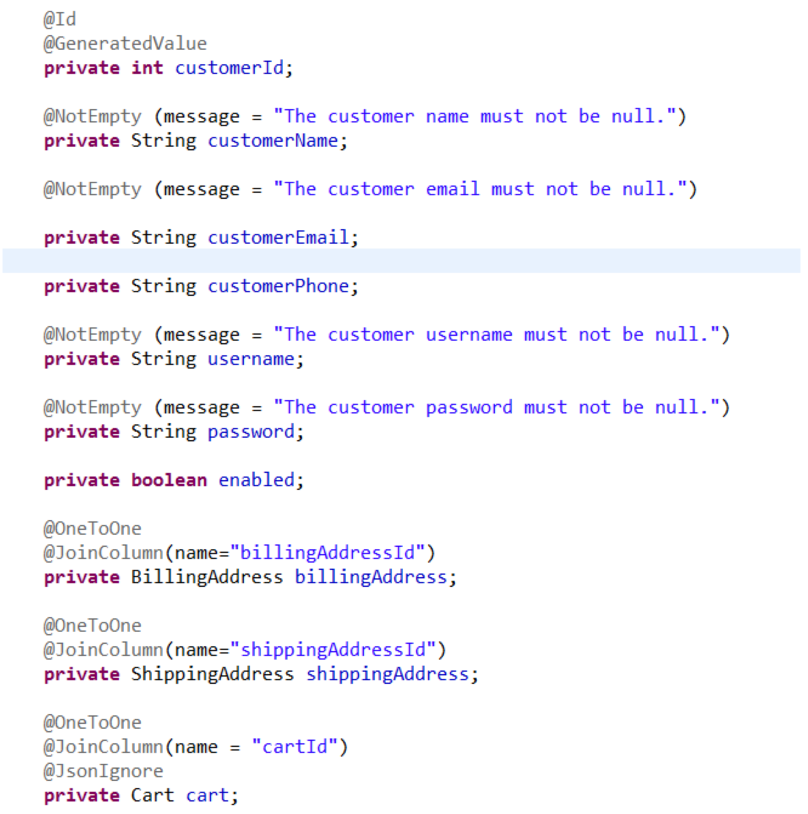
To handle the Registration Module, a controller is the first thing that is required. The controller here is named *RegistrationController* which is marked with @Controller annotation.

When a user submits the registration form, the *DispatcherServlet* checks which controller handles the registration process.

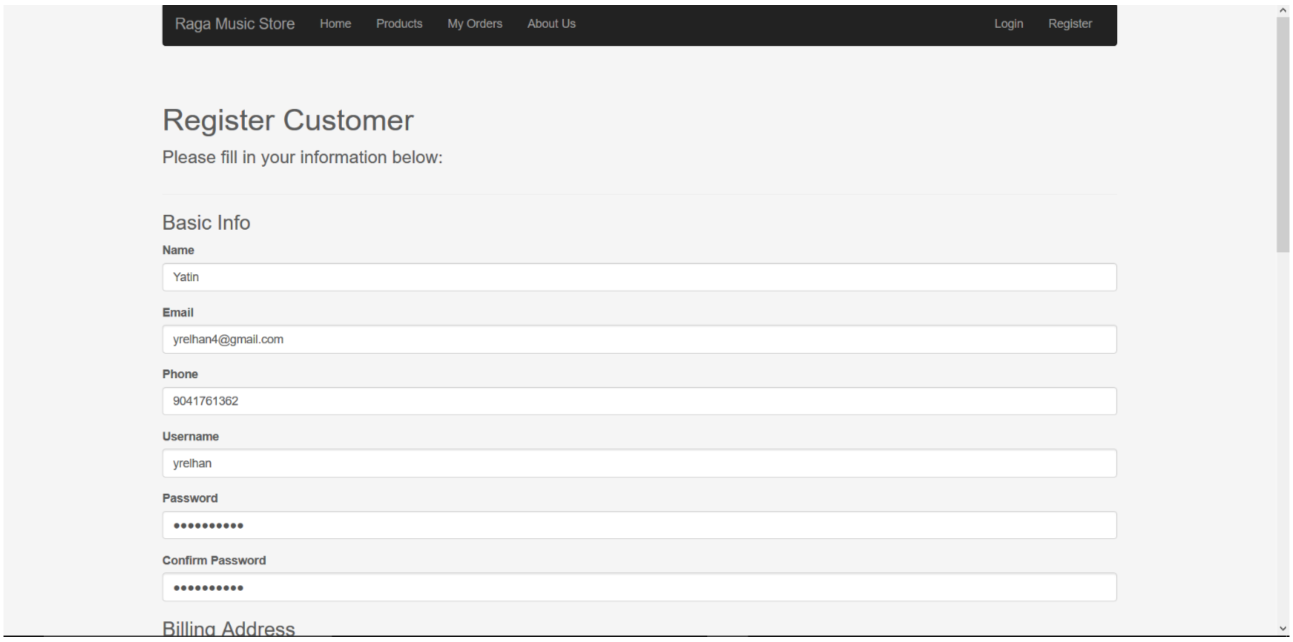
The specific called methods are executed, and if there are no errors, the customer is registered and redirected to the home page. To add a customer a *CustomerService* is called which further calls *CustomerDao* for adding the customer details to the database. To redirect to the home page the *DispatcherServlet* checks which string the method is returning and that view is returned. The *DispatcherServlet* uses *ViewResolver* for this process.

An autologin service is also provided to directly login the customer after registration.

The error checks include that each field must be filled. This is handled by Hibernate's *@NotEmpty* annotation. Also, the username and email id must be unique and valid.



```
@Id
@GeneratedValue
private int customerId;

@NotEmpty (message = "The customer name must not be null.")
private String customerName;

@NotEmpty (message = "The customer email must not be null.")

private String customerEmail;

private String customerPhone;

@NotEmpty (message = "The customer username must not be null.")
private String username;

@NotEmpty (message = "The customer password must not be null.")
private String password;

private boolean enabled;

@OneToOne
@JoinColumn(name="billingAddressId")
private BillingAddress billingAddress;

@OneToOne
@JoinColumn(name="shippingAddressId")
private ShippingAddress shippingAddress;

@OneToOne
@JoinColumn(name = "cartId")
@JsonIgnore
private Cart cart;
```



Register Customer

Please fill in your information below:

Basic Info

Name: Yatin

Email: yrelhan4@gmail.com

Phone: 9041761362

Username: yrelhan

Password: ••••••••••

Confirm Password: ••••••••••

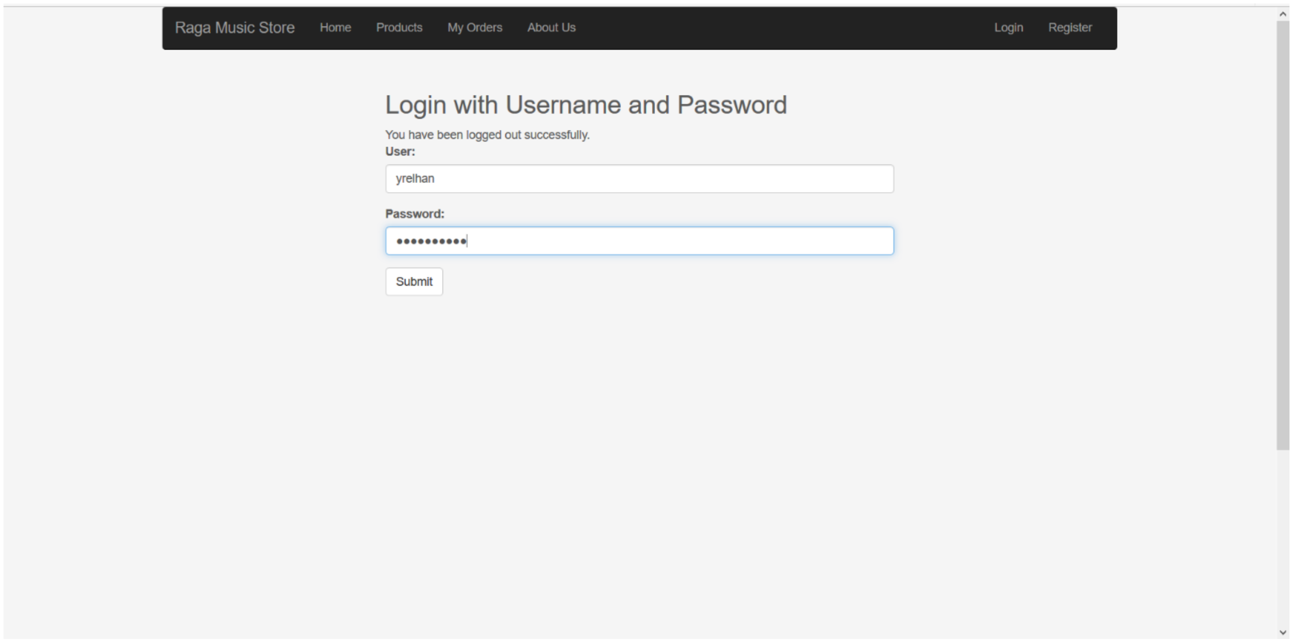
Billing Address

4.1.2 Module 2: Login and Logout

If the user is a returning customer, he can login using his registered *userid* and *password*.

The login is handled by *HomeController*. It checks whether the username-password combination exists in the database.

The *HomeController* calls the appropriate methods, and if a combination is found, logins the user.



Login with Username and Password

You have been logged out successfully.

User: yrelhan

Password: ••••••••••

For login purpose **Spring Security** is also used.

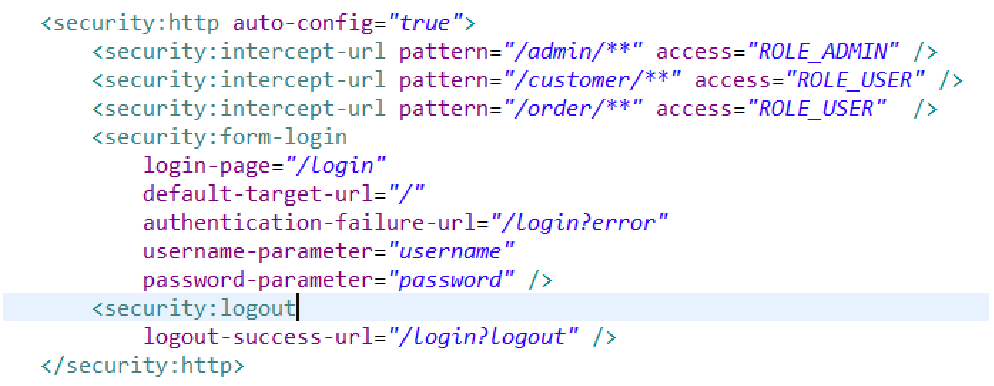
Spring Security provides security services for J2EE-based enterprise software applications.

There are two main areas for application securities.

1. **Authentication:** Process of checking the user, who they claim to be.
2. **Authorization:** Process of deciding whether an user is allowed to perform an activity within the application.

For this purpose two roles of *Admin* and *User* have been declared.

For the purposes of login, some parameters have to be declared in *Application Context.*



```
<security:http auto-config="true">
    <security:intercept-url pattern="/admin/**" access="ROLE_ADMIN" />
    <security:intercept-url pattern="/customer/**" access="ROLE_USER" />
    <security:intercept-url pattern="/order/**" access="ROLE_USER"  />
    <security:form-login
        login-page="/login"
        default-target-url="/"
        authentication-failure-url="/login?error"
        username-parameter="username"
        password-parameter="password" />
    <security:logout
        logout-success-url="/login?logout" />
</security:http>
```

To protect against CSRF attacks, a *csrfParameter and csrfToken* are included in the login page as hidden inputs.

4.1.3 Module 3: Product Catalog

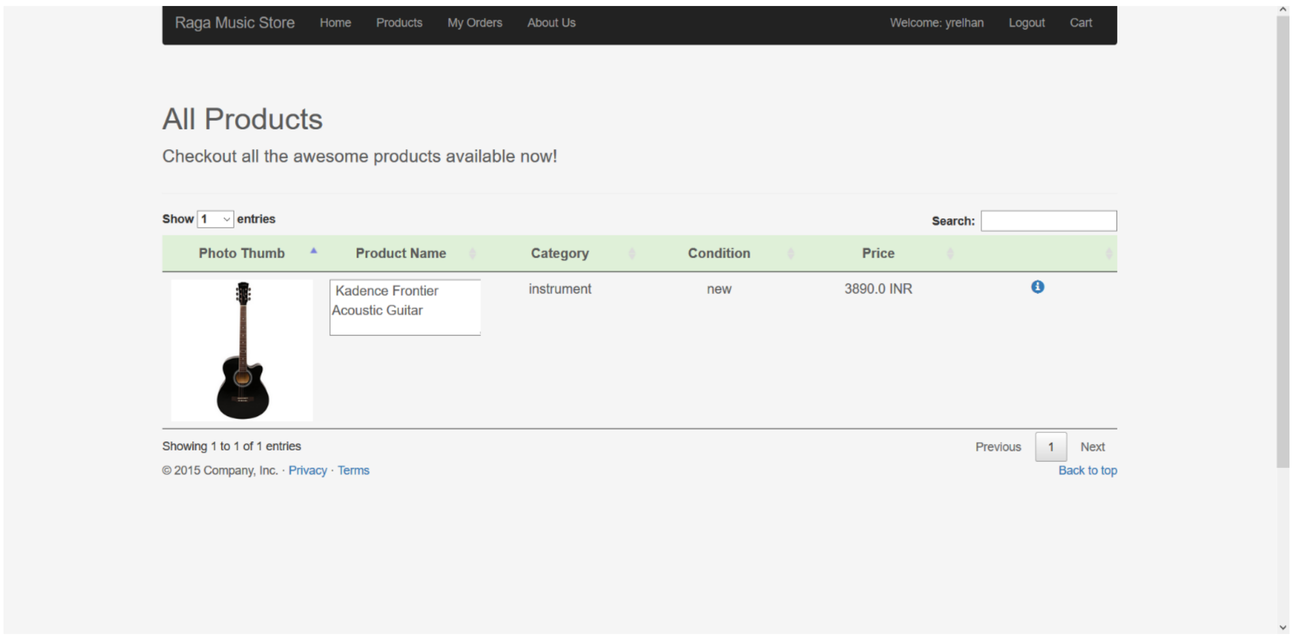
The user can click on the *Products* icon to see a list of all the products. This request is also handled by the *DispatcherServlet.*

Whenever user clicks on the Products link, the *DispatcherServlet* checks which controller handles the request, and in this case, it is the *ProductController.*

The appropriate method is then called inside the *ProductController* which further calls *ProductService* which calls *ProductDao* and the list of the products is populated from the database.

DataTables is also implemented to facilitate the searching and display of the products.

The user can also click the *OrderNow* button to send the request to accept the order to the admin. Whenever the user clicks the *OrderNow* button, an entry is added to the *MyOrders tab of the* listing the status of the order as *Pending.*


Raga Music Store Home Products My Orders About Us Welcome: yrelhan Logout Cart

All Products

Checkout all the awesome products available now!

Show 1 entries Search:

Photo Thumb	Product Name	Category	Condition	Price	
	Kadence Frontier Acoustic Guitar	instrument	new	3890.0 INR	

Showing 1 to 1 of 1 entries Previous 1 Next

© 2015 Company, Inc. · Privacy · Terms Back to top

Also, a request is sent to the administrator to accept the order. Once the administrator accepts the order, the product is added to the cart for the user to check out. All of this is handled by *CartController*, *OrderController* and *CartResources*.

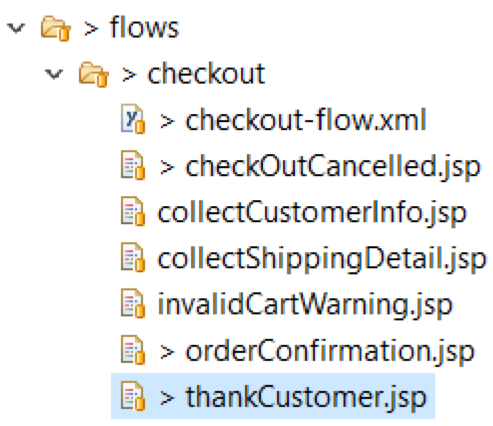
The above mentioned Controllers further call *CustomerOrderDao*, *CartDao*, *CartitemDao* to complete the order.

The request of the product is added to the *ProcessOrder* table while maintaining the *CustomerId*, *ProductId*, *CartId*, *Quantity*, *TotalPrice*, *ShippingAddress* and *BillingAddress*.

4.1.4 Module 4: Checkout

Checkout is handled by a framework called Spring Web-Flow or SWF.

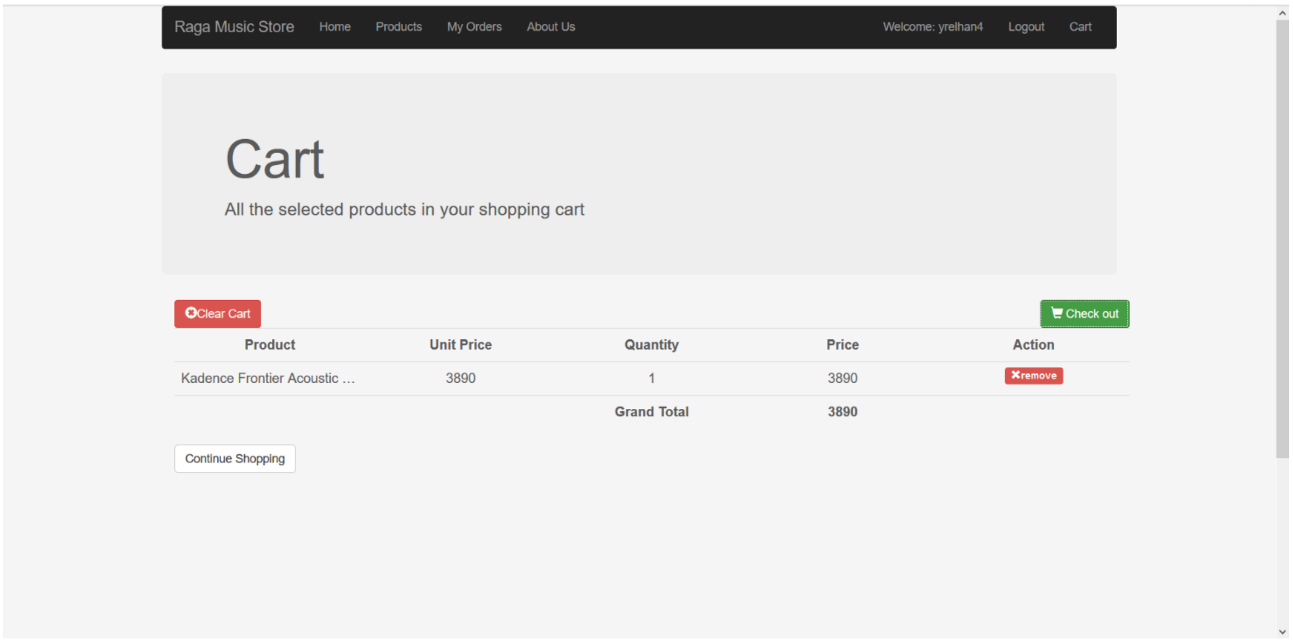
Spring Web Flow (SWF) is a component of the Spring Framework's web stack focused on the definition and execution of user interface (UI) flow within a web application. It is a module that allows you to make logical flows of your web application.


```
> flows
  > checkout
      > checkout-flow.xml
      > checkOutCancelled.jsp
      collectCustomerInfo.jsp
      collectShippingDetail.jsp
      invalidCartWarning.jsp
      > orderConfirmation.jsp
      > thankCustomer.jsp
```

Spring Web Flow is composed of a set of states (Displaying a View or executing any Action etc.). Transition of the flow from one state to another is triggered by an event. This continues till the flow completes and enters the end-state. The important Spring Web Flow states are:

1. The **start-state**, when a flow is created the initial state of the flow is defined by the start-state attribute in the Webflow.
2. An **action-state** executes an action and returns a logical result on its completion. The next to which the flow will be transitioned to depends on the result of this state A view-state when entered pauses the flow and returns the control to the client/user and the flow is resumed on the user/client event which resumes the flow and triggers the transition to the state depending on the user/client input or decision.
3. **End-state** signifies the end of the flow. When a flow enters the end-state the active flow session is terminated. If the end-state of the root flow is entered the resources associated with it are cleaned up automatically.

To evaluate any expression, that is to call any function *<evaluate expression =””>* tag is used.



Cart

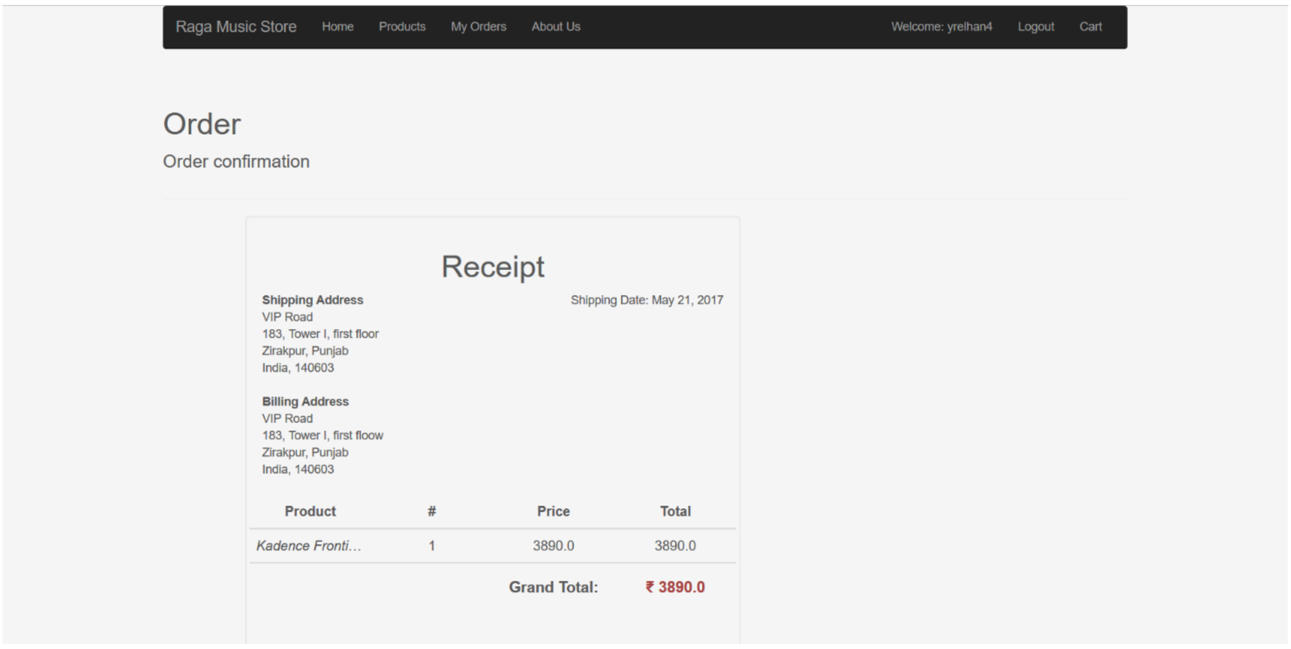
All the selected products in your shopping cart

Product	Unit Price	Quantity	Price	Action
Kadence Frontier Acoustic ...	3890	1	3890	✗remove
		Grand Total	3890	

Whenever the user checks out, the request is sent to the admin, to ship and deliver the product.

To complete the order *CartController*, *OrderController* and *CartResources* controllers are used.

The above mentioned Controllers further call *CustomerOrderDao*, *CartDao*, *CartitemDao* to complete the order.



Order

Order confirmation

Receipt

Shipping Address
VIP Road
183, Tower I, first floor
Zirakpur, Punjab
India, 140603

Shipping Date: May 21, 2017

Billing Address
VIP Road
183, Tower I, first floow
Zirakpur, Punjab
India, 140603

Product	#	Price	Total
Kadence Fronti...	1	3890.0	3890.0
		Grand Total:	**₹ 3890.0**

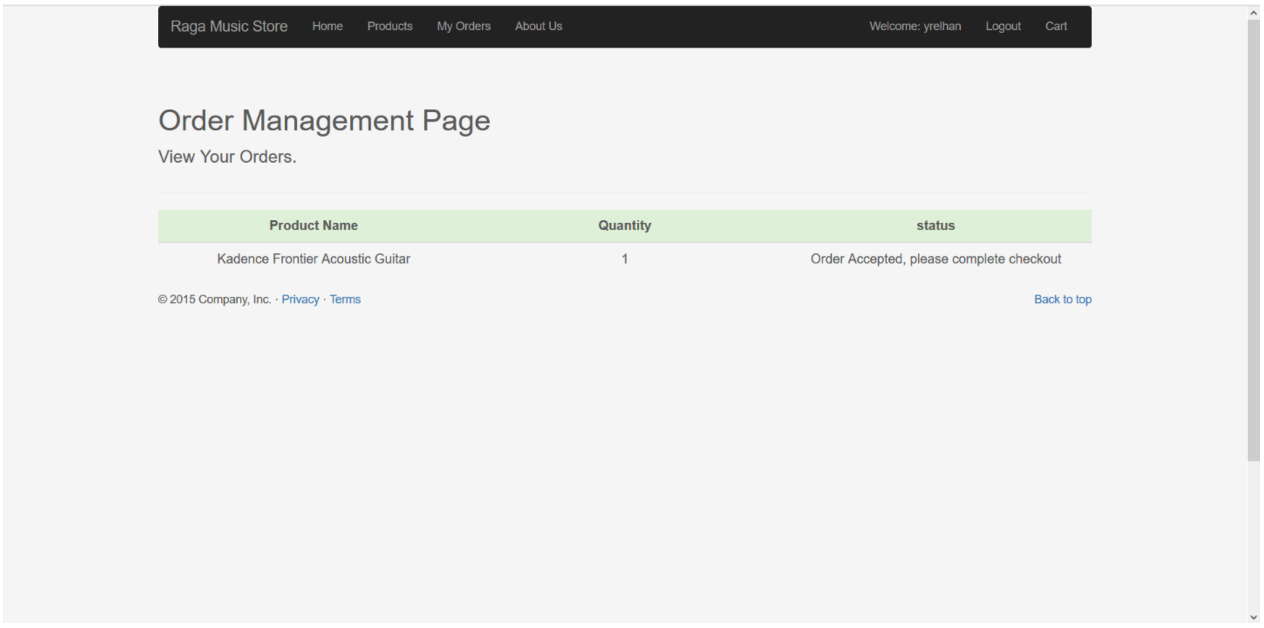
Also, whenever the user completes checkout, the quantity of products in the product inventory get decreased by the quantity that is ordered. The request goes to *CartItemDao* from where a query is run to get the quantity of the products of the specific *ProductId* in the inventory which is stored in *Product* table and another query is run to get the quantity that is being ordered which is stored in *CartItem* table in the database. The difference is the stored back in the *Product* table.

4.1.5 Module 5: Order Management

The admin has to accept the order of a customer before he/she can check out. Once the customer has checked out, the onus is on the administrator to first ship the order and then deliver the product.

The customer gets regular updates in the *My Orders* tab, such that

All this is handled by *CartResources* controller.



Order Management Page

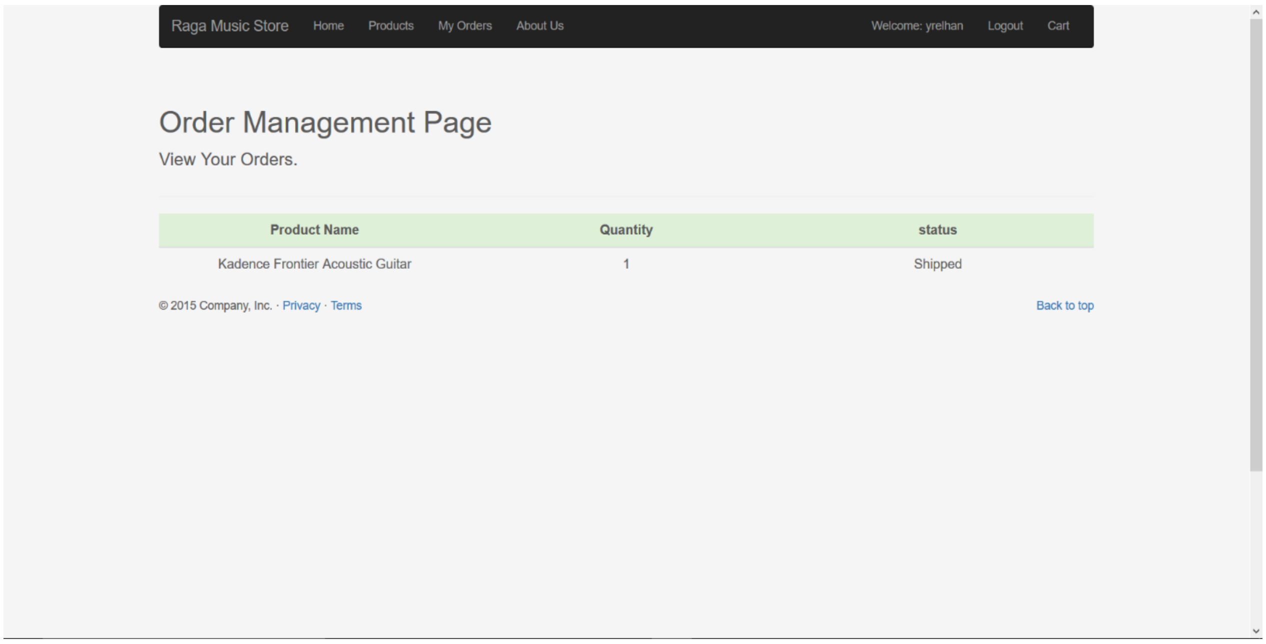
View Your Orders.

Product Name	Quantity	status
Kadence Frontier Acoustic Guitar	1	Order Accepted, please complete checkout

© 2015 Company, Inc. · Privacy · Terms

Back to top

The information is stored in *ProcessOrder* table in the database.Once the user checks out, the admin can let the user know once his order is shipped and can again update the status once the order has been delivered.



Order Management Page

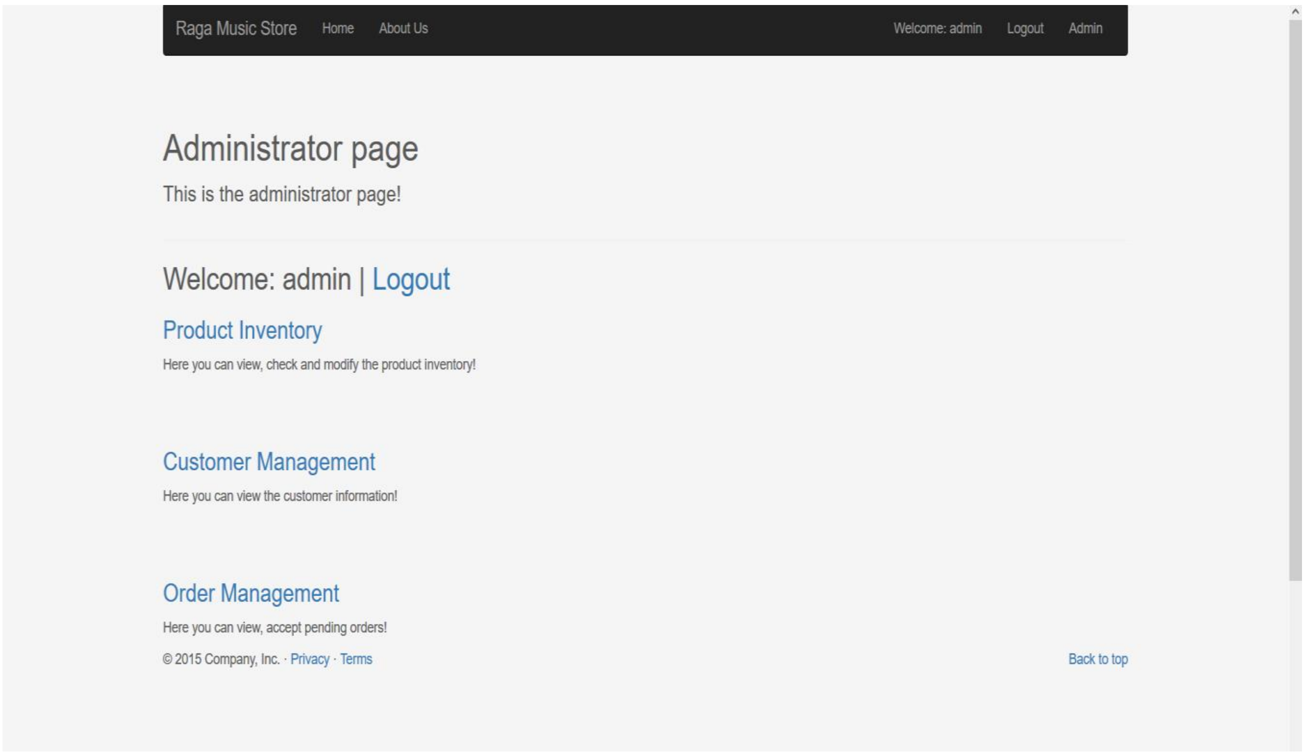
View Your Orders.

Product Name	Quantity	status
Kadence Frontier Acoustic Guitar	1	Shipped

© 2015 Company, Inc. · Privacy · Terms

Back to top

4.2 ADMINISTRATOR MODULES



Administrator page

This is the administrator page!

Welcome: admin | Logout

Product Inventory

Here you can view, check and modify the product inventory!

Customer Management

Here you can view the customer information!

Order Management

Here you can view, accept pending orders!

© 2015 Company, Inc. · Privacy · Terms

Back to top

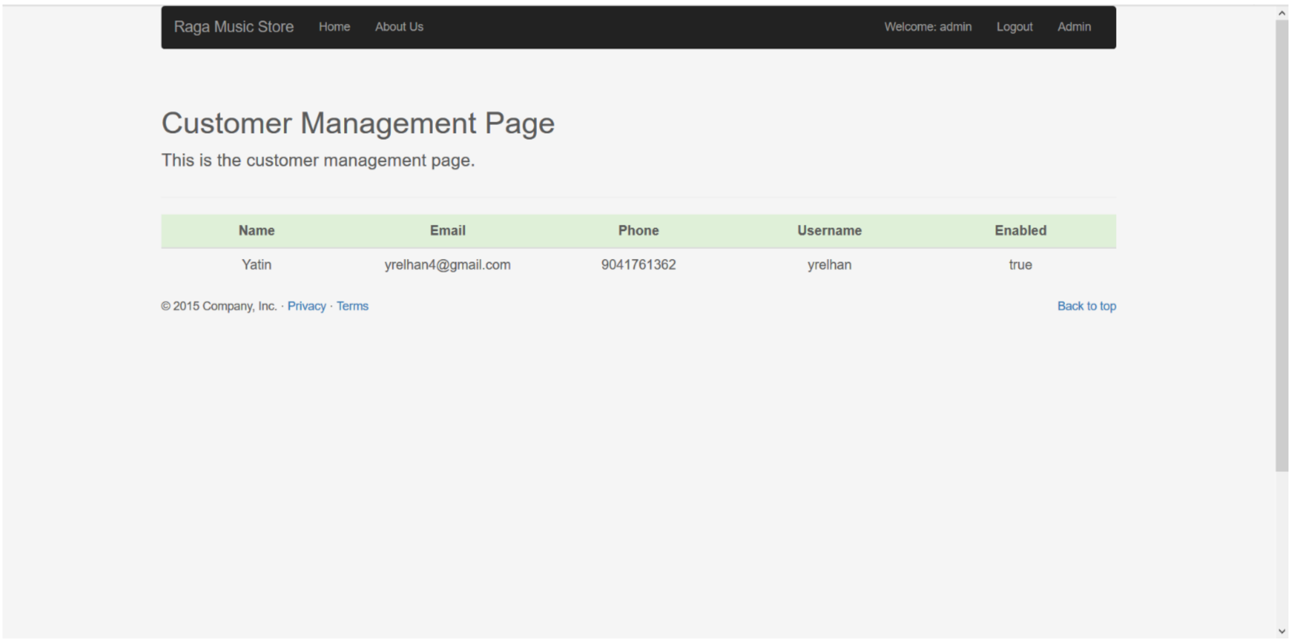
4.2.1 Module 1: Customer Management

The administrator has the responsibility of handling the customers. He can view the customer details including his name, email, phone, username.

For this purpose, the *DispatcherServlet* calls the *CustomerService* which further calls the *CustomerDao.* Inside the implementation of CustomerDao, a function to *getAllCutomers* is called to get a list of all the customers.

The *CustomerDao*, connects to the database and retrieves the list of all the customers in a list.

Inside the view, the information is collected in an AngularJS variable and displayed to the user.



Raga Music Store Home About Us Welcome: admin Logout Admin

Customer Management Page

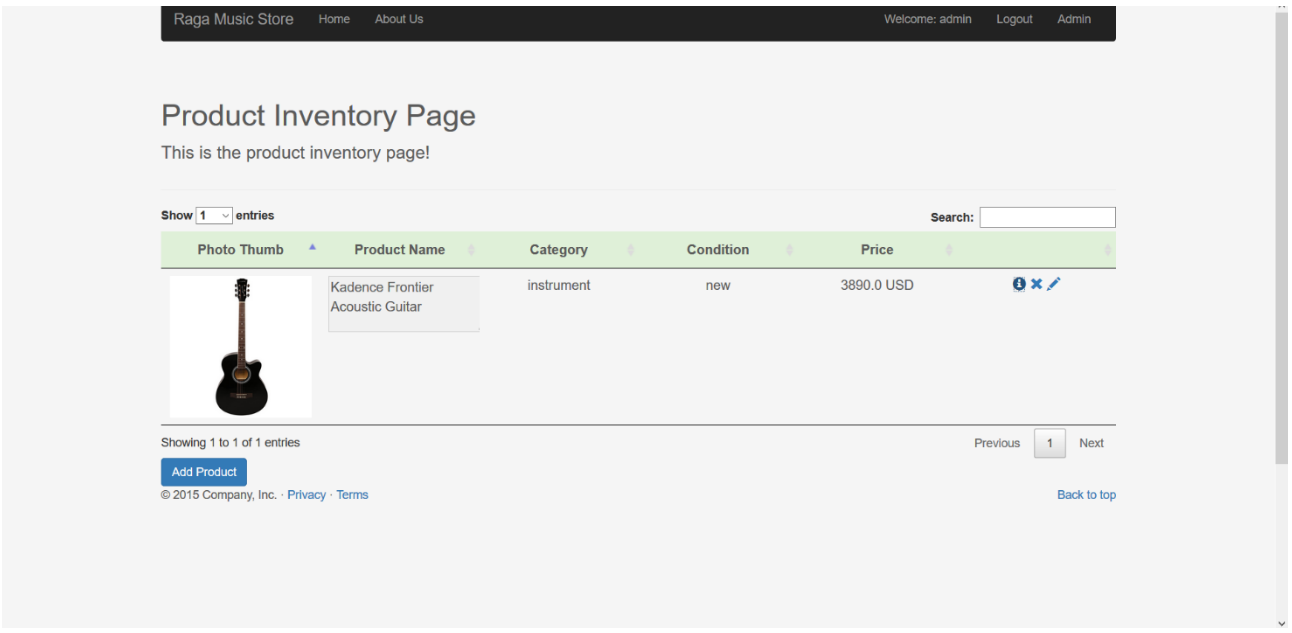
This is the customer management page.

Name	Email	Phone	Username	Enabled
Yatin	yrelhan4@gmail.com	9041761362	yrelhan	true

© 2015 Company, Inc. · Privacy · Terms Back to top

4.2.2 Module 2: Product Inventory

The administrator has the capability to handle the product inventory. He can view the products that he has added. Also he can add a new product, which includes Product name, product quantity, manufacturer details, an image.



Raga Music Store Home About Us Welcome: admin Logout Admin

Product Inventory Page

This is the product inventory page!

Show 1 entries Search:

Photo Thumb	Product Name	Category	Condition	Price	
	Kadence Frontier Acoustic Guitar	instrument	new	3890.0 USD	

Showing 1 to 1 of 1 entries Previous 1 Next

Add Product

© 2015 Company, Inc. · Privacy · Terms Back to top

To display the products, the *DispatcherServlet* calls the *AdminHome* controller which calls the method *getAllProducts* to get a list of all the products. This method calls the *productService* which further calls the *productDao* get the products details already stored in the database.

To handle the request of adding new products, the *DispatcherServlet* sends the request to *AdminProduct* controller. Inside the *AdminProduct* controller appropriate methods are called to add the product to the database.

Also inside the *AdminProduct* controller, there are methods are handle administrator's request to edit and remove the product. A similar process is followed for both too.

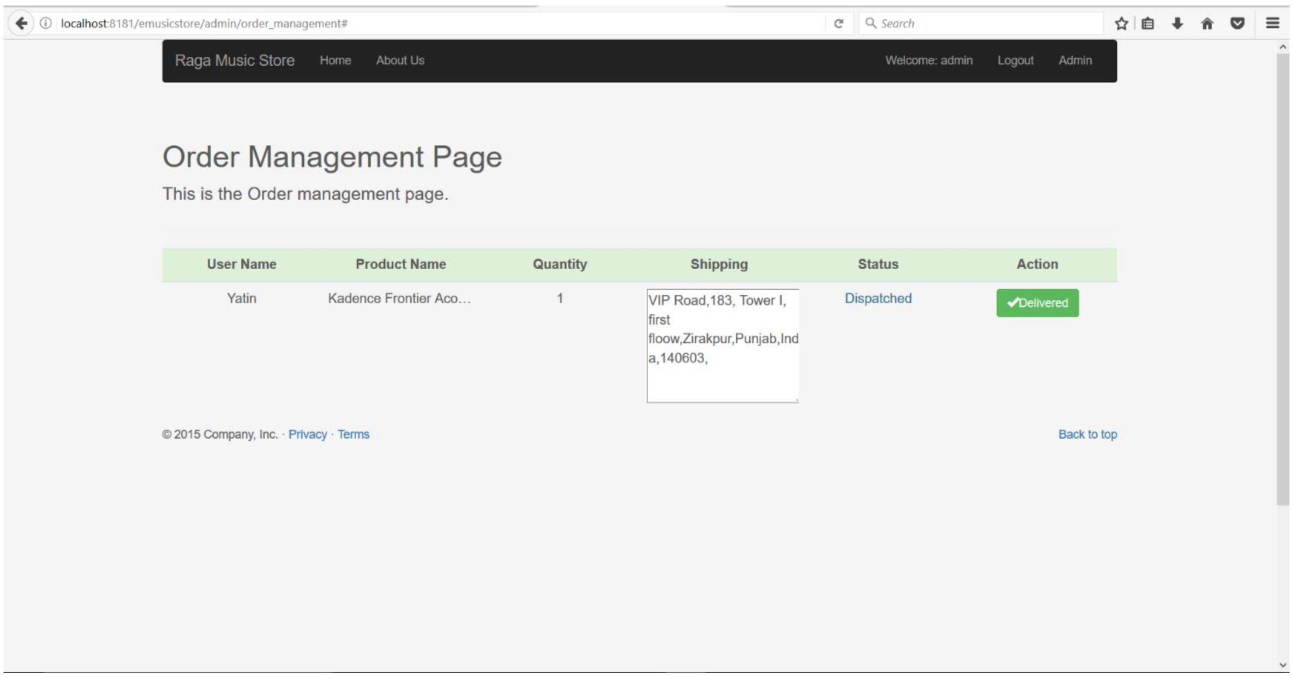
To add an image to the database, a bean named *MultiPartResolver* is added inside the *ApplicationContext* to handle the request to upload the file. The image file is uploaded inside the project folder with the same name as the *ProductId* it belongs to.

4.2.3 Module 3: Order Management

The administrator's role also includes handling the orders of the customer. Once the customer has requested a product, it is the administrator's responsibility to *Accept* the order. Once the customer checks out, the admin can then notify the user than he has *Shipped* the order and subsequently *Delivered* the order.

For this purpose, the *DispactherServlet* transfers the control to *CartResources* controller.

The controller checks if the request is to accept an order, that is to add the product to the cart or to ship the order, or to deliver the order. All these functionalities are combined in the same controller.



Order Management Page

This is the Order management page.

User Name	Product Name	Quantity	Shipping	Status	Action
Yatin	Kadence Frontier Aco...	1	VIP Road,183, Tower I, first floow,Zirakpur,Punjab,Inda,140603,	Dispatched	✓Delivered

© 2015 Company, Inc. · Privacy · Terms

Back to top

If the product is already present in the cart, the controller first calls the *customerService* to change the status of the product in the *ProcessOrder* table. Further, *cartItemService is called*, which further calls the *cartItemDao* to add the product inside the cart if not already present.

To fulfil the request of shipping the product again the request goes to the same controller *CartResources.* The controller calls the *customerService* which further calls the *customerDao* which changes the status of product to shipped.

To fulfil the request of delivering the product again the request goes to the same controller *CartResources.* The controller calls the *customerService* which further calls the *customerDao* which changes the status of product to delivered.

5 ***DESIGN***

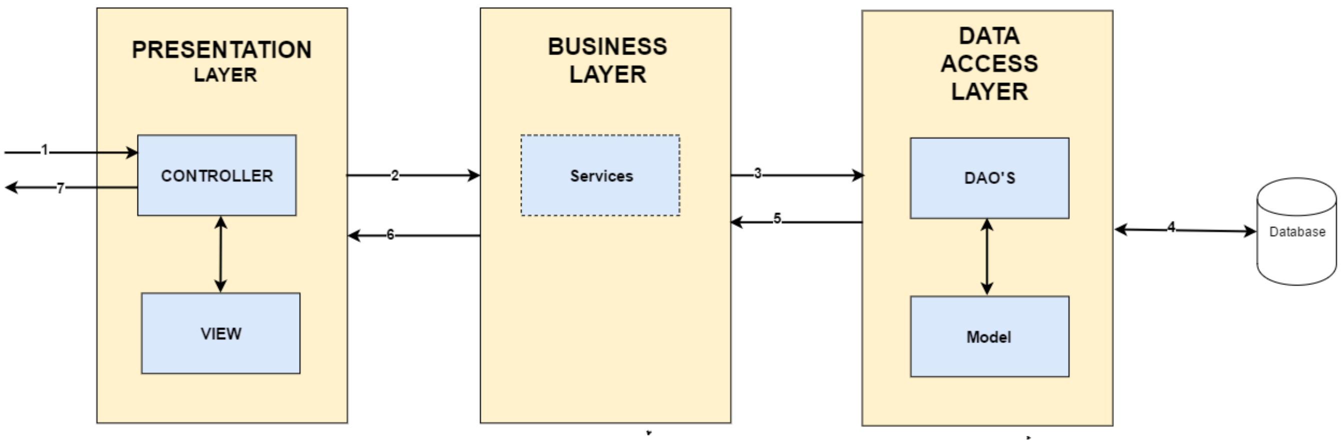
5.1 Introduction

The purpose of this document is to provide an architectural design for the Raga - Online Music Store. The design will show the presentation tier, the middle tier, composing of class and sequence diagrams, and the data tier. Each class will have a brief description about its purpose.

5.2 Architecture of the System

The architecture of the system is based on three-tier architecture. There are three logical tiers: the **Presentation Layer**, the **Business Layer**, and the **Data Layer**. The presentation tier is responsible for displaying the contents to the user while business tier communicates between presentation tier and data tier, for example receiving the search request for the product and passing it to the database and submitting the results back to presentation tier for display, or displaying the cart empty message when there is no product in the cart. It is the main component of the website since it is responsible for handling the overall logic. The data tier is responsible for storing the product information, transactions of the customer and shopping cart products. The data tier is also responsible handling the requests from the business tier and using stored procedures passing the result back.

The figure below shows the three-tier architecture of Raga – Online Music Store:



5.2.1 Presentation Layer

7.2.1.1 Views

The View's job is to translate data into a visual rendering for response to the Client (ie. web browser or other consumer). The data will be supplied primarily by the Controller; however, the View may also have a helper that can retrieve data that is associated with the rendering and not necessarily with the current request (ex. aside data, footer data).

The presentation tier for the system is JSP page with User Controls. The Eclipse IDE will be used to create the Web forms. It uses code behind code, where the code for each JSP page is encapsulated into a separate file.

JSTL or JavaServer Pages Standard Tag Library is also used. JSTL is a collection of useful JSP tags which encapsulates the core functionality common to many JSP applications. JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags.

JSP Pages	**Purpose**
Home.jsp	The Web page for welcome screen.
RegisterCustomer.jsp	The Web page where a new user can create a user and register himself.
Login.jsp	The Web page where a new user can log-in with the credentials he put in while registering.
ProductList.jsp	The Web page where the user can see all the products in the inventory. Even a user who has not logged in can see this page.
ViewProduct.jsp	The Web page where the user can view the details of a particular product.
OrderManagamentCustomer.jsp	The Web page where a user can check the status of his orders.
Cart.jsp	The web page where the user can see all the orders in his cart along with quantity and grand total.
CollectCustomerInfo.jsp	The Web page where a user has to fill in his information before submitting the order. This webpage is part of the Spring Webflow.

CollectShippingDetail.jsp	The Web page where a user has to fill in his shipping details bore submitting the order. This webpage is part of the Spring Webflow.
OrderConfirmation.jsp	The Web page where a user can confirm his order after reviewing it. This webpage is part of the Spring Webflow.
InvalidCartWarning.jsp	The Web page where a user gets an invalid cart warning because his cart is empty. This webpage is part of the Spring Webflow.
CheckOutCancelled.jsp	The Web page displayed when the user cancels his order. This webpage is part of the Spring Webflow.
Admin.jsp	The Web page where the admin has the option to chose one of the tasks.
CustomerManagement.jsp	The Web page where the admin can see the details of the customers that have registered.
OrderManagement.jsp	The Web page where the administrator can manage the orders of the customer.
ProductInventory.jsp	The Web page where the administrator can add new products, delete or edit existing products.
AddProduct.jsp	The Web page displayed when the admin choses to add new product.
EditProduct.jsp	The Web page when admin choses to edit any existing product in the inventory.
About.jsp	The Web page which has the information about the website.

7.2.1.2 Controllers

The Controller's job is to translate incoming requests into outgoing responses. In order to do this, the controller must take request data and pass it into the Service layer. The service layer then returns data that the Controller injects into a View for rendering. This view might be HTML for a standard web request; or, it might be something like JSON (JavaScript Object Notation) for a RESTful API request.

Controller	Purpose
AdminHome.java	Handles the request for the pages of product inventory, customer management and order management for the administrator.
AdminProduct.java	Handles the request for the pages of add product, edit product and delete product page for the administrator.
CartController.java	Handles the request for the functionalities of getting of the cart for a particular customer.
CartResources.java	Handles the request for the page of adding of a particular product to the cart. Also handles the functionality of changing the status of the products already in the cart.
HomeController.java	This is the main controller of the web application. Handles the home page, the login page and the about is page of the web application.
OrderController.java	This controller helps to create the order for a particular customer. Also helps to get all orders for that customer.
ProductController.java	This controller handles the request of displaying all the products. Also handles the request to view details of a particular product.
RegisterController.java	This controller handles the request to register a customer. Also once the user registers it autologins the user.

5.2.2 Business Layer

7.2.2.1 Services

This layer provides cohesive, high-level logic for related parts of an application. This layer is invoked directly by the Controller and View helpers.

Services	Purpose
ProductService.java	This service calls the ProductDao for the functions mentioned later.
CustomerService.java	It calls the CustomerDao for the fucntions mentioned later.
CustomerOrderService.java	This service computes the grand total of all the items for a particular customer. It also calls the CustomerOrderDao for the fucntions mentioned later.
CartService.java	It calls the CartDao for the functions mentioned later
CartItemService.java	It calls the CartItemDao for the functions mentioned later.

5.2.3 Data Access Layer

Data Access Objects (DAO)

This layer has Data Access Objects or DAO’s provides access to the persistence layer. This layer is only ever invoked by Service objects. Objects in the data access layer do not know about each other.

Data Access Objects	Purpose
ProductDao.java	This data access object requests the database for the following functionalities. • Get a product by id • Get all products • Add a product • Edit a product • Delete a product
CustomerDao.java	This data access object requests the database for the following functionalities. • Add a customer • Get a customer with a particular id • Get a list of all customers • Get a customer with a particular username • Get all orders for a particular customer • Get all orders for a particular customer • Accepting an order • Dispatching an order • Delivering an order
CustomerOrderDao.java	This data access object stores the order details of a particular customer to the database.
CartDao.java	This data access object handles the following functionalities. • Get a cart by id • Update a cart when a new product is added • Validating a cart
CartItemDao.java	This data access object handles the following functionalities. • Adding a product to the cart • Removing product from a cart • Removing all items from the cart • Get a particular product from the cart

Models

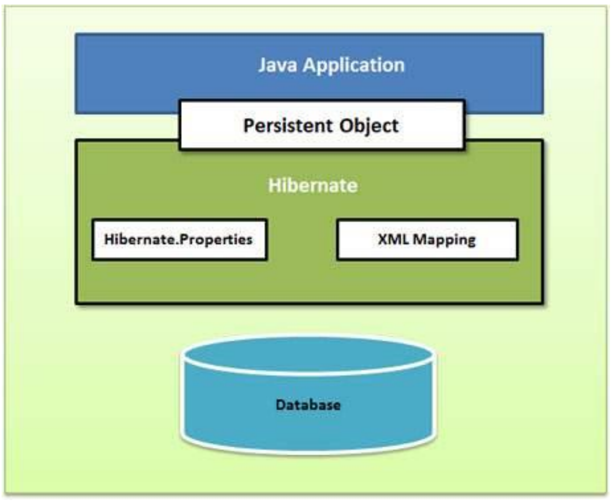
The Model's job is to represent the problem domain, maintain state, and provide methods for accessing and mutating the state of the application.

Model	Purpose
Authorities.java	This model is required by Spring Security to implement roles in the web application
Users.java	This model is required by Spring Security to implement users for the web application.
Customer.java	This model has all the fields that are required for a particular customer.
Cart.java	This model has all the fields that are required to create a cart for a particular customer.
CatItem.java	This model has all the fields that are required to create a cart item for a particular customer.
CustomerOrder.java	This model has all the fields that are required to create an order for a particular customer.
Producut.java	This model has all the fields that are required to create a product in the database.
ProcessOrder.java	This model has all the fields that are required to process orders of a particular customer.
ShippingAddress.java	This model has all the fields that are required to set the shipping address of a particular customer.
BillingAddress.java	This model has all the fields that are required to set the billing address of a particular customer.

5.2.4 Database

ORM or Object Relational Model allows you to use java objects as representation of a relational database. It maps the two concepts (object-oriented and relational)

Hibernate is an ORM framework - you describe how your objects are represented in your database, and hibernate handles the conversion.



MySQL database has been used.

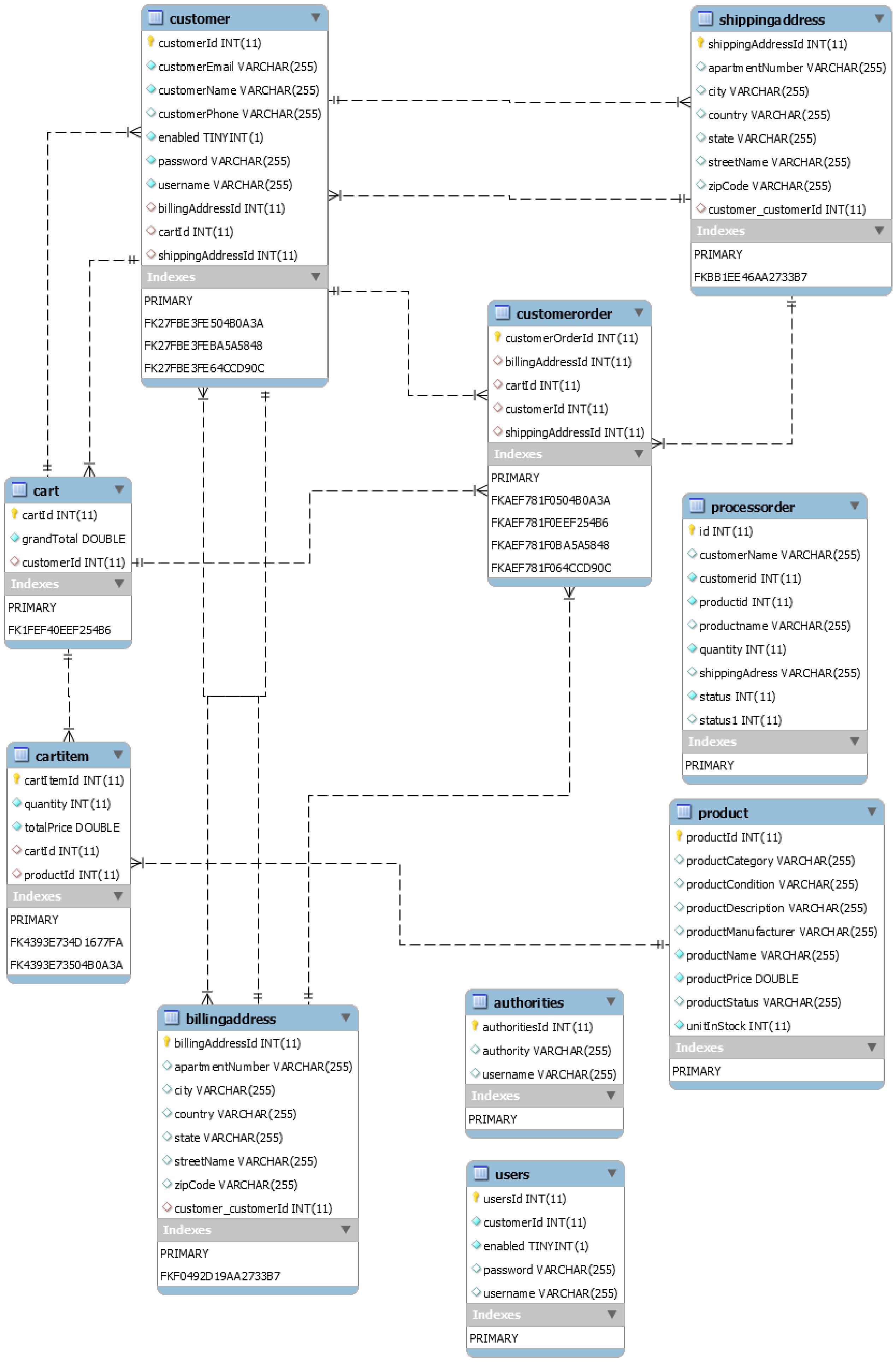
Each model had the annotation *@Entity* which directed Hibernate to automatically create the tables.

We can also mark a field in the model as *@Transient*. This annotation would skip that particular field while making the table in the database.

@OnetoMany or *@ManytoOne* annotations can be used to describe the relations between any two tables.

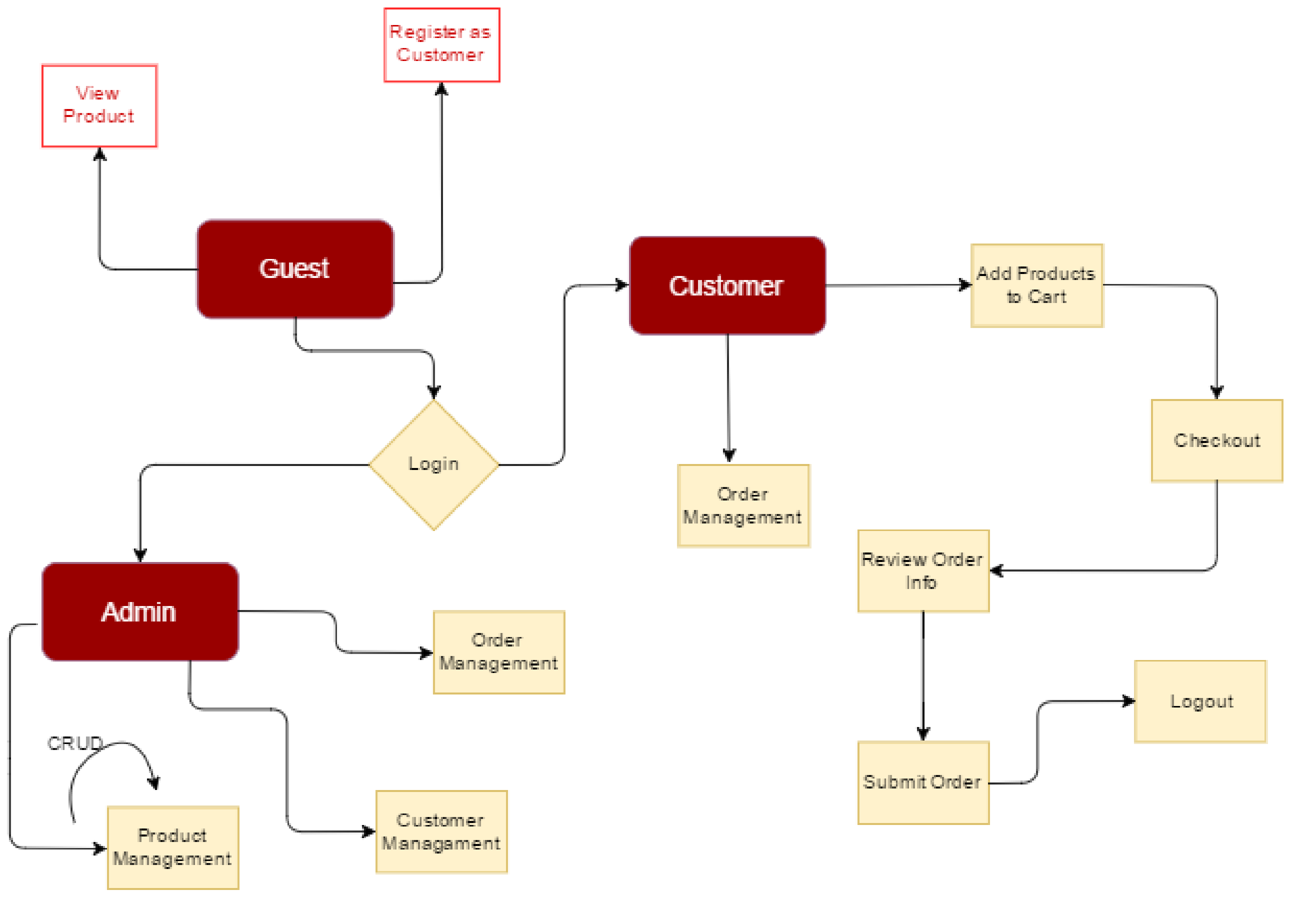
Also *@JoinColumn* can be used to join two columns of a table i.e. the primary key and foreign key relationship can be defined using these two annotations

ER diagram on the next page, shows the Primary key-Foreign Key and One-One or One-Many relationship between any two tables in the database.


customer
customerId INT(11)
customerEmail VARCHAR(255)
customerName VARCHAR(255)
customerPhone VARCHAR(255)
enabled TINYINT(1)
password VARCHAR(255)
username VARCHAR(255)
billingAddressId INT(11)
cartId INT(11)
shippingAddressId INT(11)
Indexes
PRIMARY
FK27FBE3FE504B0A3A
FK27FBE3FEBA5A5848
FK27FBE3FE64CCD90C
shippingaddress
shippingAddressId INT(11)
apartmentNumber VARCHAR(255)
city VARCHAR(255)
country VARCHAR(255)
state VARCHAR(255)
streetName VARCHAR(255)
zipCode VARCHAR(255)
customer_customerId INT(11)
Indexes
PRIMARY
FKBB1EE46AA2733B7
customerorder
customerOrderId INT(11)
billingAddressId INT(11)
cartId INT(11)
customerId INT(11)
shippingAddressId INT(11)
Indexes
PRIMARY
FKAEF781F0504B0A3A
FKAEF781F0EEF254B6
FKAEF781F0BA5A5848
FKAEF781F064CCD90C
cart
cartId INT(11)
grandTotal DOUBLE
customerId INT(11)
Indexes
PRIMARY
FK1FEF40EEF254B6
processorder
id INT(11)
customerName VARCHAR(255)
customerid INT(11)
productid INT(11)
productname VARCHAR(255)
quantity INT(11)
shippingAdress VARCHAR(255)
status INT(11)
status1 INT(11)
Indexes
PRIMARY
cartitem
cartItemId INT(11)
quantity INT(11)
totalPrice DOUBLE
cartId INT(11)
productId INT(11)
Indexes
PRIMARY
FK4393E734D1677FA
FK4393E73504B0A3A
product
productId INT(11)
productCategory VARCHAR(255)
productCondition VARCHAR(255)
productDescription VARCHAR(255)
productManufacturer VARCHAR(255)
productName VARCHAR(255)
productPrice DOUBLE
productStatus VARCHAR(255)
unitInStock INT(11)
Indexes
PRIMARY
billingaddress
billingAddressId INT(11)
apartmentNumber VARCHAR(255)
city VARCHAR(255)
country VARCHAR(255)
state VARCHAR(255)
streetName VARCHAR(255)
zipCode VARCHAR(255)
customer_customerId INT(11)
Indexes
PRIMARY
FKF0492D19AA2733B7
authorities
authoritiesId INT(11)
authority VARCHAR(255)
username VARCHAR(255)
Indexes
PRIMARY
users
usersId INT(11)
customerId INT(11)
enabled TINYINT(1)
password VARCHAR(255)
username VARCHAR(255)
Indexes
PRIMARY

5.3 Flow and Sequence Diagrams

5.3.1 Flow Diagram of the Web-App



The above flow diagram represents what actions a Guest, User or an Administrator can perform.

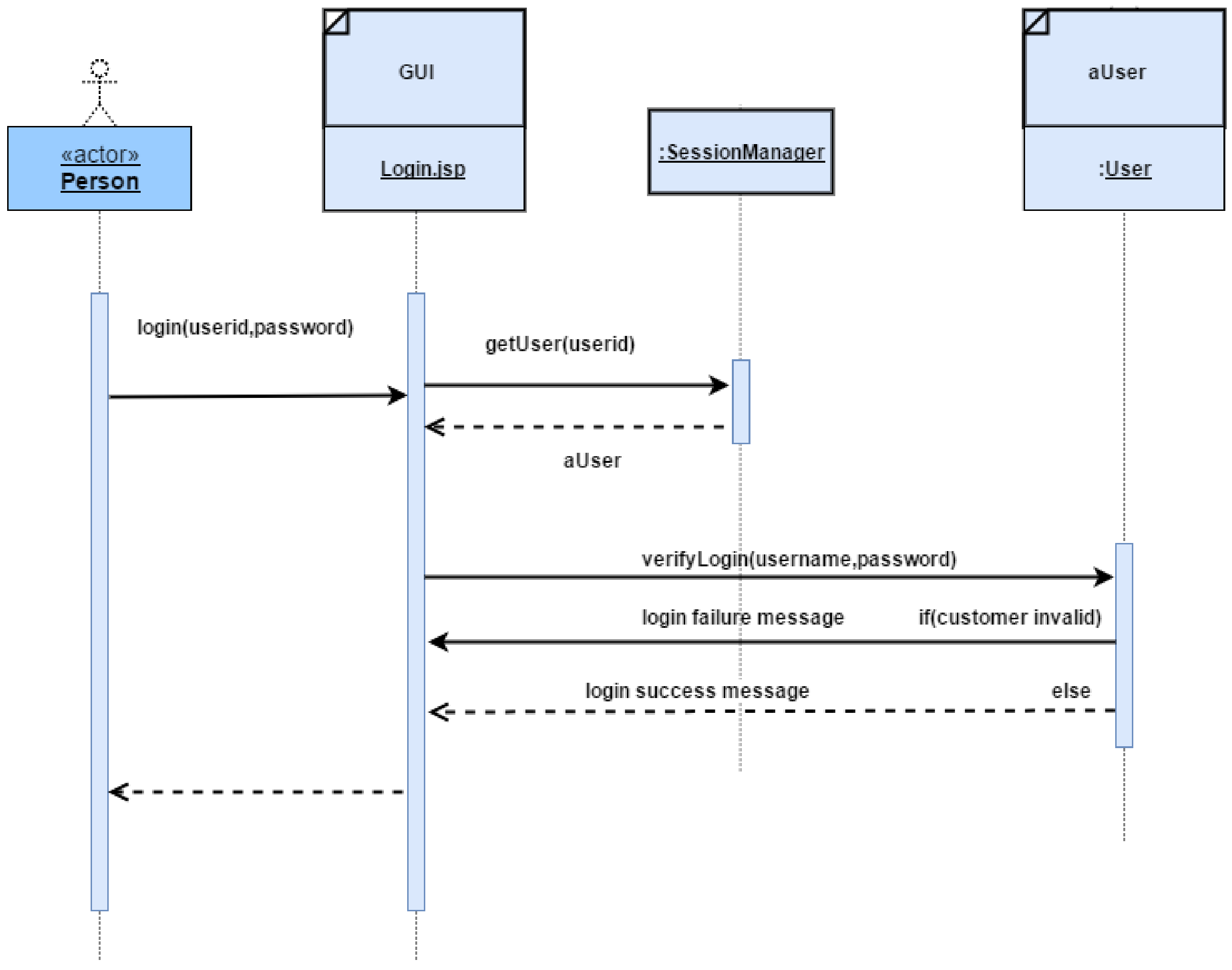
The guest can view the product list or register as a customer.

After the guest registers and then logins, he can perform various tasks as described in the flow chart.

The administrator can also login with his credentials and perform his tasks.

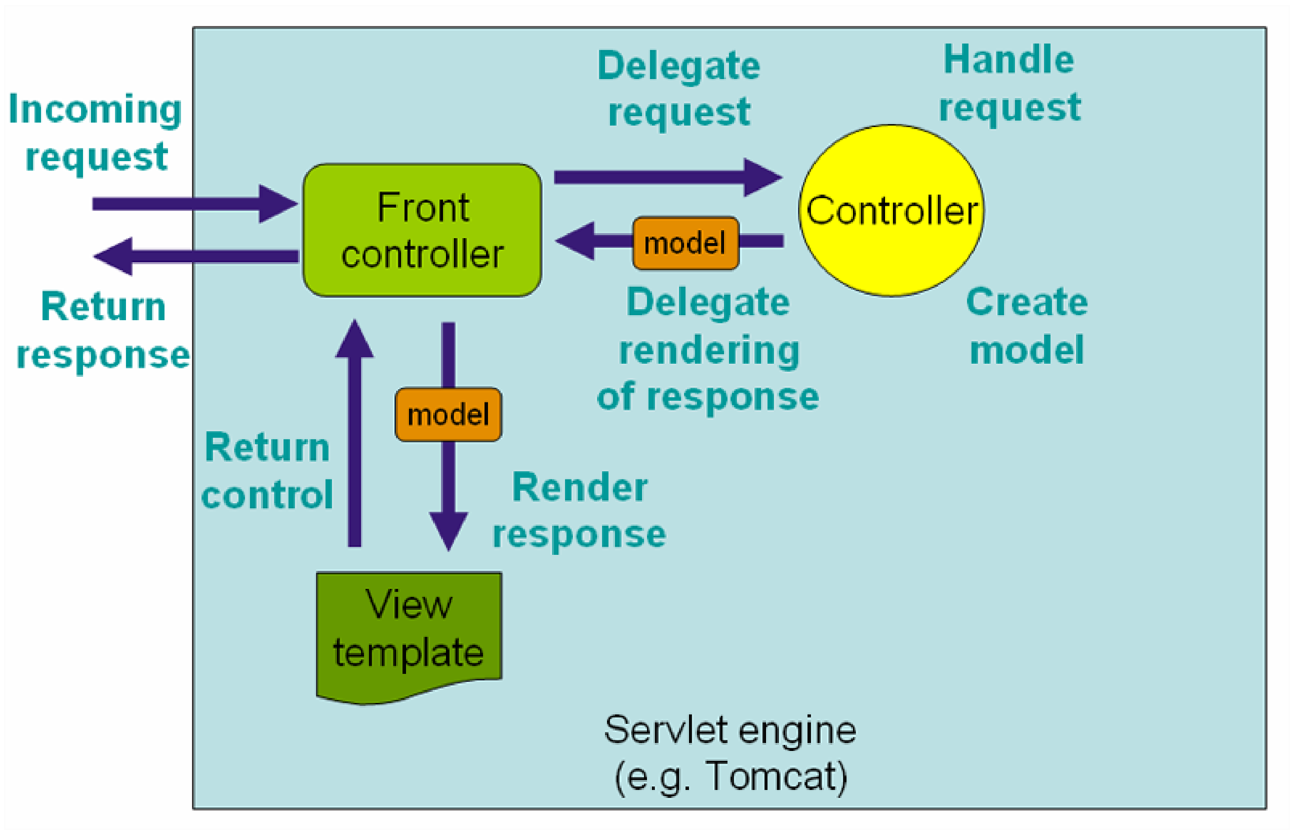
5.3.2 Sequence Diagrams

7.3.2.1 System Login Sequence Diagram



The user needs to log in to the system for accessing secure sites. The user will enter his/her userid and password then clicks on the login button on the Login.aspx Web form. The SessionManager class gets the user form the User class and its password provided is verified with the User class instance it gets back. If the userid and password are correct, the system will direct the user to appropriate Web page. Otherwise, the system will prompt the user to check the userid and password.

5.3.3 Diagram showing how Dispatcher Servlet works



The central component of Spring MVC is a Spring controller. A controller is the only servlet you need to configure in a Java web deployment descriptor web.xml file.

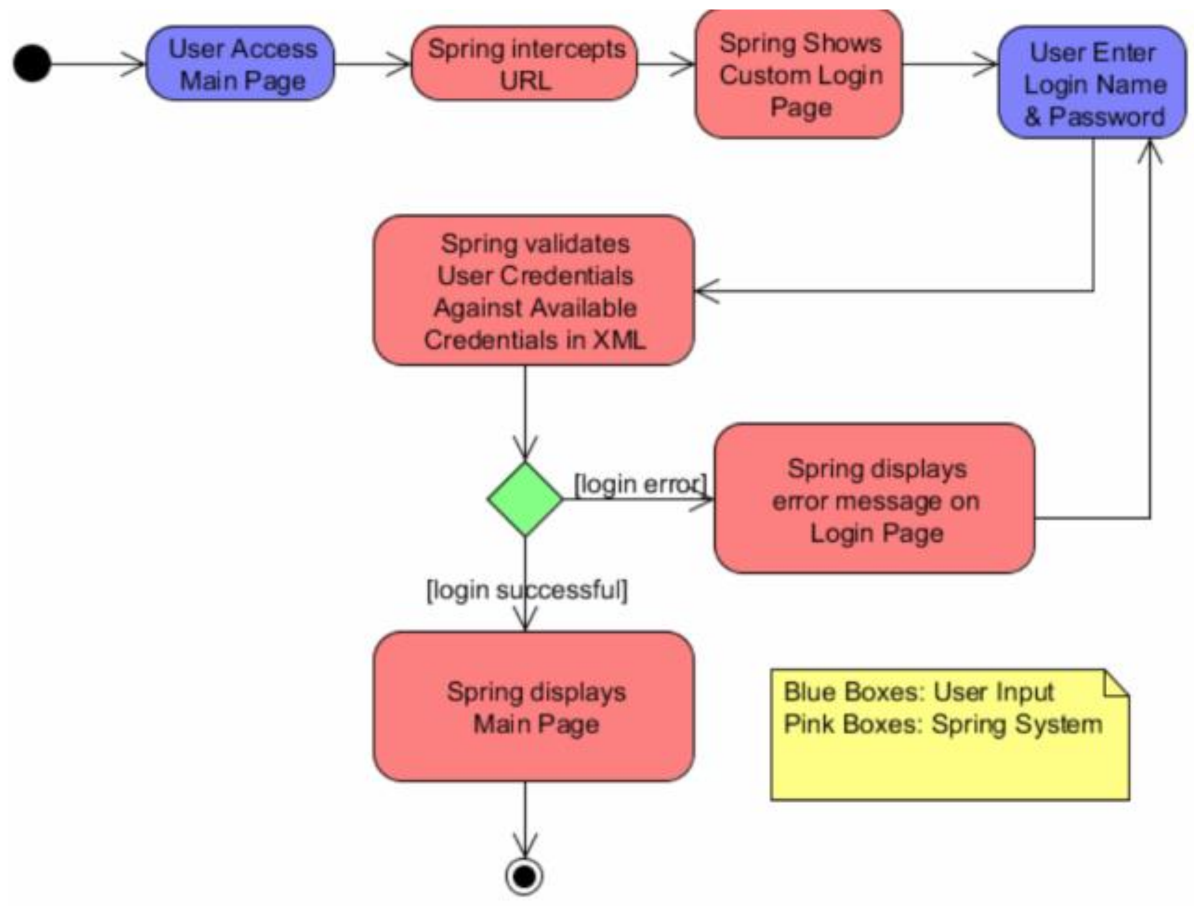
A Spring MVC controller is otherwise called as front controller generally referred to a single servlet called Dispatcher Servlet. The front controller manages the entire request handling process and every web request must go through it.

When a web request is sent to a Spring MVC application, a controller first receives the request. Then it organizes the different components configured in Spring's web application context or annotations present in the controller itself, all needed to handle the request.

Role of a Dispatcher Servlet

- Dispatcher Servlet is used to handle all incoming requests and route them through Spring
- Uses customizable logic to determine which controllers should handle which incoming requests
- Forwards all responses to through view handlers to determine the correct views to route responses to
- Exposes all beans defined in Spring to controllers for dependency injection

5.3.4 Diagram showing how Spring Security works


User Access Main Page
Spring intercepts URL
Spring Shows Custom Login Page
User Enter Login Name & Password
Spring validates User Credentials Against Available Credentials in XML
[login error]
Spring displays error message on Login Page
[login successful]
Spring displays Main Page
Blue Boxes: User Input
Pink Boxes: Spring System