Sri Lanka Institute of Information Technology

Shopping Platform
Report

Distributed System
Assignment 02

Group ID: 2021S1_REG_WD_25

Submitted by:

1. IT19155708– Shanghavi.R
2. IT19049878 - E. Jathusanan
3. IT19033174 - M.Vithursan
4. IT19007748 - M.Praveen

Introduction

"Spring Digital Mart" is an online shopping website developed for buying and selling products over the Internet and provides services for both buyers and sellers to fulfill their shopping needs. Customers can easily search and buy one or more product through this online platform.

This application is developed using spring boot for the backend services while react js and redux for the frontend development. MySQL Workbench has been used as the database.

In our application sellers can add, update, delete and view items and also buyer can search, view and buy the items added by the seller. Once the Items are selected by the buyer, they will be added to the cart. Buyer can proceed to checkout to finalize the order. Once buyer checks out, they will receive a verification mail to confirm the purchase. From that mail, buyers will be redirected to payment gateway. Buyers have the facilities to do the payment via credit cards.

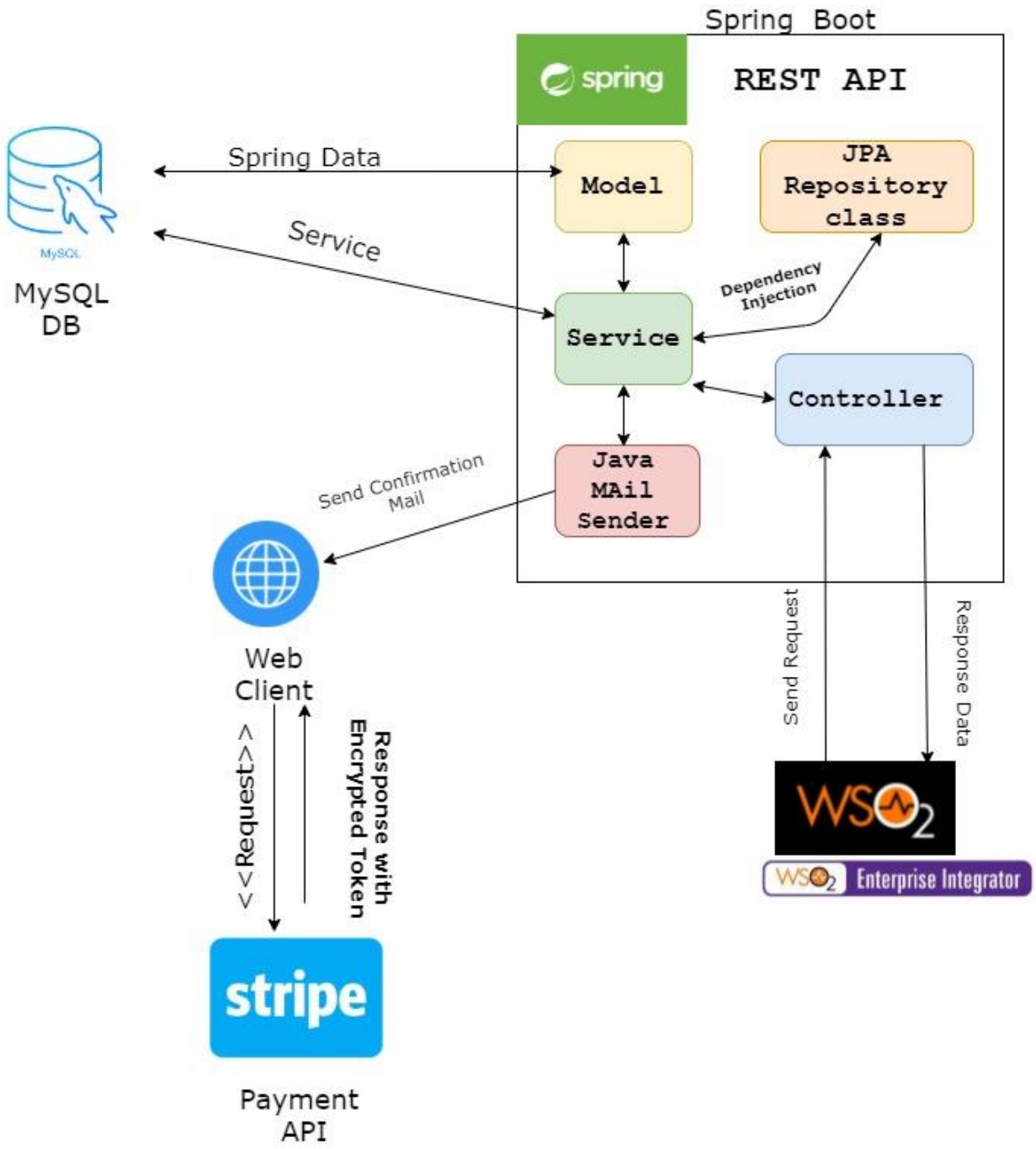
Several authentication and security mechanisms has been implemented to verify the security and reliability of the shopping platform.

This report will elaborate the functionalities of this shopping platform with the aid of diagrams.

Tools And Technologies

1. Web Client
    - ReactJS
    - Bootstrap
    - Redux
    - VS Code
2. REST API
    - Spring Boot
    - MySQL
    - Eclipse IDE
3. Java Mail Sender
4. Stripe payment gateway
5. JSON Web Token

High Level Architectural Diagram



Services

1. Web Client

Frontend of the web application is developed using React, Redux and Bootstrap libraries. Backend of the application communicates using Axios library. The Screenshot below shows how the buyer component functions.

```
const Items_API_BASE_URL = "http://localhost:8080/API/SpringBoot/Items";

class ItemService {

    getItemsById(ItemId){
        return axios.get(Items_API_BASE_URL + '/' + ItemId);
    }
```

```
onSearch = ()=> {
    return axios.get(`http://localhost:8080/API/SpringBoot/search/${this.state.searchWord}`).then(res => {
        this.setState({Items: res.data})
    }).catch(err => {
        console.log(err);
    })
}
```

```
    this.state = {
        myurl:'http://localhost:3000/',
        searchWord: "",
        items: []
    }
}
```

```
//add cart action
export const addToCart= (id)=>{

    return{
        type: ADD_TO_CART,

        id
    }
}
```

```
const cartReducer= (state = initState,action)=>{

    //INSIDE HOME COMPONENT
    if(action.type === ADD_TO_CART){
        console.log("reducer");
        console.log(action.id);

         let addedItem = state.Items.find(Item=> Item.id === action.id)
         //check if the action id exists in the addedItems
        let existed_Item= state.addedItems.find(Item=> action.id === Item.id)

        if(existed_Item)
        {
           addedItem.qty += 1
            return{
               ...state,
               total: state.total + addedItem.price
                }
        }
        else{
           addedItem.qty = 1;
           //calculating the total
           let newTotal = state.total + addedItem.price
           console.log("added item");

           return{
               ...state,
               addedItems: [...state.addedItems, addedItem],
               total : newTotal
           }
```

2. Item service

Buyer and Seller has two different interfaces. When they request the application for the item service, they will be directed to the pages according to their role and can perform relevant tasks.

1. **getAllItems() –** This API is implemented to display all the items and their details on the home page. Name of the item, description of the item, price, quantity and the image of all the items..

```
// get all Items
@GetMapping("/Items")
public List<Item> getAllItems(){
    return ItemRepository.findAll();
}
```

2. **getItemById() –**This API is implemented to display the item seller is wishing to see. In this API take the item id parameter which is given to the item automatically when the item is added by the seller.

```
// get Item by id rest api
@GetMapping("/Items/{id}")
public ResponseEntity<Item> getItemById(@PathVariable Long id) {
    Item Item = ItemRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Item not exist with id :" + id));
    return ResponseEntity.ok(Item);
}
```

3. **createItem() –** This API is implemented for the seller to add a new item to the database to sell. This API takes the item object parameter, which is used to add a new item in the database.

```
// create Item rest api
@PostMapping("/Items")
public Item createItem(@RequestBody Item Item) {
    return ItemRepository.save(Item);
}
```

4. **updateItem()** –This API is implemented to update an existing item's details. Item id parameter is passed to find the specific item, so the seller can edit the details of the item.

```
// update Item rest api

@PutMapping("/Items/{id}")
public ResponseEntity<Item> updateItem(@PathVariable Long id, @RequestBody Item ItemDetails){
    Item Item = ItemRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Item not exist with id :" + id));

    Item.setItemName(ItemDetails.getItemName());
    Item.setQty(ItemDetails.getQty());
    Item.setPrice(ItemDetails.getPrice());
    Item.setDescription(ItemDetails.getDescription());
    Item.setPath(ItemDetails.getPath());
    Item updatedItem = ItemRepository.save(Item);
    return ResponseEntity.ok(updatedItem);
}
```

5. **deleteItem()** –This API is implemented for the sellers to delete the items as they prefer.

```
// delete Item rest api
@DeleteMapping("/Items/{id}")
public ResponseEntity<Map<String, Boolean>> deleteItem(@PathVariable Long id){
    Item Item = ItemRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Item not exist with id :" + id));

    ItemRepository.delete(Item);
    Map<String, Boolean> response = new HashMap<>();
    response.put("deleted", Boolean.TRUE);
    return ResponseEntity.ok(response);
}
```

6. **searchItem()** – This API is implemented for the buyer to search for the item they prefer. This API passes the search term entered by the buyer to compare with all the items' details to find the best results for the user.

```
@GetMapping("/search/{key}")
public List<Item> searchItem(@PathVariable("key") String key){
   return ItemRepository.findByNameLike("%"+key+"%");
}
```

3. Delivery service

When the buyer requests the system for the delivery service that is provided in the shopping system,

1. **getAllItems()**- This API is implemented to get all the delivery details for the seller to view. Address, city and zip code of the delivery will be displayed by the API for the seller to carry out the deliveries.

```
@GetMapping("/GetDetails")
public List<DeliveryDetail> getAllItems(){
    return DeliveryRepository.findAll();
}
```

2. **createDeliveryDetail()**- This API is implemented for a new delivery to be added. A DeliveryDetail object is passed as the parameter to create a new record in the database.

```
@PostMapping("/Delivery")
public String createDeliveryDetail(@RequestBody DeliveryDetail DeliveryDetail) {
    DeliveryRepository.save(DeliveryDetail);
    return "success";
}
```

3. **deleteDelivery()**-This API is implemented for the seller to delete a delivery record. this API passes the id of the specific record to be deleted.

```
@DeleteMapping("/GetDetails/{id}")
public ResponseEntity<Map<String, Boolean>> deleteDelivery(@PathVariable Long id){

    DeliveryDetail Detail = DeliveryRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Item not exist with id :" + id));

    DeliveryRepository.delete(Detail);
    Map<String, Boolean> response = new HashMap<>();
    response.put("deleted", Boolean.TRUE);
    return ResponseEntity.ok(response);
}
```

4. Payment service

When the buyer requests the system for payment service system will offer **Credit Card Payment.**

When the buyer proceeds to pay, he/ she has to add their card number, CVC, expired date and email. The system will update the database and will send a confirmation message via email..

```
@Service
public class PaymentService {

    @Value("${STRIPE_SECRET_KEY}")
    private String secretKey;

    @PostConstruct
    public void init() {
        Stripe.apiKey = secretKey;
    }
    public String charge(PaymentRequest chargeRequest) throws StripeException {
         Map<String, Object> chargeParams = new HashMap<>();
         chargeParams.put("amount", chargeRequest.getAmount());
         chargeParams.put("currency", PaymentRequest.Currency.INR);
         chargeParams.put("source", chargeRequest.getToken().getId());

        Charge charge = Charge.create(chargeParams);
        return charge.getId();
    }
}
```

```
@RestController
@RequestMapping("/payment")
public class PaymentController {

    @Autowired
    PaymentService service;

    @PostMapping
    public ResponseEntity<String> completePayment(@RequestBody PaymentRequest request) throws StripeException {
        String chargeId= service.charge(request);
        return chargeId!=null? new ResponseEntity<String>(chargeId,HttpStatus.OK):
            new ResponseEntity<String>("Please check the credit card details entered",HttpStatus.BAD_REQUEST);
    }

    @ExceptionHandler
    public String handleError(StripeException ex) {
        return ex.getMessage();
    }
}
```

5. Authentication / Security Mechanism

To improve the level of security and protection of the shopping application, We have developed several security mechanisms.

1. **User Authentication**

When entering to the application the users must be registered to the system after that they have to give their credentials (Username and password) to enter into the system.

2. **User Authorization**

According to our system, there are two types of users are using our system. They are Buyers and Sellers. According to their username and password they will be re-directed to their relevant home pages. Buyers only have the access to add or modify the orders while the sellers have the access to add and modify the items to be sold.

Appendix

CreateItemComponent.js

```
import React, { Component } from 'react'
import ItemService from '../services/ItemService';

class CreateItemComponent extends Component {
    constructor(props) {
        super(props)

        this.state = {
            // step 2
            id: this.props.match.params.id,
            itemName: '',
            qty: '',
            price: '',
            description: '',
            path: ''
        }
        this.changeItemNameHandler = this.changeItemNameHandler.bind(this);
        this.changeQtyHandler = this.changeQtyHandler.bind(this);
        this.changePriceHandler = this.changePriceHandler.bind(this);
        this.changeDescriptionHandler = this.changeDescriptionHandler.bind(this);
        this.changePathHandler = this.changePathHandler.bind(this);
        this.saveOrUpdateItem = this.saveOrUpdateItem.bind(this);
    }

    // step 3
    componentDidMount(){

        // step 4
        if(this.state.id === '_add'){
            return
        }else{
            ItemService.getItemsById(this.state.id).then( (res) =>{
                let Item = res.data;
                this.setState({itemName: Item.itemName,
                    qty: Item.qty,
                    price: Item.price,
                    description: Item.description,
                    path: Item.path
                });
            });
        }
```

```
    }
    saveOrUpdateItem = (e) => {
        e.preventDefault();
        let Item = {itemName: this.state.itemName, qty: this.state.qty, price: th
is.state.price, description: this.state.description, path: this.state.path };
        console.log('Item => ' + JSON.stringify(Item));

        // step 5
        if(this.state.id === '_add'){
            ItemService.createItems(Item).then(res =>{
                this.props.history.push('/Items');
            });
        }else{
            console.log(this.state.id +" "+Item.path);
            ItemService.updateItems(Item, this.state.id).then( res => {
                this.props.history.push('/Items');
            });
        }
    }

    changeItemNameHandler= (event) => {
        this.setState({itemName: event.target.value});
    }

    changeQtyHandler= (event) => {
        this.setState({qty: event.target.value});
    }

    changePriceHandler= (event) => {
        this.setState({price: event.target.value});
    }
    changeDescriptionHandler= (event) => {
        this.setState({description: event.target.value});
    }
    changePathHandler= (event) => {
        this.setState({path: event.target.value});
    }

    cancel(){
        this.props.history.push('/Items');
    }

    getTitle(){
        if(this.state.id === '_add'){
            return <h3 className="text-center">Add Item</h3>
```

```
        }else{
            return <h3 className="text-center">Update Item</h3>
        }
    }
    render() {
        return (
            <div>
                <br></br>
                   <div className = "container">
                        <div className = "row">
                            <div className = "card col-md-6 offset-md-3 offset-
md-3">
                                {
                                    this.getTitle()
                                }
                                <div className = "card-body">
                                    <form>
                                        <div className = "form-group">
                                            <label> Item Name: </label>
                                            <input placeholder="Item Name" name="
firstName" className="form-control"
                                                value={this.state.itemName} onCha
nge={this.changeItemNameHandler}/>
                                        </div>
                                        <div className = "form-group">
                                            <label> Item Qty: </label>
                                            <input placeholder="Item Qty" name="q
ty" className="form-control"
                                                value={this.state.qty} onChange={
this.changeQtyHandler}/>
                                        </div>
                                        <div className = "form-group">
                                            <label> Item Price: </label>
                                            <input placeholder="Item Price" name=
"price" className="form-control"
                                                value={this.state.price} onChange
={this.changePriceHandler}/>
                                        </div>
                                        <div className = "form-group">
                                            <label> Item Description: </label>
                                            <input placeholder="Item Description"
 name="description" className="form-control"
                                                value={this.state.description} on
Change={this.changeDescriptionHandler}/>
                                        </div>
```

```
                                        <div className = "form-group">
                                            <label> Item Path: </label>
                                            <input placeholder="Item Path" name="
path" className="form-control"
                                                value={this.state.path} onChange=
{this.changePathHandler}/>
                                        </div>

                                        <button className="btn btn-
success" onClick={this.saveOrUpdateItem}>Save</button>
                                        <button className="btn btn-
danger" onClick={this.cancel.bind(this)} style={{marginLeft: "10px"}}>Cancel</but
ton>
                                    </form>
                                </div>
                            </div>
                        </div>

                    </div>
                </div>
        )
    }
}

export default CreateItemComponent
```

ListItemComponents.js

```
import React, { Component } from 'react'
import ItemService from '../services/ItemService'

class ListItemComponent extends Component {
    constructor(props) {
        super(props)

        this.state = {
                Items: []
        }
        this.addItem = this.addItem.bind(this);
        this.editItem = this.editItem.bind(this);
        this.deleteItem = this.deleteItem.bind(this);
```

```
    }

    deleteItem(id){
        ItemService.deleteItems(id).then( res => {
            this.setState({Items: this.state.Items.filter(Item => Item.id !== id)
});
        });
    }
    viewItem(id){
        this.props.history.push(`/view-item/${id}`);
    }
    editItem(id){
        this.props.history.push(`/add-item/${id}`);
    }

    componentDidMount(){
        ItemService.getItems().then((res) => {
            this.setState({ Items: res.data});
        });
    }

    addItem(){
        this.props.history.push('/add-item/_add');
    }

    render() {
        return (
            <div>
                 <h2 className="text-center">Item List</h2>
                 <div className = "row">
                    <button className="btn btn-
primary" onClick={this.addItem}> Add Item</button>
                 </div>
                 <br></br>
                 <div className = "row">
                        <table className = "table table-striped table-bordered">

                            <thead>
                                <tr>
                                    <th> Item</th>
                                    <th> Item Name</th>
                                    <th> Item Qty</th>
                                    <th> Item Price</th>
                                    <th> Item Description</th>
                                    <th> Actions</th>
```

```
                                </tr>
                            </thead>
                            <tbody>
                                {
                                    this.state.Items.map(
                                        Item =>
                                        <tr key = {Item.id}>
                                            <td><img src={Item.path} alt="" heig
ht="150" width="150"/></td>
                                            <td> { Item.itemName} </td>
                                            <td> {Item.qty}</td>
                                            <td> {Item.price}</td>
                                            <td> {Item.description}</td>
                                            <td>
                                                <button onClick={ () => this.edi
tItem(Item.id)} className="btn btn-info">Update </button>
                                                <button style={{marginLeft: "10p
x"}} onClick={ () => this.deleteItem(Item.id)} className="btn btn-
danger">Delete </button>
                                                <button style={{marginLeft: "10p
x"}} onClick={ () => this.viewItem(Item.id)} className="btn btn-
info">View </button>
                                            </td>
                                        </tr>
                                    )
                                }
                            </tbody>
                        </table>

                    </div>

                </div>
            )
        }
}

export default ListItemComponent
```

ViewItemComponent.js

```
import React, { Component } from 'react'
import ItemService from '../services/ItemService'

class ViewItemComponent extends Component {
    constructor(props) {
        super(props)

        this.state = {
            id: this.props.match.params.id,
            item: {}
        }
    }

    componentDidMount(){
        ItemService.getItemsById(this.state.id).then( res => {
            this.setState({item: res.data});
        })
    }

    render() {
        return (
            <div>
                <br></br>
                <div className = "card col-md-6 offset-md-3">
                    <h3 className = "text-center"> View Item Details</h3>
                    <div className = "card-body">
                        <div className = "row">
                            <label> Item Name: </label>
                            <div> { this.state.item.itemName }</div>
                        </div>
                        <div className = "row">
                            <label> Item Qty: </label>
                            <div> { this.state.item.qty }</div>
                        </div>
                        <div className = "row">
                            <label> Item Price: </label>
                            <div> { this.state.item.price }</div>
                        </div>
                        <div className = "row">
                            <label> Item Description </label>
                            <div> { this.state.item.description  }</div>
                        </div>
                        <div className = "row">
```

```
                            <label> Item path: </label>
                            <img src="lg.jpg"  height="150" width="150" alt="prod
uct"/>
                            <div>{ this.state.item.path  }</div>
                        </div>

                    </div>

                </div>
            </div>
        )
    }
}

export default ViewItemComponent
```

BuyerHome.js

```
import React, { Component } from 'react'
import Card from 'react-bootstrap/Card'
import StripeButton from './stripebutton.component'
import Button from 'react-bootstrap/Button'
import Form from 'react-bootstrap/Form'
import FormControl from 'react-bootstrap/FormControl'
import {addToCart} from '../actions/action-types/cartActions';
import { connect } from "react-redux";
import axios from "axios";
import ItemService from "../services/ItemService"


class BuyerHome extends Component {

 constructor(props) {
    super(props)

    this.state = {
        myurl:'http://localhost:3000/',
        searchWord: "",
        items: []
    }
}

componentDidMount(){
    this.goBack();
```

```
}

onSearch = ()=> {
    return axios.get(`http://localhost:8080/API/SpringBoot/search/${this.state.se
archWord}`).then(res => {
        this.setState({Items: res.data})
    }).catch(err => {
        console.log(err);
    })
}

goBack = () => {
    ItemService.getItems().then((res) => {
        this.setState({ Items: res.data});
        this.setState({ searchWord: ""});
    });
}

handleClick = (id)=> {
    console.log("hello");
    console.log(id)
    this.props.addToCart(id);
}

    render() {
        return (
            <div>
                 <Form inline>
                <FormControl
                 type="text"
                 placeholder="Search Here"
                 className="mr-sm-2"
                 value={this.state.searchWord}
                 onChange={(e)=> this.setState({
                    searchWord: e.target.value
                 })}
                 />
                 {this.state.searchWord != "" ? <Button variant="outline-
info" onClick={this.goBack}>X</Button> : null}
                <Button variant="outline-
info" onClick={this.onSearch}>Search</Button>
                 </Form>

                <div className="card">
```

```
                {this.props.Items.map(
                Item =>

                    <Card key={Item.id} style={{ width: '18rem' }}>
                    <Card.Img variant="top" src={this.state.myurl+Item.path} widt
h="200px" height="300px"/>
                    <Card.Body>
                    <Card.Title>{Item.itemName}</Card.Title>
                    <Card.Text>
                    {Item.description}
                     </Card.Text>

                     <Card.Text>
                    {Item.price}
                    </Card.Text>

                    <StripeButton price={Item.price} variant="primary">Buy Now</S
tripeButton>
                    <Button variant="primary" onClick={()=>{this.handleClick(Item
.id)}}>Add to Cart</Button>
                    </Card.Body>
                    </Card>

                    )}

                </div>
            </div>
        )
    }
}

const mapStateToProps = (state)=>{
    return {
        Items: state.Items
         }
}

const mapDispatchToProps= (dispatch)=>{
    return{
       addToCart: (id)=>{dispatch(addToCart(id))}
    }
}

export default connect(mapStateToProps,mapDispatchToProps)(BuyerHome)
```

cart.js

```
import React, { Component } from 'react';
import { connect } from 'react-redux'
import { Link } from 'react-router-dom'
import { removeItem,addQuantity,subtractQuantity} from '../actions/action-
types/cartActions'
import Recipe from './Recipe'

class Cart extends Component{

     //to remove the item completely
     handleRemove = (id)=>{
        this.props.removeItem(id);
    }
    //to add the quantity
    handleAddQuantity = (id)=>{
        this.props.addQuantity(id);
    }
    //to substruct from the quantity
    handleSubtractQuantity = (id)=>{
        this.props.subtractQuantity(id);
    }

    render(){

        console.log("cart");
        console.log(this.props.Items);

        let addedItems = this.props.Items.length ?
            (
                this.props.Items.map(Item=>{
                    return(

                        <li className="collection-item avatar" key={Item.id}>
                                    <div className="item-img">
                                        <img src={Item.img} alt={Item.img} classN
ame=""/>
                                    </div>

                                    <div className="item-desc">
                                        <span className="title">{Item.title}</spa
n>
                                        <p>{Item.desc}</p>
                                        <p><b>Price: {Item.price}$</b></p>
                                        <p>
```

```
                                        <b>Quantity: {Item.qty}</b>
                                    </p>
                                    <div className="add-remove">
                                        <Link to="/cart"><i className="materi
al-
icons" onClick={()=>{this.handleAddQuantity(Item.id)}}>arrow_drop_up</i></Link>
                                        <Link to="/cart"><i className="materi
al-
icons" onClick={()=>{this.handleSubtractQuantity(Item.id)}}>arrow_drop_down</i></
Link>
                                    </div>
                                    <button className="waves-effect waves-
light btn pink remove" onClick={()=>{this.handleRemove(Item.id)}}>Remove</button>
                                </div>

                            </li>
                    )
                })
            ):

             (
                <p>Nothing.</p>
             )
       return(
            <div className="container">
                <div className="cart">
                    <h5>You have ordered:</h5>
                    <ul className="collection">
                        {addedItems}
                    </ul>
                </div>
                <Recipe />
            </div>
       )
    }
}

const mapStateToProps = (state)=>{
    console.log("card");
    console.log(state.addedItems);
    return{
        Items: state.addedItems
    }
}
const mapDispatchToProps = (dispatch)=>{
```

```
    return{
        removeItem: (id)=>{dispatch(removeItem(id))},
        addQuantity: (id)=>{dispatch(addQuantity(id))},
        subtractQuantity: (id)=>{dispatch(subtractQuantity(id))}
    }
}

export default connect(mapStateToProps,mapDispatchToProps)(Cart)
```

StripeButtonComponent.js

```
import React from "react";
import StripeCheckout from "react-stripe-checkout";
import axios from "axios";

const StripeButton = ({ price }) => {
  const publishableKey = "pk_test_vODJNnbks07dYxUhFM5MOXJs00N7hKX8qb";
  const stripePrice = price * 100;

  const onToken = (token) => {
    console.log(token);
    axios
      .post("http://localhost:8083/payment", {
        amount: stripePrice,
        token,
      })
      .then((response) => {
        console.log(response);
        alert("payment success");
      })
      .catch((error) => {
        console.log(error);
        alert("Payment failed");
      });
  };

  return (
    <StripeCheckout
      amount={stripePrice}
      label="Pay Now"
      name="SPRING DIGITAL MART"
      image="https://svgshare.com/i/CUz.svg"
      description={`Your total is ${price}`}
      panelLabel="Pay Now"
      token={onToken}
```

```
      stripeKey={publishableKey}
      currency="INR"
    />
  );
};

export default StripeButton;
```

DeliveryDetails.js

```
import React, { Component } from 'react'
import Form from 'react-bootstrap/Form'
import FormControl from 'react-bootstrap/FormControl'
import Button from 'react-bootstrap/Button'
import axios from "axios";
import { Col}  from 'react-bootstrap';

export default class DeliveryDetails extends Component {

    constructor(props) {

        super(props);



        this.onChangeAddress = this.onChangeAddress.bind(this);

        this.onChangeCity = this.onChangeCity.bind(this);

        this.onChangeZip = this.onChangeZip.bind(this);


        this.onSubmit = this.onSubmit.bind(this);



        this.state = {

          address: "",

          city: "",

          zipcode: "",
```

```
    };
  }


  onChangeAddress(e) {
    this.setState({
      address: e.target.value,
    });
  }


  onChangeCity(e) {
    this.setState({
      city: e.target.value,
    });
  }


  onChangeZip(e) {
    this.setState({
      zipcode: e.target.value,
    });
  }


  onSubmit(e) {
    e.preventDefault();
    console.log("Inside onsubmit");
```

```
    const DeliveryDetails = {

      address: this.state.address,

      city: this.state.city,

      zipcode: this.state.zipcode,


    };



    axios

      .post("http://localhost:8080/API/SpringBoot/Delivery", DeliveryDetails)

      .then((res) => {

        console.log(res.data);
        console.log("Promise Succeed")

      });



    this.setState({

      address: "",

      city: "",

      zipcode: "",


    });
}
render() {
    return (
        <div>
```

```
   <Form onSubmit={this.onSubmit}>

  <Form.Group controlId="formGridAddress1">
    <Form.Label>Address</Form.Label>
    <Form.Control placeholder="Enter your address" value={this.state.address} onC
hange={this.onChangeAddress} />

  </Form.Group>



  <Form.Row>
    <Form.Group as={Col} controlId="formGridCity">
      <Form.Label>City</Form.Label>
      <Form.Control placeholder="Enter your city" value={this.state.city} onChang
e={this.onChangeCity}/>

    </Form.Group>


    <Form.Group as={Col} controlId="formGridZip">
      <Form.Label>Zip</Form.Label>
      <Form.Control placeholder="Enter your zip" value={this.state.zipcode} onCha
nge={this.onChangeZip}/>

    </Form.Group>
  </Form.Row>

  <Form.Group id="formGridCheckbox">
    <Form.Check type="checkbox" label="Check me out" />
  </Form.Group>

  <Button variant="primary" type="submit">
    Submit
  </Button>
</Form>

            </div>
        )
    }
}
```

viewDelivery.js

```
import axios from 'axios';
```

```
import React, { Component } from 'react'
import Table from 'react-bootstrap/Table'
import Button from 'react-bootstrap/Button'


export default class ViewDelivery extends Component {
    constructor(props){
        super(props);
        this.deleteDelivery = this.deleteDelivery.bind(this);
        this.state = {
            DeliveryDetails:[]

        }
    }

    componentDidMount(){
        const url = "http://localhost:8080/API/SpringBoot/GetDetails"
        axios
        .get(url)
        .then((res) => {
            this.setState({DeliveryDetails : res.data});

            console.log("Promise Succeed")
            console.log(res.data)
        })
    }
    deleteDelivery(id){

        axios
        .delete("http://localhost:8080/API/SpringBoot/GetDetails/" + id)
        .then((res) => {
            console.log(res.data)
            this.setState({


                DeliveryDetails:this.state.DeliveryDetails.filter((el) =>el.id !=
= id),

            });
        })

    }

    render() {
        return (
```

```
            <div>
                <Table striped bordered hover>
                <thead>
    <tr>
      <th>#</th>
      <th>Address</th>
      <th>City</th>
      <th>ZipCode</th>

    </tr>
  </thead>
  <tbody>
  {
                        this.state.DeliveryDetails.map(
                            (DeliveryDetail) =>
                            <tr key = {DeliveryDetail.id}>
                                <td>{DeliveryDetail.id}</td>
                                <td>{DeliveryDetail.address}</td>
                                <td>{DeliveryDetail.city}</td>
                                <td>{DeliveryDetail.zipcode}</td>
                                <td><Button variant="success" onClick={ () => thi
s.deleteDelivery(DeliveryDetail.id)}> DeliveryCompleted</Button></td>

                            </tr>
                        )
                    }
  </tbody>
</Table>


            </div>
        )
    }
}
```

AppNavBar.js

```
import React, { Component } from 'react';
import { Collapse, Nav, Navbar, NavbarBrand, NavbarToggler, NavbarText, NavItem,
NavLink } from 'reactstrap';
import { Link } from 'react-router-dom';
import {Button} from 'react-bootstrap'
import { withRouter } from 'react-router-dom';

import AuthenticationService from '../services/AuthenticationService';
```

```
class AppNavbar extends Component {
  constructor(props) {
    super(props);
    this.state = {isOpen: false};
    this.toggle = this.toggle.bind(this);

    this.state = {
      showUser: false,
      showPM: false,
      showAdmin: false,
      username: undefined,
      login: false
    };
  }

  componentDidMount() {
    const user = AuthenticationService.getCurrentUser();

    if (user) {
      const roles = [];

      user.authorities.forEach(authority => {
        roles.push(authority.authority)
      });

      this.setState({
        showUser: true,
        showPM: roles.includes("ROLE_PM") || roles.includes("ROLE_ADMIN"),
        showAdmin: roles.includes("ROLE_ADMIN"),
        login: true,
        username: user.username
      });
    }
  }

  signOut = () => {
    AuthenticationService.signOut();
    this.props.history.push('/home');
    window.location.reload();
  }

  toggle() {
    this.setState({
      isOpen: !this.state.isOpen
```

```
    });
  }

  render() {
    return <Navbar color="dark" dark expand="md">
      <NavbarBrand tag={Link} to="/home">SPRING DIGITAL MART</NavbarBrand>
      <Nav className="mr-auto">
        <NavLink href="/home">Home</NavLink>
        {this.state.showUser && <NavLink href="/user">User</NavLink>}
        {this.state.showPM && <NavLink href="/pm">PM</NavLink>}
        {this.state.showAdmin && <NavLink href="/admin">Admin</NavLink>}
      </Nav>
      <NavbarToggler onClick={this.toggle}/>
      <Collapse isOpen={this.state.isOpen} navbar>
        {
          this.state.login ? (
            <Nav className="ml-auto" navbar>
              <NavItem>
                  <NavbarText>
                    Signed in as: <a href="/profile">{this.state.username}</a>
                  </NavbarText>
              </NavItem>
              <NavItem>
                <NavLink href="#" onClick={this.signOut}>SignOut</NavLink>
              </NavItem>
            </Nav>
          ) : (
            <Nav className="ml-auto" navbar>
              <NavItem>
                <NavLink href="/signin">Login</NavLink>
              </NavItem>
              <NavItem>
                <NavLink href="/signup">SignUp</NavLink>
              </NavItem>

              <ul type="none" className="right">
                      <Link to="/cart" > <Button variant="danger" className="myc
lass">My Cart</Button> </Link>

                    </ul>

            </Nav>

          )
        }
```

```
      </Collapse>
    </Navbar>;
  }
}

export default withRouter(AppNavbar);
```

Login.js

```
import React, { Component } from 'react';

import { Container } from 'reactstrap';
import { Form, Alert, FormGroup, Input, Label, Row, Col } from "reactstrap";
import {Button} from 'react-bootstrap';
import AuthenticationService from "../services/AuthenticationService";
import avatar from '../avatar.png';
import '../App.css';

class Login extends Component {

  constructor(props) {
    super(props);

    this.state = {
      username: "",
      password: "",
      error: ""
    };
  }

  changeHandler = (event) => {
    let nam = event.target.name;
    let val = event.target.value;
    this.setState({[nam]: val});
  }

  doLogin = async (event) => {
    event.preventDefault();

    AuthenticationService
        .signin(this.state.username,
                  this.state.password)
      .then(
        () => {
          this.props.history.push('/profile');
```

```
        },
        error => {
          console.log("Login fail: error = { " + error.toString() + " }");
          this.setState({error: "Can not signin successfully ! Please check usern
ame/password again"});
        }
    );
  }

  render() {
    return (
      <div>

        <Container fluid>
          <Row style={{marginTop:"20px"}}>
          <Col sm="12" md={{ size: 3, offset: 4 }}>
            <div style={{marginBottom: "10px"}}>
              <img src={avatar} alt="Avatar" className="avatar center"
                style={{width: "50%", height: "auto"}}/>
            </div>
            <br/>
            <Form  onSubmit={this.doLogin}>
              <FormGroup>
                <Label for="username"><strong>Username</strong></Label>
                <Input autoFocus
                  type="text"
                  name="username" id="username"
                  value={this.state.username}
                  placeholder="Enter Username"
                  autoComplete="username"
                  onChange={this.changeHandler}
                />
              </FormGroup>
              <br/>
              <FormGroup>
                <Label for="password"><strong>Password</strong></Label>
                <Input type="password"
                  name="password" id="password"
                  value={this.state.password}
                  placeholder="Enter Password"
                  autoComplete="password"
                  onChange={this.changeHandler}
                />
              </FormGroup>
                <br/> <br/>
```

```
              <Button type="submit" variant="primary" size="lg" block>
                Sign In
              </Button>
              {
                this.state.error && (
                  <Alert color="danger">
                    {this.state.error}
                  </Alert>
                )
              }
            </Form>
            </Col>
          </Row>
        </Container>
      </div>);
  }
}

export default Login;
```

SignUp.js

```
import React, { Component } from 'react';
import { Container } from 'reactstrap';
import { Button, Form, FormGroup, Input, Label, Row, Col } from "reactstrap";
import { Alert } from "react-bootstrap"

import Authentication from '../services/AuthenticationService'

const validEmailRegex = RegExp(/^(([^<>()\[\]\.,;:\s@\"]+(\.[^<>()\[\]\.,;:\s@\"]
+)*)|(\".+\"))@(([^<>()[\]\.,;:\s@\"]+\.)+[^<>()[\]\.,;:\s@\"]{2,})$/i);

const validateForm = (errors) => {
  let valid = true;
  Object.values(errors).forEach(
    (val) => val.length > 0 && (valid = false)
  );
  return valid;
}

class SignUp extends Component {

  constructor(props) {
    super(props);
    this.state = {
```

```
    firstname: "",
    lastname: "",
    username: "",
    email: "",
    password: "",
    message: "",
    successful: false,
    validForm: true,
    errors: {
      firstname: '',
      lastname: '',
      username: '',
      email: '',
      password: ''
    }
  };
}

changeHandler = (event) => {
  const { name, value } = event.target;

  let errors = this.state.errors;

  switch (name) {
    case 'firstname':
      errors.firstname =
        value.length < 3
          ? 'FirstName must be 3 characters long!'
          : '';
      break;
    case 'lastname':
      errors.lastname =
        value.length < 3
          ? 'LastName must be 3 characters long!'
          : '';
      break;
    case 'username':
      errors.username =
        value.length < 5
          ? 'Username must be 5 characters long!'
          : '';
      break;
    case 'email':
      errors.email =
        validEmailRegex.test(value)
```

```
            ? ''
            : 'Email is not valid!';
        break;
      case 'password':
        errors.password =
          value.length < 8
            ? 'Password must be 8 characters long!'
            : '';
        break;
      default:
        break;
    }

    this.setState({errors, [name]: value}, ()=> {
        console.log(errors)
    })
  }

  signUp = (e) => {
    e.preventDefault();
    const valid = validateForm(this.state.errors);
    this.setState({validForm: valid});
    if(valid){
      Authentication.register(
        this.state.firstname,
        this.state.lastname,
        this.state.username,
        this.state.email,
        this.state.password
      ).then(
        response => {
          this.setState({
            message: response.data.message,
            successful: true
          });
        },
        error => {
          console.log("Fail! Error = " + error.toString());

          this.setState({
            successful: false,
            message: error.toString()
          });
        }
      );
```

```
  }
}

render() {
  const title = <h2>Register User</h2>;
  const errors = this.state.errors;

  let alert = "";

  if(this.state.message){
    if(this.state.successful){
      alert = (
                <Alert variant="success">
                  {this.state.message}
                </Alert>
              );
    }else{
      alert = (
                <Alert variant="danger">
                  {this.state.message}
                </Alert>
              );
    }
  }

  return (
    <div>

      <Container fluid>
        <Row>
        <Col sm="12" md={{ size: 4, offset: 4 }}>
        {title}
          <Form onSubmit={this.signUp}>
            <FormGroup controlId="forFirstname">
              <Label for="firstname">First Name</Label>
              <Input
                type="text"
                placeholder="Enter First Name"
                name="firstname" id="firstname"
                value={this.state.firstname}
                autoComplete="firstname"
                onChange={this.changeHandler}
              />
              {
                errors.firstname && (
```

```
            <Alert variant="danger">
              {errors.firstname}
            </Alert>
          )
        }
      </FormGroup>

      <FormGroup controlId="forLastname">
        <Label for="lastname">Last Name</Label>
        <Input
          type="text"
          placeholder="Enter Last Name"
          name="lastname" id="lastname"
          value={this.state.lastname}
          autoComplete="lastname"
          onChange={this.changeHandler}
        />
        {
          errors.lastname && (
            <Alert variant="danger">
              {errors.firstname}
            </Alert>
          )
        }
      </FormGroup>

      <FormGroup controlId="forUsername">
        <Label for="username">Username</Label>
        <Input
          type="text"
          placeholder="Enter UserName"
          name="username" id="username"
          value={this.state.username}
          autoComplete="username"
          onChange={this.changeHandler}
        />
        {
          errors.username && (
            <Alert variant="danger">
              {errors.username}
            </Alert>
          )
        }
      </FormGroup>
```

```
<FormGroup controlId="formEmail">
  <Label for="email">Email</Label>
  <Input required
    type="text"
    placeholder="Enter Email"
    name="email" id="email"
    value={this.state.email}
    autoComplete="email"
    onChange={this.changeHandler}
  />
  {
    errors.email && (
        <Alert variant="danger">
          {errors.email}
        </Alert>
      )
  }
</FormGroup>

<FormGroup controlId="formPassword">
  <Label for="password">Password</Label>
  <Input required
    type="password"
    placeholder="Enter Password"
    name="password" id="password"
    value={this.state.password}
    autoComplete="password"
    onChange={this.changeHandler}
  />
  {
    errors.password && (
        <Alert key="errorspassword" variant="danger">
          {errors.password}
        </Alert>
      )
  }
</FormGroup>

<Button variant="primary" type="submit">
  Create
</Button>
{
  !this.state.validForm && (
    <Alert key="validForm" variant="danger">
      Please check the inputs again!
```

```
                </Alert>
              )
            }

            {alert}
          </Form>
          </Col>
        </Row>
      </Container>
    </div>);
  }
}

export default SignUp;
```

AuthendicationService.js

```
import axios from "axios";



class AuthenticationService {
  signin = (username, password) => {
      return axios.post("/API/SpringBoot/auth/signin", {username, password})
        .then(response => {
          if (response.data.accessToken) {
            localStorage.setItem("user", JSON.stringify(response.data));
          }
          return response.data;
        })
        .catch(err => {
          console.log(err);
          throw err;
        });
  }

  signOut() {
    localStorage.removeItem("user");
  }

  register = async(firstname, lastname, username, email, password) => {
    return axios.post("/API/SpringBoot/auth/signup", {
      firstname,
      lastname,
      username,
```

```
      email,
      password
    });
  }

  getCurrentUser() {
    return JSON.parse(localStorage.getItem('user'));;
  }
}

export default new AuthenticationService();
```

BackendService.js

```
import axios from 'axios';

// Add a request interceptor
axios.interceptors.request.use( config => {
  const user = JSON.parse(localStorage.getItem('user'));

  if(user && user.accessToken){
    const token = 'Bearer ' + user.accessToken;
    config.headers.Authorization =  token;
  }

  return config;
});

class BackendService {
  async getUserBoard() {
    return await axios.get("/API/SpringBoot/test/user");
  }

  async getPmBoard() {
    return await axios.get("/API/SpringBoot/test/pm");
  }

  async getAdminBoard() {
    return await axios.get("/API/SpringBoot/test/admin");
  }
}

export default new BackendService();
```

ItemService.js

```
import axios from 'axios';

const Items_API_BASE_URL = "http://localhost:8080/API/SpringBoot/Items";

class ItemService {

    getItemsById(ItemId){
        return axios.get(Items_API_BASE_URL + '/' + ItemId);
    }

    getItems(){
        return axios.get(Items_API_BASE_URL);
    }

    createItems(Item){
        return axios.post(Items_API_BASE_URL, Item);
    }



    updateItems(Item, ItemId){
        return axios.put(Items_API_BASE_URL + '/' + ItemId, Item);
    }

    deleteItems(ItemId){
        return axios.delete(Items_API_BASE_URL + '/' + ItemId);
    }
    searchItems(Item, key){
        return axios.put(Items_API_BASE_URL + '/' + key, Item);
    }
}

export default new ItemService()
```

CartActions.js

```
import { ADD_TO_CART,REMOVE_ITEM,SUB_QUANTITY,ADD_QUANTITY} from '..'



//add cart action
export const addToCart= (id)=>{
```

```
    return{
        type: ADD_TO_CART,

        id
    }
}
//remove item action
export const removeItem=(id)=>{
    return{
        type: REMOVE_ITEM,
        id
    }
}
//subtract qt action
export const subtractQuantity=(id)=>{
    return{
        type: SUB_QUANTITY,
        id
    }
}
//add qt action
export const addQuantity=(id)=>{
    return{
        type: ADD_QUANTITY,
        id
    }
}
```

Index.js

```
//Types should be in const to avoid typos and duplication since it's a string and
 could be easily miss spelled
export const ADD_TO_CART = 'ADD_TO_CART';
export const REMOVE_ITEM = 'REMOVE_ITEM';
export const SUB_QUANTITY = 'SUB_QUANTITY';
export const ADD_QUANTITY = 'ADD_QUANTITY';
export const ADD_SHIPPING = 'ADD_SHIPPING';
```

CartReducer.js

```
import { ADD_TO_CART,REMOVE_ITEM,SUB_QUANTITY,ADD_QUANTITY,ADD_SHIPPING } from '.
./actions/index'
import ItemService from './../services/ItemService'

const initState = {

    Items: [ ],
    addedItems:[],
    total: 0
}
const cartReducer= (state = initState,action)=>{

    //INSIDE HOME COMPONENT
    if(action.type === ADD_TO_CART){
        console.log("reducer");
        console.log(action.id);

          let addedItem = state.Items.find(Item=> Item.id === action.id)
          //check if the action id exists in the addedItems
         let existed_Item= state.addedItems.find(Item=> action.id === Item.id)

         if(existed_Item)
         {
            addedItem.qty += 1
             return{
                ...state,
                 total: state.total + addedItem.price
                  }
        }
         else{
            addedItem.qty = 1;
            //calculating the total
            let newTotal = state.total + addedItem.price
            console.log("added item");

            return{
                ...state,
                addedItems: [...state.addedItems, addedItem],
                total : newTotal
            }

        }
    }
```

```
    if(action.type === REMOVE_ITEM){
        let itemToRemove= state.addedItems.find(Item=> action.id === Item.id)
        let new_items = state.addedItems.filter(Item=> action.id !== Item.id)

        //calculating the total
        let newTotal = state.total - (itemToRemove.price * itemToRemove.qty )
        console.log(itemToRemove)
        return{
            ...state,
            addedItems: new_items,
            total: newTotal
        }
    }
    //INSIDE CART COMPONENT
    if(action.type=== ADD_QUANTITY){
        let addedItem = state.Items.find(Item=> Item.id === action.id)
          addedItem.qty += 1
          let newTotal = state.total + addedItem.price
          return{
              ...state,
              total: newTotal
          }
    }
    if(action.type=== SUB_QUANTITY){
        let addedItem = state.Items.find(Item=> Item.id === action.id)
        //if the qt == 0 then it should be removed
        if(addedItem.qty === 1){
            let new_items = state.addedItems.filter(Item=>Item.id !== action.id)
            let newTotal = state.total - addedItem.price
            return{
                ...state,
                addedItems: new_items,
                total: newTotal
            }
        }
        else {
            addedItem.qty -= 1
            let newTotal = state.total - addedItem.price
            return{
                ...state,
                total: newTotal
            }
        }

    }
```

```
    if(action.type=== ADD_SHIPPING){
          return{
              ...state,
              total: state.total + 6
          }
    }

    if(action.type=== 'SUB_SHIPPING'){
        return{
            ...state,
            total: state.total - 6
        }
  }

  else{
    return state
    }

}

export default cartReducer
```

App.js

```
import React from 'react';
import './App.css';
import {BrowserRouter as Router, Route, Switch} from 'react-router-dom'
import ListItemComponent from './components/ListItemComponent';
//import HeaderComponent from './Login/HeaderComponent';
import CreateItemComponent from './components/CreateItemComponent';
import ViewItemComponent from './components/ViewItemComponent';
import BuyerHome from './Buyer/BuyerHome';
import StripeButton from './Buyer/stripebutton.component';
import AppNavbar from './Login/AppNavbar';
import Login from './Login/Login';
import Signup from './Login/SignUp';
import ProjectManagerPage from './Login/ProjectManagerPage';
import Cart from './Buyer/Cart';
import DeliveryDetails from './Buyer/DeliveryDetails';
import ViewDelDetails from './Buyer/ViewDelivery'

function App() {
  return (
    <div>
```

```
        <Router>
              <AppNavbar />
                    <div className="container">
                    <Switch>

                    <Route path='/' exact={true} component={ViewDelDetails}/>
                    <Route path="/cart" component={Cart}/>
                    <Route path="/DeliveryDetails" component={DeliveryDetails}/>
                    <Route path='/signin' exact={true} component={Login}/>
                    <Route path='/signup' exact={true} component={Signup}/>
                    <Route path='/pm' exact={true} component={ProjectManagerPage}
/>



                   <Route path = "/Stripebutton" exact component = {StripeButton}
></Route>
                   <Route path = "/Items" component = {ListItemComponent}></Route
>
                   <Route path = "/add-
item/:id" component = {CreateItemComponent}></Route>
                   <Route path = "/view-
item/:id" component = {ViewItemComponent}></Route>
                   <Route path = "/ViewDelDetails" component = {ViewDelDetails}><
/Route>



                    </Switch>
                </div>

        </Router>
    </div>

  );
}

export default App;
```

SpringBootBackendApplication.java

```
package com.springboot.project;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import
org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@SpringBootApplication
public class SpringBootBackEndApplication {

	public static void main(String[] args) {
		SpringApplication.run(SpringBootBackEndApplication.class,
args);
	}

}
```

BuyerController.java

```
package com.springboot.project.controller;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
import com.springboot.project.exception.ResourceNotFoundException;
import com.springboot.project.model.Item;
import com.springboot.project.repository.ItemRepository;

@CrossOrigin(origins = "http://localhost:3000")
@RestController
@RequestMapping(value = "/API/SpringBoot/")

public class BuyerController {


	@Autowired
		private ItemRepository ItemRepository;

		// get all Items
		@GetMapping("/Buyer")
		public List<Item> getAllItems(){
			return ItemRepository.findAll();
		}


}
```

ItemController.java

```
package com.springboot.project.controller;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.repository.query.Param;
import org.springframework.http.ResponseEntity;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
```

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import com.springboot.project.exception.ResourceNotFoundException;
import com.springboot.project.model.Item;
import com.springboot.project.repository.ItemRepository;

@CrossOrigin(origins = "http://localhost:3000")
@RestController
@RequestMapping(value = "/API/SpringBoot/")
public class ItemController {

@Autowired
	private ItemRepository ItemRepository;

	// get all Items
	@GetMapping("/Items")
	public List<Item> getAllItems(){
		return ItemRepository.findAll();
	}

	// create Item rest api
	@PostMapping("/Items")
	public Item createItem(@RequestBody Item Item) {
		return ItemRepository.save(Item);
	}

	// get Item by id rest api
	@GetMapping("/Items/{id}")
	public ResponseEntity<Item> getItemById(@PathVariable Long id) {
		Item Item = ItemRepository.findById(id)
				.orElseThrow(() -> new
ResourceNotFoundException("Item not exist with id :" + id));
		return ResponseEntity.ok(Item);
	}
```

```
        // update Item rest api

        @PutMapping("/Items/{id}")
        public ResponseEntity<Item> updateItem(@PathVariable Long id,
    @RequestBody Item ItemDetails){
                Item Item = ItemRepository.findById(id)
                        .orElseThrow(() -> new
    ResourceNotFoundException("Item not exist with id :" + id));

                Item.setItemName(ItemDetails.getItemName());
                Item.setQty(ItemDetails.getQty());
                Item.setPrice(ItemDetails.getPrice());
                Item.setDescription(ItemDetails.getDescription());
                Item.setPath(ItemDetails.getPath());
                Item updatedItem = ItemRepository.save(Item);
                return ResponseEntity.ok(updatedItem);
        }

        // delete Item rest api
        @DeleteMapping("/Items/{id}")
        public ResponseEntity<Map<String, Boolean>>
    deleteItem(@PathVariable Long id){
                Item Item = ItemRepository.findById(id)
                        .orElseThrow(() -> new
    ResourceNotFoundException("Item not exist with id :" + id));

                ItemRepository.delete(Item);
                Map<String, Boolean> response = new HashMap<>();
                response.put("deleted", Boolean.TRUE);
                return ResponseEntity.ok(response);
        }

        @GetMapping("/search/{key}")
      public List<Item> searchItem(@PathVariable("key") String key){
        return ItemRepository.findByNameLike("%"+key+"%");
      }


}
```

ResourceNotFoundException.java

```
package com.springboot.project.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException{

	private static final long serialVersionUID = 1L;

	public ResourceNotFoundException(String message) {
		super(message);
	}
}
```

Item.java

```
package com.springboot.project.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "Items")
public class Item {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private long id;

	@Column(name = "Item_Name")
	private String itemName;
```

```
	@Column(name = "Item_Qty")
	private int qty;

	@Column(name = "Item_Price")
	private double price;

	@Column(name = "Item_Description")
	private String description;

	@Column(name = "Item_Path")
	private String path;



	public Item() {

	}

	public Item(String itemName, int qty, double price, String
description,String path) {
		super();
		this.itemName = itemName;
		this.qty = qty;
		this.price = price;
		this.description = description;
		this.path = path;
	}

	public String getPath() {
		return path;
	}

	public void setPath(String path) {
		this.path = path;
	}
	public long getId() {
		return id;
	}

	public void setId(long id) {
```

```
        this.id = id;
    }

    public String getItemName() {
        return itemName;
    }

    public void setItemName(String itemName) {
        this.itemName = itemName;
    }

    public int getQty() {
        return qty;
    }

    public void setQty(int qty) {
        this.qty = qty;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }



}
```

Itemrepository.java

```
package com.springboot.project.repository;

import java.util.List;

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

import com.springboot.project.model.Item;

public interface ItemRepository extends JpaRepository<Item, Long> {

	@Query(value = "SELECT * FROM mydb.items where "
			+ "match(item_description, item_name) "
			+ "against(?1);", nativeQuery = true)
	List<Item> findByNameLike(String key);

}
```

PaymentController.java

```
package com.springmart.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.springmart.model.PaymentRequest;
import com.springmart.service.PaymentService;
import com.stripe.exception.StripeException;
import com.stripe.model.Charge;
```

```
@RestController
@RequestMapping("/payment")
public class PaymentController {

	@Autowired
	PaymentService service;

	@PostMapping
	public ResponseEntity<String> completePayment(@RequestBody
PaymentRequest request) throws StripeException {
		String chargeId= service.charge(request);
		return chargeId!=null? new
ResponseEntity<String>(chargeId,HttpStatus.OK):
			new ResponseEntity<String>("Please check the credit
card details entered",HttpStatus.BAD_REQUEST);
	}

	@ExceptionHandler
	public String handleError(StripeException ex) {
		return ex.getMessage();
	}
}
```

PaymentRequest.java

```
package com.springmart.model;

public class PaymentRequest {

	public enum Currency{
		INR,USD;
	}
	   private String description;
	    private int amount;
	    private Currency currency;
	    private String stripeEmail;
	    private Token token;

		public String getDescription() {
			return description;
		}
		public void setDescription(String description) {
			this.description = description;
```

```
	}
	public int getAmount() {
		return amount;
	}
	public void setAmount(int amount) {
		this.amount = amount;
	}
	public Currency getCurrency() {
		return currency;
	}
	public void setCurrency(Currency currency) {
		this.currency = currency;
	}
	public String getStripeEmail() {
		return stripeEmail;
	}
	public void setStripeEmail(String stripeEmail) {
		this.stripeEmail = stripeEmail;
	}
	public Token getToken() {
		return token;
	}
	public void setToken(Token stripeToken) {
		this.token = stripeToken;
	}


}
```

Token.java

```
package com.springmart.model;

public class Token {

	private String id;

	public String getId() {
		return id;
	}

	public void setId(String id) {
		this.id = id;
	}

}
```

PaymentApplication.java

package com.springmart.payment;

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import
org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@SpringBootApplication
public class PaymentApplication {

	public static void main(String[] args) {
		SpringApplication.run(PaymentApplication.class,args);
	}


	@Bean
	public WebMvcConfigurer corsConfigurer() {
		return new WebMvcConfigurer() {
			public void addCorsMappings(CorsRegistry registry) {

	registry.addMapping("/payment").allowedOrigins("http://localhost:30
00");
			}
		};
	}

}
```

PaymentService.java

```
package com.springmart.service;

import java.util.HashMap;
import java.util.Map;

import javax.annotation.PostConstruct;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
```

```
import com.springmart.model.PaymentRequest;
import com.stripe.Stripe;
import com.stripe.exception.StripeException;
import com.stripe.model.Charge;

@Service
public class PaymentService {

    @Value("${STRIPE_SECRET_KEY}")
    private String secretKey;

   @PostConstruct
   public void init() {
      Stripe.apiKey = secretKey;
   }
     public String charge(PaymentRequest chargeRequest) throws
StripeException {
          Map<String, Object> chargeParams = new HashMap<>();
        chargeParams.put("amount", chargeRequest.getAmount());
        chargeParams.put("currency", PaymentRequest.Currency.INR);
        chargeParams.put("source", chargeRequest.getToken().getId());

          Charge charge = Charge.create(chargeParams);
          return charge.getId();
     }
}
```

DeliveryApplication.java

```
package com.DeliveryService.Delivery;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DeliveryApplication {
```

```
	public static void main(String[] args) {
		SpringApplication.run(DeliveryApplication.class, args);
	}

}
```

DeliveryController.java

```
package com.DeliveryService.Delivery.Controller;


import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.DeliveryService.Delivery.Controller.ResourceNotFoundException;
import com.DeliveryService.Delivery.Controller.DeliveryDetail;


@RestController
@RequestMapping(value = "/API/SpringBoot")
@CrossOrigin("*")
public class DeliveryController {

	@Autowired
	private DeliveryRepository DeliveryRepository;

	@GetMapping("/GetDetails")
	public List<DeliveryDetail> getAllItems(){
		return DeliveryRepository.findAll();
	}


	@DeleteMapping("/GetDetails/{id}")
	public ResponseEntity<Map<String, Boolean>> deleteDelivery(@PathVariable Long
id){

		DeliveryDetail Detail = DeliveryRepository.findById(id).orElseThrow(() -
> new ResourceNotFoundException("Item not exist with id :" + id));
```

```
            DeliveryRepository.delete(Detail);
            Map<String, Boolean> response = new HashMap<>();
            response.put("deleted", Boolean.TRUE);
            return ResponseEntity.ok(response);
      }

            @PostMapping("/Delivery")
            public String createDeliveryDetail(@RequestBody DeliveryDetail
DeliveryDetail) {
                  DeliveryRepository.save(DeliveryDetail);
                  return "success";
            }

}
```

DeliveryDetail.java

```
package com.DeliveryService.Delivery.Controller;


import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "delivery")
public class DeliveryDetail {

      @Id
      @GeneratedValue(strategy = GenerationType.IDENTITY)
      private long id;

      @Column(name = "delivery_address")
      private String address;

      @Column(name = "delivery_city")
      private String city;

      @Column(name = "zip")
```

```
	private int zipcode;



	public DeliveryDetail() {
		super();
		// TODO Auto-generated constructor stub
	}

	public DeliveryDetail(long id, String address, String city, int zipcode)
{
		super();
		this.id = id;
		this.address = address;
		this.city = city;
		this.zipcode = zipcode;
	}

	public long getId() {
		return id;
	}

	public void setId(long id) {
		this.id = id;
	}

	public String getAddress() {
		return address;
	}

	public void setAddress(String address) {
		this.address = address;
	}

	public String getCity() {
		return city;
	}

	public void setCity(String city) {
```

```
        this.city = city;
    }

    public int getZipcode() {
        return zipcode;
    }

    public void setZipcode(int zipcode) {
        this.zipcode = zipcode;
    }

}
```

DeliveryRepository.java

```
package com.DeliveryService.Delivery.Controller;

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


public interface DeliveryRepository extends JpaRepository<DeliveryDetail, Long>{

}
```

SpringBootJwtAuthenticationExamplesApplication.java

```
package com.springmart.jwtauthentication;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootJwtAuthenticationExamplesApplication {

    public static void main(String[] args) {

    SpringApplication.run(SpringBootJwtAuthenticationExamplesApplication.c
lass, args);
```

```
        }

}
```

AuthRestAPIs.java

```
package com.springmart.jwtauthentication.controller;

import java.util.HashSet;
import java.util.Set;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationTo
ken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.springmart.jwtauthentication.message.request.LoginForm;
import com.springmart.jwtauthentication.message.request.SignUpForm;
import com.springmart.jwtauthentication.message.response.JwtResponse;
import com.springmart.jwtauthentication.message.response.ResponseMessage;
import com.springmart.jwtauthentication.model.Role;
import com.springmart.jwtauthentication.model.RoleName;
import com.springmart.jwtauthentication.model.User;
import com.springmart.jwtauthentication.repository.RoleRepository;
```

```
import com.springmart.jwtauthentication.repository.UserRepository;
import com.springmart.jwtauthentication.security.jwt.JwtProvider;

@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
@RequestMapping("/API/SpringBoot/auth")
public class AuthRestAPIs {

	@Autowired
	AuthenticationManager authenticationManager;

	@Autowired
	UserRepository userRepository;

	@Autowired
	RoleRepository roleRepository;

	@Autowired
	PasswordEncoder encoder;

	@Autowired
	JwtProvider jwtProvider;

	@PostMapping("/signin")
	public ResponseEntity<?> authenticateUser(@Valid @RequestBody
LoginForm loginRequest) {

		Authentication authentication = authenticationManager.authenticate(
				new
UsernamePasswordAuthenticationToken(loginRequest.getUsername(),
loginRequest.getPassword()));


	SecurityContextHolder.getContext().setAuthentication(authentication);

		String jwt = jwtProvider.generateJwtToken(authentication);
		UserDetails userDetails = (UserDetails) authentication.getPrincipal();

		return ResponseEntity.ok(new JwtResponse(jwt,
userDetails.getUsername(), userDetails.getAuthorities()));
```

```
	}

	@PostMapping("/signup")
	public ResponseEntity<?> registerUser(@Valid @RequestBody
SignUpForm signUpRequest) {
		if
(userRepository.existsByUsername(signUpRequest.getUsername())) {
			return new ResponseEntity<>(new ResponseMessage("Fail ->
Username is already taken!"),
					HttpStatus.BAD_REQUEST);
		}

		if (userRepository.existsByEmail(signUpRequest.getEmail())) {
			return new ResponseEntity<>(new ResponseMessage("Fail ->
Email is already in use!"),
					HttpStatus.BAD_REQUEST);
		}

		// Creating user's account
		User user = new User(signUpRequest.getFirstname(),
signUpRequest.getLastname(),
							signUpRequest.getUsername(),
signUpRequest.getEmail(),
				encoder.encode(signUpRequest.getPassword()));

		Set<Role> roles = new HashSet<>();
		if(signUpRequest.getRole() != null) {
			Set<String> strRoles = signUpRequest.getRole();

			strRoles.forEach(role -> {
				switch (role) {
				case "admin":
					Role adminRole =
roleRepository.findByName(RoleName.ROLE_ADMIN)
							.orElseThrow(() -> new
RuntimeException("Fail! -> Cause: User Role not find."));
					roles.add(adminRole);

					break;
```

```
                        default:
                            Role userRole =
roleRepository.findByName(RoleName.ROLE_USER)
                                    .orElseThrow(() -> new
RuntimeException("Fail! -> Cause: User Role not find."));
                            roles.add(userRole);
                        }
                    });
                } else {
                    // default mode : User register
                    Role userRole =
roleRepository.findByName(RoleName.ROLE_USER)
                            .orElseThrow(() -> new RuntimeException("Fail! -
> Cause: User Role not find."));
                    roles.add(userRole);
                }

                user.setRoles(roles);
                userRepository.save(user);

                return new ResponseEntity<>(new ResponseMessage("User "+
signUpRequest.getFirstname() + " is registered successfully!"), HttpStatus.OK);
        }
}
```

TestRestAPIs.java

```
package com.springmart.jwtauthentication.controller;

import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
public class TestRestAPIs {
```

```
	@GetMapping("/API/SpringBoot/test/user")
	@PreAuthorize("hasRole('USER') or hasRole('ADMIN')")
	public String userAccess() {
		return ">>> User Contents!";
	}



	@GetMapping("/API/SpringBoot/test/admin")
	@PreAuthorize("hasRole('ADMIN')")
	public String adminAccess() {
		return ">>> Admin Contents";
	}
}
```

LoginForm.java

```
package com.springmart.jwtauthentication.message.request;

import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Size;

public class LoginForm {
    @NotBlank
    @Size(min=3, max = 60)
    private String username;

    @NotBlank
    @Size(min = 6, max = 40)
    private String 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;
    }
}
```

SignUpForm.java

```
package com.springmart.jwtauthentication.message.request;

import java.util.Set;

import javax.validation.constraints.*;

public class SignUpForm {
    @NotBlank
    @Size(min = 3, max = 50)
    private String firstname;

    @NotBlank
    @Size(min = 3, max = 50)
    private String lastname;

    @NotBlank
    @Size(min = 3, max = 50)
    private String username;

    @NotBlank
    @Size(max = 60)
    @Email
    private String email;

    private Set<String> role;
```

```
@NotBlank
@Size(min = 6, max = 40)
private String password;

public String getFirstname() {
   return firstname;
}

public void setFirstname(String firstname) {
   this.firstname = firstname;
}

public String getLastname() {
   return lastname;
}

public void setLastname(String lastname) {
   this.lastname = lastname;
}

public String getUsername() {
   return username;
}

public void setUsername(String username) {
   this.username = username;
}

public String getEmail() {
   return email;
}

public void setEmail(String email) {
   this.email = email;
}

public String getPassword() {
   return password;
}
```

```
    public void setPassword(String password) {
       this.password = password;
    }

    public Set<String> getRole() {
         return this.role;
    }

    public void setRole(Set<String> role) {
         this.role = role;
    }
}
```

JwtResponse.java

```
package com.springmart.jwtauthentication.message.response;

import java.util.Collection;

import org.springframework.security.core.GrantedAuthority;

public class JwtResponse {
        private String token;
        private String type = "Bearer";
        private String username;
        private Collection<? extends GrantedAuthority> authorities;

        public JwtResponse(String accessToken, String username, Collection<?
extends GrantedAuthority> authorities) {
                this.token = accessToken;
                this.username = username;
                this.authorities = authorities;
        }

        public String getAccessToken() {
                return token;
        }

        public void setAccessToken(String accessToken) {
```

```
        this.token = accessToken;
    }

    public String getTokenType() {
        return type;
    }

    public void setTokenType(String tokenType) {
        this.type = tokenType;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

  public Collection<? extends GrantedAuthority> getAuthorities() {
    return authorities;
  }
}
```

ResponseMessage.java

```
package com.springmart.jwtauthentication.message.response;

public class ResponseMessage {
	private String message;

	public ResponseMessage(String message) {
		this.message = message;
	}

	public String getMessage() {
		return message;
	}

	public void setMessage(String message) {
		this.message = message;
	}
}
```

Role.java

```
package com.springmart.jwtauthentication.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

import org.hibernate.annotations.NaturalId;

@Entity
@Table(name = "roles")
public class Role {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   @Enumerated(EnumType.STRING)
   @NaturalId
   @Column(length = 60)
   private RoleName name;

   public Role() {}

   public Role(RoleName name) {
      this.name = name;
   }

   public Long getId() {
      return id;
   }

   public void setId(Long id) {
      this.id = id;
   }
```

```
    public RoleName getName() {
        return name;
    }

    public void setName(RoleName name) {
        this.name = name;
    }
}
```

RoleName.java

```
package com.springmart.jwtauthentication.model;

public enum  RoleName {
    ROLE_USER,
    ROLE_ADMIN
}
```

User.java

```
package com.springmart.jwtauthentication.model;

import java.util.HashSet;
import java.util.Set;

import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;
import javax.validation.constraints.Email;
```

```
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Size;

import org.hibernate.annotations.NaturalId;

@Entity
@Table(name = "users", uniqueConstraints = {
        @UniqueConstraint(columnNames = {
            "username"
        }),
        @UniqueConstraint(columnNames = {
            "email"
        })
})
public class User{
          @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank
    @Size(min=3, max = 50)
    private String firstname;

    @NotBlank
    @Size(min=3, max = 50)
    private String lastname;

    @NotBlank
    @Size(min=3, max = 50)
    private String username;

    @NaturalId
    @NotBlank
    @Size(max = 50)
    @Email
    private String email;

    @NotBlank
    @Size(min=6, max = 100)
    private String password;
```

```
@ManyToMany(fetch = FetchType.LAZY)
@JoinTable(name = "user_roles",
      joinColumns = @JoinColumn(name = "user_id"),
      inverseJoinColumns = @JoinColumn(name = "role_id"))
private Set<Role> roles = new HashSet<>();

public User() {}

public User(String firstname, String lastname,
                            String username, String email, String password) {
   this.firstname = firstname;
   this.lastname = lastname;
   this.username = username;
   this.email = email;
   this.password = password;
}

public Long getId() {
   return id;
}

public void setId(Long id) {
   this.id = id;
}

public String getUsername() {
   return username;
}

public void setUsername(String username) {
   this.username = username;
}

public String getFirstname() {
   return firstname;
}

public void setFirstname(String firstname) {
   this.firstname = firstname;
```

```
    }

    public String getLastname() {
        return lastname;
    }

    public void setLastname(String lastname) {
        this.lastname = lastname;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Set<Role> getRoles() {
        return roles;
    }

    public void setRoles(Set<Role> roles) {
        this.roles = roles;
    }
}
```

RoleRepository.java

```
package com.springmart.jwtauthentication.repository;
```

```
import java.util.Optional;

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

import com.springmart.jwtauthentication.model.Role;
import com.springmart.jwtauthentication.model.RoleName;

@Repository
public interface RoleRepository extends JpaRepository<Role, Long> {
   Optional<Role> findByName(RoleName roleName);
}
```

UserRepository.java

```
package com.springmart.jwtauthentication.repository;

import java.util.Optional;

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

import com.springmart.jwtauthentication.model.User;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
   Optional<User> findByUsername(String username);
   Boolean existsByUsername(String username);
   Boolean existsByEmail(String email);
}
```

WebSecurityConfig.java

```
package com.springmart.jwtauthentication.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.builders.Authentica
tionManagerBuilder;
import
org.springframework.security.config.annotation.method.configuration.EnableGlob
alMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSec
urity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityCo
nfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticatio
nFilter;

import com.springmart.jwtauthentication.security.jwt.JwtAuthEntryPoint;
import com.springmart.jwtauthentication.security.jwt.JwtAuthTokenFilter;
import
com.springmart.jwtauthentication.security.services.UserDetailsServiceImpl;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(
                prePostEnabled = true
)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
   @Autowired
   UserDetailsServiceImpl userDetailsService;

   @Autowired
   private JwtAuthEntryPoint unauthorizedHandler;

   @Bean
   public JwtAuthTokenFilter authenticationJwtTokenFilter() {
```

```
        return new JwtAuthTokenFilter();
    }

    @Override
    public void configure(AuthenticationManagerBuilder
authenticationManagerBuilder) throws Exception {
        authenticationManagerBuilder
                .userDetailsService(userDetailsService)
                .passwordEncoder(passwordEncoder());
    }

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception
{
        return super.authenticationManagerBean();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.cors().and().csrf().disable().
                authorizeRequests()
                .antMatchers("/API/SpringBoot/auth/**").permitAll()
                .anyRequest().authenticated()
                .and()

.exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS
);

        http.addFilterBefore(authenticationJwtTokenFilter(),
UsernamePasswordAuthenticationFilter.class);
    }
}
```

JwtAuthEntryPoint.java

```
package com.springmart.jwtauthentication.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.builders.Authentica
tionManagerBuilder;
import
org.springframework.security.config.annotation.method.configuration.EnableGlob
alMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSec
urity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityCo
nfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticatio
nFilter;

import com.springmart.jwtauthentication.security.jwt.JwtAuthEntryPoint;
import com.springmart.jwtauthentication.security.jwt.JwtAuthTokenFilter;
import
com.springmart.jwtauthentication.security.services.UserDetailsServiceImpl;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(
                prePostEnabled = true
)
```

```
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
   @Autowired
   UserDetailsServiceImpl userDetailsService;

   @Autowired
   private JwtAuthEntryPoint unauthorizedHandler;

   @Bean
   public JwtAuthTokenFilter authenticationJwtTokenFilter() {
      return new JwtAuthTokenFilter();
   }

   @Override
   public void configure(AuthenticationManagerBuilder
authenticationManagerBuilder) throws Exception {
      authenticationManagerBuilder
            .userDetailsService(userDetailsService)
            .passwordEncoder(passwordEncoder());
   }

   @Bean
   @Override
   public AuthenticationManager authenticationManagerBean() throws Exception
{
      return super.authenticationManagerBean();
   }

   @Bean
   public PasswordEncoder passwordEncoder() {
      return new BCryptPasswordEncoder();
   }

   @Override
   protected void configure(HttpSecurity http) throws Exception {
      http.cors().and().csrf().disable().
            authorizeRequests()
            .antMatchers("/API/SpringBoot/auth/**").permitAll()
            .anyRequest().authenticated()
            .and()
```

```
.exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS
);

      http.addFilterBefore(authenticationJwtTokenFilter(),
UsernamePasswordAuthenticationFilter.class);
   }
}
```

JwtAuthTokenFilter.java

```
package com.springmart.jwtauthentication.security.jwt;

import java.io.IOException;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationTo
ken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.web.authentication.WebAuthenticationDetailsSource
;
import org.springframework.web.filter.OncePerRequestFilter;

import
com.springmart.jwtauthentication.security.services.UserDetailsServiceImpl;

public class JwtAuthTokenFilter extends OncePerRequestFilter {
```

```
	@Autowired
	private JwtProvider tokenProvider;

	@Autowired
	private UserDetailsServiceImpl userDetailsService;

	private static final Logger logger =
LoggerFactory.getLogger(JwtAuthTokenFilter.class);

	@Override
	protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain)
			throws ServletException, IOException {
		try {

			String jwt = getJwt(request);
			if (jwt != null && tokenProvider.validateJwtToken(jwt)) {
				String username =
tokenProvider.getUserNameFromJwtToken(jwt);

				UserDetails userDetails =
userDetailsService.loadUserByUsername(username);
				UsernamePasswordAuthenticationToken authentication =
new UsernamePasswordAuthenticationToken(
						userDetails, null,
userDetails.getAuthorities());
				authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));


	SecurityContextHolder.getContext().setAuthentication(authentication);
			}
		} catch (Exception e) {
			logger.error("Can NOT set user authentication -> Message: {}",
e);
		}

		filterChain.doFilter(request, response);
	}
```

```
	private String getJwt(HttpServletRequest request) {
		String authHeader = request.getHeader("Authorization");

		if (authHeader != null && authHeader.startsWith("Bearer ")) {
			return authHeader.replace("Bearer ", "");
		}

		return null;
	}
}
```

JwtProvider.java

```
package com.springmart.jwtauthentication.security.jwt;

import java.util.Date;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Component;

import com.springmart.jwtauthentication.security.services.UserPrinciple;

import io.jsonwebtoken.ExpiredJwtException;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.MalformedJwtException;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.SignatureException;
import io.jsonwebtoken.UnsupportedJwtException;

@Component
public class JwtProvider {

   private static final Logger logger =
LoggerFactory.getLogger(JwtProvider.class);
```

```
    @Value("${loizenai.app.jwtSecret}")
    private String jwtSecret;

    @Value("${loizenai.app.jwtExpiration}")
    private int jwtExpiration;

    public String generateJwtToken(Authentication authentication) {

        UserPrinciple userPrincipal = (UserPrinciple) authentication.getPrincipal();

        return Jwts.builder()
                            .setSubject((userPrincipal.getUsername()))
                            .setIssuedAt(new Date())
                            .setExpiration(new Date((new Date()).getTime() +
jwtExpiration*1000))
                            .signWith(SignatureAlgorithm.HS512, jwtSecret)
                            .compact();
    }

    public boolean validateJwtToken(String authToken) {
        try {
            Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(authToken);
            return true;
        } catch (SignatureException e) {
            logger.error("Invalid JWT signature -> Message: {} ", e);
        } catch (MalformedJwtException e) {
            logger.error("Invalid JWT token -> Message: {}", e);
        } catch (ExpiredJwtException e) {
            logger.error("Expired JWT token -> Message: {}", e);
        } catch (UnsupportedJwtException e) {
            logger.error("Unsupported JWT token -> Message: {}", e);
        } catch (IllegalArgumentException e) {
            logger.error("JWT claims string is empty -> Message: {}", e);
        }

        return false;
    }

    public String getUserNameFromJwtToken(String token) {
        return Jwts.parser()
```

```
                .setSigningKey(jwtSecret)
                .parseClaimsJws(token)
                .getBody().getSubject();
    }
}
```

UserDetailsServiceImpl.java

```
package com.springmart.jwtauthentication.security.services;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.springmart.jwtauthentication.model.User;
import com.springmart.jwtauthentication.repository.UserRepository;

@Service
public class UserDetailsServiceImpl implements UserDetailsService {

    @Autowired
    UserRepository userRepository;

    @Override
    @Transactional
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {

        User user =
userRepository.findByUsername(username).orElseThrow(
                () -> new UsernameNotFoundException("User Not
Found with -> username or email : " + username));

        return UserPrinciple.build(user);
    }
```

}

UserPrinciple.java

```
package com.springmart.jwtauthentication.security.services;

import java.util.Collection;
import java.util.List;
import java.util.Objects;
import java.util.stream.Collectors;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.springmart.jwtauthentication.model.User;

public class UserPrinciple implements UserDetails {
	private static final long serialVersionUID = 1L;

	private Long id;

    private String username;

    private String email;

    @JsonIgnore
    private String password;

    private Collection<? extends GrantedAuthority> authorities;

    public UserPrinciple(Long id, String username, String email, String password,
                                        Collection<? extends GrantedAuthority>
authorities) {
        this.id = id;
        this.username = username;
        this.email = email;
        this.password = password;
        this.authorities = authorities;
```

```
}

public static UserPrinciple build(User user) {
   List<GrantedAuthority> authorities = user.getRoles().stream().map(role ->
         new SimpleGrantedAuthority(role.getName().name())
   ).collect(Collectors.toList());

   return new UserPrinciple(
         user.getId(),
         user.getUsername(),
         user.getEmail(),
         user.getPassword(),
         authorities
   );
}

public Long getId() {
   return id;
}

public String getEmail() {
   return email;
}

@Override
public String getUsername() {
   return username;
}

@Override
public String getPassword() {
   return password;
}

@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
   return authorities;
}

@Override
```

```
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        UserPrinciple user = (UserPrinciple) o;
        return Objects.equals(id, user.id);
    }
}
```