**Source Code**

Capstone Project - Medicare

Programmer: **Luis A. Silva Soto**

**Backend**

**com.controller package**

**AccountController**

package com.controller;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import com.medicare.Account;

import com.medicare.Login;

import com.service.AccountService;

@RestController

@RequestMapping("/account")

public class AccountController {

@Autowired

private AccountService accountService;

@PostMapping("/create")

public ResponseEntity<String> createAccount(@RequestBody Account account) {

String result = accountService.createAccount(account);

return ResponseEntity.ok(result);

}

@GetMapping("/amount")

public ResponseEntity<Float> getAccountAmount(@RequestParam String email) {

float amount = accountService.getAccountAmount(email);

return ResponseEntity.ok(amount);

}

@PutMapping("/modify")

public ResponseEntity<Float> modifyAccountAmount(@RequestParam String email, @RequestParam float newAmount) {

float modifiedAmount = accountService.modifyAccountAmount(email, newAmount);

return ResponseEntity.ok(modifiedAmount);

}

@PutMapping("/topUp")

public ResponseEntity<String> modifyAccountTopUp(@RequestBody Map<String, Object> requestMap) {

String email = (String) requestMap.get("email");

float newAmount = Float.parseFloat(requestMap.get("newAmount").toString());

accountService.modifyAccountAmount(email, newAmount);

return ResponseEntity.ok("Top Up Successfull");

}

}

**LoginController**

package com.controller;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.MediaType;

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.medicare.Login;

import com.service.LoginService;

@RestController

@RequestMapping("Login")

@CrossOrigin

public class LoginController {

@Autowired

LoginService loginService;

@PostMapping(value="signIn",consumes=MediaType.APPLICATION\_JSON\_VALUE)

public String signIn(@RequestBody Login login) {

return loginService.signIn(login);

}

@PostMapping(value="signUp",consumes=MediaType.APPLICATION\_JSON\_VALUE)

public String signUp(@RequestBody Login login) {

return loginService.signUp(login);

}

}

**MedicineController**

package com.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.MediaType;

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.RequestMethod;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

import com.medicare.Medicine;

import com.service.MedicineService;

@RestController

@RequestMapping("Medicine")

@CrossOrigin(origins = "http://localhost:3000")

public class MedicineController {

@Autowired

MedicineService medicineService;

@PostMapping(value="addMedicine",consumes=MediaType.APPLICATION\_JSON\_VALUE)

public String addMedicine(@RequestBody Medicine medicine) {

return medicineService.createMedicine(medicine);

}

@GetMapping(value = "viewAllMedicine",produces = MediaType.APPLICATION\_JSON\_VALUE)

public List<Medicine> getAllMedicines() {

return medicineService.findAllMedicine();

}

@PutMapping("/{medicineid}")

public ResponseEntity<String> updateProduct(@PathVariable int medicineid, @RequestBody Medicine updatedMedicine) {

medicineService.updateProduct(medicineid, updatedMedicine);

return ResponseEntity.ok("Product updated successfully");

}

@DeleteMapping("/{medicineid}")

public ResponseEntity<String> deleteMedicine(@PathVariable int medicineid) {

medicineService.deleteMedicine(medicineid);

return ResponseEntity.ok("Medicine deleted successfully");

}

@GetMapping(value = "searchMedicine")

public List<Medicine> searchMedicine(@RequestParam String query) {

return medicineService.searchMedicine(query);

}

@PutMapping("/enable/{medicineid}")

public ResponseEntity<String> enableMedicine(@PathVariable int medicineid) {

String result = medicineService.enableMedicine(medicineid);

if (result.equals("Medicine enabled")) {

return ResponseEntity.ok(result);

} else {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body(result);

}

}

@PutMapping("/disable/{medicineid}")

public ResponseEntity<String> disableMedicine(@PathVariable int medicineid) {

String result = medicineService.disableMedicine(medicineid);

if (result.equals("Medicine disabled")) {

return ResponseEntity.ok(result);

} else {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body(result);

}

}

@GetMapping("/{id}")

public ResponseEntity<Medicine> getMedicineById(@PathVariable int id) {

Medicine medicine = medicineService.findMedicineById(id);

if (medicine != null) {

return new ResponseEntity<>(medicine, HttpStatus.OK);

} else {

return new ResponseEntity<>(HttpStatus.NOT\_FOUND);

}

}

@PutMapping("/updateInventory/{medicineId}")

public ResponseEntity<Medicine> updateInventory(@PathVariable int medicineId, @RequestParam int newInventory) {

try {

Medicine updatedMedicine = medicineService.updateInventory(medicineId, newInventory);

return ResponseEntity.ok(updatedMedicine);

} catch (Exception e) {

return ResponseEntity.badRequest().build();

}

}

}

**OrdersController**

Note: There are a lot of functions commented out, this where explorative implementation I have performed, decided to leave it in the code to show effort.

package com.controller;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import com.repository.AccountRepository;

import com.service.OrdersService;

import com.medicare.\*;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;

import java.util.List;

import java.util.stream.Collectors;

@RestController

@RequestMapping("/orders")

public class OrdersController {

private final OrdersService orderService;

private final AccountRepository accountRepository;

@Autowired

public OrdersController(OrdersService orderService, AccountRepository accountRepository) {

this.orderService = orderService;

this.accountRepository = accountRepository;

}

@PostMapping("/checkout")

public ResponseEntity<String> checkout(@RequestBody CheckOutRequest checkoutRequest) {

String userEmail = checkoutRequest.getUserEmail();

Account account = accountRepository.findByEmailid(userEmail);

if (account == null) {

return ResponseEntity.status(HttpStatus.BAD\_REQUEST).body("Account not found with email: " + userEmail);

}

try {

Orders order = orderService.createOrder(account, checkoutRequest.getMedicines(), checkoutRequest.getTotalCost());

// Deduct the total cost from the account's balance

float updatedBalance = account.getAmount() - checkoutRequest.getTotalCost();

account.setAmount(updatedBalance);

accountRepository.save(account);

return ResponseEntity.ok("Order created successfully. Order ID: " + order.getOrderid());

} catch (Exception e) {

return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body("Error creating order: " + e.getMessage());

}

}

// @PostMapping("/checkout")

// public ResponseEntity<String> checkout(@RequestBody CheckOutRequest checkoutRequest) {

// try {

// String userEmail = checkoutRequest.getUserEmail();

// Account account = accountRepository.findByEmailid(userEmail);

// if (account == null) {

// return ResponseEntity.status(HttpStatus.BAD\_REQUEST).body("Account not found with email: " + userEmail);

// }

//

// Orders order = orderService.createOrder(account, checkoutRequest.getMedicines(), checkoutRequest.getTotalCost());

//

// // Deduct the total cost from the account's balance

// float updatedBalance = account.getAmount() - checkoutRequest.getTotalCost();

// account.setAmount(updatedBalance);

// accountRepository.save(account);

//

// return ResponseEntity.ok("Order created successfully. Order ID: " + order.getOrderid());

// } catch (Exception e) {

// return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body("Error creating order: " + e.getMessage());

// }

// }

// @PostMapping("/checkout")

// public ResponseEntity<String> checkout(@RequestBody CheckOutRequest checkoutRequest) {

// try {

// String userEmail = checkoutRequest.getUserEmail();

// Account account = accountRepository.findByEmailid(userEmail);

// if (account == null) {

// return ResponseEntity.status(HttpStatus.BAD\_REQUEST).body("Account not found with email: " + userEmail);

// }

//

// Orders order = orderService.createOrder(account, checkoutRequest.getMedicines(), checkoutRequest.getTotalCost());

//

// // Deduct the total cost from the account's balance

// float updatedBalance = account.getAmount() - checkoutRequest.getTotalCost();

// account.setAmount(updatedBalance);

// accountRepository.save(account);

//

// return ResponseEntity.ok("Order created successfully. Order ID: " + order.getOrderid());

// } catch (Exception e) {

// return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body("Error creating order: " + e.getMessage());

// }

//}

// @GetMapping("/history")

// public ResponseEntity<List<OrderResponse>> getOrderHistory(@RequestParam String email) {

// try {

// List<Orders> orders = orderService.getOrderHistory(email);

//

// List<OrderResponse> orderResponses = orders.stream()

// .map(this::convertToOrderResponse)

// .collect(Collectors.toList());

//

// return ResponseEntity.ok(orderResponses);

// } catch (Exception e) {

// List<OrderResponse> emptyList = new ArrayList<>();

// return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body(emptyList);

// }

// }

// // Convert Order to OrderResponse

// private OrderResponse convertToOrderResponse(Orders order) {

// OrderResponse orderResponse = new OrderResponse();

// orderResponse.setId(order.getOrderid());

// orderResponse.setEmailId(order.getEmailid());

// orderResponse.setTotalCost(order.getTotalCost());

// orderResponse.setOrderDate(order.getOrderDate());

//

// List<OrderItemResponse> orderItemResponses = order.getItems().stream()

// .map(this::convertToOrderItemResponse)

// .collect(Collectors.toList());

//

// orderResponse.setItems(orderItemResponses);

//

// return orderResponse;

// }

//

// private OrderItemResponse convertToOrderItemResponse(OrderItem orderItem) {

// OrderItemResponse orderItemResponse = new OrderItemResponse();

// orderItemResponse.setId(orderItem.getId());

// orderItemResponse.setMedicineId(orderItem.getMedicineId());

// orderItemResponse.setQuantity(orderItem.getQuantity());

// return orderItemResponse;

// }

// @GetMapping("/history")

// public ResponseEntity<List<OrderResponse>> getOrderHistory(@RequestParam String email) {

// try {

// List<Orders> orders = orderService.getOrderHistory(email);

//

// List<OrderResponse> orderResponses = orders.stream()

// .map(this::convertToOrderResponse)

// .collect(Collectors.toList());

//

// return ResponseEntity.ok(orderResponses);

// } catch (Exception e) {

// return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body(Collections.emptyList());

// }

// }

// @GetMapping("/history")

// public ResponseEntity<List<Orders>> getOrderHistory(@RequestParam String email) {

// try {

// List<Orders> orders = orderService.getOrderHistory(email);

// return ResponseEntity.ok(orders);

// } catch (Exception e) {

// return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body(Collections.emptyList());

// }

// }

@GetMapping("/history")

public ResponseEntity<List<Orders>> getOrderHistory(@RequestParam String email) {

List<Orders> orders = orderService.getOrderHistory(email);

return ResponseEntity.ok(orders);

}

@GetMapping("/allOrders")

public ResponseEntity<List<Orders>> getAllOrders(){

List<Orders> orders = orderService.getAllOrders();

return ResponseEntity.ok(orders);

}

//

// private OrderResponse convertToOrderResponse(Orders order) {

// OrderResponse orderResponse = new OrderResponse();

// orderResponse.setId(order.getOrderid());

// orderResponse.setEmailId(order.getEmailid());

// orderResponse.setTotalCost(order.getTotalCost());

// orderResponse.setOrderDate(order.getOrderDate());

// orderResponse.setOrder(order); // Pass the entire order to the frontend

//

// return orderResponse;

// }

// private OrderResponse convertToOrderResponse(Orders order) {

// OrderResponse orderResponse = new OrderResponse();

// orderResponse.setId(order.getOrderid());

// orderResponse.setEmailId(order.getEmailid());

// orderResponse.setTotalCost(order.getTotalCost());

// orderResponse.setOrderDate(order.getOrderDate());

//

// // Convert order item strings to order item responses

// List<OrderItemResponse> orderItemResponses = Arrays.stream(order.getOrderItemStrings().split(";"))

// .map(this::convertToOrderItemResponse)

// .collect(Collectors.toList());

//

//

// orderResponse.setItems(orderItemResponses);

//

// return orderResponse;

// }

//

// private OrderItemResponse convertToOrderItemResponse(String orderItemString) {

// // Parse the order item string and create an OrderItemResponse

// //logger.debug("Parsing order item string: {}", orderItemString);

// String[] parts = orderItemString.split(",");

// String medicineName = parts[0].substring(parts[0].indexOf(":") + 1).trim();

// int quantity = Integer.parseInt(parts[1].substring(parts[1].indexOf(":") + 1).trim());

// float price = Float.parseFloat(parts[2].substring(parts[2].indexOf(":") + 1).trim());

//

// OrderItemResponse orderItemResponse = new OrderItemResponse();

// orderItemResponse.setMedicineName(medicineName);

// orderItemResponse.setQuantity(quantity);

// orderItemResponse.setPrice(price);

//

// return orderItemResponse;

// }

// private OrderItemResponse convertToOrderItemResponse(String orderItemString) {

// // Parse the order item string and create an OrderItemResponse

// String[] parts = orderItemString.split(",");

// String medicineName = parts[0].substring(parts[0].indexOf(":") + 1).trim();

// int quantity = Integer.parseInt(parts[1].substring(parts[1].indexOf(":") + 1).trim());

// float price = Float.parseFloat(parts[2].substring(parts[2].indexOf(":") + 1).trim());

//

// OrderItemResponse orderItemResponse = new OrderItemResponse();

// orderItemResponse.setMedicineName(medicineName);

// orderItemResponse.setQuantity(quantity);

// orderItemResponse.setPrice(price);

//

// return orderItemResponse;

// }

}

**com.exception package**

**NotFoundException**

package com.exception;

public class NotFoundException extends RuntimeException {

public NotFoundException(String message) {

super(message);

}

}

**com.medicare package**

**Account**

package com.medicare;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Account {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int accno;

private String emailid;

private float amount;

public int getAccno() {

return accno;

}

public void setAccno(int accno) {

this.accno = accno;

}

public String getEmailid() {

return emailid;

}

public void setEmailid(String emailid) {

this.emailid = emailid;

}

public float getAmount() {

return amount;

}

public void setAmount(float amount) {

this.amount = amount;

}

}

**CheckOutRequest**

package com.medicare;

import java.util.List;

public class CheckOutRequest {

private String userEmail;

private List<Medicine> medicines;

private float totalCost;

public String getUserEmail() {

return userEmail;

}

public void setUserEmail(String userEmail) {

this.userEmail = userEmail;

}

public List<Medicine> getMedicines() {

return medicines;

}

public void setMedicines(List<Medicine> medicines) {

this.medicines = medicines;

}

public float getTotalCost() {

return totalCost;

}

public void setTotalCost(float totalCost) {

this.totalCost = totalCost;

}

}

**Login**

package com.medicare;

import java.util.List;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.OneToMany;

import javax.persistence.OneToOne;

@Entity

public class Login {

@Id

private String emailid;

private String password;

private String typeOfUser;

@OneToMany

@JoinColumn(name="emailid")

private List<Account> account;

public List<Account> getAccount() {

return account;

}

public void setAccount(List<Account> account) {

this.account = account;

}

public String getEmailid() {

return emailid;

}

public void setEmailid(String emailid) {

this.emailid = emailid;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getTypeOfUser() {

return typeOfUser;

}

public void setTypeOfUser(String typeOfUser) {

this.typeOfUser = typeOfUser;

}

}

**Medicine**

package com.medicare;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Medicine {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int medicineid;

private int inventory;

private float price;

private String name;

private String description;

private String imageurl;

private boolean enable = true;

private double offer;

private float originalprice;

public float getOriginalprice() {

return originalprice;

}

public void setOriginalprice(float originalprice) {

this.originalprice = originalprice;

}

public double getOffer() {

return offer;

}

public void setOffer(double offer) {

this.offer = offer;

}

public boolean isEnable() {

return enable;

}

public void setEnable(boolean enable) {

this.enable = enable;

}

public int getMedicineid() {

return medicineid;

}

public void setMedicineid(int medicineid) {

this.medicineid = medicineid;

}

public int getInventory() {

return inventory;

}

public void setInventory(int inventory) {

this.inventory = inventory;

}

public float getPrice() {

return price;

}

public void setPrice(float price) {

this.price = price;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

public String getImageurl() {

return imageurl;

}

public void setImageurl(String imageurl) {

this.imageurl = imageurl;

}

}

**OrderItemResponse**

package com.medicare;

public class OrderItemResponse {

private String medicineName;

private int quantity;

private float price;

public String getMedicineName() {

return medicineName;

}

public void setMedicineName(String medicineName) {

this.medicineName = medicineName;

}

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public float getPrice() {

return price;

}

public void setPrice(float price) {

this.price = price;

}

}

**OrderResponse**

package com.medicare;

import java.util.Date;

import java.util.List;

public class OrderResponse {

private int id;

private String emailId;

private float totalCost;

private Date orderDate;

private Orders order;

public Orders getOrder() {

return order;

}

public void setOrder(Orders order) {

this.order = order;

}

private List<OrderItemResponse> items;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getEmailId() {

return emailId;

}

public void setEmailId(String emailId) {

this.emailId = emailId;

}

public float getTotalCost() {

return totalCost;

}

public void setTotalCost(float totalCost) {

this.totalCost = totalCost;

}

public Date getOrderDate() {

return orderDate;

}

public void setOrderDate(Date orderDate) {

this.orderDate = orderDate;

}

public List<OrderItemResponse> getItems() {

return items;

}

public void setItems(List<OrderItemResponse> items) {

this.items = items;

}

}

**Orders**

package com.medicare;

import javax.persistence.\*;

import java.util.Date;

import java.util.List;

@Entity

public class Orders {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int orderid;

private String emailid;

@ManyToOne

@JoinColumn(name="accno")

private Account account;

@Column(columnDefinition = "TEXT")

private String orderItemStrings;

private float totalCost;

@Temporal(TemporalType.TIMESTAMP)

private Date orderDate;

public int getOrderid() {

return orderid;

}

public void setOrderid(int orderid) {

this.orderid = orderid;

}

public String getEmailid() {

return emailid;

}

public void setEmailid(String emailid) {

this.emailid = emailid;

}

public Account getAccount() {

return account;

}

public void setAccount(Account account) {

this.account = account;

}

// public List<OrderItem> getItems() {

// return items;

// }

// public void setItems(List<OrderItem> items) {

// this.items = items;

// }

public float getTotalCost() {

return totalCost;

}

public String getOrderItemStrings() {

return orderItemStrings;

}

public void setOrderItemStrings(String orderItemsJson) {

this.orderItemStrings = orderItemsJson;

}

public void setTotalCost(float totalCost) {

this.totalCost = totalCost;

}

public Date getOrderDate() {

return orderDate;

}

public void setOrderDate(Date orderDate) {

this.orderDate = orderDate;

}

}

**com.repository package**

**AccountRepository**

package com.repository;

import java.util.List;

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

import org.springframework.stereotype.Repository;

import com.medicare.Account;

@Repository

public interface AccountRepository extends JpaRepository<Account,Integer> {

Account findByEmailid(String emailid);

}

**LoginRepository**

package com.repository;

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

import org.springframework.stereotype.Repository;

import com.medicare.Login;

@Repository

public interface LoginRepository extends JpaRepository<Login, String>{

}

**MedicineRepository**

package com.repository;

import java.util.List;

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

import org.springframework.stereotype.Repository;

import com.medicare.Medicine;

@Repository

public interface MedicineRepository extends JpaRepository<Medicine,Integer> {

List<Medicine> findByNameContainingIgnoreCase(String name);

}

**OrdersRepository**

package com.repository;

import java.util.List;

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

import com.medicare.Orders;

public interface OrdersRepository extends JpaRepository<Orders, Long> {

List<Orders> findByemailidOrderByOrderDateDesc(String emailid);

List<Orders> findByAccountOrderByOrderDateDesc(String email);

List<Orders> findByEmailid(String emailid);

}

**com.service package**

**AccountService**

package com.service;

import java.util.List;

import java.util.Map;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.web.bind.annotation.RequestBody;

import com.medicare.Account;

import com.medicare.Login;

import com.medicare.Medicine;

import com.repository.AccountRepository;

@Service

public class AccountService {

@Autowired

AccountRepository accountRepository;

public String createAccount(Account account) {

accountRepository.save(account);

return "Account created";

}

public float getAccountAmount(String email) {

Account account = accountRepository.findByEmailid(email);

if (account != null) {

return account.getAmount();

} else {

return -1;

}

}

public float modifyAccountAmount(String email, Float newAmount) {

Account account = accountRepository.findByEmailid(email);

if (account != null) {

account.setAmount(newAmount);

accountRepository.save(account);

return account.getAmount();

} else {

return -1;

}

}

}

**LoginService**

package com.service;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.medicare.Account;

import com.medicare.Login;

import com.repository.LoginRepository;

@Service

public class LoginService {

@Autowired

LoginRepository loginRepository;

@Autowired

AccountService accountService;

public String signIn(Login login) {

Optional<Login> result = loginRepository.findById(login.getEmailid());

if(result.isPresent()) {

Login ll = result.get();

if(ll.getPassword().equals(login.getPassword())) {

if(ll.getTypeOfUser().equals("admin")) {

return "Admin Success";

}else {

return "Customer success";

}

}else {

return "Password is wrong";

}

}else {

return "EmailId is wrong";

}

}

public String signUp(Login login) {

if (login.getTypeOfUser().contentEquals("admin")) {

return "Admin Account cannot be created";

}else {

Optional<Login> result=loginRepository.findById(login.getEmailid());

if (result.isPresent()) {

return "Account already exist";

}else {

Account acc = new Account();

acc.setAmount(1000);

acc.setEmailid(login.getEmailid());

loginRepository.save(login);

accountService.createAccount(acc);

return "Account created successfully";

}

}

}

}

**MedicineService**

package com.service;

import java.util.List;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.repository.MedicineRepository;

import com.medicare.Medicine;

import com.exception.\*;

@Service

public class MedicineService {

@Autowired

MedicineRepository medicineRepository;

public String createMedicine(Medicine medicine) {

medicine.setOriginalprice(medicine.getPrice());

medicine.setEnable(true);

medicineRepository.save(medicine);

return "Medicine created";

}

public List<Medicine> findAllMedicine() {

return medicineRepository.findAll();

}

public String deleteMedicine(int mid) {

Optional<Medicine> result = medicineRepository.findById(mid);

if(result.isPresent()) {

Medicine m = result.get();

medicineRepository.delete(m);

return "Medicine deleted successfully";

}else {

return "Medicine not present";

}

}

public void updateProduct(int medicineid, Medicine updatedProduct) {

Medicine existingProduct = medicineRepository.findById(medicineid)

.orElseThrow();

existingProduct.setDescription(updatedProduct.getDescription());

existingProduct.setInventory(updatedProduct.getInventory());

existingProduct.setName(updatedProduct.getName());

existingProduct.setPrice(updatedProduct.getPrice());

existingProduct.setEnable(updatedProduct.isEnable());

existingProduct.setImageurl(updatedProduct.getImageurl());

existingProduct.setOffer(updatedProduct.getOffer());

if (updatedProduct.getOffer() == 0) {

existingProduct.setPrice(updatedProduct.getOriginalprice());

}

medicineRepository.save(existingProduct);

}

public List<Medicine> searchMedicine(String query) {

return medicineRepository.findByNameContainingIgnoreCase(query);

}

public String enableMedicine(int medicineId) {

Optional<Medicine> optionalMedicine = medicineRepository.findById(medicineId);

if (optionalMedicine.isPresent()) {

Medicine medicine = optionalMedicine.get();

medicine.setEnable(true);

medicineRepository.save(medicine);

return "Medicine enabled";

} else {

return "Medicine not found";

}

}

public String disableMedicine(int medicineId) {

Optional<Medicine> optionalMedicine = medicineRepository.findById(medicineId);

if (optionalMedicine.isPresent()) {

Medicine medicine = optionalMedicine.get();

medicine.setEnable(false);

medicineRepository.save(medicine);

return "Medicine disabled";

} else {

return "Medicine not found";

}

}

public Medicine findMedicineById(int medicineId) {

return medicineRepository.findById(medicineId).orElse(null);

}

public Medicine updateInventory(int medicineId, int newInventory) {

Optional<Medicine> optionalMedicine = medicineRepository.findById(medicineId);

if (optionalMedicine.isPresent()) {

Medicine medicine = optionalMedicine.get();

medicine.setInventory(newInventory);

return medicineRepository.save(medicine);

} else {

throw new NotFoundException("Medicine not found");

}

}

}

**OrdersService**

package com.service;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.medicare.Account;

import com.medicare.Medicine;

import com.medicare.OrderItem;

import com.medicare.Orders;

import com.repository.AccountRepository;

import com.repository.OrdersRepository;

//import com.repository.OrderItemRepository;

import java.util.Arrays;

import java.util.Date;

import java.util.List;

import java.util.stream.Collectors;

@Service

public class OrdersService {

private final OrdersRepository orderRepository;

private final AccountRepository accountRepository;

//private final OrderItemRepository orderItemRepository;

@Autowired

public OrdersService(OrdersRepository orderRepository, AccountRepository accountRepository) {//, OrderItemRepository orderItemRepository) {

this.orderRepository = orderRepository;

this.accountRepository = accountRepository;

// this.orderItemRepository = orderItemRepository;

}

// @Transactional

// public Orders createOrder(Account account, List<Medicine> medicines, float totalCost) {

// Orders order = new Orders();

// order.setAccount(account);

// order.setEmailid(account.getEmailid());

// order.setItems(createOrderItems(medicines, order)); // Pass the order to createOrderItems

// order.setTotalCost(totalCost);

// order.setOrderDate(new Date());

// // Save the order to the database

// Orders savedOrder = orderRepository.save(order);

//

// // Update the order ID in the associated order items

// List<OrderItem> orderItems = savedOrder.getItems();

// for (OrderItem orderItem : orderItems) {

// orderItem.setOrder(savedOrder);

// }

// return savedOrder;

// }

//

// private List<OrderItem> createOrderItems(List<Medicine> medicines, Orders order) {

// return medicines.stream()

// .map(medicine -> {

// OrderItem orderItem = new OrderItem();

// orderItem.setMedicineId(medicine.getMedicineid());

// orderItem.setQuantity(1); // Assuming each medicine adds 1 quantity to the order

// orderItem.setOrder(order); // Set the order for the order item

// return orderItem;

// })

// .collect(Collectors.toList());

// }

private List<String> createOrderItemStrings(List<Medicine> medicines) {

return medicines.stream()

.map(medicine -> {

String orderItemString = String.format(

"Medicine: %s, Quantity: %d, Price: %.2f",

medicine.getName(),

1, // Assuming each medicine adds 1 quantity to the order

medicine.getPrice()

);

return orderItemString;

})

.collect(Collectors.toList());

}

public Orders createOrder(Account account, List<Medicine> medicines, float totalCost) {

Orders order = new Orders();

order.setAccount(account);

order.setEmailid(account.getEmailid());

order.setTotalCost(totalCost);

order.setOrderDate(new Date());

List<String> orderItems = createOrderItemStrings(medicines);

ObjectMapper objectMapper = new ObjectMapper();

try {

String orderItemsJson = objectMapper.writeValueAsString(orderItems);

order.setOrderItemStrings(orderItemsJson);

} catch (JsonProcessingException e) {

// Handle the exception (e.g., log an error)

}

Orders savedOrder = orderRepository.save(order);

return savedOrder;

}

// public Orders createOrder(Account account, List<Medicine> medicines, float totalCost) {

// Orders order = new Orders();

// order.setAccount(account);

// order.setEmailid(account.getEmailid());

// order.setTotalCost(totalCost);

// order.setOrderDate(new Date());

//

// List<String> orderItems = createOrderItemStrings(medicines);

// order.setOrderItemStrings(orderItems);

//

// Orders savedOrder = orderRepository.save(order);

// return savedOrder;

// }

//

// public List<Orders> getOrderHistory(String email) {

// // Fetch order history for the given account

// List<Orders> orders = orderRepository.findByemailidOrderByOrderDateDesc(email);

// System.out.println(Arrays.toString(orders.toArray()));

// // Fetch order items for each order

// for (Orders order : orders) {

// List<OrderItem> orderItems = fetchOrderItemsForOrder(order);

// System.out.println("Order ID: " + order.getOrderid());

// System.out.println("Order Items: " + orderItems.toString());

// order.setItems(orderItems);

// }

//

// return orders;

// }

//

// private List<OrderItem> fetchOrderItemsForOrder(Orders order) {

// // Fetch order items for the given order ID

// return orderItemRepository.findByOrderOrderid(order);

// //return orderItemRepository.findByOrderid(order);

// }

//

public List<Orders> getOrderHistory(String email) {

List<Orders> orders = orderRepository.findByEmailid(email);

List<Orders> orders1 = orderRepository.findAll();

//System.out.println("Orders received with email "+email+" in getOrderHistory: " + orders);

return orders;

}

public List<Orders> getAllOrders() {

List<Orders> orders1 = orderRepository.findAll();

//System.out.println("Orders received with email "+email+" in getOrderHistory: " + orders);

return orders1;

}

//

}

**com package**

**CorsConfig**

package com;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.config.annotation.CorsRegistry;

import org.springframework.web.servlet.config.annotation.EnableWebMvc;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration

@EnableWebMvc

public class CorsConfig implements WebMvcConfigurer {

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/\*\*")

.allowedOrigins("http://localhost:3000") // Update this with your frontend's origin

.allowedMethods("GET", "POST", "PUT", "DELETE")

.allowCredentials(true);

}

}

**MedicareBackendApplication**

package com;

import javax.annotation.PostConstruct;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.autoconfigure.domain.EntityScan;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

import com.medicare.Login;

import com.repository.LoginRepository;

@SpringBootApplication(scanBasePackages ="com")

@EntityScan(basePackages = "com.medicare")

@EnableJpaRepositories(basePackages ="com.repository")

public class MedicareBackendApplication {

@Autowired

LoginRepository loginRepository;

@PostConstruct

public void adminAccountCreation( ) {

Login ll = new Login();

ll.setEmailid("admin@gmail.com");

ll.setPassword("admin@123");

ll.setTypeOfUser("admin");

loginRepository.save(ll);

}

public static void main(String[] args) {

SpringApplication.run(MedicareBackendApplication.class, args);

System.out.println("spring boot backend ready");

}

}

**application.properties**

spring.datasource.url=jdbc\:mysql\://localhost\:3306/medicaredb

logging.level.com.controller.OrdersController=DEBUG

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.username=root

spring.datasource.password=London\!23

spring.jpa.hibernate.ddl-auto=update

**MedicareBackendApplicationTests**

package com;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest

class MedicareBackendApplicationTests {

@Test

void contextLoads() {

}

}

**Frontend**

**Components**

**AddMedicine.js**

import React, { useState } from 'react';

import axios from 'axios';

import { Link, useNavigate } from 'react-router-dom';

import '../AddMedicine.css';

function AddMedicine() {

const [name, setName] = useState('');

const [description, setDescription] = useState('');

const [price, setPrice] = useState('');

const [inventory, setInventory] = useState('');

const [imageurl, setImageURL] = useState('');

const [enable, setEnable] = useState('');

const navigate = useNavigate();

const addMedicine = async (event) => {

event.preventDefault();

const medicine = {

name: name,

description: description,

price: price,

inventory: inventory,

imageurl: imageurl,

enable: enable,

};

try {

const result = await axios.post('http://localhost:8080/Medicine/addMedicine', medicine);

if (result.data === 'Medicine created') {

alert('Medicine Created Successfully!!');

} else {

alert('Please try again');

}

} catch (ex) {

console.log(ex);

}

};

return (

<div className="add-medicine-container">

<h2 className="add-medicine-header">Add Medicine</h2>

<form className="add-medicine-form" onSubmit={addMedicine}>

<label>Name</label>

<input type="text" name="name" onChange={(e) => setName(e.target.value)} />

<br />

<label>Description</label>

<input type="text" name="description" onChange={(e) => setDescription(e.target.value)} />

<br />

<label>Price</label>

<input type="number" name="price" onChange={(e) => setPrice(e.target.value)} />

<br />

<label>Quantity</label>

<input type="number" name="inventory" onChange={(e) => setInventory(e.target.value)} />

<br />

<label>Image URL</label>

<input type="text" name="imageurl" onChange={(e) => setImageURL(e.target.value)} />

<br />

<label>Enable</label>

<input type="checkbox" name="enable" onChange={(e) => setEnable(e.target.checked)} />

<br />

<input type="submit" value="Submit" />

<input type="reset" value="Reset" />

<br />

</form>

</div>

);

}

export default AddMedicine;

**Admin.js**

import React from 'react';

import { Link, Outlet } from 'react-router-dom';

import '../Admin.css';

function Admin() {

return (

<div className="admin-container">

<h2 className="admin-header">Welcome to Admin Home Page</h2>

<nav className="admin-nav">

<Link to="addMedicine">Add Medicine</Link>

<Link to="viewMedicine">View all Products</Link>

<Link to="allOrders">View All Placed Orders</Link>

<Link to="/">Logout</Link>

</nav>

{/\* <hr className="admin-hr" /> \*/}

<Outlet />

</div>

);

}

export default Admin;

**AllOrders.js**

import React, { useState, useEffect } from "react";

import axios from "axios";

import "../Orders.css";

function AllOrders() {

const [orderHistory, setOrderHistory] = useState([]);

const [filteredOrders, setFilteredOrders] = useState([]);

const [sortBy, setSortBy] = useState("orderDate");

const [searchQuery, setSearchQuery] = useState("");

const [currentPage, setCurrentPage] = useState(1);

const itemsPerPage = 10;

useEffect(() => {

fetchOrderHistory();

}, []);

useEffect(() => {

filterOrders();

}, [orderHistory, searchQuery]);

const fetchOrderHistory = async () => {

try {

const response = await axios.get("http://localhost:8080/orders/allOrders");

setOrderHistory(response.data);

} catch (error) {

console.error("Error fetching order history:", error);

}

};

const filterOrders = () => {

const filtered = orderHistory.filter(

(order) =>

order.orderid.toString().includes(searchQuery) ||

order.totalCost.toFixed(2).includes(searchQuery) ||

new Date(order.orderDate).toLocaleDateString().includes(searchQuery)

);

setFilteredOrders(filtered);

};

const handleSortChange = (event) => {

setSortBy(event.target.value);

};

const compareOrders = (a, b) => {

if (sortBy === "orderid") {

return a.orderid - b.orderid;

} else if (sortBy === "totalCost") {

return a.totalCost - b.totalCost;

} else {

const dateA = new Date(a.orderDate).getTime();

const dateB = new Date(b.orderDate).getTime();

return dateA - dateB;

}

};

const totalPages = Math.ceil(filteredOrders.length / itemsPerPage);

const startIndex = (currentPage - 1) \* itemsPerPage;

const endIndex = startIndex + itemsPerPage;

const currentOrders = filteredOrders.slice(startIndex, endIndex);

const handlePageChange = (pageNumber) => {

setCurrentPage(pageNumber);

};

return (

<div className="container mt-4">

<h2 className="orders-title">All Order History</h2>

<div className="filter-sort">

<input

type="text"

placeholder="Search by order ID, total cost, or date"

value={searchQuery}

onChange={(e) => setSearchQuery(e.target.value)}

/>

<label>Sort by:</label>

<select value={sortBy} onChange={handleSortChange}>

<option value="orderDate">Order Date</option>

<option value="orderid">Order ID</option>

<option value="totalCost">Total Cost</option>

</select>

</div>

<table className="table orders-table">

<thead>

<tr>

<th>Order ID</th>

<th>Total Cost (£)</th>

<th>Order Date</th>

<th>Items</th>

</tr>

</thead>

<tbody>

{currentOrders.length > 0 ? (

currentOrders.sort(compareOrders).map((order) => (

<tr key={order.orderid}>

<td>{order.orderid}</td>

<td>{order.totalCost.toFixed(2)}</td>

<td>{new Date(order.orderDate).toLocaleDateString()}</td>

<td>

<ul className="order-items-list">

{order.orderItemStrings && order.orderItemStrings.length > 0 ? (

JSON.parse(order.orderItemStrings).map((item, index) => (

<li key={index}>{item}</li>

))

) : (

<li>No items</li>

)}

</ul>

</td>

</tr>

))

) : (

<tr>

<td colSpan="4" className="no-orders-text">

You have no order history.

</td>

</tr>

)}

</tbody>

</table>

<div className="pagination">

{Array.from({ length: totalPages }, (\_, index) => (

<button

key={index}

onClick={() => handlePageChange(index + 1)}

className={currentPage === index + 1 ? "active" : ""}

>

{index + 1}

</button>

))}

</div>

</div>

);

}

export default AllOrders;

**Basket.js**

import React from "react";

import "../Basket.css";

function Basket({ basketItems, onRemoveItem }) {

return (

<div className="basket-container">

<div className="basket-header">

<h2>Basket</h2>

</div>

<ul className="basket-items">

{basketItems.map((item) => (

<li className="basket-item" key={item.medicineid}>

<div className="basket-item-details">

<span className="basket-item-name">{item.name}</span>

<span className="basket-item-price">£{item.price}</span>

</div>

<span

className="basket-item-remove"

onClick={() => onRemoveItem(item.medicineid)}

>

Remove

</span>

</li>

))}

</ul>

</div>

);

}

export default Basket;

**BrowseMedical.js**

import axios from "axios";

import React, { useState, useEffect } from "react";

import Basket from "./Basket";

import { useUser } from '../UserContext';

import "../BrowseMedical.css";

import { Link, useNavigate } from "react-router-dom";

function BrowseMedical() {

const [medicines, setMedicines] = useState([]);

const [basket, setBasket] = useState([]);

const [searchQuery, setSearchQuery] = useState("");

const [totalBasketAmount, setTotalBasketAmount] = useState(0);

const [accountAmount, setAccountAmount] = useState(0);

const user = useUser();

const navigate = useNavigate();

useEffect(() => {

console.log("Basket Items:", basket);

}, [basket]);

const loadMedicines = async () => {

try {

const response = await axios.get("http://localhost:8080/Medicine/viewAllMedicine");

setMedicines(response.data);

} catch (error) {

console.error("Error fetching medicines:", error);

}

};

const handleSearch = async () => {

if (searchQuery.trim() === "") {

loadMedicines();

} else {

try {

const response = await axios.get(`http://localhost:8080/Medicine/searchMedicine?query=${searchQuery}`);

setMedicines(response.data);

} catch (error) {

console.error("Error searching medicines:", error);

}

}

};

const addToBasket = async (medicineId) => {

try {

const response = await axios.get(`http://localhost:8080/Medicine/${medicineId}`);

const selectedMedicine = response.data;

setBasket((prevBasket) => [...prevBasket, selectedMedicine]);

setTotalBasketAmount((prevTotal) => prevTotal + selectedMedicine.price);

console.log("Added to basket:", selectedMedicine);

} catch (error) {

console.error("Error adding to basket:", error);

}

};

const loadAccountAmount = async () => {

try {

const usersEmails = user.userEmail;

const accountResponse = await axios.get(`http://localhost:8080/account/amount?email=${usersEmails}`);

const fetchedAccountAmount = accountResponse.data;

setAccountAmount(fetchedAccountAmount);

} catch (error) {

console.error("Error fetching Account Amount:", error);

}

};

useEffect(() => {

if (user) {

loadAccountAmount();

}

}, [user]);

const removeItemFromBasket = (medicineId) => {

const updatedBasket = basket.filter(item => item.medicineid !== medicineId);

setBasket(updatedBasket);

const removedItem = basket.find(item => item.medicineid === medicineId);

setTotalBasketAmount(prevTotal => prevTotal - removedItem.price);

};

const handlesCheckout = async () => {

try {

if (!user) {

console.log("User not logged in");

return;

}

const usersEmail = user.userEmail;

console.log("User Email:", usersEmail);

// Calculate the total cost of items in the basket

const totalCost = basket.reduce((total, item) => total + item.price, 0);

// Fetch the current inventory of each medicine

const fetchMedicinesPromises = basket.map(async (item) => {

const medicineResponse = await axios.get(`http://localhost:8080/Medicine/${item.medicineid}`);

const medicine = medicineResponse.data;

return medicine;

});

const medicines = await Promise.all(fetchMedicinesPromises);

// Check if account has sufficient balance and medicines have enough inventory

const canCheckout = accountAmount >= totalCost && medicines.every((medicine) => medicine.inventory > 0);

if (canCheckout) {

const checkoutRequest = {

userEmail: usersEmail,

medicines: basket.map(item => ({

medicineid: item.medicineid,

quantity: 1 ,

name: item.name,

price: item.price

})),

totalCost: totalCost

};

if (canCheckout){

const updateMedicinePromises = medicines.map(async (medicine) => {

const updatedInventory = medicine.inventory - 1;

await axios.put(`http://localhost:8080/Medicine/updateInventory/${medicine.medicineid}?newInventory=${updatedInventory}`);

});

};

const updatedAccountAmount = accountAmount - totalCost;

// Send the checkout request to create an order

const createOrderResponse = await axios.post("http://localhost:8080/orders/checkout", checkoutRequest);

console.log("Order created successfully. Order ID:", createOrderResponse.data);

console.log(basket);

navigate("./order-summary", { state: { orderItems: basket } });

// Clear the basket after successful checkout

setBasket([]);

loadMedicines();

setTotalBasketAmount(0);

setAccountAmount(updatedAccountAmount);

} else {

console.log("Checkout Failed: Insufficient Balance or Inventory");

}

//navigate("order-summary", { state: { orderItems: basket } });

} catch (error) {

console.error("Error during checkout:", error);

}

};

useEffect(() => {

console.log("Basket Items:", basket);

}, [basket]);

const medicineRecord = medicines.map((medicine) => (

<tr key={medicine.medicineid}>

{medicine.enable ? (

<>

<td>{medicine.medicineid}</td>

<td>{medicine.name}</td>

<td>{medicine.description}</td>

<td>£{medicine.price}</td>

<td>{medicine.inventory}</td>

<td>

<img src={medicine.imageurl} alt={medicine.name} width="100px" height="100px" />

</td>

<td>{medicine.offer}%</td>

<td>

<button onClick={() => addToBasket(medicine.medicineid)}>Add to Basket</button>

</td>

</>

) : null}

</tr>

));

return (

<div className="browse-medical-container">

<h2>View all Medicine Products</h2>

<div className="search-bar">

<input

type="text"

placeholder="Search by name"

value={searchQuery}

onChange={(e) => setSearchQuery(e.target.value)}

/>

<button className="search-button" onClick={handleSearch}>Search</button>

<button className="show-all-button" onClick={loadMedicines}>Show All</button>

</div>

<table className="medicine-table">

<thead>

<tr>

<th>ID</th>

<th>Name</th>

<th>Description</th>

<th>Price</th>

<th>Stock</th>

<th>Image</th>

<th>Offer % discount (Already Applied in Price)</th>

<th></th>

</tr>

</thead>

<tbody>{medicineRecord}</tbody>

</table>

<div className="basket-checkout">

<Basket basketItems={basket} onRemoveItem={removeItemFromBasket} />

<div className="basket-summary">

<p>Total Basket Amount: £{totalBasketAmount.toFixed(2)}</p>

<br></br>

<p>Available to spend from your account: £{accountAmount.toFixed(2)}</p>

<button className="checkout-button" onClick={handlesCheckout}>

Checkout

</button>

</div>

</div>

</div>

);

}

export default BrowseMedical;

**Customer.js**

import { Link ,Outlet} from "react-router-dom";

import '../Admin.css';

import React from 'react';

function Customer(){

return(

<div className="admin-container">

<h2 className="admin-header">Welcome to Customer Home Page</h2>

<nav className="admin-nav">

<Link to="browseMedical">Purchase Medicines!</Link>

<Link to ="orders">View Past Orders</Link>

<Link to ="topUpAccount">Top Up Account</Link>

<Link to="/">Logout</Link>

</nav>

{/\* <hr className="admin-hr" /> \*/}

<Outlet />

</div>

);

export default Customer;

**HomePage.js**

import React from "react";

import image from "../Images/Medicare1.jpg";

import image2 from "../Images/Medicare2.jpg";

import "../HomePage.css";

const HomePage = () => {

return (

<div className="home-page">

<h2 className="home-title">Welcome to Medicare - Your Medical Supplies Shop</h2>

<div className="image-container">

<img src={image} alt="Medical Supplies 1" className="responsive-image" />

<img src={image2} alt="Medical Supplies 2" className="responsive-image" />

</div>

<p className="home-text">

At Medicare, we provide a wide range of high-quality medical supplies to meet your needs. Whether you're looking for personal protective equipment, medical devices, or healthcare essentials, we have you covered. Our mission is to ensure you have access to top-notch products that contribute to your well-being.

</p>

<p className="home-text">

Browse through our collection and experience the convenience of shopping for medical supplies online. We're committed to providing reliable products and exceptional service to our customers. Start exploring our shop today!

</p>

</div>

);

};export default HomePage;

**OrderSummary.js**

import React from "react";

import { useLocation, Link } from "react-router-dom";

import "../OrderSummary.css";

function OrderSummary() {

const location = useLocation();

const { orderItems } = location.state || { orderItems: [] };

const totalCost = orderItems.reduce((total, item) => total + item.price, 0);

console.log(orderItems);

return (

<div className="order-summary">

<h2>Order Summary</h2>

{orderItems.length > 0 ? (

<ul className="order-items">

{orderItems.map((item, index) => (

<li key={index}>

{item.name} - £{item.price}

</li>

))}

</ul>

) : (

<p className="empty-message">No items in the basket.</p>

)}

<p className="total-cost">Total Cost: £{totalCost.toFixed(2)}</p>

<Link to="/customer/browseMedical" className="back-link">

Back to Browse Medical

</Link>

</div>

);

}

export default OrderSummary;

**Orders.js**

import React, { useState, useEffect } from "react";

import axios from "axios";

import { useUser } from '../UserContext';

import "../Orders.css";

function Orders() {

const [orderHistory, setOrderHistory] = useState([]);

const user = useUser();

useEffect(() => {

fetchOrderHistory();

}, []);

const fetchOrderHistory = async () => {

try {

if (!user) {

console.log("User not logged in");

return;

}

const usersEmail = user.userEmail;

const response = await axios.get(`http://localhost:8080/orders/history?email=${usersEmail}`);

setOrderHistory(response.data);

} catch (error) {

console.error("Error fetching order history:", error);

}

};

return (

<div className="container mt-4">

<h2 className="orders-title">Your Order History</h2>

<table className="table orders-table">

<thead>

<tr>

<th>Order ID</th>

<th>Total Cost (£)</th>

<th>Order Date</th>

<th>Items</th>

</tr>

</thead>

<tbody>

{orderHistory.length > 0 ? (

orderHistory.map(order => (

<tr key={order.orderid}>

<td>{order.orderid}</td>

<td>£{order.totalCost.toFixed(2)}</td>

<td>{new Date(order.orderDate).toLocaleDateString()}</td>

<td>

<ul className="order-items-list">

{order.orderItemStrings && order.orderItemStrings.length > 0 ? (

JSON.parse(order.orderItemStrings).map((item, index) => (

<li key={index}>{item}</li>

))

) : (

<li>No items</li>

)}

</ul>

</td>

</tr>

))

) : (

<tr>

<td colSpan="4" className="no-orders-text">

You have no order history.

</td>

</tr>

)}

</tbody>

</table>

</div>

);

}

export default Orders;

**TopUpAccount.js**

import axios from "axios";

import { useUser } from '../UserContext';

import React, { useState, useEffect } from "react";

import "../TopUoAccount.css";

function TopUpAccount() {

const user = useUser();

const [newAmount, setNewAmount] = useState(0);

const [creditCardNumber, setCreditCardNumber] = useState("");

const [expiryDate, setExpiryDate] = useState("");

const [csv, setCSV] = useState("");

const [successMessage, setSuccessMessage] = useState("");

const [accountAmount, setAccountAmount] = useState(0);

const [errorMessage, setErrorMessage] = useState("");

const [totalAmount, setTotalAmount] = useState(0);

const handleTopUp = async () => {

const newAmount = accountAmount + parseFloat(totalAmount);

console.log(totalAmount);

console.log(newAmount);

try {

const usersEmail = user.userEmail;

console.log(usersEmail);

const response = await axios.put("http://localhost:8080/account/topUp", {

email: usersEmail,

newAmount: parseFloat(newAmount)

});

if (response.data) {

setSuccessMessage("Top-up successful!");

setErrorMessage("");

setTotalAmount(0);

setCreditCardNumber("");

setExpiryDate("");

setCSV("");

loadAccountAmount();

} else {

setSuccessMessage("");

setErrorMessage("Top-up failed. Please check your details and try again.");

}

} catch (error) {

console.error("Error topping up account:", error);

setSuccessMessage("");

setErrorMessage("An error occurred. Please try again later.");

}

};

const loadAccountAmount = async () => {

try {

const usersEmails = user.userEmail;

console.log(usersEmails);

const accountResponse = await axios.get(`http://localhost:8080/account/amount?email=${usersEmails}`);

const fetchedAccountAmount = accountResponse.data;

setAccountAmount(fetchedAccountAmount);

console.log(accountAmount);

} catch (error) {

console.error("Error fetching Account Amount:", error);

}

};

useEffect(() => {

if (user) {

loadAccountAmount();

}

}, [user]);

return (

<div className="top-up-container">

<h2>Top Up Account</h2>

{successMessage && <p className="success-message">{successMessage}</p>}

{errorMessage && <p className="error-message">{errorMessage}</p>}

<div className="input-group">

<label>Amount to Deposit (£):</label>

<input

type="number"

value={totalAmount}

onChange={(e) => setTotalAmount(e.target.value)}

/>

</div>

<div className="input-group">

<label>Credit Card Number:</label>

<input

type="text"

value={creditCardNumber}

onChange={(e) => setCreditCardNumber(e.target.value)}

/>

</div>

<div className="input-group">

<label>Expiry Date:</label>

<input

type="text"

value={expiryDate}

onChange={(e) => setExpiryDate(e.target.value)}

/>

</div>

<div className="input-group">

<label>CSV:</label>

<input

type="text"

value={csv}

onChange={(e) => setCSV(e.target.value)}

/>

</div>

<button className="top-up-button" onClick={handleTopUp}>Top Up</button>

</div>

);

}

export default TopUpAccount;

**ViewMedicine.js**

import axios from "axios";

import React, { useState, useEffect } from "react";

import '../ViewMedicine.css';

function ViewMedicine() {

const [medicines, setMedicines] = useState([]);

const [editMedicine, setEditMedicine] = useState(null);

const [editedValues, setEditedValues] = useState({});

const [searchQuery, setSearchQuery] = useState("");

const loadMedicines = async () => {

try {

const response = await axios.get("http://localhost:8080/Medicine/viewAllMedicine");

setMedicines(response.data);

} catch (error) {

console.error("Error fetching medicines:", error);

}

};

const handleDelete = async (medicineid) => {

try {

await axios.delete(`http://localhost:8080/Medicine/${medicineid}`);

loadMedicines(); // Refresh product list after deletion

} catch (error) {

console.error("Error deleting product:", error);

}

};

const handleEdit = (medicine) => {

setEditedValues({ ...medicine });

setEditMedicine(medicine);

};

const handleUpdate = async (updatedMedicine) => {

try {

const updatedPrice = updatedMedicine.originalprice \* (1 - updatedMedicine.offer / 100);

updatedMedicine.price = updatedPrice;

await axios.put(`http://localhost:8080/Medicine/${updatedMedicine.medicineid}`, updatedMedicine);

loadMedicines(); // Refresh medicine list after update

setEditMedicine(null); // Clear edit mode

} catch (error) {

console.error("Error updating medicine:", error);

}

};

const handleSearch = async () => {

if (searchQuery.trim() === "") {

loadMedicines(); // Reset to all medicines

} else {

try {

const response = await axios.get(`http://localhost:8080/Medicine/searchMedicine?query=${searchQuery}`);

setMedicines(response.data);

} catch (error) {

console.error("Error searching medicines:", error);

}

}

};

const handleEnable = async (medicineid) => {

try {

await axios.put(`http://localhost:8080/Medicine/enable/${medicineid}`);

loadMedicines(); // Refresh medicine list after enabling

} catch (error) {

console.error("Error enabling medicine:", error);

}

};

const handleDisable = async (medicineid) => {

try {

await axios.put(`http://localhost:8080/Medicine/disable/${medicineid}`);

loadMedicines(); // Refresh medicine list after disabling

} catch (error) {

console.error("Error disabling medicine:", error);

}

};

const medicineRecord = medicines.map((medicine) => (

<tr key={medicine.medicineid}>

<td>{medicine.medicineid}</td>

<td>{editMedicine === medicine ? (

<input

type="text"

value={editedValues.name}

onChange={(e) => setEditedValues({ ...editedValues, name: e.target.value })}

/>

) : (

medicine.name

)}</td>

<td>{editMedicine === medicine ? (

<input

type="text"

value={editedValues.description}

onChange={(e) => setEditedValues({ ...editedValues, description: e.target.value })}

/>

) : (

medicine.description

)}</td>

<td>{editMedicine === medicine ? (

<input

type="number"

value={editedValues.price}

onChange={(e) => setEditedValues({ ...editedValues, price: e.target.value})}

/>

) : (

medicine.price

)}</td>

<td>{editMedicine === medicine ? (

<input

type="number"

value={editedValues.inventory}

onChange={(e) => setEditedValues({ ...editedValues, inventory: e.target.value})}

/>

) : (

medicine.inventory

)}</td>

<td>{editMedicine === medicine ? (

<input

type="number"

value={editedValues.offer}

onChange={(e) => setEditedValues({ ...editedValues, offer: e.target.value})}

/>

) : (

medicine.offer

)}</td>

<td>

<img src={medicine.imageurl} alt={medicine.name} width="100px" height="100px" />

</td>

<td>

{editMedicine === medicine ? (

<button onClick={() => handleUpdate(editedValues)}>Update</button>

) : (

<button onClick={() => handleEdit(medicine)}>Edit</button>

)}

<button onClick={() => handleDelete(medicine.medicineid)}>Delete</button>

</td>

{/\* <td>{medicine.offer}%</td> \*/}

{/\* <td>

<button onClick={() => handleDelete(medicine.medicineid)}>Delete</button>

<button onClick={() => handleEdit(medicine)}>Edit</button>

</td> \*/}

<td>

{medicine.enable ? (

<button onClick={() => handleDisable(medicine.medicineid)}>Disable</button>

) : (

<button onClick={() => handleEnable(medicine.medicineid)}>Enable</button>

)}

</td>

</tr>

));

return (

<div>

<h2>Search Medicine Products</h2>

<div>

<input

type="text"

placeholder="Search by name"

value={searchQuery}

onChange={(e) => setSearchQuery(e.target.value)}

/>

<button onClick={handleSearch}>Search</button>

<button onClick={loadMedicines}>Show All</button>

</div>

<table className="medicine-table">

<thead>

<tr>

<th>ID</th>

<th>Name</th>

<th>Description</th>

<th>Price</th>

<th>Stock</th>

{/\* <th>Image</th> \*/}

<th>Offer % discount</th>

<th>Image</th>

<th>Edit / Delete Record</th>

<th>Enable / Disable</th>

</tr>

</thead>

<tbody>{medicineRecord}</tbody>

</table>

</div>

);

}

export default ViewMedicine;

**Login.js**

import React, { useState } from 'react';

import { useUser } from './UserContext';

import axios from 'axios';

import { Link, useNavigate } from 'react-router-dom';

import './Login.css';

function Login() {

const { setUserEmail } = useUser();

const [emailid, setEmailid] = useState('');

const [password, setPassword] = useState('');

const [typeofuser, setTypeOfUser] = useState('');

const navigate = useNavigate();

const signIn = async (event) => {

event.preventDefault();

let login = { emailid: emailid, password: password, typeofuser: typeofuser };

try {

let result = await axios.post('http://localhost:8080/Login/signIn', login);

if (result.data === 'Admin Success') {

navigate('/admin');

} else if (result.data === 'Customer success') {

setUserEmail(emailid);

navigate('/customer');

} else {

alert(result.data);

}

} catch (ex) {

console.log(ex);

}

};

return (

<div className="login-container">

<div className="login-header">Login Page</div>

<form className="login-form" onSubmit={signIn}>

<label>Email</label>

<input

className="login-input"

type="email"

name="emailid"

onChange={(e) => setEmailid(e.target.value)}

/>

<label>Password</label>

<input

className="login-input"

type="password"

name="password"

onChange={(e) => setPassword(e.target.value)}

/>

<div className="radio-group">

<input

type="radio"

name="typeofuser"

value="admin"

onChange={(e) => setTypeOfUser(e.target.value)}

/>

<span>Admin</span>

<input

type="radio"

name="typeofuser"

value="customer"

onChange={(e) => setTypeOfUser(e.target.value)}

/>

<span>Customer</span>

</div>

<input className="submit-button" type="submit" value="Submit" />

<input className="reset-button" type="reset" value="Reset" />

<Link className="signup-link" to="/signup">

SignUp

</Link>

</form>

</div>

);

}

export default Login;

**SignUp.js**

import React, { useState } from 'react';

import axios from 'axios';

import { Link } from 'react-router-dom';

import './Signup.css';

function SignUp() {

const [emailid, setEmailid] = useState('');

const [password, setPassword] = useState('');

const [typeOfUser, setTypeOfUser] = useState('customer');

const signIn = async (event) => {

event.preventDefault();

const login = { emailid, password, typeOfUser };

try {

const result = await axios.post('http://localhost:8080/Login/signUp', login);

console.log(result.data);

alert('Account Created Successfully!! Please sign in.');

} catch (ex) {

console.log(ex);

}

};

return (

<div className="signup-container">

<div className="signup-header">Account Creation</div>

<p>As a welcome bonus, you account will be credited £1000 to spend in the shop!</p>

<form className="signup-form" onSubmit={signIn}>

<label>Email</label>

<input

type="email"

name="emailid"

className="signup-input"

onChange={(e) => setEmailid(e.target.value)}

/>

<br />

<label>Password</label>

<input

type="password"

name="password"

className="signup-input"

onChange={(e) => setPassword(e.target.value)}

/>

<br />

{/\* <div className="radio-group">

<input

type="radio"

name="User"

value="customer"

onChange={(e) => setTypeOfUser(e.target.value)}

/>

<label>Customer</label>

</div> \*/}

<input type="submit" className="submit-button" value="Submit" />

<input type="reset" className="reset-button" value="Reset" />

<br />

<Link to="/Login" className="login-link">

Login

</Link>

</form>

</div>

);

}

export default SignUp;

**UserContext.js**

import { createContext, useContext, useState } from 'react';

const UserContext = createContext();

export const UserProvider = ({ children }) => {

const [userEmail, setUserEmail] = useState(null);

console.log(userEmail);

return (

<UserContext.Provider value={{ userEmail, setUserEmail }}>

{children}

</UserContext.Provider>

);

};

export const useUser = () => useContext(UserContext);

**App.js**

import "./bootstrap.min.css";

import './App.css';

import Login from './Login';

import SignUp from './SignUp';

import { Routes,Route, Link } from 'react-router-dom';

import Admin from './Components/Admin'

import Customer from './Components/Customer'

import Homepage from './Components/Homepage'

import AddMedicine from './Components/AddMedicine';

import ViewMedicine from './Components/ViewMedicine';

import BrowseMedical from "./Components/BrowseMedical";

import React, { useState } from "react";

import { UserProvider } from './UserContext';

import Orders from './Components/Orders'

import OrderSummary from './Components/OrderSummary'

import AllOrders from './Components/AllOrders'

import TopUpAccount from './Components/TopUpAccount'

function App() {

const [basket, setBasket] = useState([]);

return (

<div className="App">

<nav className="navbar navbar-expand navbar-dark bg-dark">

<Link to={"/"} className="navbar-brand">

Medicare Website - your One Stop for Medical supplies!

</Link>

<div className="navbar-nav mr-auto">

<li className ="nav-item">

<Link to={"/login"} className="nav-link">

Login

</Link>

</li>

<li className ="nav-item">

<Link to={"/signup"} className="nav-link">

Signup

</Link>

</li>

</div>

</nav>

<UserProvider>

<Routes>

<Route path="/" element={<Homepage/>}/>

<Route path="/signup" element={<SignUp/>}/>

<Route path="/login" element={<Login/>}/>

<Route path='/admin' element={<Admin/>}>

<Route path ="addMedicine" element={<AddMedicine/>}/>

<Route path ="viewMedicine" element={<ViewMedicine/>}/>

<Route path ="allOrders" element={<AllOrders/>}/>

</Route>

<Route path="/customer" element={<Customer basket={basket} setBasket={setBasket} />}>

<Route path="browseMedical" element={<BrowseMedical basket={basket} setBasket={setBasket} />} />

<Route path="orders" element={<Orders/>}/>

<Route path="topUpAccount" element={<TopUpAccount/>}/>

</Route>

<Route path="/customer/browseMedical/order-summary" element={<OrderSummary/>} />

{/\* <Route path="/customer/checkout" element={<Checkout basketItems={basket} />} /> \*/}

</Routes>

</UserProvider>

</div>

);

}

export default App;

**Index.js**

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

import { BrowserRouter } from 'react-router-dom';

import { MemoryRouter } from 'react-router-dom';

import { UserProvider } from './UserContext';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<React.StrictMode>

<MemoryRouter>

<UserProvider>

<App />

</UserProvider>

</MemoryRouter>

</React.StrictMode>

);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

reportWebVitals();