UE21CS352B - Object Oriented Analysis & Design using Java

Mini Project Report

Pharmacy Management System

Submitted by:

Mohammed Taha Roshan	**PES1UG21CS339**
Mohd Faizan Khan	**PES2UG21CS302**
NP. Yashwanth	**PES1UG21CS352**
M. Surya Gagan Reddy	**PES1UG21CS347**

6th Semester F Section

Prof. Bhargavi Mokashi

January - May 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Problem Statement: Pharmacy Management System

This project is a comprehensive Pharmacy Management System which facilitates efficient operations and enhanced user experiences. The system empowers customers to browse, select, and purchase Medicines and other medicinal products online seamlessly while enabling administrators to manage products, orders, and inventory effectively. Specifically, administrators will have the capability to oversee product inventory, update stock levels, add/remove products, and manage order .

Key Features:

1)User Interface:

Created a user-friendly web interface that enables customers to effortlessly browse products, add items to their carts, and complete orders.

Designed an intuitive dashboard for administrators to efficiently manage products, orders, and inventory

Here's a rephrased version to avoid plagiarism:

2. Product Administration:

Empowers administrators to introduce new products, modify existing product details like name, description, and price, and eliminate products when necessary.

Implemented sorting and filtering options to aid customers in finding products based on categories, brands, or keywords.

3. Cart and Order Supervision:

Facilitates customers in adding products to their carts, reviewing cart contents, and securely completing orders.

Equips administrators to efficiently view, process, and manage orders, covering order confirmation.

4. Inventory Control for Administrators:

Empowers administrators to manage product inventory effectively, including stock level updates, monitoring product availability.

Offers inventory management tools such as adding new stock, adjusting stock levels, and removing outdated products from inventory.

5. User Verification and Protection:

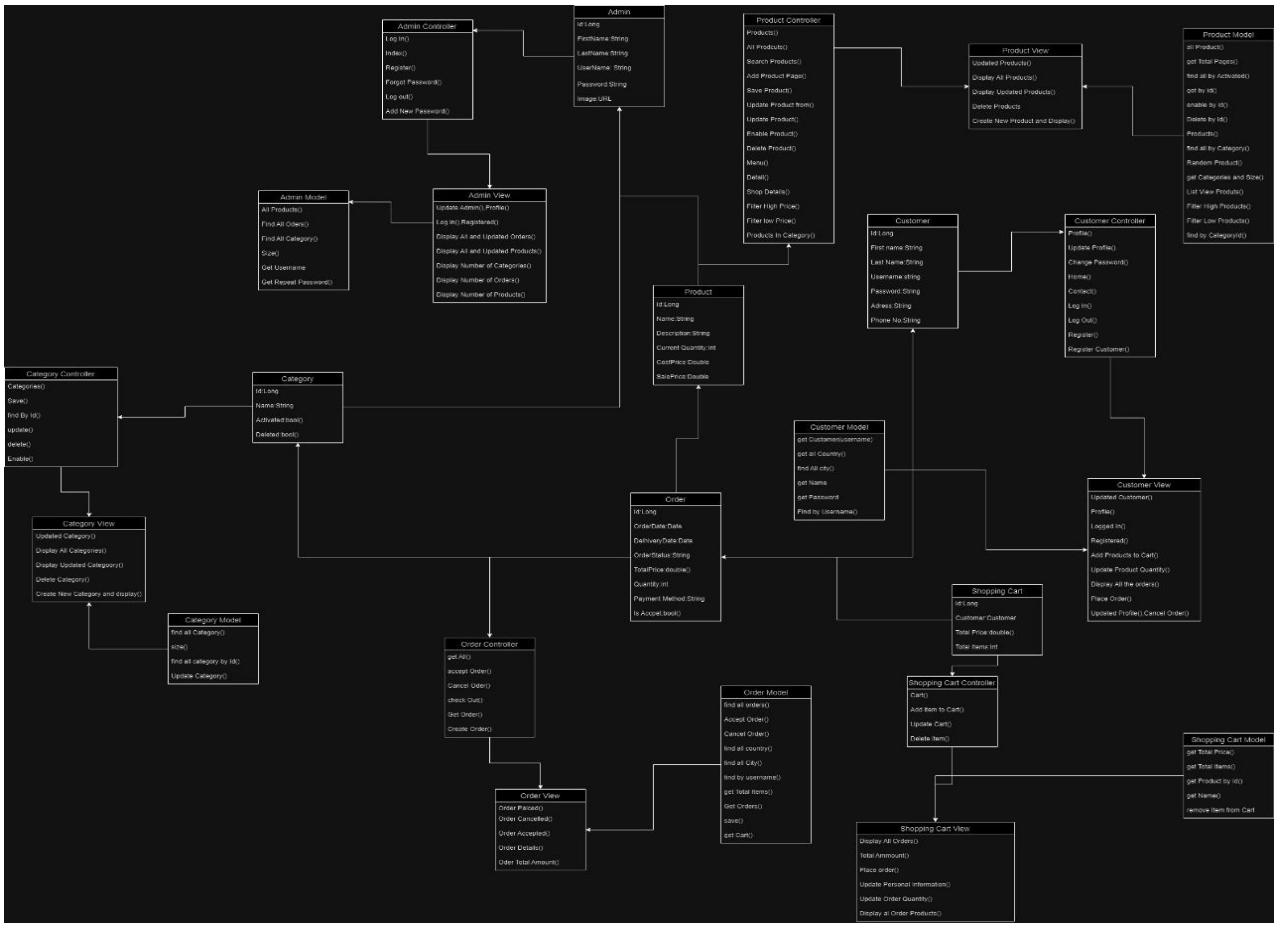
Implements distinct and secure authentication procedures for users and administrators, ensuring that sensitive functions are appropriately controlled.

Allows customers to create accounts, manage profiles, and securely store payment information for future purchases.

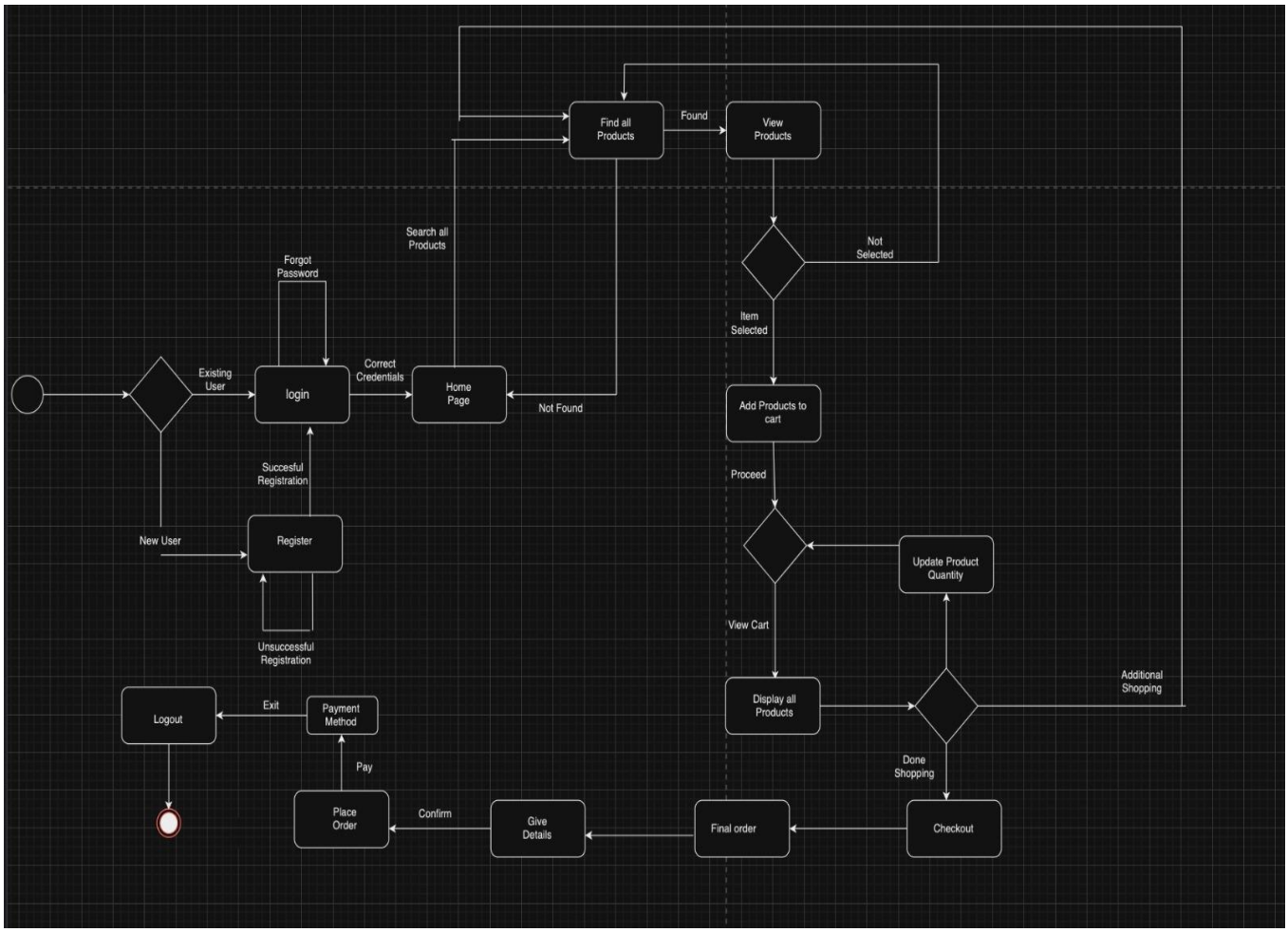
Grants administrators exclusive access to administrative functions through a distinct login process, maintaining a clear distinction between user and admin privileges.

UML DIAGRAMS:

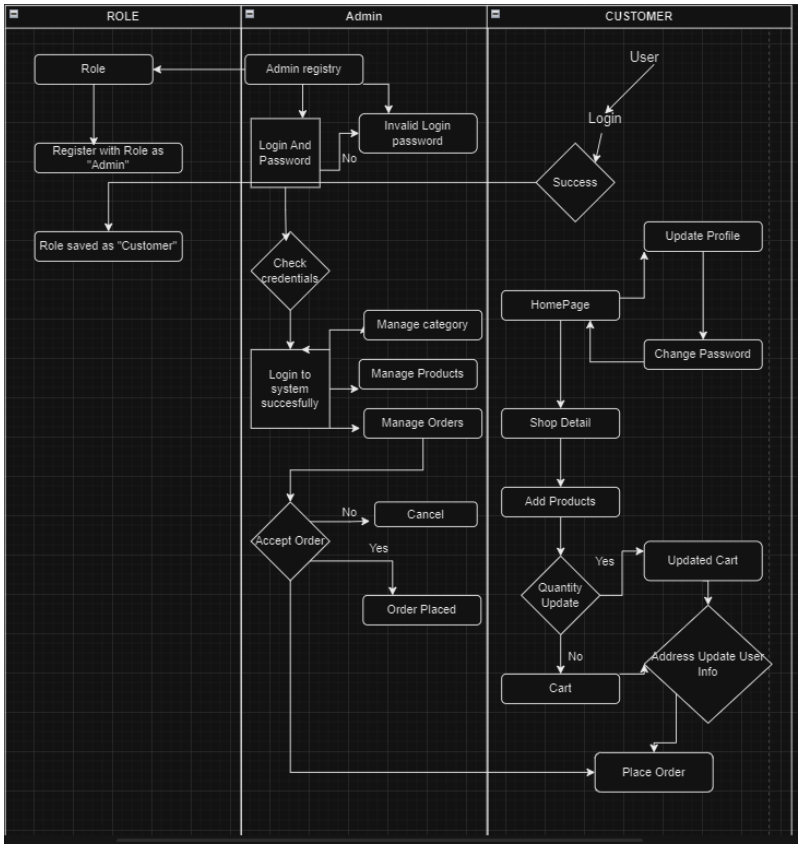
Class Diagram:



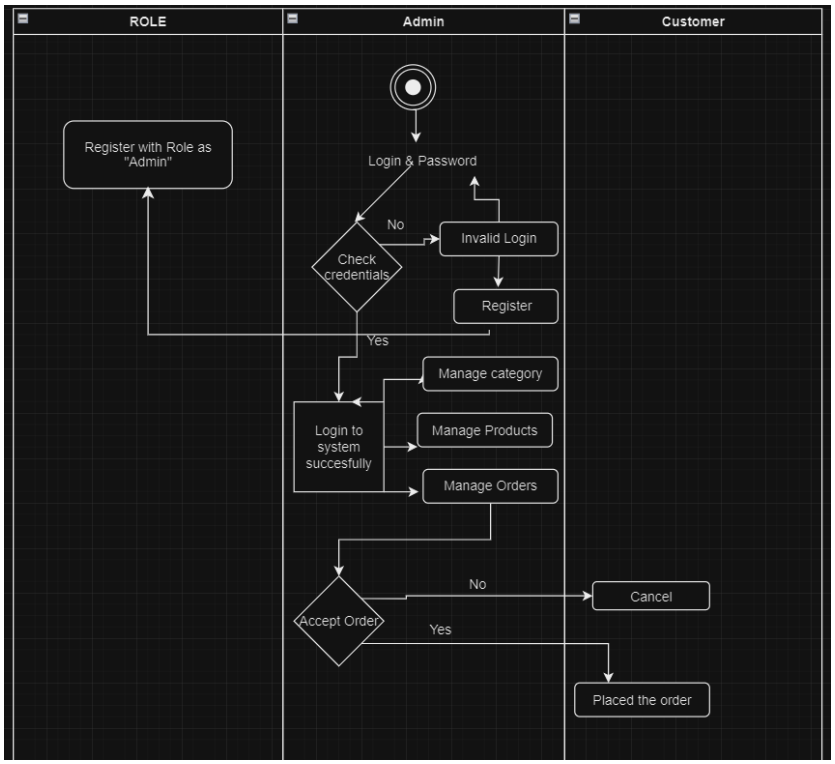
State Diagram:



Activity Diagram(major):



Activity Diagram(Minor):



Principles:

SRP-Single Responsibility Principle

```java
@Controller
@RequiredArgsConstructor
public class CategoryController {

    private final CategoryService categoryService;

    @GetMapping("/categories")
    public String categories(Model model) {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        if (authentication == null || authentication instanceof AnonymousAuthenticationToken) {
            return "redirect:/login";
        }
        model.addAttribute("title", "Manage Category");
        List<Category> categories = categoryService.findAll();
        model.addAttribute("categories", categories);
        model.addAttribute("size", categories.size());
        model.addAttribute("categoryNew", new Category());
        return "categories";
    }

    @PostMapping("/save-category")
    public String save(@ModelAttribute("categoryNew") Category category, Model model, RedirectAttributes redirectAttributes) {
        try {
            categoryService.save(category);
            model.addAttribute("categoryNew", category);
            redirectAttributes.addFlashAttribute("success", "Add successfully!");
        } catch (DataIntegrityViolationException e1) {
            e1.printStackTrace();
            redirectAttributes.addFlashAttribute("error", "Duplicate name of category, please check again!");
        } catch (Exception e2) {
            e2.printStackTrace();
            model.addAttribute("categoryNew", category);
            redirectAttributes.addFlashAttribute("error",
                    "Error server");
        }
        return "redirect:/categories";
    }

    @RequestMapping(value = "/findById", method = {RequestMethod.PUT, RequestMethod.GET})
    @ResponseBody
    public Optional<Category> findById(Long id) {
        return categoryService.findById(id);
    }

    @GetMapping("/update-category")
    public String update(Category category, RedirectAttributes redirectAttributes) {
        try {
            categoryService.update(category);
            redirectAttributes.addFlashAttribute("success", "Update successfully!");
        } catch (DataIntegrityViolationException e1) {
            e1.printStackTrace();
            redirectAttributes.addFlashAttribute("error", "Duplicate name of category, please check again!");
        } catch (Exception e2) {
            e2.printStackTrace();
            redirectAttributes.addFlashAttribute("error", "Error from server or duplicate name of category, please check again!");
        }
        return "redirect:/categories";
    }


    @RequestMapping(value = "/delete-category", method = {RequestMethod.GET, RequestMethod.PUT})
    public String delete(Long id, RedirectAttributes redirectAttributes) {
        try {
            categoryService.deleteById(id);
            redirectAttributes.addFlashAttribute("success", "Deleted successfully!");
        } catch (DataIntegrityViolationException e1) {
            e1.printStackTrace();
            redirectAttributes.addFlashAttribute("error", "Duplicate name of category, please check again!");
        } catch (Exception e2) {
            e2.printStackTrace();
            redirectAttributes.addFlashAttribute("error", "Error server");
        }
        return "redirect:/categories";
    }

    @RequestMapping(value = "/enable-category", method = {RequestMethod.PUT, RequestMethod.GET})
    public String enable(Long id, RedirectAttributes redirectAttributes) {
        try {
            categoryService.enableById(id);
            redirectAttributes.addFlashAttribute("success", "Enable successfully");
        } catch (DataIntegrityViolationException e1) {
            e1.printStackTrace();
            redirectAttributes.addFlashAttribute("error", "Duplicate name of category, please check again!");
        } catch (Exception e2) {
            e2.printStackTrace();
            redirectAttributes.addFlashAttribute("error", "Error server");
        }
        return "redirect:/categories";
    }


}
```

Category Management: The primary responsibility of the CategoryController class is to handle category-related operations such as displaying categories, adding new categories, updating existing categories, and deleting categories. It's responsible for managing the CRUD (Create, Read, Update, Delete) operations related to categories.

While the CategoryController class does perform multiple tasks, they all revolve around the management of categories within the application. Therefore, it follows the SRP because all its responsibilities are related to the same conceptual entity (category management).

OCP:

Interface registerlogin: The register login interface defines two methods: login() and register(). This interface is open for extension because new classes can implement it to provide different implementations for login and registration functionalities without modifying the interface itself. For example, both LoginController and AuthController implement this interface to provide login and registration functionalities.

LoginController and AuthController Classes: Both classes provide implementations for the login() and register() methods defined in the register login interface. These classes are closed for modification because their functionalities can be extended by adding new methods or classes, without needing to modify the existing code. For example, if you want to add additional authentication methods or registration processes, you can create new classes or methods without altering the existing controllers.

Methods for Handling Login and Registration: The methods login() and register() in both LoginController and AuthController classes handle the login and registration functionalities, respectively. These methods are specific to their respective controllers and can be extended or replaced by subclassing or implementing new controllers.

Overall, the provided code demonstrates adherence to the Open/Closed Principle by allowing for extension of login and registration functionalities through subclassing or implementing new classes, without requiring modifications to the existing codebase we can add forgot password and many other ethods to register login interface. New functionalities can be added without altering the existing controllers, making the codebase more maintainable and extensible.

Registerlogin.java

```
public interface registerlogin {
      public String login(Model model);
      public String register(Model model);
}
```

LoginController.java

```
@Controller
@RequiredArgsConstructor
public class LoginController implements registerlogin {
    private final CustomerService customerService;
    private final BCryptPasswordEncoder passwordEncoder;
```

```
    @RequestMapping(value = "/login", method = RequestMethod.GET)
    public String login(Model model) {
        model.addAttribute("title", "Login Page");
        model.addAttribute("page", "Home");
        return "login";
    }


    @GetMapping("/register")
    public String register(Model model) {
        model.addAttribute("title", "Register");
        model.addAttribute("page", "Register");
        model.addAttribute("customerDto", new CustomerDto());
        return "register";
    }


    @PostMapping("/do-register")
    public String registerCustomer(@Valid @ModelAttribute("customerDto")
CustomerDto customerDto,
                                   BindingResult result,
                                   Model model) {
        try {
            if (result.hasErrors()) {
                model.addAttribute("customerDto", customerDto);
                return "register";
            }
            String username = customerDto.getUsername();
            Customer customer = customerService.findByUsername(username);
            if (customer != null) {
                model.addAttribute("customerDto", customerDto);
                model.addAttribute("error", "Email has been register!");
                return "register";
            }
            if
(customerDto.getPassword().equals(customerDto.getConfirmPassword())) {

customerDto.setPassword(passwordEncoder.encode(customerDto.getPassword()));
                customerService.save(customerDto);
                model.addAttribute("success", "Register successfully!");
            } else {
                model.addAttribute("error", "Password is incorrect");
                model.addAttribute("customerDto", customerDto);
                return "register";
            }
        } catch (Exception e) {
            e.printStackTrace();
            model.addAttribute("error", "Server is error, try again
later!");
        }
        return "register";
    }


}
```

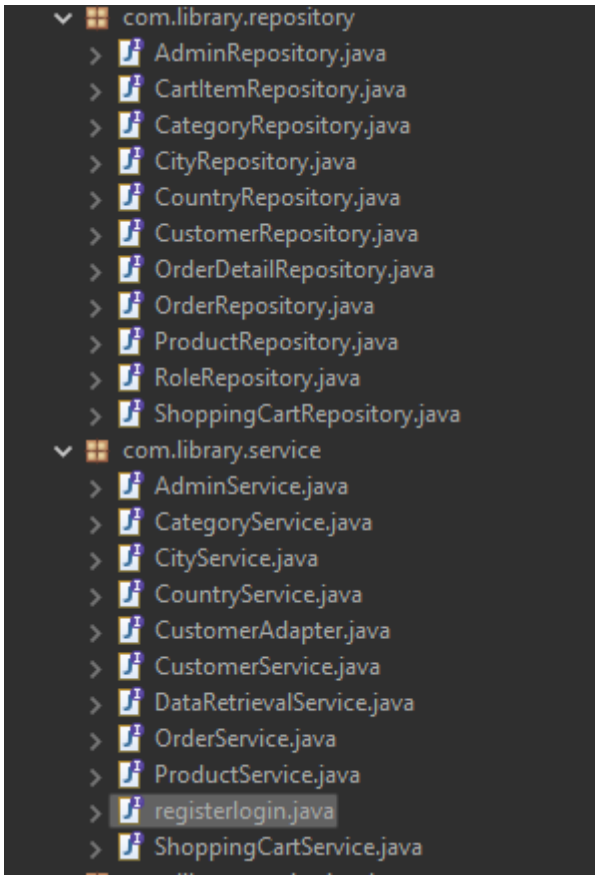
AuthController.java

```
@Controller
@RequiredArgsConstructor
```

```
public class AuthController implements registerlogin{
    private final AdminService adminService;
    private final ProductService productService;
    private final BCryptPasswordEncoder passwordEncoder;
    private final OrderService orderService;
    private final CategoryService categoryService;
    @RequestMapping("/login")
    public String login(Model model) {
        model.addAttribute("title", "Login Page");
        return "login";
    }

    @RequestMapping("/index")
    public String index(Model model) {
        List<ProductDto> products = productService.allProduct();
        List<Order> orders = orderService.findALlOrders();
        List<Category> category=categoryService.findALl();
      model.addAttribute("orders", orders.size());
      model.addAttribute("categories",category.size());
      model.addAttribute("products", products.size());
        model.addAttribute("title", "Home Page");
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        if (authentication == null || authentication instanceof
AnonymousAuthenticationToken) {
            return "redirect:/login";
        }
        return "index";
    }

    @GetMapping("/register")
    public String register(Model model) {
        model.addAttribute("title", "Register");
        model.addAttribute("adminDto", new AdminDto());
        return "register";
    }

    @GetMapping("/forgot-password")
    public String forgotPassword(Model model) {
        model.addAttribute("title", "Forgot Password");
        return "forgot-password";
    }

    @GetMapping("/logout")
    public String logout(Model model) {
        return "redirect:/login";
    }

    @PostMapping("/register-new")
    public String addNewAdmin(@Valid @ModelAttribute("adminDto") AdminDto
adminDto,
                              BindingResult result,
                              Model model) {

        try {

            if (result.hasErrors()) {
                model.addAttribute("adminDto", adminDto);
                return "register";
            }
            String username = adminDto.getUsername();
```

```
            Admin admin = adminService.findByUsername(username);
            if (admin != null) {
                model.addAttribute("adminDto", adminDto);
                System.out.println("admin not null");
                model.addAttribute("emailError", "Your email has been
registered!");
                return "register";
            }
            if
(adminDto.getPassword().equals(adminDto.getRepeatPassword())) {

adminDto.setPassword(passwordEncoder.encode(adminDto.getPassword()));
                adminService.save(adminDto);
                System.out.println("success");
                model.addAttribute("success", "Register successfully!");
                model.addAttribute("adminDto", adminDto);
            } else {
                model.addAttribute("adminDto", adminDto);
                model.addAttribute("passwordError", "Your password maybe
wrong! Check again!");
                System.out.println("password not same");
            }
        } catch (Exception e) {
            e.printStackTrace();
            model.addAttribute("errors", "The server has been wrong!");
        }
        return "register";

    }
}
```

ISP-Interface Segregation Principle



For each specific functionality or requirement, we design separate interfaces. For example, if we have an operation like finding an admin by username, we define a specific interface such as AdminRepo that includes methods related to admin-related queries only. This approach ensures that each interface is focused on a particular set of functionalities, promoting better code organization and adherence to the ISP.

LSP: Liskov Substitution Principle

The Dto interface defines common behavior for all DTO classes.Subclasses like CartItemDto, ProductDto, etc., extend the Dto interface, inheriting its common behavior and adding specialized functionality as needed.Which is later on used in service and that is implemented in the serviceImpl.

Dto.java

```
public interface dto {
    // Define common behavior for DTOs
    Long getId();
    void setId(Long id);
}
```

CartItemDto.java

```
public class CartItemDto implements dto {
    private Long id;
```

```
    private ShoppingCartDto cart;

    private ProductDto product;

    private int quantity;

    private double unitPrice;

}
```

ProductDto.java

```
public class ProductDto implements dto{
    private Long id;
    private String name;
    private String description;
    private int currentQuantity;
    private double costPrice;
    private double salePrice;
    private String image;
    private Category category;
    private boolean activated;
    private boolean deleted;
    private String currentPage;
}
```

CustomerDto

```
public class CustomerDto {
    @Size(min = 3, max = 10, message = "First name contains 3-10
characters")
    private String firstName;

    @Size(min = 3, max = 10, message = "Last name contains 3-10
characters")
    private String lastName;
    private String username;
    @Size(min = 3, max = 15, message = "Password contains 3-10 characters")
    private String password;

    @Size(min = 10, max = 15, message = "Phone number contains 10-15
characters")
    private String phoneNumber;

    private String address;
    private String confirmPassword;
    private City city;
    private String image;
    private String country;

}
```

CategoryDto.java

```
public class CategoryDto implements dto {
    private Long id;
```

```
    private String name;
    private Long productSize;
}
```

ProductServiceImpl it implements all the dto

```
@Service
@RequiredArgsConstructor
public class ProductServiceImpl implements ProductService {
    private final ProductRepository productRepository;


    private final ImageUpload imageUpload;

    @Override
    public List<Product> findAll() {
        return productRepository.findAll();
    }

    @Override
    public List<ProductDto> products() {
        return transferData(productRepository.getAllProduct());
    }

    @Override
    public List<ProductDto> allProduct() {
        List<Product> products = productRepository.findAll();
        List<ProductDto> productDtos = transferData(products);
        return productDtos;
    }

    @Override
    public Product save(MultipartFile imageProduct, ProductDto productDto)
{
        Product product = new Product();
        try {
            if (imageProduct == null) {
                product.setImage(null);
            } else {
                imageUpload.uploadFile(imageProduct);

product.setImage(Base64.getEncoder().encodeToString(imageProduct.getBytes()
));
            }
            product.setName(productDto.getName());
            product.setDescription(productDto.getDescription());
            product.setCurrentQuantity(productDto.getCurrentQuantity());
            product.setCostPrice(productDto.getCostPrice());
            product.setCategory(productDto.getCategory());
            product.set_deleted(false);
            product.set_activated(true);
            return productRepository.save(product);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    @Override
    public Product update(MultipartFile imageProduct, ProductDto
productDto) {
        try {
```

```
            Product productUpdate =
productRepository.getReferenceById(productDto.getId());
            if (imageProduct.getBytes().length > 0) {
                if (imageUpload.checkExist(imageProduct)) {
                    productUpdate.setImage(productUpdate.getImage());
                } else {
                    imageUpload.uploadFile(imageProduct);

productUpdate.setImage(Base64.getEncoder().encodeToString(imageProduct.getB
ytes()));
                }
            }
            productUpdate.setCategory(productDto.getCategory());
            productUpdate.setId(productUpdate.getId());
            productUpdate.setName(productDto.getName());
            productUpdate.setDescription(productDto.getDescription());
            productUpdate.setCostPrice(productDto.getCostPrice());
            productUpdate.setSalePrice(productDto.getSalePrice());

productUpdate.setCurrentQuantity(productDto.getCurrentQuantity());
            return productRepository.save(productUpdate);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    @Override
    public void enableById(Long id) {
        Product product = productRepository.getById(id);
        product.set_activated(true);
        product.set_deleted(false);
        productRepository.save(product);
    }

    @Override
    public void deleteById(Long id) {
        Product product = productRepository.getById(id);
        product.set_deleted(true);
        product.set_activated(false);
        productRepository.save(product);
    }

    @Override
    public ProductDto getById(Long id) {
        ProductDto productDto = new ProductDto();
        Product product = productRepository.getById(id);
        productDto.setId(product.getId());
        productDto.setName(product.getName());
        productDto.setDescription(product.getDescription());
        productDto.setCostPrice(product.getCostPrice());
        productDto.setSalePrice(product.getSalePrice());
        productDto.setCurrentQuantity(product.getCurrentQuantity());
        productDto.setCategory(product.getCategory());
        productDto.setImage(product.getImage());
        return productDto;
    }

    @Override
    public Product findById(Long id) {
        return productRepository.findById(id).get();
```

```
    }

    @Override
    public List<ProductDto> randomProduct() {
        return transferData(productRepository.randomProduct());
    }

    @Override
    public Page<ProductDto> searchProducts(int pageNo, String keyword) {
        List<Product> products =
productRepository.findAllByNameOrDescription(keyword);
        List<ProductDto> productDtoList = transferData(products);
        Pageable pageable = PageRequest.of(pageNo, 5);
        Page<ProductDto> dtoPage = toPage(productDtoList, pageable);
        return dtoPage;
    }

    @Override
    public Page<ProductDto> getAllProducts(int pageNo) {
        Pageable pageable = PageRequest.of(pageNo, 6);
        List<ProductDto> productDtoLists = this.allProduct();
        Page<ProductDto> productDtoPage = toPage(productDtoLists,
pageable);
        return productDtoPage;
    }

    @Override
    public Page<ProductDto> getAllProductsForCustomer(int pageNo) {
        return null;
    }

    @Override
    public List<ProductDto> findAllByCategory(String category) {
        return transferData(productRepository.findAllByCategory(category));
    }

    @Override
    public List<ProductDto> filterHighProducts() {
        return transferData(productRepository.filterHighProducts());
    }

    @Override
    public List<ProductDto> filterLowerProducts() {
        return transferData(productRepository.filterLowerProducts());
    }

    @Override
    public List<ProductDto> listViewProducts() {
        return transferData(productRepository.listViewProduct());
    }

    @Override
    public List<ProductDto> findByCategoryId(Long id) {
        return transferData(productRepository.getProductByCategoryId(id));
    }

    @Override
    public List<ProductDto> searchProducts(String keyword) {
        return transferData(productRepository.searchProducts(keyword));
    }
```

```
    private Page toPage(List list, Pageable pageable) {
        if (pageable.getOffset() >= list.size()) {
            return Page.empty();
        }
        int startIndex = (int) pageable.getOffset();
        int endIndex = ((pageable.getOffset() + pageable.getPageSize()) >
list.size())
                ? list.size()
                : (int) (pageable.getOffset() + pageable.getPageSize());
        List subList = list.subList(startIndex, endIndex);
        return new PageImpl(subList, pageable, list.size());
    }

    private List<ProductDto> transferData(List<Product> products) {
        List<ProductDto> productDtos = new ArrayList<>();
        for (Product product : products) {
            ProductDto productDto = new ProductDto();
            productDto.setId(product.getId());
            productDto.setName(product.getName());
            productDto.setCurrentQuantity(product.getCurrentQuantity());
            productDto.setCostPrice(product.getCostPrice());
            productDto.setSalePrice(product.getSalePrice());
            productDto.setDescription(product.getDescription());
            productDto.setImage(product.getImage());
            productDto.setCategory(product.getCategory());
            productDto.setActivated(product.is_activated());
            productDto.setDeleted(product.is_deleted());
            productDtos.add(productDto);
        }
        return productDtos;
    }

}
```

DIP: Dependency Inversion Principle

DataRetrievalService.java

```
public interface DataRetrievalService<T> {
    List<T> findAll();
}
```

CityServiceImpl.java

```
public class CityServiceImpl implements DataRetrievalService<City> {
    private final CityRepository cityRepository;

    @Override
    public List<City> findAll() {
        return cityRepository.findAll();
    }
}
```

CountryServiceImpl

```
public class CountryServiceImpl implements DataRetrievalService<Country> {
```

```
    private final CountryRepository countryRepository;

    @Override
    public List<Country> findAll() {
        return countryRepository.findAll();
    }
}
```

Abstraction through Interface (DataRetrievalService):

The DataRetrievalService interface serves as an abstraction that defines a contract for data retrieval operations.

By using an interface, the implementation classes (CityServiceImpl and CountryServiceImpl) are decoupled from the specific data retrieval logic.

Dependency Inversion:

Both CityServiceImpl and CountryServiceImpl depend on the DataRetrievalService interface rather than concrete implementations (CityRepository and CountryRepository).

This inversion of dependencies allows the implementation classes to depend on abstractions rather than concrete implementations, promoting flexibility and extensibility.

Ease of Extension and Modification:

If a new type of data retrieval service needs to be added (e.g., RegionServiceImpl), it can simply implement the DataRetrievalService interface without affecting existing code.

Similarly, if there are changes or updates to the data retrieval logic (e.g., switching from CityRepository to a different repository implementation), it can be done without modifying the implementation classes.

Overall, the code demonstrates adherence to the Dependency Inversion Principle by relying on abstractions (interfaces) and promoting loose coupling between components. It allows for easier maintenance, extension, and testing of the codebase.

Patterns:

Singleton:

AdminServiceImpl

We have used Volatile and Syncronization

```
@Service
@RequiredArgsConstructor
public class AdminServiceImpl implements AdminService {
    private static volatile AdminServiceImpl instance;

    private final AdminRepository adminRepository;
    private final RoleRepository roleRepository;

    public static synchronized AdminServiceImpl getInstance(AdminRepository
adminRepository, RoleRepository roleRepository) {
        if (instance == null) {
            instance = new AdminServiceImpl(adminRepository,
roleRepository);
        }
        return instance;
    }

    @Override
    public Admin save(AdminDto adminDto) {
        Admin admin = new Admin();
        admin.setFirstName(adminDto.getFirstName());
        admin.setLastName(adminDto.getLastName());
        admin.setUsername(adminDto.getUsername());
        admin.setPassword(adminDto.getPassword());
        admin.setRoles(Arrays.asList(roleRepository.findByName("ADMIN")));
        return adminRepository.save(admin);
    }

    @Override
    public Admin findByUsername(String username) {
        return adminRepository.findByUsername(username);
    }
}
```

Builder:

Category.java

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "categories", uniqueConstraints =
@UniqueConstraint(columnNames = "name"))
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "category_id")
    private Long id;
```

```java
    private String name;
    @Column(name = "is_activated")
    private boolean activated;
    @Column(name = "is_deleted")
    private boolean deleted;

    // Constructor, getters, setters, etc.

    // Builder class
    public static class Builder {
        private Long id;
        private String name;
        private boolean activated;
        private boolean deleted;

        public Builder id(Long id) {
            this.id = id;
            return this;
        }

        public Builder name(String name) {
            this.name = name;
            return this;
        }

        public Builder activated(boolean activated) {
            this.activated = activated;
            return this;
        }

        public Builder deleted(boolean deleted) {
            this.deleted = deleted;
            return this;
        }

        public Category build() {
            Category category = new Category();
            category.setId(this.id);
            category.setName(this.name);
            category.setActivated(this.activated);
            category.setDeleted(this.deleted);
            return category;
        }
    }

    // Getter for builder
    public static Builder builder() {
        return new Builder();
    }
}
```

CategoryServiceImpl.java

```java
@Service
@RequiredArgsConstructor
public class CategoryServiceImpl implements CategoryService {
    private final CategoryRepository categoryRepository;

    @Override
```

```java
    public Category save(Category category) {
        Category categoryToSave = Category.builder()
                .name(category.getName())
                .activated(true)
                .deleted(false)
                .build();
        return categoryRepository.save(categoryToSave);

    }

    @Override
    public Category update(Category category) {
        Category categoryUpdate =
categoryRepository.getReferenceById(category.getId());
        categoryUpdate.setName(category.getName());
        return categoryRepository.save(categoryUpdate);
    }

    @Override
    public List<Category> findAllByActivatedTrue() {
        return categoryRepository.findAllByActivatedTrue();
    }

    @Override
    public List<Category> findALl() {
        return categoryRepository.findAll();
    }

    @Override
    public Optional<Category> findById(Long id) {
        return categoryRepository.findById(id);
    }

    @Override
    public void deleteById(Long id) {
        Category category = categoryRepository.getById(id);
        category.setActivated(false);
        category.setDeleted(true);
        categoryRepository.save(category);
    }

    @Override
    public void enableById(Long id) {
        Category category = categoryRepository.getById(id);
        category.setActivated(true);
        category.setDeleted(false);
        categoryRepository.save(category);
    }

    @Override
    public List<CategoryDto> getCategoriesAndSize() {
        List<CategoryDto> categories =
categoryRepository.getCategoriesAndSize();
        return categories;
    }

}
```

Factory:

OrderServiceFactory.java

```
public interface OrderServiceFactory {
    OrderService createOrderService();
}
```

OrderController.java

```
@Controller
@RequiredArgsConstructor
public class OrderController {
    private final CustomerService customerService;
//    private final OrderService orderService;
    private final OrderServiceFactory orderServiceFactory;
    private OrderService orderService; // No need to initialize here

    // Use @PostConstruct to initialize orderService
    @PostConstruct
    public void init() {
        orderService = orderServiceFactory.createOrderService();
    }
    private final ShoppingCartService cartService;

    private final DataRetrievalService<Country> countryService;
    private final DataRetrievalService<City> cityService;

    @GetMapping("/check-out")
    public String checkOut(Principal principal, Model model) {
        if (principal == null) {
            return "redirect:/login";
        } else {
            CustomerDto customer =
customerService.getCustomer(principal.getName());
            if (customer.getAddress() == null || customer.getCity() == null
|| customer.getPhoneNumber() == null) {
                model.addAttribute("information", "You need update your
information before check out");
                List<Country> countryList = countryService.findAll();
                List<City> cities = cityService.findAll();
                model.addAttribute("customer", customer);
                model.addAttribute("cities", cities);
                model.addAttribute("countries", countryList);
                model.addAttribute("title", "Profile");
                model.addAttribute("page", "Profile");
                return "customer-information";
            } else {
                ShoppingCart cart =
customerService.findByUsername(principal.getName()).getCart();
                model.addAttribute("customer", customer);
                model.addAttribute("title", "Check-Out");
                model.addAttribute("page", "Check-Out");
                model.addAttribute("shoppingCart", cart);
                model.addAttribute("grandTotal", cart.getTotalItems());
                return "checkout";
            }
        }
    }
```

```
    @GetMapping("/orders")
    public String getOrders(Model model, Principal principal) {
        if (principal == null) {
            return "redirect:/login";
        } else {
            Customer customer =
customerService.findByUsername(principal.getName());
            List<Order> orderList = customer.getOrders();
            model.addAttribute("orders", orderList);
            model.addAttribute("title", "Order");
            model.addAttribute("page", "Order");
            return "order";
        }
    }

    @RequestMapping(value = "/cancel-order", method = {RequestMethod.PUT,
RequestMethod.GET})
    public String cancelOrder(Long id, RedirectAttributes attributes) {
        orderService.cancelOrder(id);
        attributes.addFlashAttribute("success", "Cancel order
successfully!");
        return "redirect:/orders";
    }


    @RequestMapping(value = "/add-order", method = {RequestMethod.POST})
    public String createOrder(Principal principal,
                              Model model,
                              HttpSession session) {
        if (principal == null) {
            return "redirect:/login";
        } else {
            Customer customer =
customerService.findByUsername(principal.getName());
            ShoppingCart cart = customer.getCart();
            Order order = orderService.save(cart);
            session.removeAttribute("totalItems");
            model.addAttribute("order", order);
            model.addAttribute("title", "Order Detail");
            model.addAttribute("page", "Order Detail");
            model.addAttribute("success", "Add order successfully");
            return "order-detail";
        }
    }

}
```

OrderServiceFactory Interface:

The OrderServiceFactory interface defines a contract for creating OrderService instances.

It declares a single method createOrderService() responsible for creating an OrderService.

OrderController's Dependency on OrderServiceFactory:

The OrderController class depends on the OrderServiceFactory interface rather than concrete implementations of OrderService.

It receives an instance of OrderServiceFactory through constructor injection (OrderController(OrderServiceFactory orderServiceFactory)).

Creation of OrderService:

Instead of directly instantiating an OrderService, the OrderController class delegates the responsibility of creating OrderService instances to the OrderServiceFactory.

During initialization (init() method annotated with @PostConstruct), the OrderController invokes the createOrderService() method of the injected OrderServiceFactory to obtain an OrderService instance.

This approach allows for flexibility in creating different implementations of OrderService without modifying the OrderController class.

Encapsulation of Creation Logic:

The creation logic for OrderService is encapsulated within the implementation of OrderServiceFactory.

This encapsulation hides the details of how OrderService instances are created, promoting loose coupling and adherence to the Open/Closed Principle.

Flexibility and Extensibility:

The Factory Pattern allows for easy extension and modification of the system by introducing new implementations of OrderService without impacting the existing code.

Different implementations of OrderService can be created and plugged into the application by providing corresponding implementations of OrderServiceFactory.

In summary, the provided code follows the Factory Pattern by abstracting the creation of OrderService instances behind a factory interface (OrderServiceFactory). This approach enhances flexibility, promotes encapsulation, and facilitates easier maintenance and extension of the codebase

Adapter:

CustomerAdapter.java

```
public interface CustomerAdapter {
    CustomerDto customerToDto(Customer customer);
    Customer dtoToCustomer(CustomerDto dto);
}
```

CustomerAdapterImpl.java

```
@Service
@RequiredArgsConstructor
```

```java
public class CustomerAdapterImpl implements CustomerAdapter {
    private final RoleRepository roleRepository;

    @Override
    public CustomerDto customerToDto(Customer customer) {
        CustomerDto customerDto = new CustomerDto();
        customerDto.setFirstName(customer.getFirstName());
        customerDto.setLastName(customer.getLastName());
        customerDto.setUsername(customer.getUsername());
        customerDto.setPassword(customer.getPassword());
        customerDto.setAddress(customer.getAddress());
        customerDto.setPhoneNumber(customer.getPhoneNumber());
        customerDto.setCity(customer.getCity());
        customerDto.setCountry(customer.getCountry());
        return customerDto;
    }

    @Override
    public Customer dtoToCustomer(CustomerDto dto) {
        Customer customer = new Customer();
        customer.setFirstName(dto.getFirstName());
        customer.setLastName(dto.getLastName());
        customer.setPassword(dto.getPassword());
        customer.setUsername(dto.getUsername());

customer.setRoles(Collections.singletonList(roleRepository.findByName("CUST
OMER")));
        customer.setAddress(dto.getAddress());
        customer.setPhoneNumber(dto.getPhoneNumber());
        customer.setCity(dto.getCity());
        customer.setCountry(dto.getCountry());
        return customer;
    }
}
```

CustomerServiceImpl

```java
@Service
@RequiredArgsConstructor
public class CustomerServiceImpl implements CustomerService {
    private final CustomerRepository customerRepository;
    private final CustomerAdapter customerAdapter;

    @Override
    public Customer save(CustomerDto customerDto) {
        Customer customer = customerAdapter.dtoToCustomer(customerDto);
        return customerRepository.save(customer);
    }

    @Override
    public Customer findByUsername(String username) {
        return customerRepository.findByUsername(username);
    }

    @Override
    public CustomerDto getCustomer(String username) {
        Customer customer = customerRepository.findByUsername(username);
        return customerAdapter.customerToDto(customer);
    }
```

```
    @Override
    public Customer changePass(CustomerDto customerDto) {
        Customer customer =
customerRepository.findByUsername(customerDto.getUsername());
        customer.setPassword(customerDto.getPassword());
        return customerRepository.save(customer);
    }

    @Override
    public Customer update(CustomerDto dto) {
        Customer customer =
customerRepository.findByUsername(dto.getUsername());
        customer.setAddress(dto.getAddress());
        customer.setCity(dto.getCity());
        customer.setCountry(dto.getCountry());
        customer.setPhoneNumber(dto.getPhoneNumber());
        return customerRepository.save(customer);
    }
}
```

Adapter Pattern:

Definition: The Adapter Pattern allows incompatible interfaces to work together by providing a bridge between them.

In the Code:

The CustomerAdapter interface defines methods customerToDto() and dtoToCustomer() for converting between Customer objects and CustomerDto objects.

The CustomerAdapterImpl class implements the CustomerAdapter interface and provides concrete implementations of the conversion methods.

This pattern allows the CustomerService implementation (CustomerServiceImpl) to work with Customer entities while utilizing CustomerDto objects, abstracting away the conversion logic.

Low Coupling:

Definition: Low coupling refers to the degree of interdependence between modules or classes within a system. Modules with low coupling are less reliant on each other.

In the Code:

The CustomerServiceImpl class depends on the CustomerAdapter interface rather than concrete implementations of conversion logic.

Dependency injection is used to inject the CustomerAdapter instance into CustomerServiceImpl, promoting loose coupling between the CustomerService and CustomerAdapter implementations.

The CustomerService implementation interacts with the CustomerAdapter interface, abstracting the details of the conversion process and reducing direct dependencies.

High Cohesion:

Definition: High cohesion refers to the degree to which elements within a module or class are related to each other. Modules with high cohesion exhibit a strong logical relationship between their members.

In the Code:

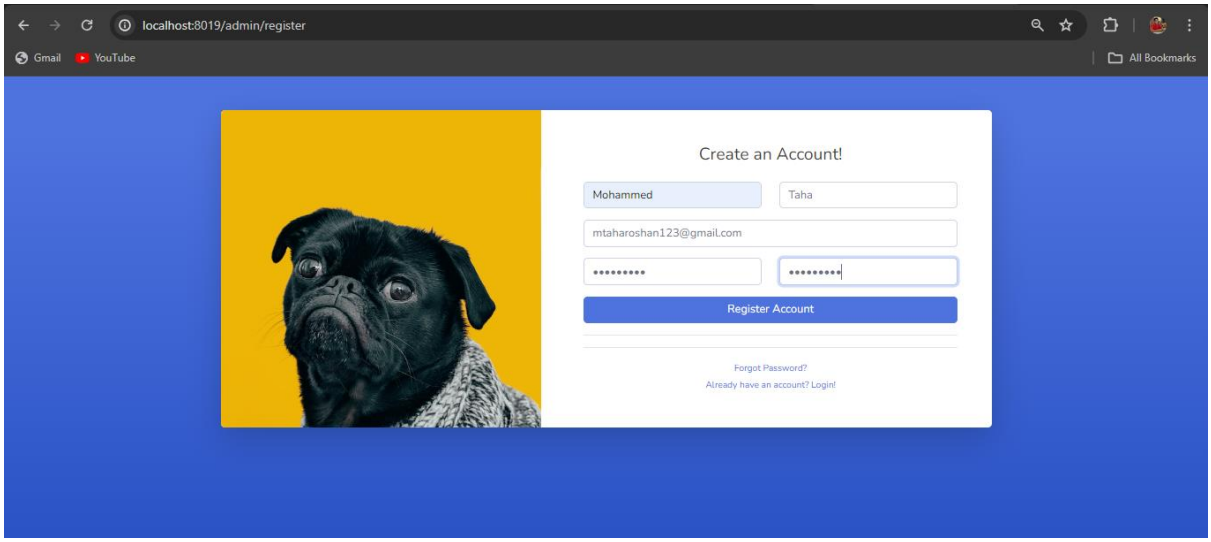
The CustomerAdapterImpl class encapsulates the logic for converting between Customer and CustomerDto objects within a single cohesive unit.

Each method in the CustomerAdapterImpl class is focused on a specific conversion task, ensuring that related operations are grouped together.

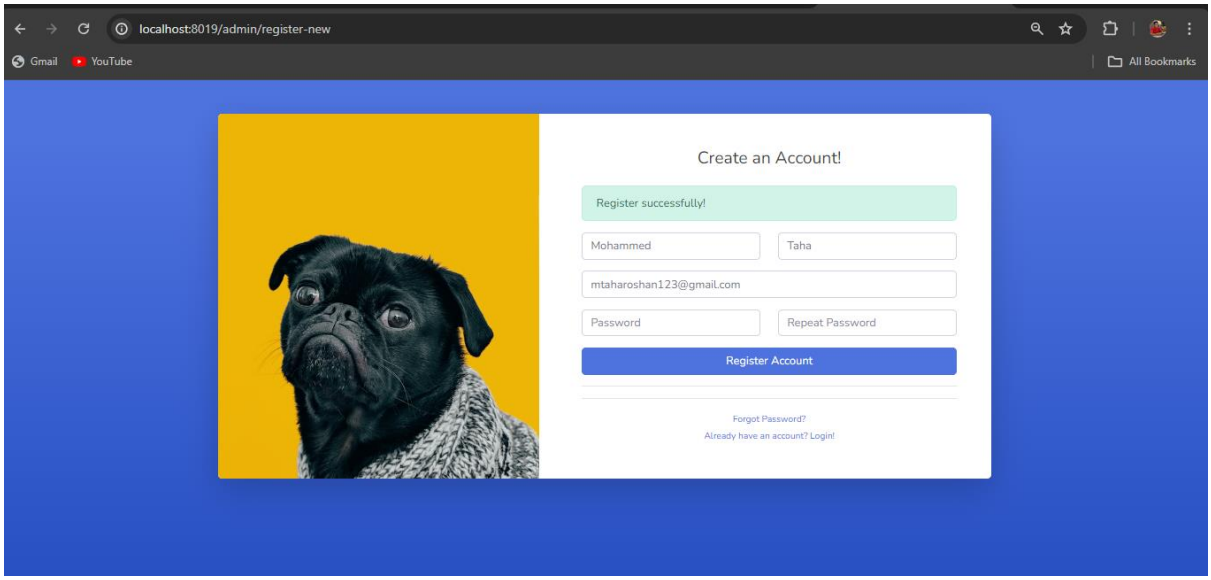
The CustomerServiceImpl class utilizes the CustomerAdapter interface to perform conversion tasks, maintaining a clear separation of concerns and promoting modular design.

In summary, the provided code demonstrates the Adapter Pattern by facilitating communication between Customer and CustomerDto objects through the CustomerAdapter interface. Additionally, it exhibits low coupling by relying on abstractions and high cohesion by organizing related operations within cohesive unit

Admin Register



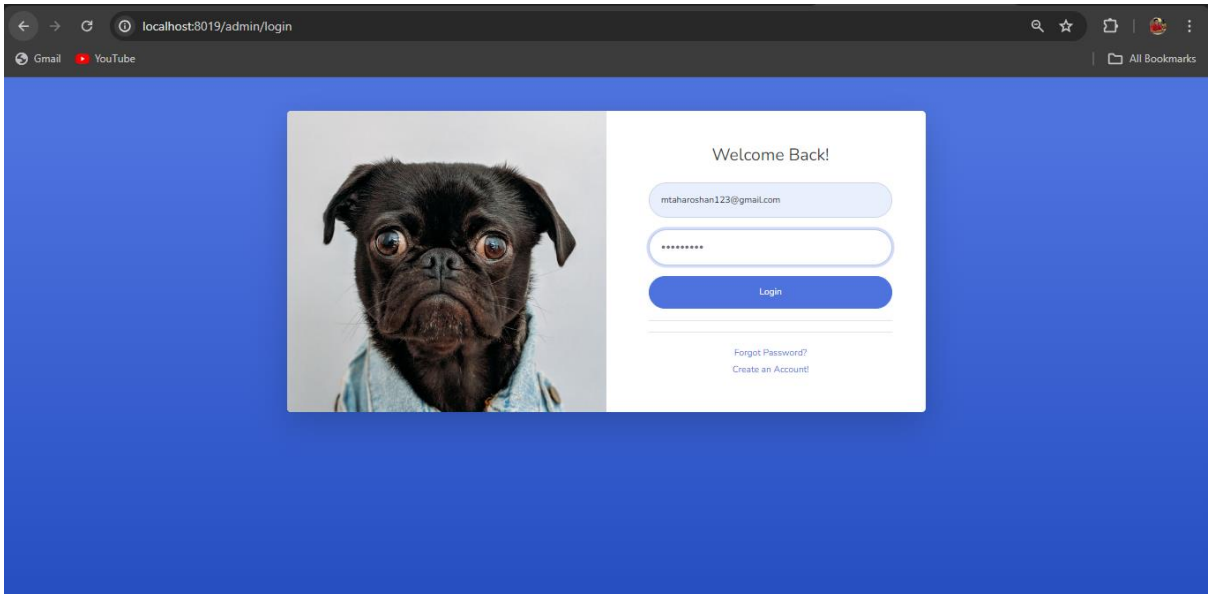
Registered



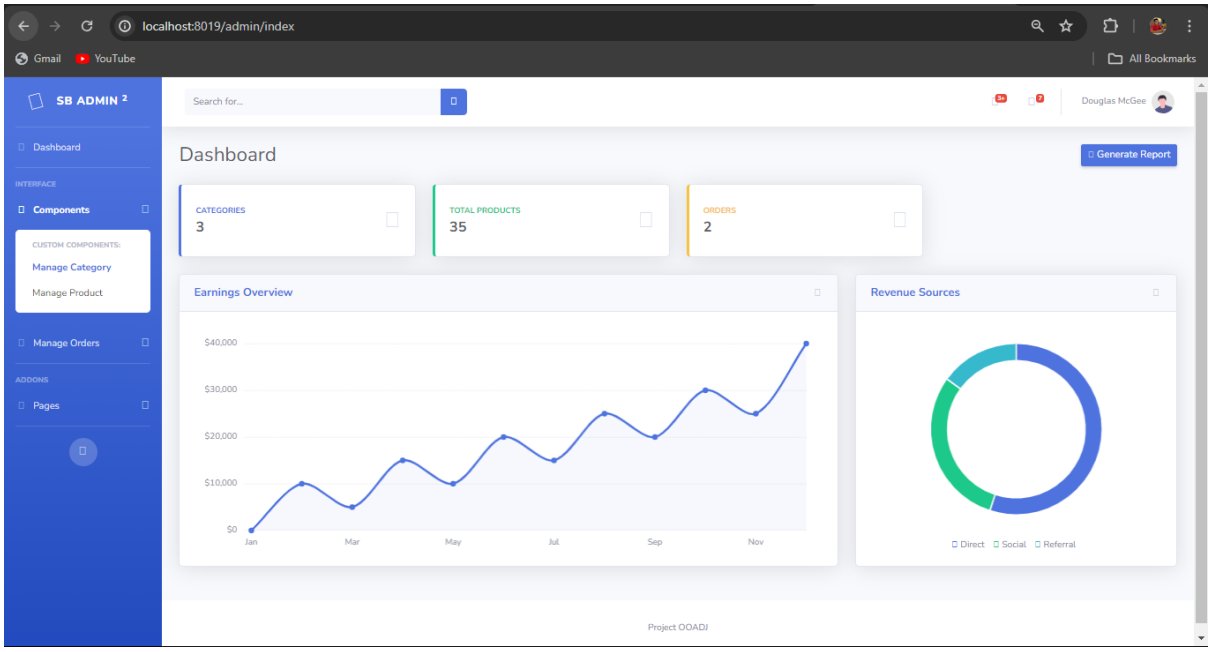
We have Encrypted the password for both customer and admin

admin_id	first_name	image	last_name	password	username
1	Mohammed	NULL	Taha	$2a$10$swXWtLUYaWEAg4LGQ36IMuKZoqrYtx...	mtaharoshan@gmail.com
2	Mohammed	NULL	Taha	$2a$10$3B9pmxYNoSx77e4bQx/jD.nXfTOyxmx...	mtaharoshan123@gmail.com
NULL	NULL	NULL	NULL	NULL	NULL

Admin Login

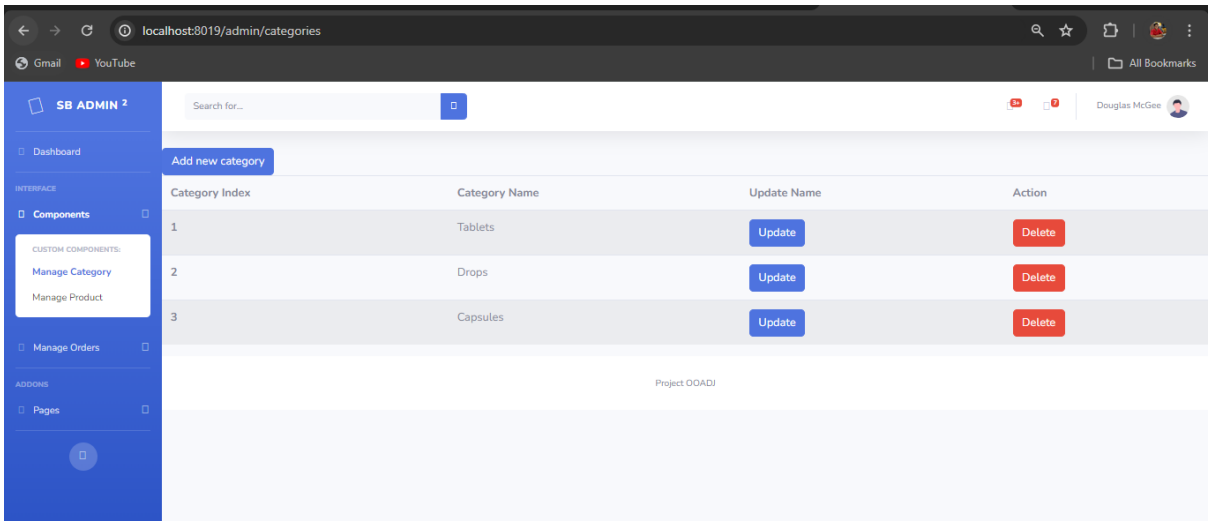


After Login direct redirect to homepage

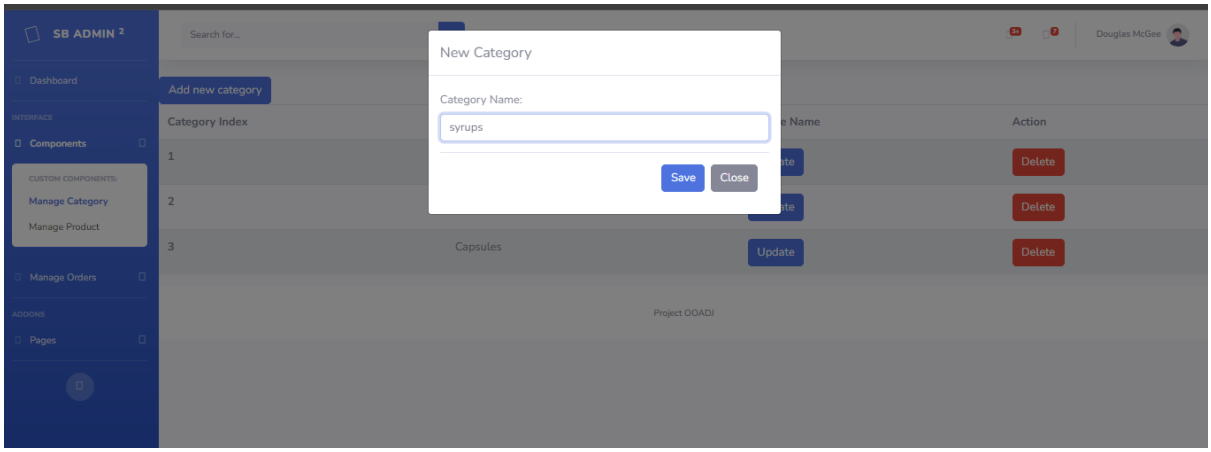


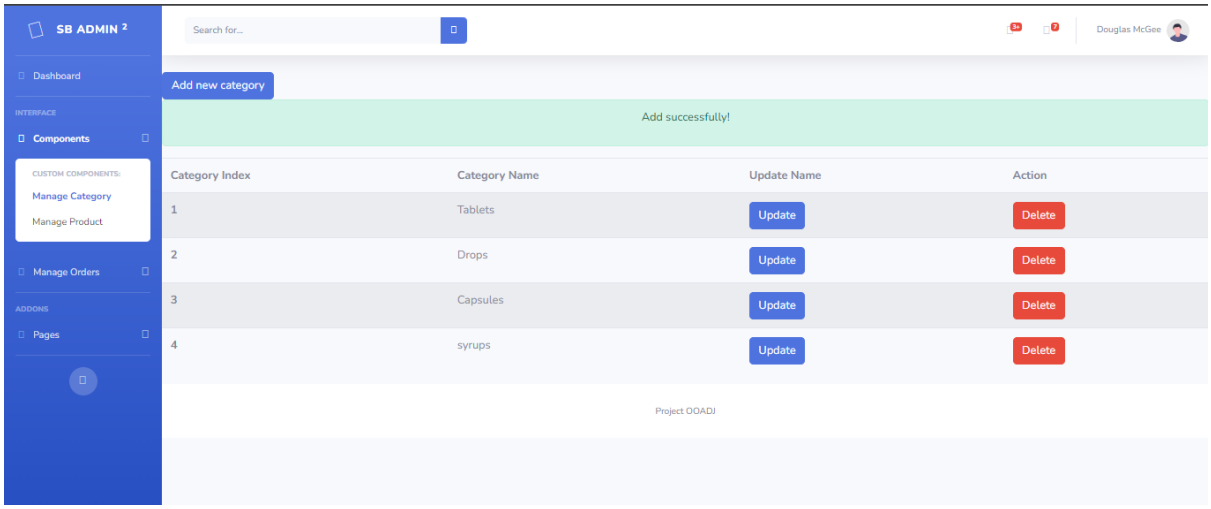
Here it shows number of orders,products,categories and it has

Categories page it has three categories tablets,drops and capsules

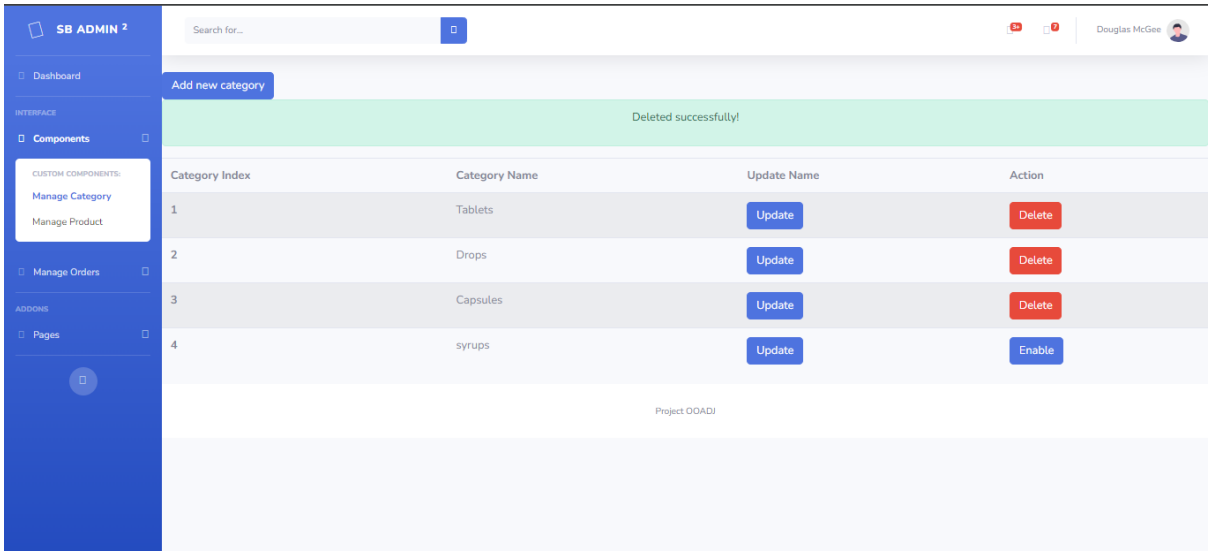


Add new category

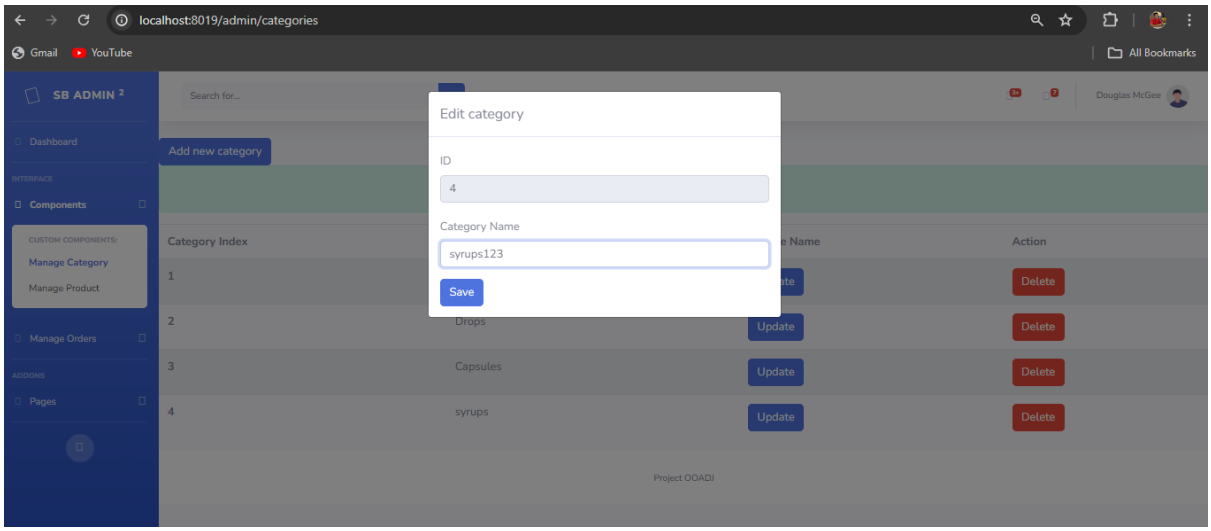




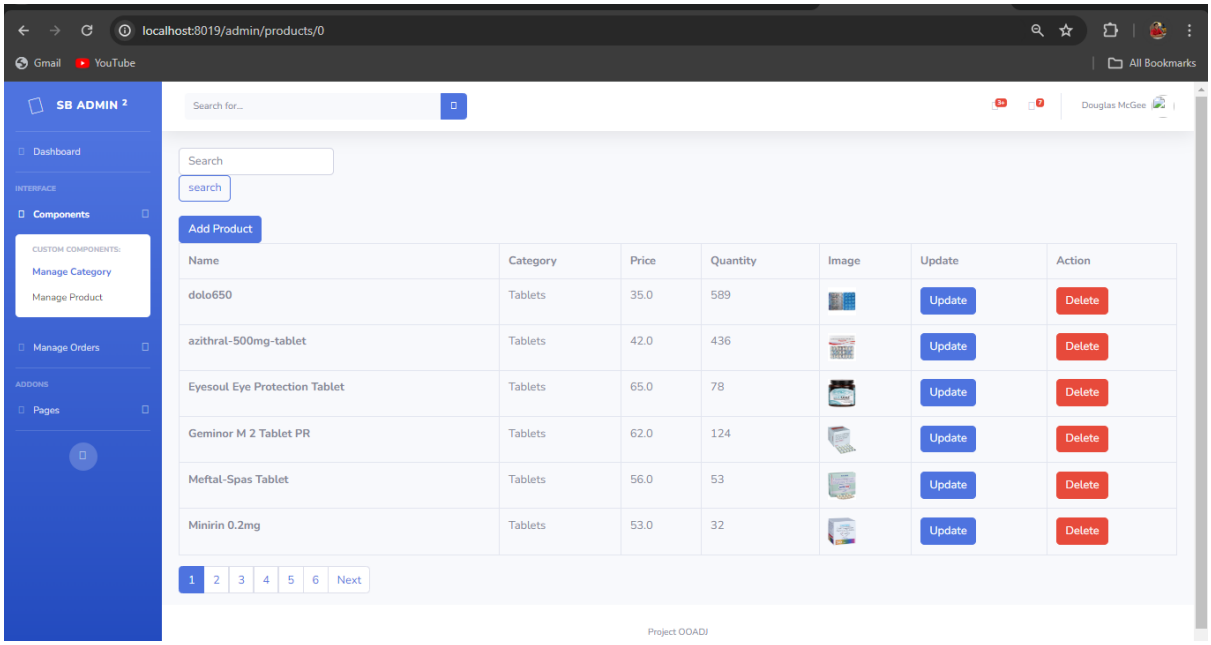
Deleting syrups it will disable and wont be considered as a catgories we can later enable it rather than creating the same category



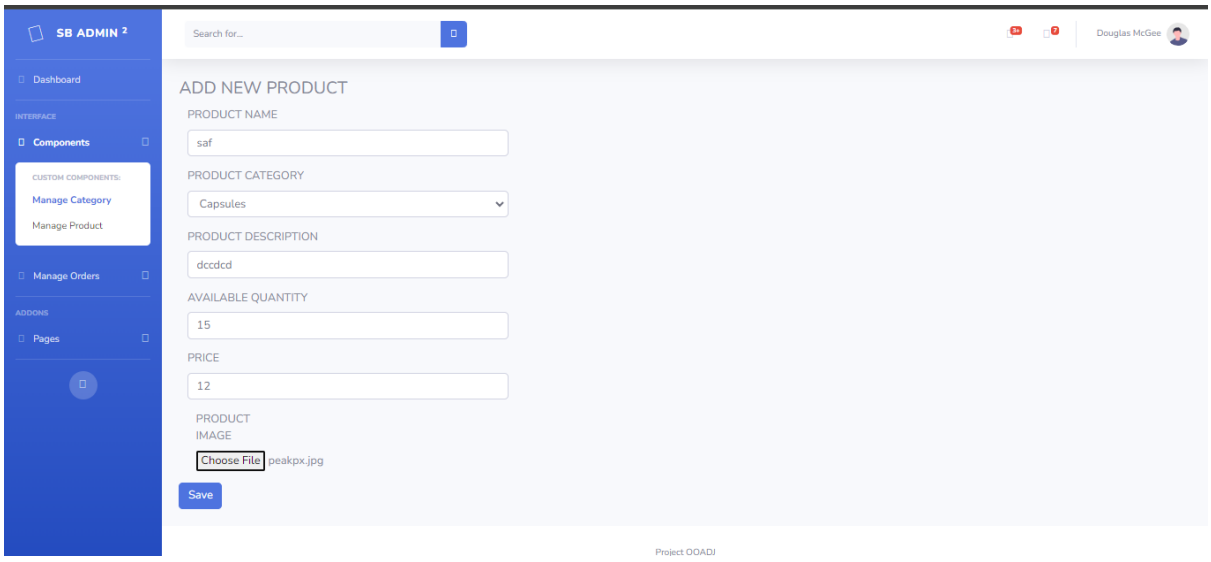
Updating the category



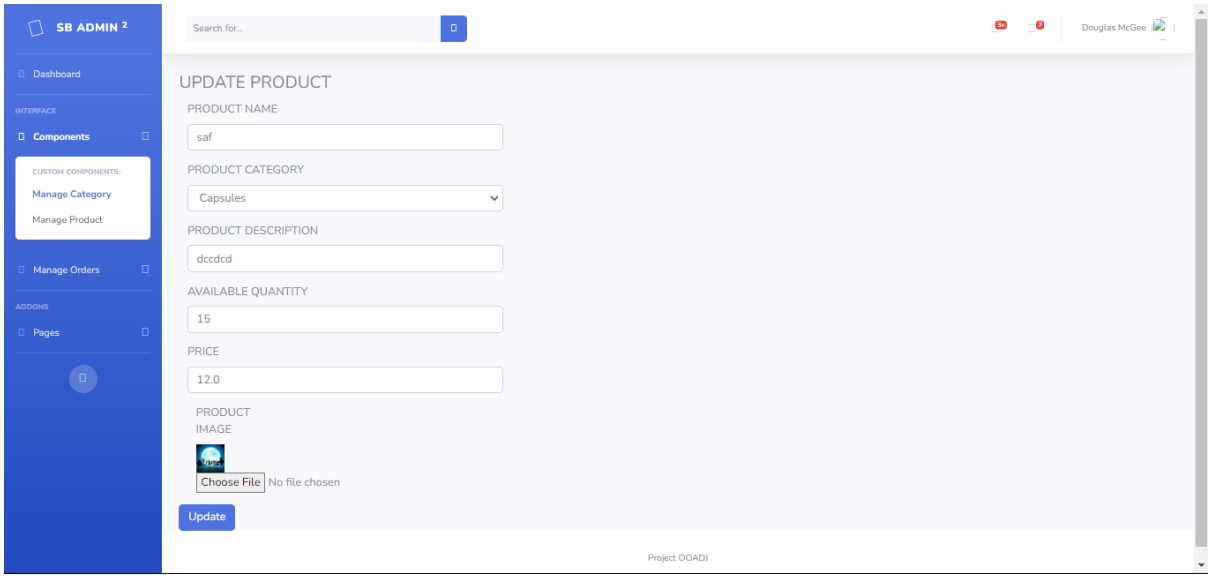
Product Page



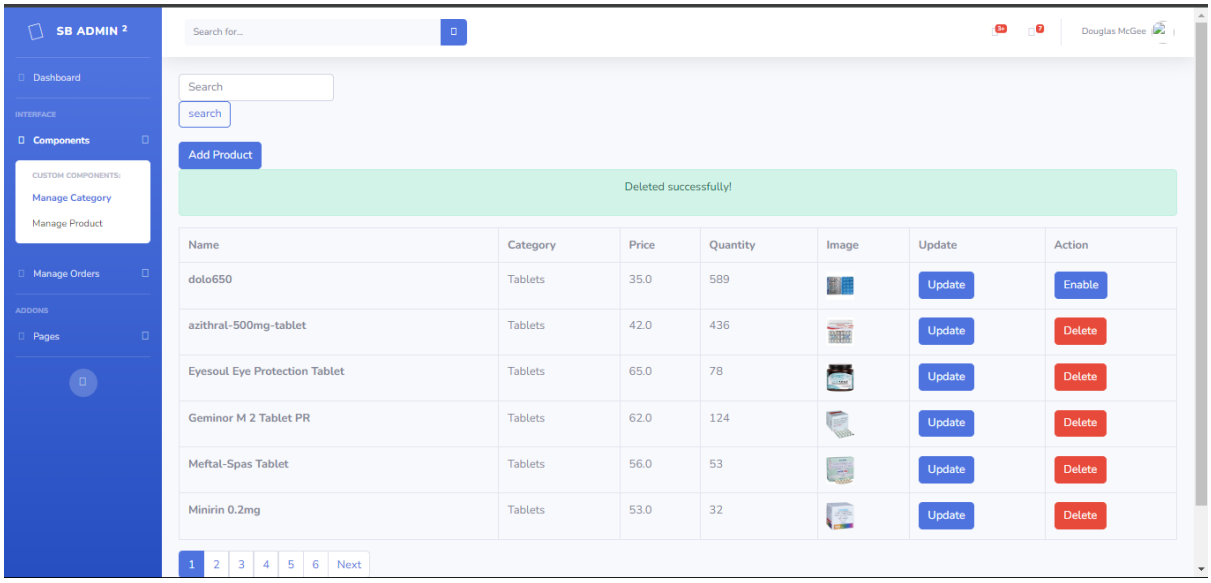
Adding a new product



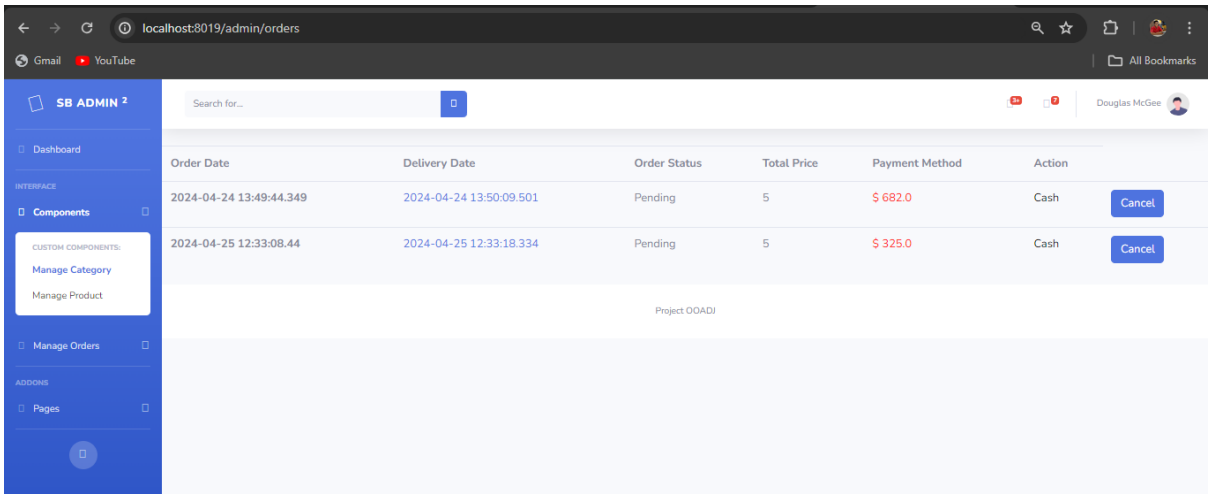
Update page



Deleting dolo650



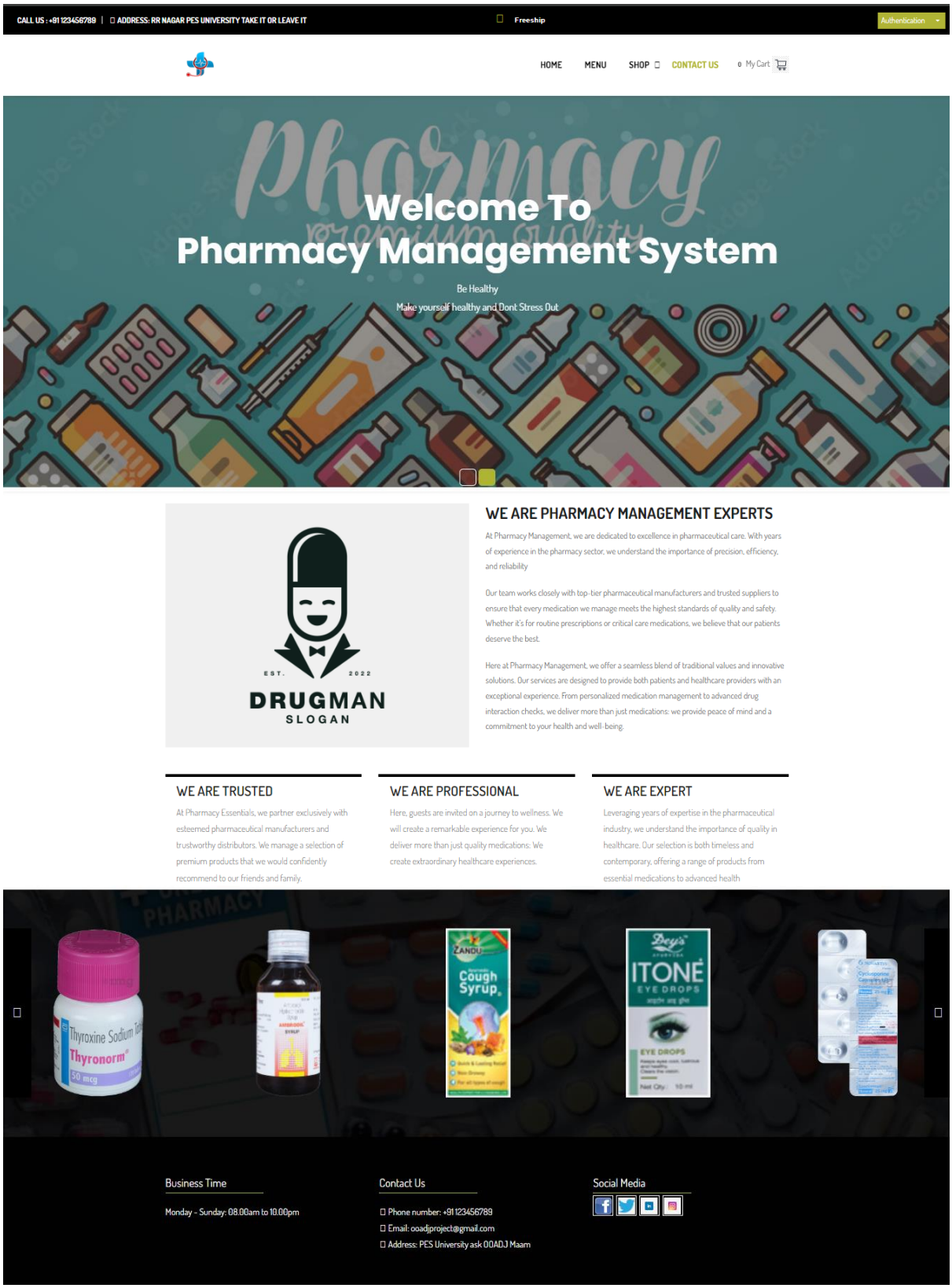
Orders from the Customer is accepted here in the admin page



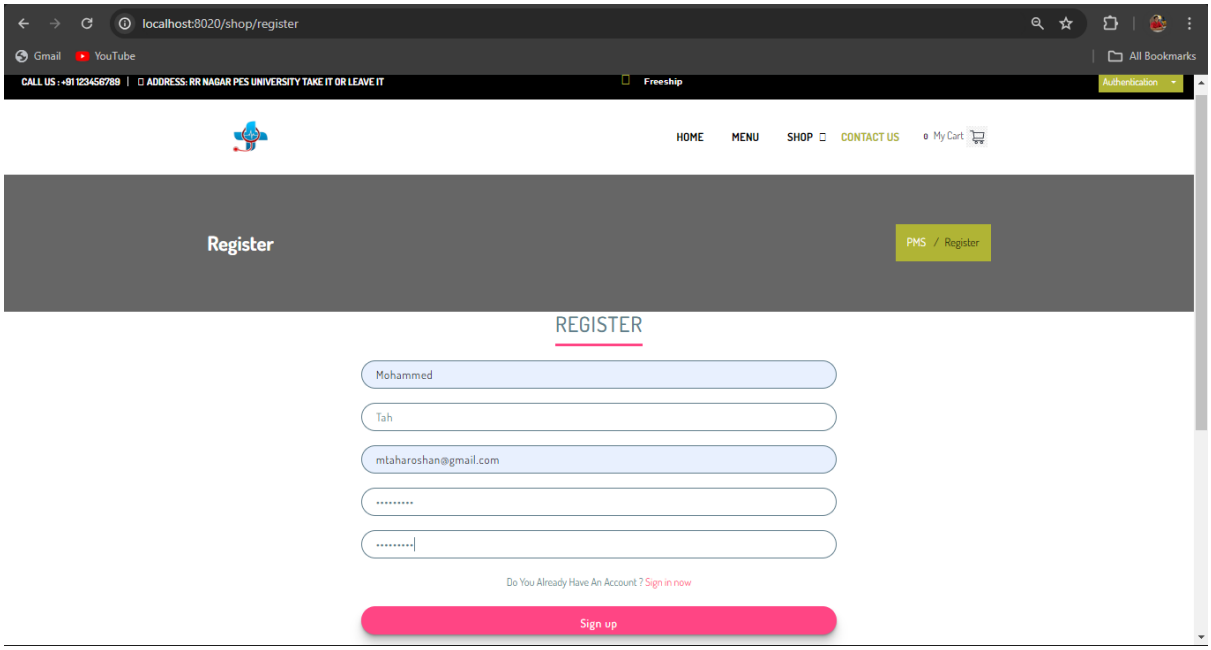
It can even be cancelled

CUSTOMER port 8020

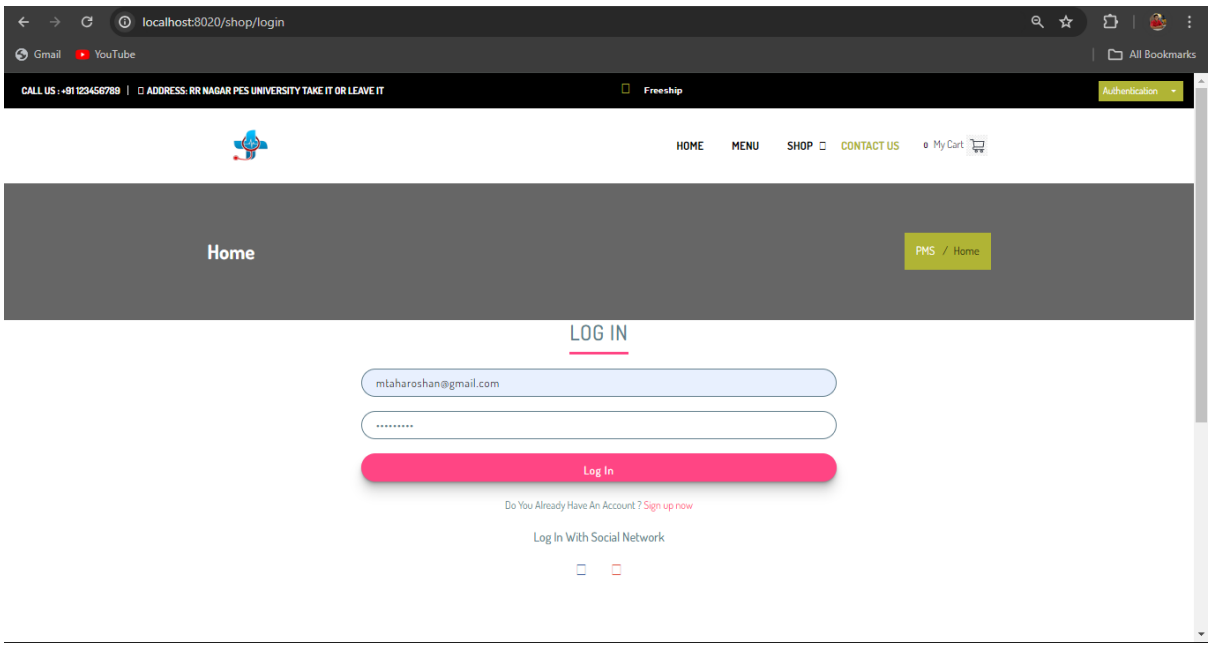
Home Page



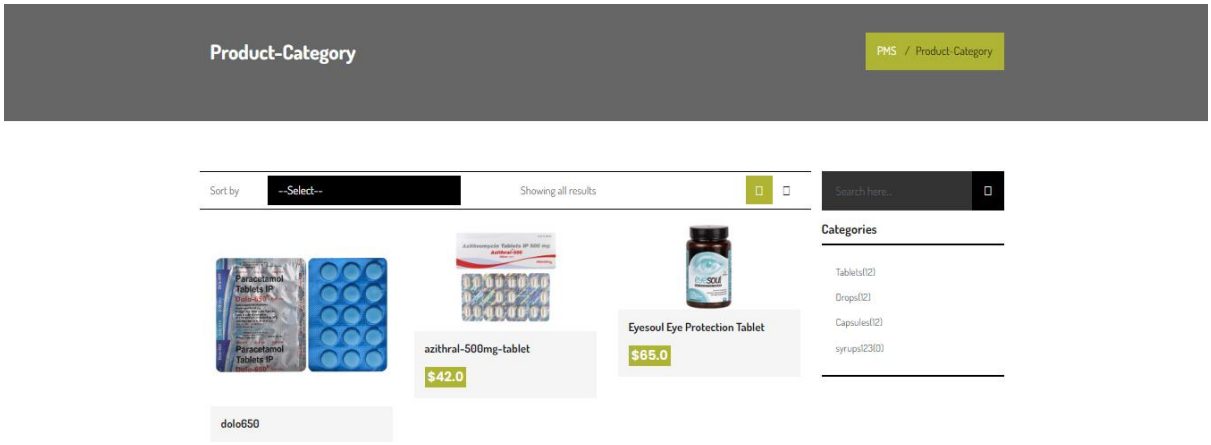
Register Page



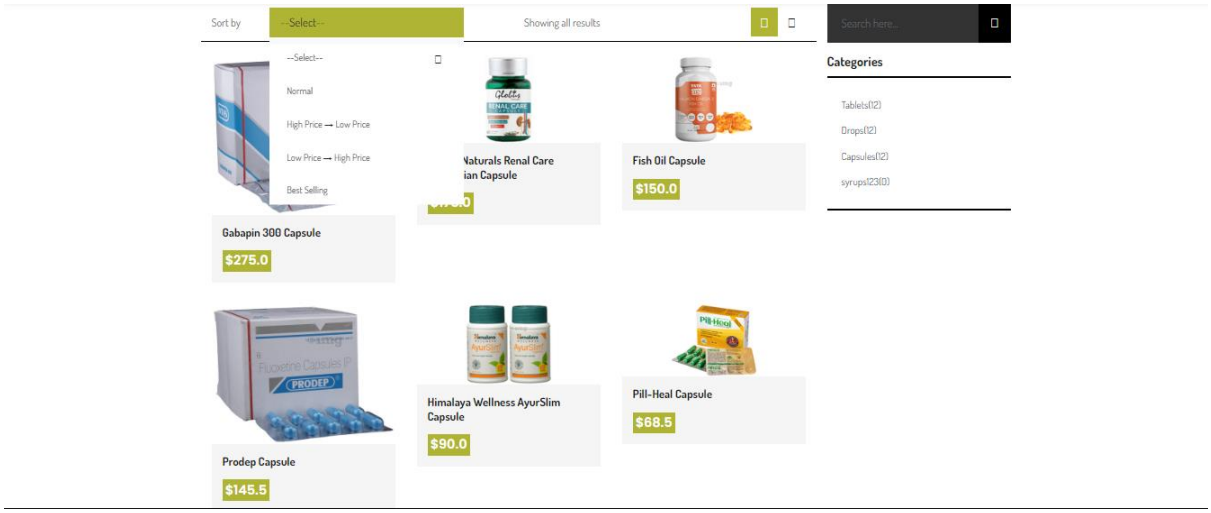
Login



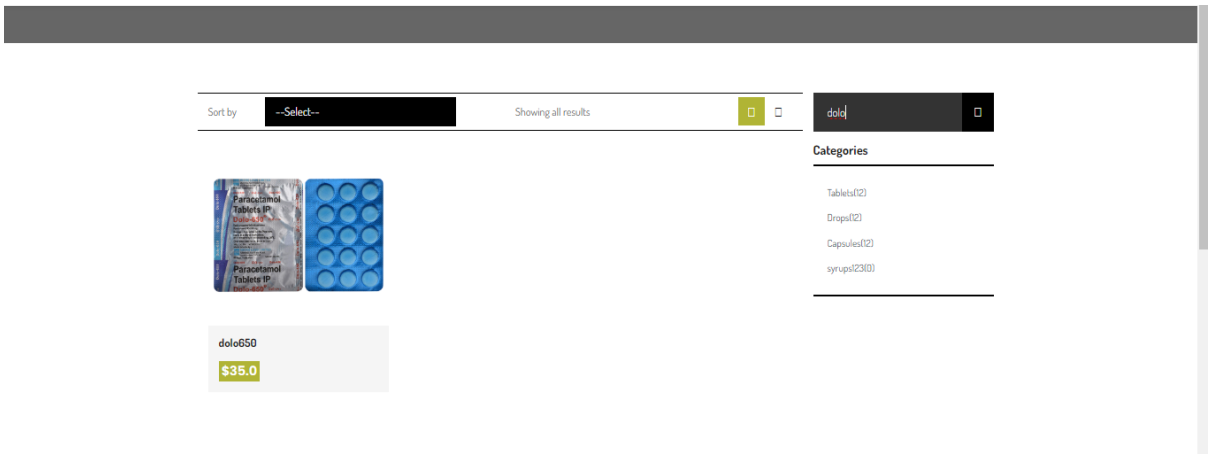
Products



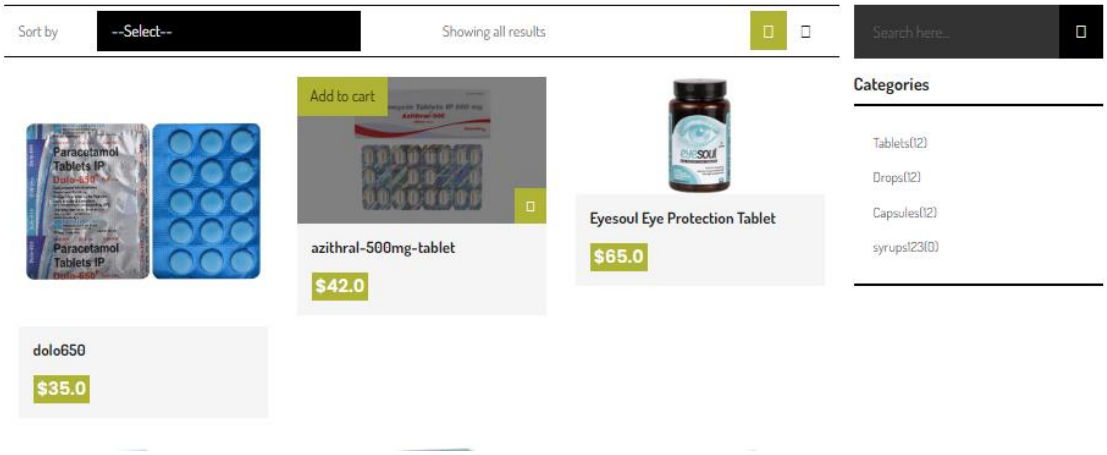
Sort the products from high price to low price



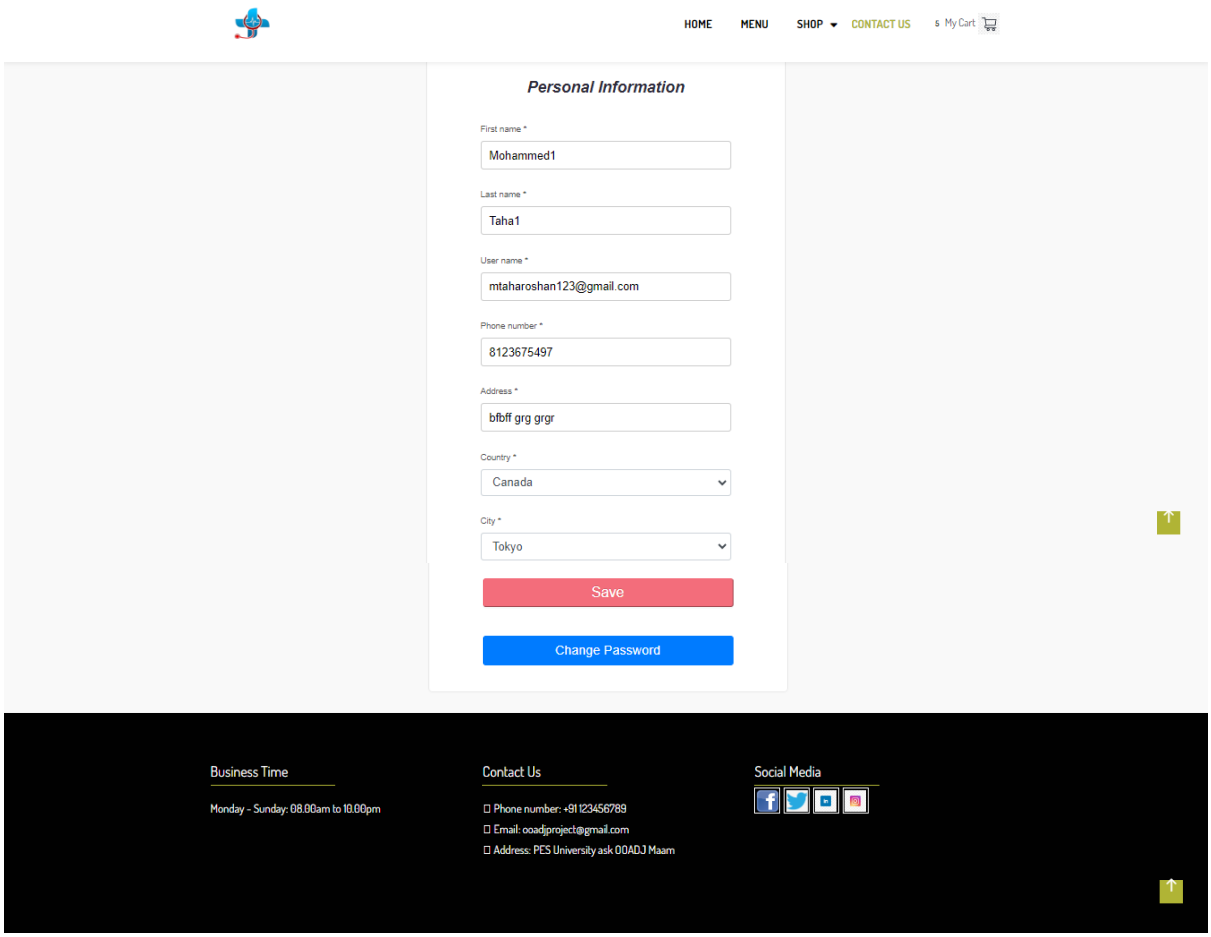
Search dolo



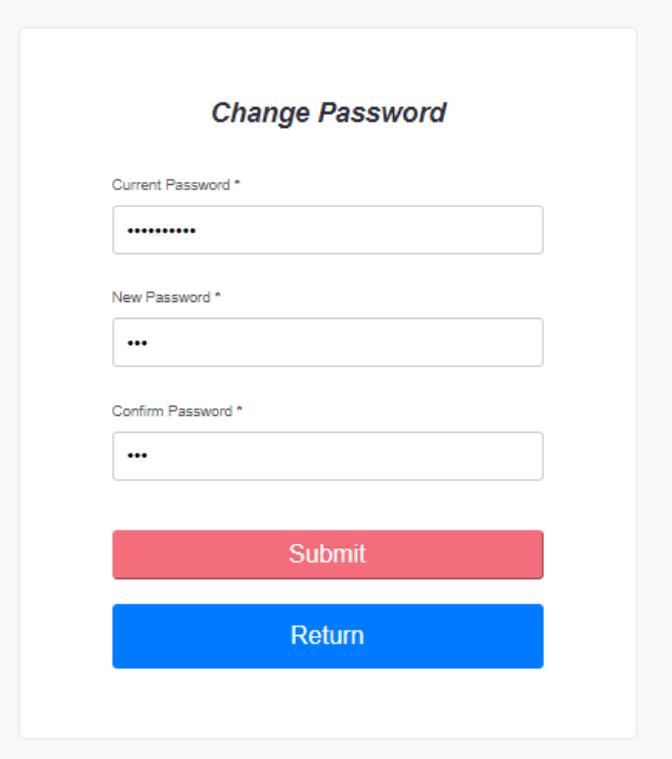
Adding items to cart



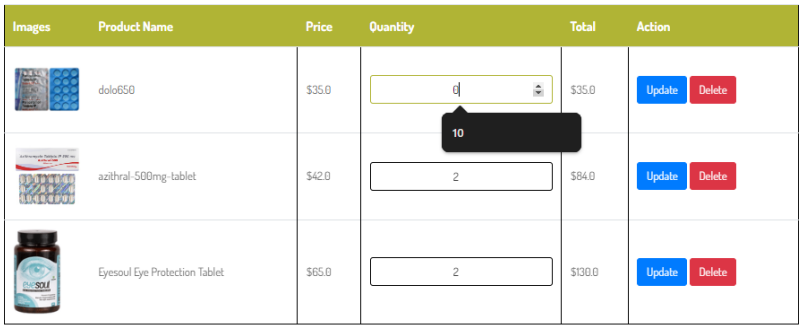
Personal Info(Profile of Customer)



We can change the password of customer



Shopping cart here we can update the product quantity as well as delete it



Images	Product Name	Price	Quantity	Total	Action
	dolo650	$35.0	0 (10)	$35.0	Update Delete
	azithral-500mg-tablet	$42.0	2	$84.0	Update Delete
	Eyesoul Eye Protection Tablet	$65.0	2	$130.0	Update Delete

Increased the quantity to 2

Images	Product Name	Price	Quantity	Total	Action
	Eyesoul Eye Protection Tablet	$65.0	2	$130.0	Update Delete
	dolo650	$35.0	2	$70.0	Update Delete
	azithral-500mg-tablet	$42.0	2	$84.0	Update Delete

Order summary

Sub Total	
Tax	$2
Shipping Cost	Free
Grand Total	$286.0

Checkout

Checkout and we can even update the address here by clicking change information

Billing address

Username *

mtaharoshan123@gmail.com

Phone Number *

8123675497

Address *

bfbff grg grgr 9 2

Change information

Payment Methods

Cash

Shopping cart

dolo650

Price: 35.0$ | Qty: 2 | Subtotal: $ = 70.0

Eyesoul Eye Protection Tablet

Price: 65.0$ | Qty: 2 | Subtotal: $ = 130.0

azithral-500mg-tablet

Price: 42.0$ | Qty: 2 | Subtotal: $ = 84.0

Your order

Product	Total
Sub Total	284.0
Tax	$2
Shipping Cost	Free
Grand Total	286.0

Place Order

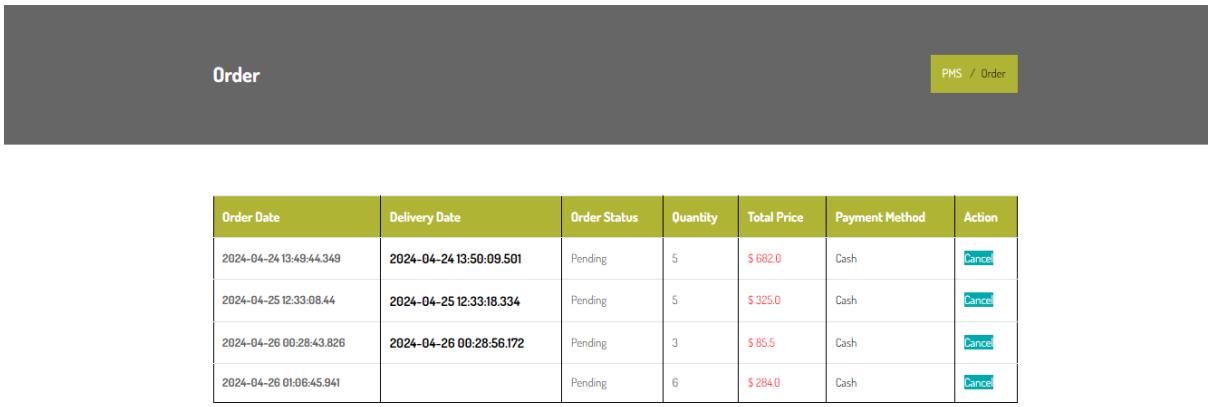
Order Detail page

Images	Product Name	Price
	azithral-500mg-tablet	$42.0
	dolo650	$35.0
	Eyesoul Eye Protection Tablet	$65.0

Order summary

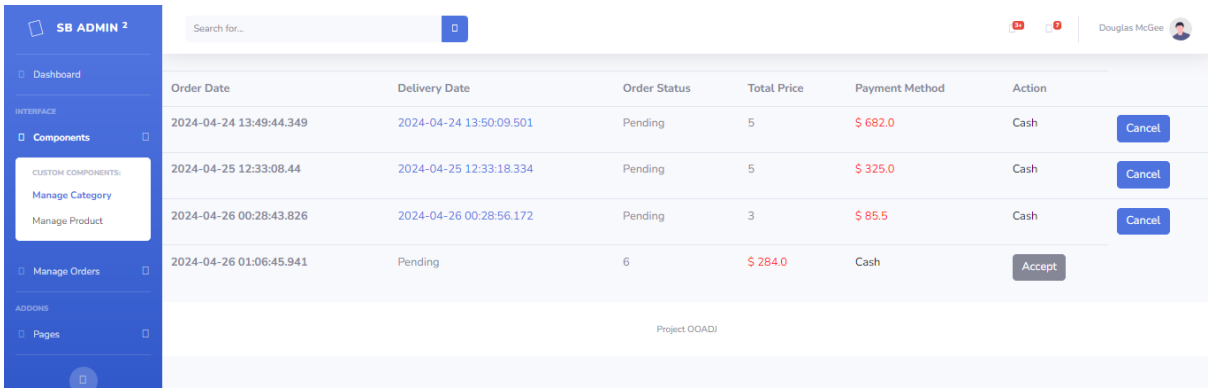
Total Quantity 6 Sub Total	284.0

Your Order – the new order is not yet accepted by Admin after accepting the delivery date will be updated

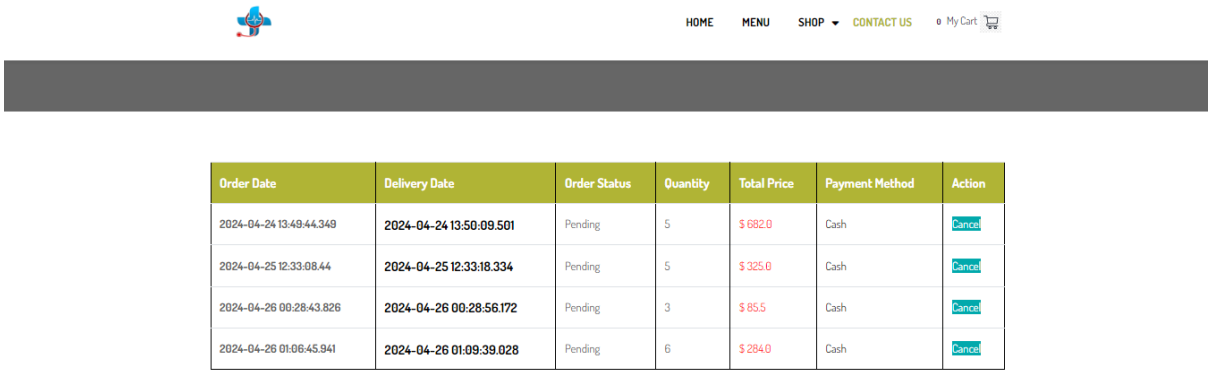


Order Date	Delivery Date	Order Status	Quantity	Total Price	Payment Method	Action
2024-04-24 13:49:44.349	2024-04-24 13:50:09.501	Pending	5	$ 682.0	Cash	Cancel
2024-04-25 12:33:08.44	2024-04-25 12:33:18.334	Pending	5	$ 325.0	Cash	Cancel
2024-04-26 00:28:43.826	2024-04-26 00:28:56.172	Pending	3	$ 85.5	Cash	Cancel
2024-04-26 01:06:45.941		Pending	6	$ 284.0	Cash	Cancel

ADMIN Order page



After Accept from Admin



You can later Cancel after accepting by admin too