**Shoe Ecommerce**

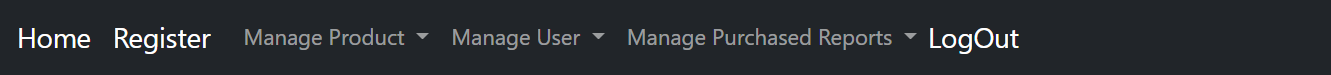
**Index:**

**Git URL:** [**https://github.com/RajaRajeswari12/Phase3ShoeEcommerce.git**](https://github.com/RajaRajeswari12/Phase3ShoeEcommerce.git)

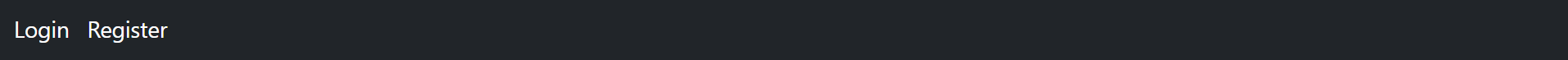
**Jsp Pages:**

1. header.html
2. addProduct.html
3. adminLogin.html
4. adminRegistration.html
5. deleteProduct.html
6. displayUserDetail.html
7. homePage.html
8. searchByUser.html
9. searchPurchaseReportsByCategory.html
10. searchPurchaseReportsByDate.html
11. updateProduct.html
12. updateProductPage.html
13. viewCustomer.html
14. viewProduct.html
15. viewPurchaseReport.html
16. welcome.html
17. **Servlets [com.sportyshoes.ecommerce.controller]**
    1. AdminUserController.java
    2. CustomerController.java
    3. ProductCategoryController.java
    4. ProductController.java
    5. PurchaseReportController.java
18. **Repositories [com.sportyshoes.ecommerce.repository]**
    1. AdminUserRepository.java
    2. CustomerRepository.java
    3. ProductCategoryRepository.java
    4. ProductRepository.java
    5. PurchaseReportRepository.java
19. **Models[com.sportyshoes.ecommerce.models]**
    1. Airline.java
    2. AdminUser.java
    3. Customer.java
    4. Product.java
    5. ProductCategory.java
    6. PurchaseReport.java
20. **Service[com.sportyshoes.ecommerce.service]**
    1. AdminUserService.java
    2. CustomerService.java
    3. ProductCategoryService.java
    4. ProductService.java
    5. PurchasedReportService.java
21. **Service Implementation [com.sportyshoes.ecommerce.service.implementation]**
    1. AdminUserServiceImpl.java
    2. CustomerServiceImpl.java
    3. ProductCategoryServiceImpl.java
    4. ProductServiceImpl.java
    5. PurchasedReportServiceImpl.java
22. **POJOS[com.sportyshoes.ecommerce.POJO]**
    1. CustomerNamePOJO.java
    2. GetProductIdPOJO.java
    3. PurchaseReportByDatePOJO.java
23. **Validators[com.sportyshoes.ecommerce.validator]**
    1. AdminUserValidator.java
24. Jsp Pages
25. **header.html**

Header Html contains the link to all the pages. It checks whether the user has logged in or not. If the Admin logs in then it allows access to the following header links.

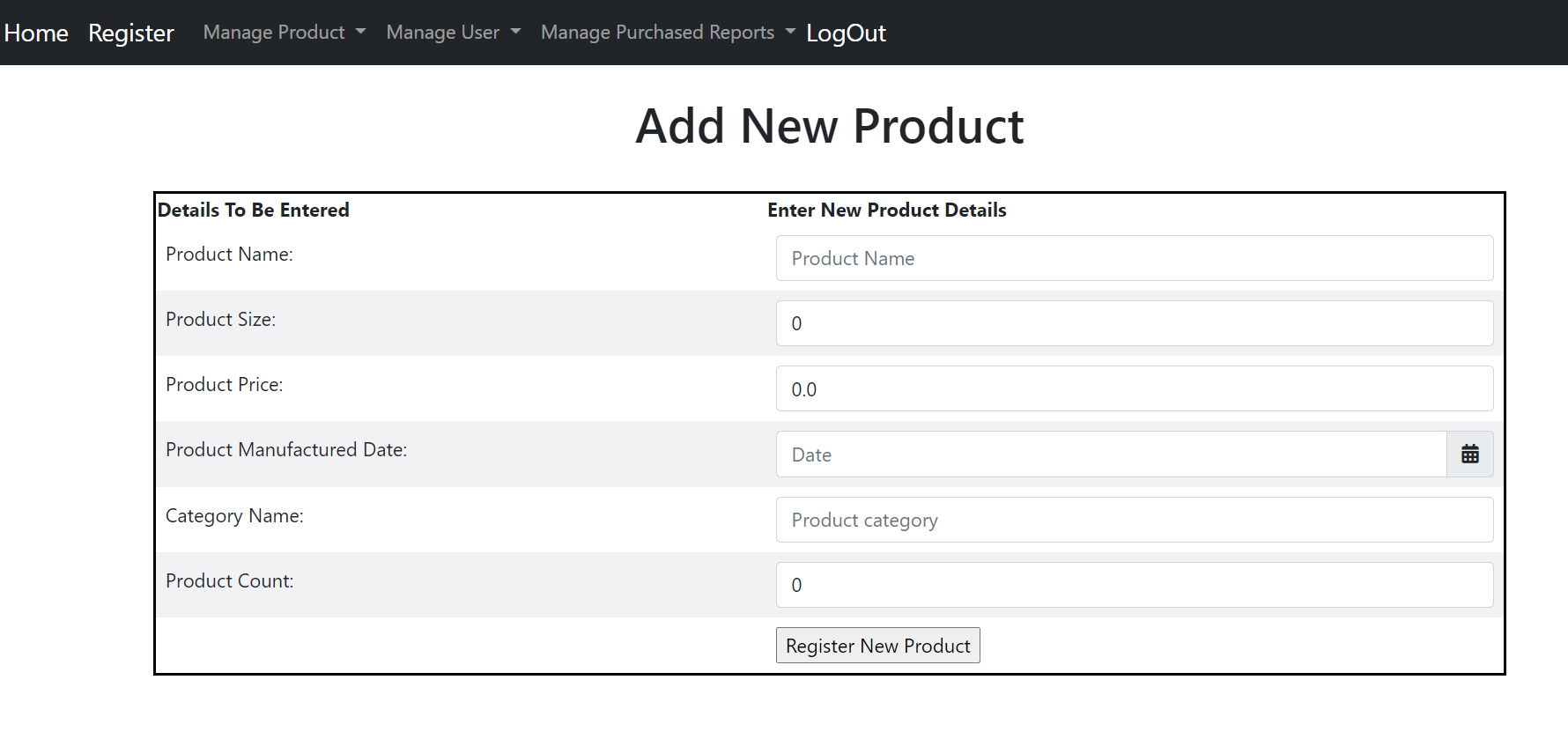
****

Or else only the below header links are active.

****

1. **addProduct.html**

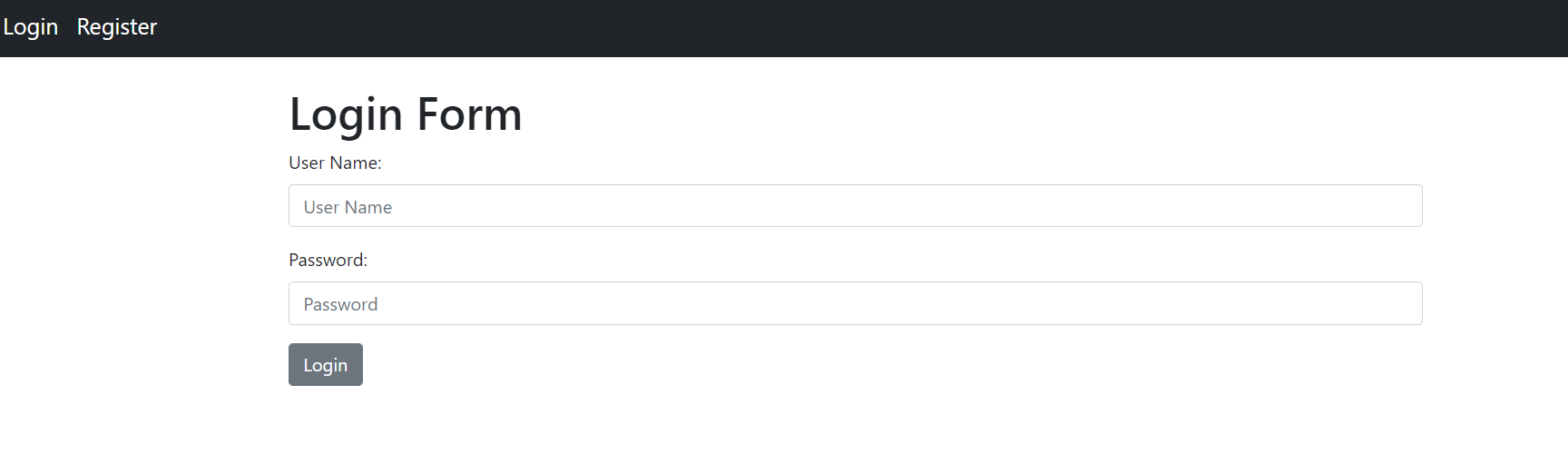
Add Product Page is used to add new product details along with Category to the backend. Once successfully inserted the data without any error. The page navigates to the viewProduct page where all the products are displayed.

****

1. **adminLogin.html**

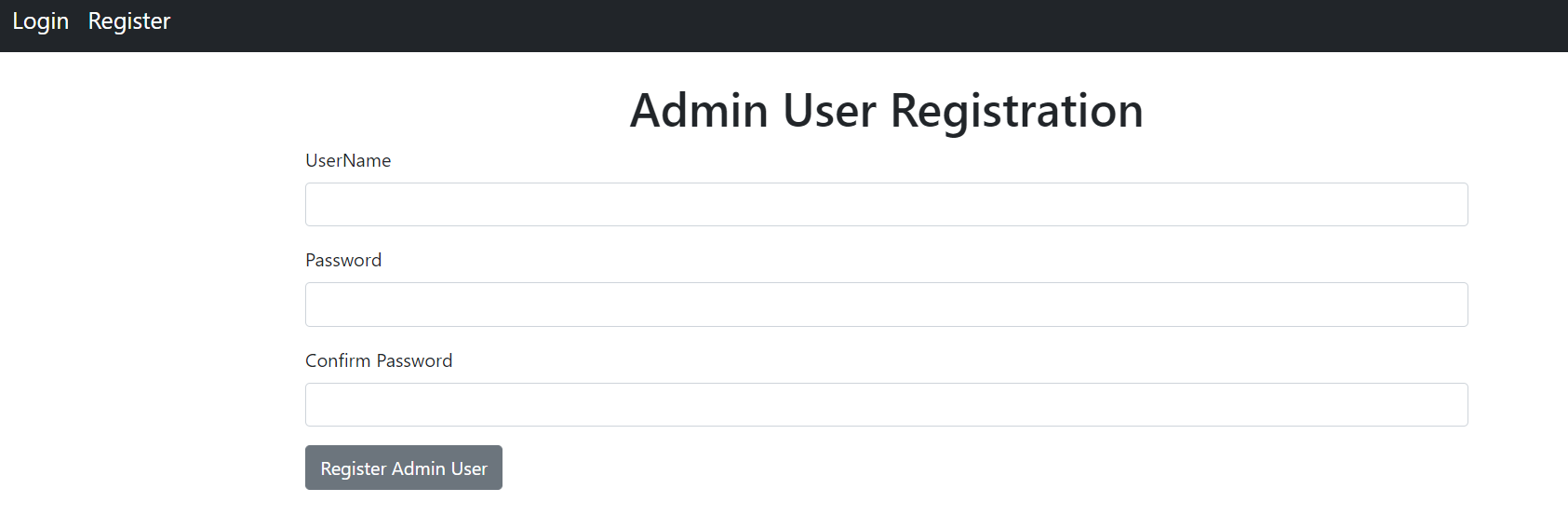
Login Page is used by the Admin to Login into shoe’s E-commerce portal.

If the Admin logins with his/her credentials then it is navigated to the Admin page where the admin can manage the products, view the list of customers who have signed up and see their details. Also view the purchase reports .

****

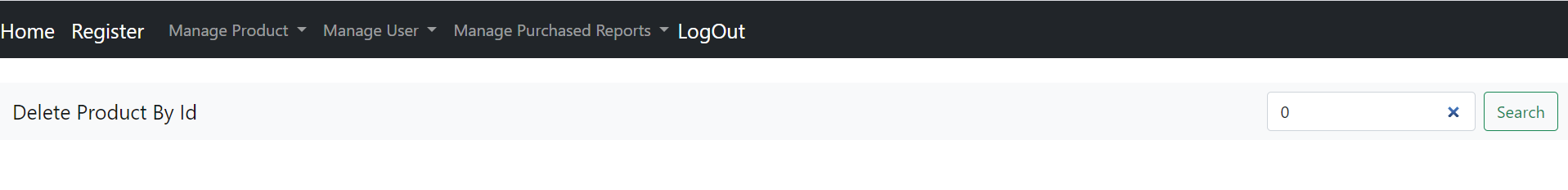
1. **adminRegistration.html**

Admin Registration page is used to register new user as Admin. And the Admin details are stored on the admin\_user.



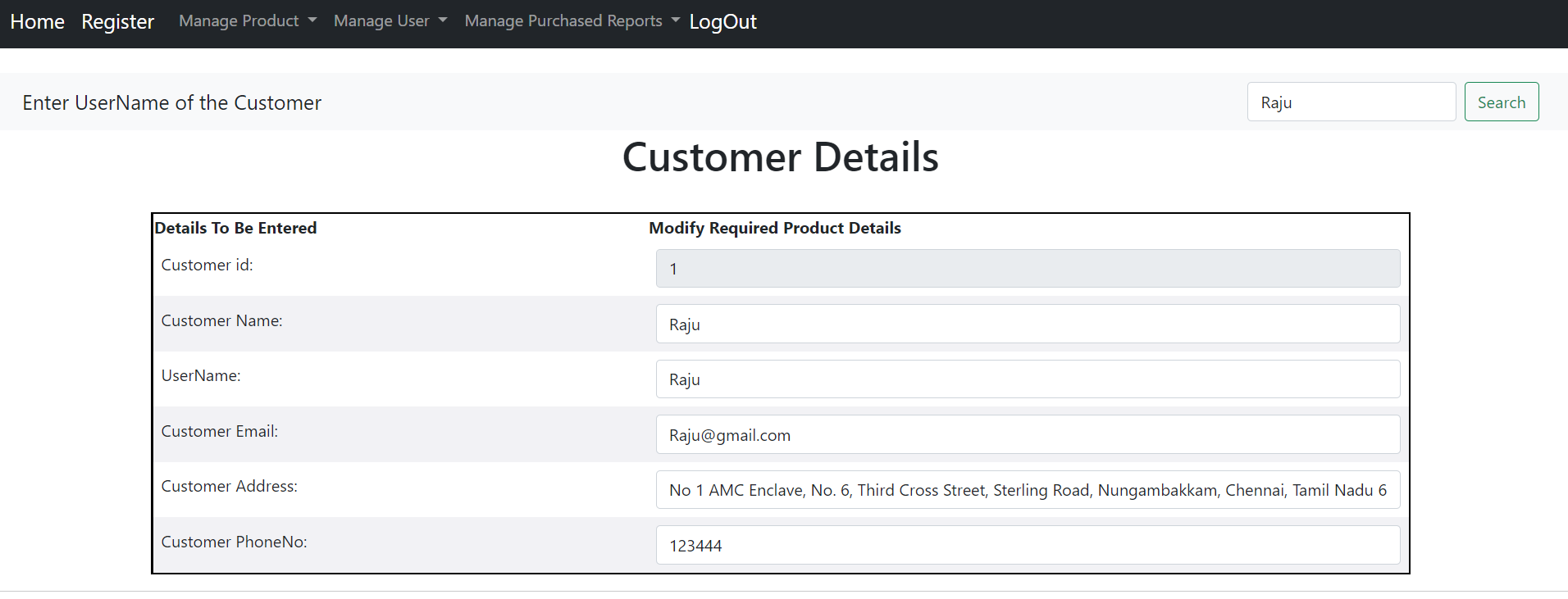
1. **deleteProduct.html**

Delete Product page is used to delete the product details by using the Product Id.

****

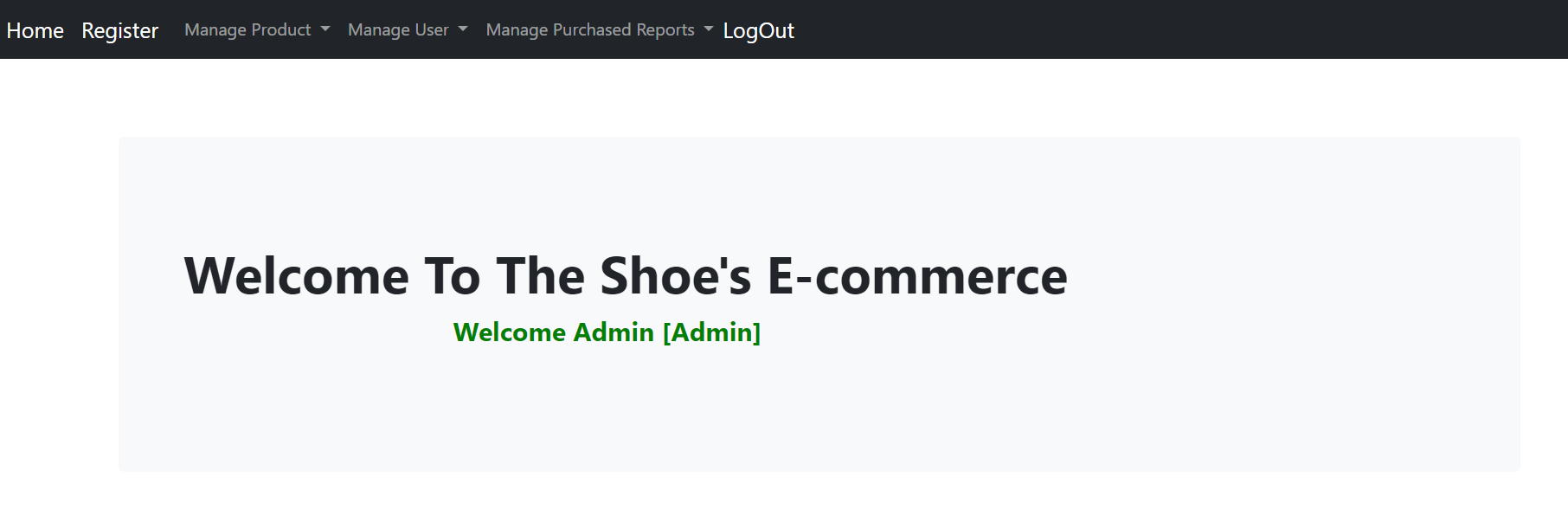
1. **displayUserDetail.html**

DisplayUserDetail page is used to display the details of the Customer by giving the cutomers username.



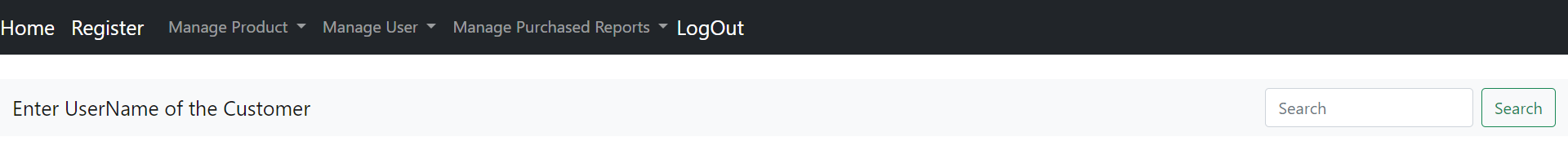
1. **homePage.html**

Home Page is the first page comes when you access the link <http://localhost:8081/home>



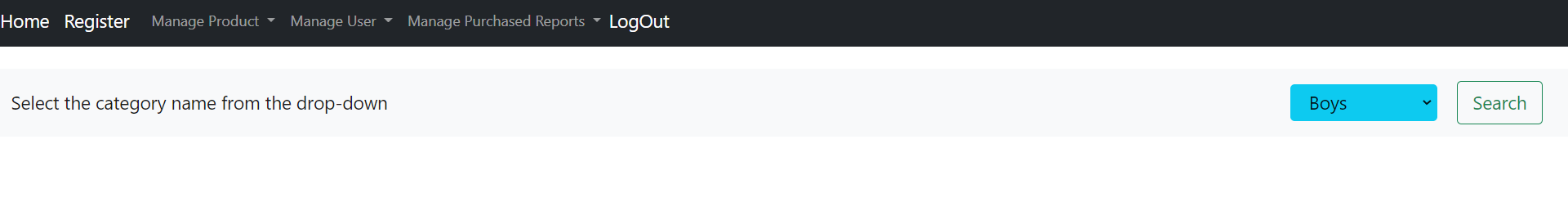
1. **searchByUser.html**

This Page is used to search the customer by giving username.

****

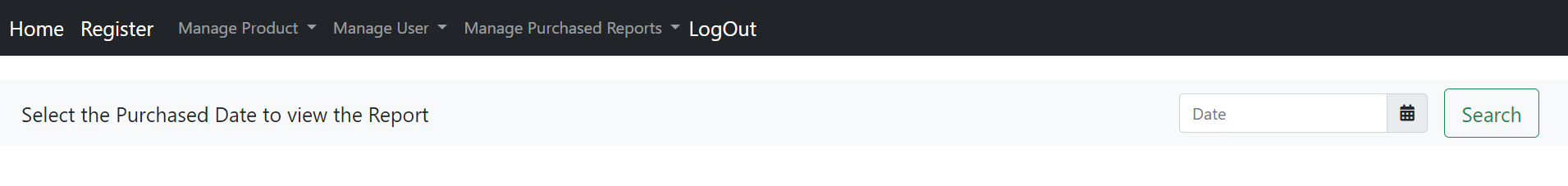
1. **searchPurchaseReportsByCategory.html**

This page is used to get all the Purchase Reports by choosing the shoe’s category from the drop down list.

****

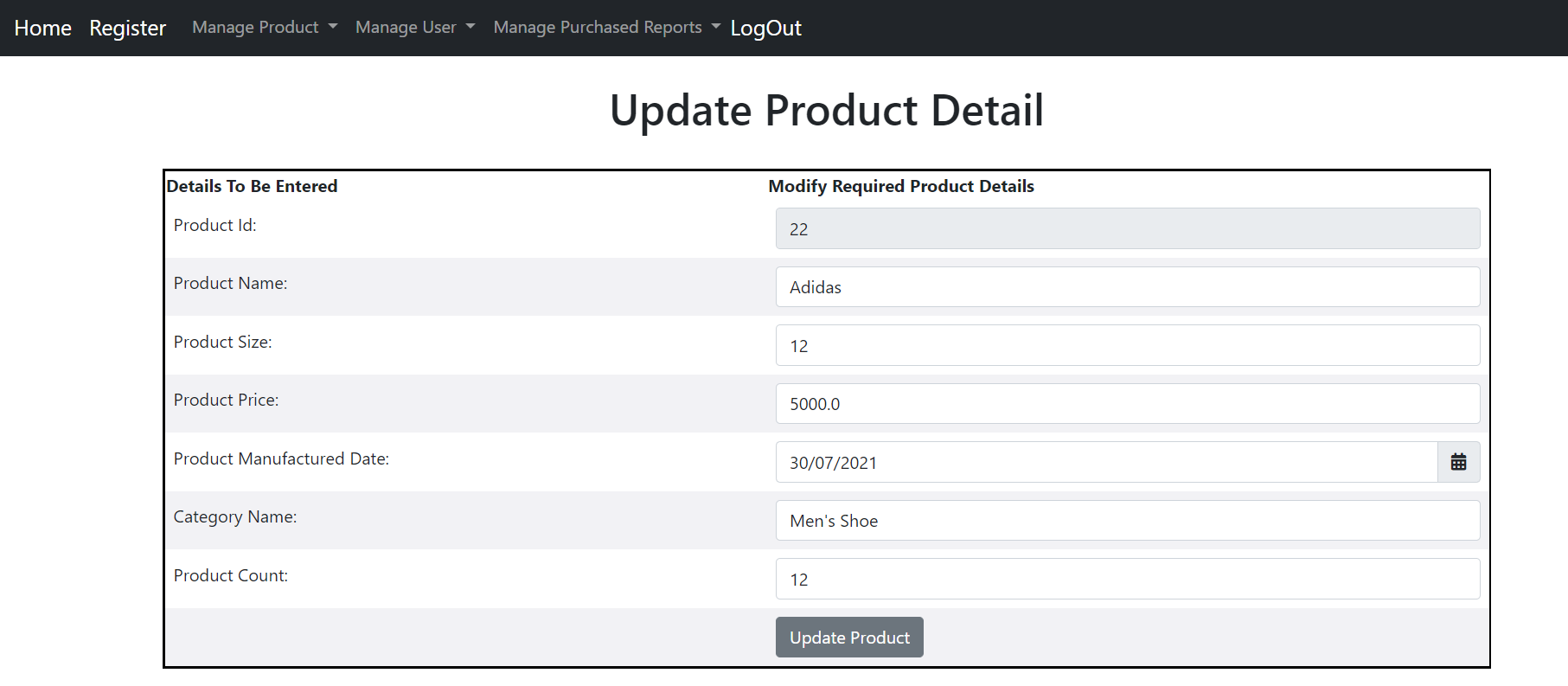
1. **searchPurchaseReportsByDate.html**

This page is used to get all the Purchase Reports by selecting the purchased Date.

****

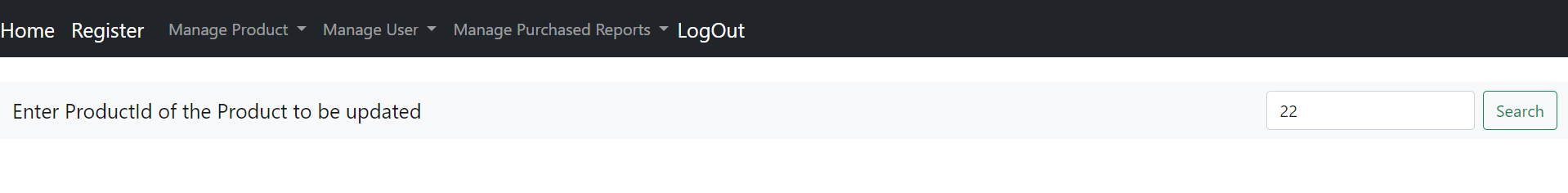
1. **updateProduct.html**

This Page is used to update the details of the added products.

****

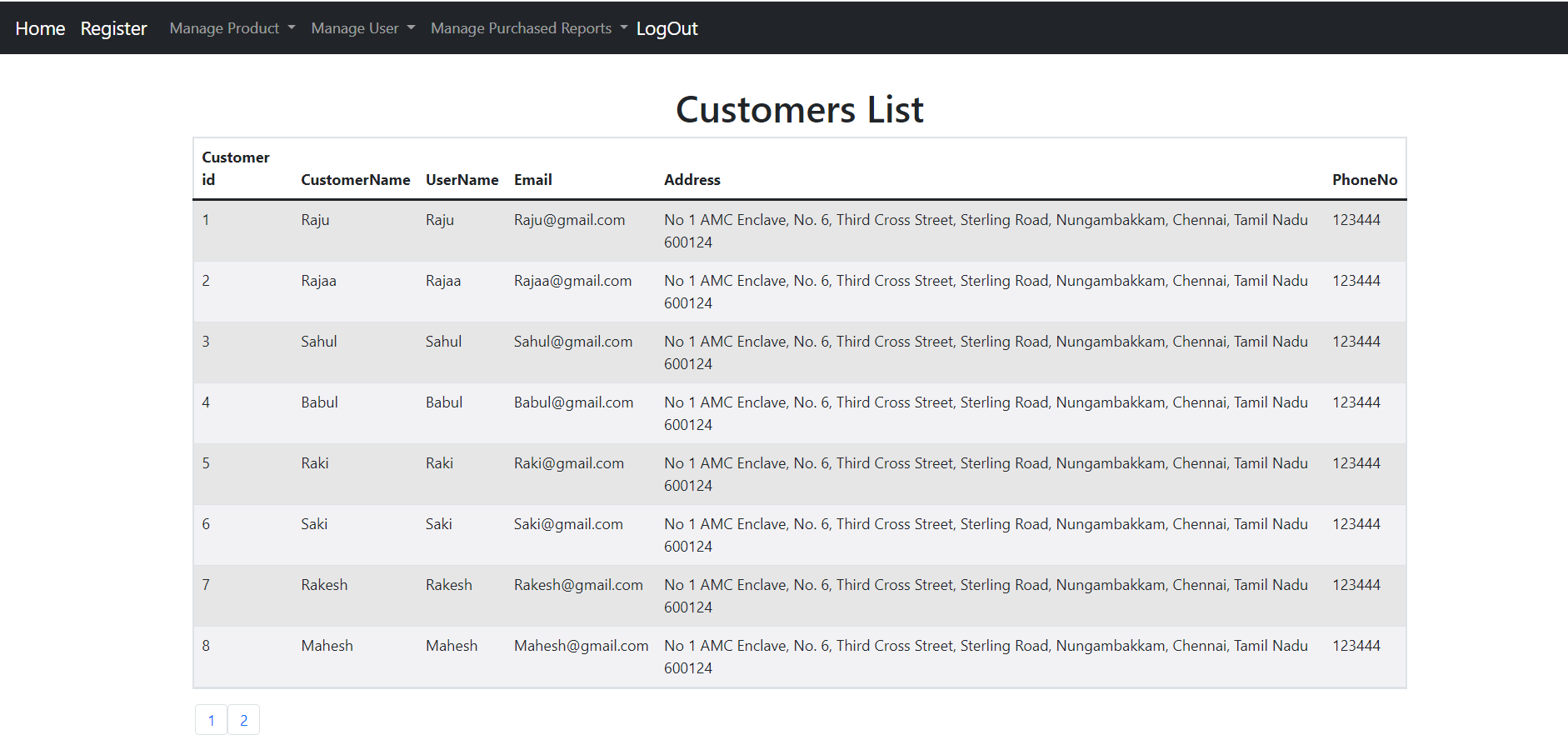
1. **updateProductPage.html**

Delete Product page is used to delete the product details by using the Product Id.

****

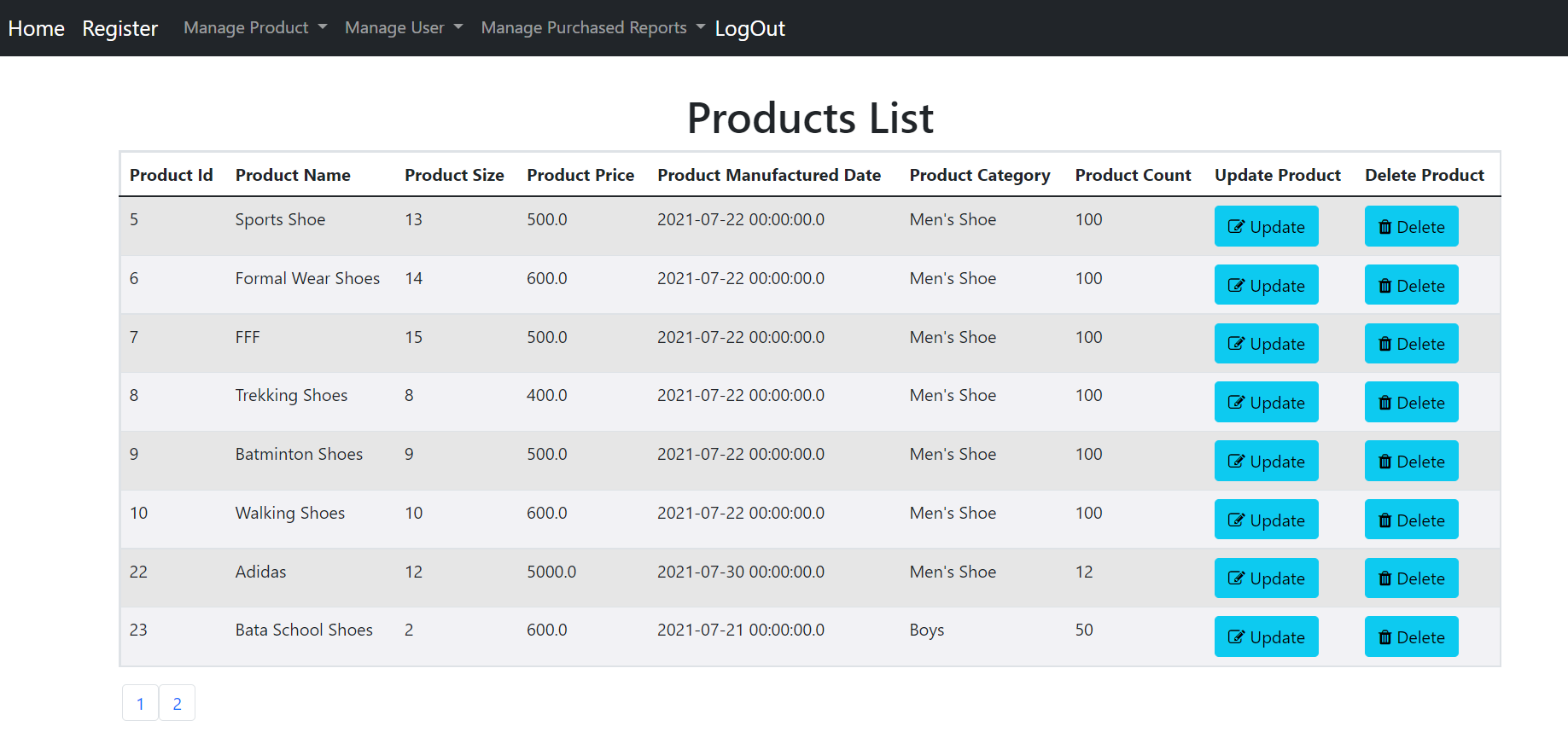
1. **viewCustomer.html**

List all the cutomers details who have signed up.



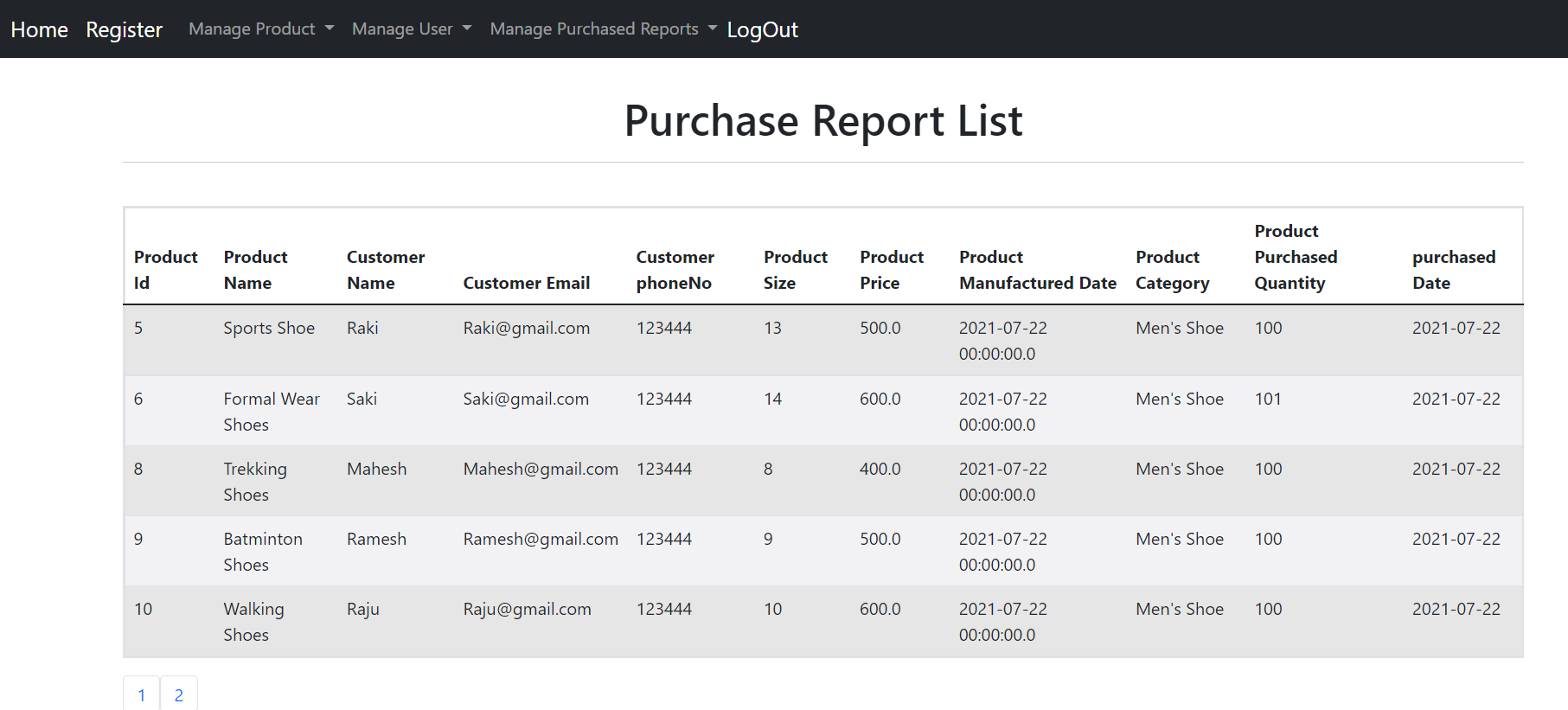
1. **viewProduct.html**

List all the Products added.

****

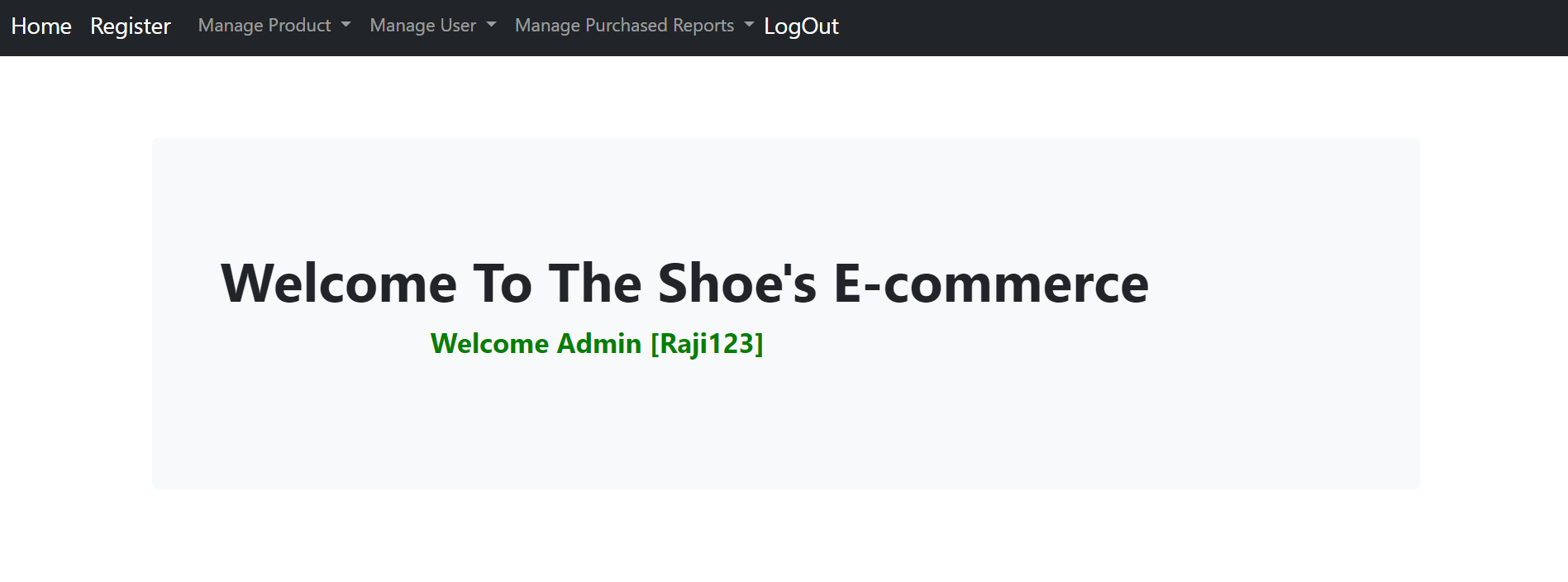
1. **viewPurchaseReport.html**

List all the details of the purchase made through this website.



1. **welcome.html**

This page is displayed, when the admin registers successfully.

****

1. **Controller[com.sportyshoes.ecommerce.controller]**
2. AdminUserController.java
3. CustomerController.java
4. ProductCategoryController.java
5. ProductController.java
6. PurchaseReportController.java
7. **Repositories [com.sportyshoes.ecommerce.repository]**
8. AdminUserRepository.java
9. CustomerRepository.java
10. ProductCategoryRepository.java
11. ProductRepository.java
12. PurchaseReportRepository.java
13. **Models[com.sportyshoes.ecommerce.models]**
14. Airline.java
15. AdminUser.java
16. Customer.java
17. Product.java
18. ProductCategory.java
19. PurchaseReport.java
20. **Service[com.sportyshoes.ecommerce.service]**
21. AdminUserService.java
22. CustomerService.java
23. ProductCategoryService.java
24. ProductService.java
25. PurchasedReportService.java
26. **Service Implementation [com.sportyshoes.ecommerce.service.implementation]**
27. AdminUserServiceImpl.java
28. CustomerServiceImpl.java
29. ProductCategoryServiceImpl.java
30. ProductServiceImpl.java
31. PurchasedReportServiceImpl.java
32. **POJOS[com.sportyshoes.ecommerce.POJO]**
33. CustomerNamePOJO.java
34. GetProductIdPOJO.java
35. PurchaseReportByDatePOJO.java
36. **Validators[com.sportyshoes.ecommerce.validator]**
37. AdminUserValidator.java
    1. com/controller/FlightDetailServlet.java

FlightDetailServlet controller is used to process the data according to the request. The methods used in the controller are

* + 1. **addNewFlightDetail**

Receive the flightDetail through the session and insert it in the backend (Database). Finally after successful creation of the record redirect the request to the FlightList.jsp

* + 1. **deleteFlightByFlightId**

Retrieve the Id of the flight to be deleted from the request URL and delete the flight record in the Database.

* + 1. **updateFlightDetail**

Receive the updated flight Detail through the session and update it in the backend (Database). After successful updation of the record redirect the request to the FlightList.jsp

* + 1. **editFlightDetail**

Receive the edit request along with the details of the flight to be edited and redirect it to the EditFlightDetail.jsp with that flight details.

* + 1. **deleteListedFlight**

Retrieve the Id List of the flights record to be deleted from the request parameter and delete all the flights record in the Database.

* + 1. **getFlightDetailList**

Receive the listing request and connect to the Database to retrieve all the flight records and redirect to the FlightList.jsp

* 1. com/controller/FlightScheduleServlet.java
     1. **getFlightDetailsByTripDateList**

Receive the listing request and connect to the Database to retrieve all the flight details records available by Date and redirect to the ListFlightTicketsAvailability.jsp

* + 1. **populateFlightDetail**

Receives the request along with two inputs and directs the stored procedure to populate the data for the specified days from the given date.

* + 1. **deleteListedFlightByTripDate**

Retrieve the Id of the flightTrip to be deleted from the request URL and delete the flightTrip in the Database.

* + 1. **deleteListedFlightByTripDateById**

Retrieve the Id List of the flightTrip records to be deleted from the request parameter and delete all the flightTrip records in the Database.

* 1. com/controller/FlightTicketSearchServlet.java
     1. **searchForm**

Contact DAO and get the source, destination list , set it in the session attribute and redirect to the SearchForm.jsp

* + 1. **searchFlight**

Receives the request with the details given by the user and redirect it to the corresponding DAO.

* + 1. **bookFlightTicket**

Retrieves the detail of the flight in which the user is going to book the flight ticket and redirect it to the PassengersDetail.jsp

* + 1. **payAndBookTicket**

Retrieves the detail required for booking the flight ticket and redirect to the Dao and then direct to the Ticket.jsp page.

* + 1. **getBookedTicketDetail**

Retrieve the ticket id from the request, redirects to the Dao and with the records returned by the dao , it redirect to the Ticket.jsp page.

* 1. /com/controller/UserRegisterServlet.java
     1. validateUser

Receive username and password from the user, redirects to the DAO and with result returned by the dao it redirects to the respective pages and set the session attributes.

* + 1. registerUser

Receives the details through the request and redirects to the DAO to insert new user record.

* + 1. logoutUser

Receives the request and removes the session attributes and set session invalidate.

1. **DataAccessObject [Dao]**
   1. **com/dao/FlightDetailDao.java**
      1. **registerFlightDetail**

Insert new flightDetails into the table. Before inserting into FlightDetail table checks whether the airline and Source,Destination already exist in their respective table with the help of the checkFlightDetail() method.

* + 1. **checkFlightDetail**

checks whether the airline and Source,Destination already exist in their respective table.If it exist then update the flightDetail with the respective record details.

* + 1. **deleteFlightDetails**

Deletes the record first from FlightAvailabilityByDate, then FlightDetail and finally from FlightRunningDays.

* + 1. **getFlightDetail**

Retrieves all the records available in the FLightDetail Table along with the respective records from airline and TripSourceDestination Table

* + 1. **getFlightDetailById**

Retrieves the record in the FLightDetail Table along with the respective records from airline and TripSourceDestination Table for the given flightId.

* + 1. **updateFlightDetail**

Update the record in the FLightDetail Table along with the respective records in airline and TripSourceDestination Table .

* 1. **com/dao/FlightScheduleDao.java**
     1. **deleteFlightDetailsByTripIds**

Deletes the record from the FlightAvailabilityByDate Table for the given FlightScheduleIds.

* + 1. **populateFlightDetail**

Populate the data by calling the stored procedure with the given data. [specified No. Of days, given start date]

* + 1. **getFlightAvailabilityByDate**

Retrieves all the records available in the FlightAvailabilityByDate Table.

* 1. **com/dao/FlightTicketSearchDao.java**
     1. **getSourceList**

Retrieves all the distinct sourceCityName available in the TripSourceDestination Table.

* + 1. **getDestinationList**

Retrieves all the distinct destinationCityName available in the TripSourceDestination Table.

* + 1. **getAvailableFlight**

Retrieves all the flights available on the given date for the given source destination.

* + 1. **bookTicket**

Deduct the no.Of seats on that flight trip[FlightAvailabilityByDate] and insert the records in the FlightTicket Table and corresponding passengers detail in the PassengersDetail

* + 1. **deductFlightSeat**

Simple method to Deduct the no. of seats on the FlightAvailabilityByDate table.

* + 1. **getFlightTicket**

Retrieves the ticket details from the FlightTicket table for the given ticket Number.

* 1. **com/dao/UserDao.java**
     1. **saveUser**

This method saves the new user details in the user table.

* + 1. **validate**

This method checks whether the given username and password exists in the table.And returns their respective user type.

1. **Models[Hibernate]**
   1. com/model/Airline.java
   2. com/model/FlightAvailabilityByDate.java
   3. com/model/FlightDetail.java
   4. com/model/FlightRunningDays.java
   5. com/model/FlightTicket.java
   6. com/model/PassengersDetail.java
   7. com/model/SearchFlightDetailPojo.java
   8. com/model/TripSourceDestination.java
   9. com/model/User.java