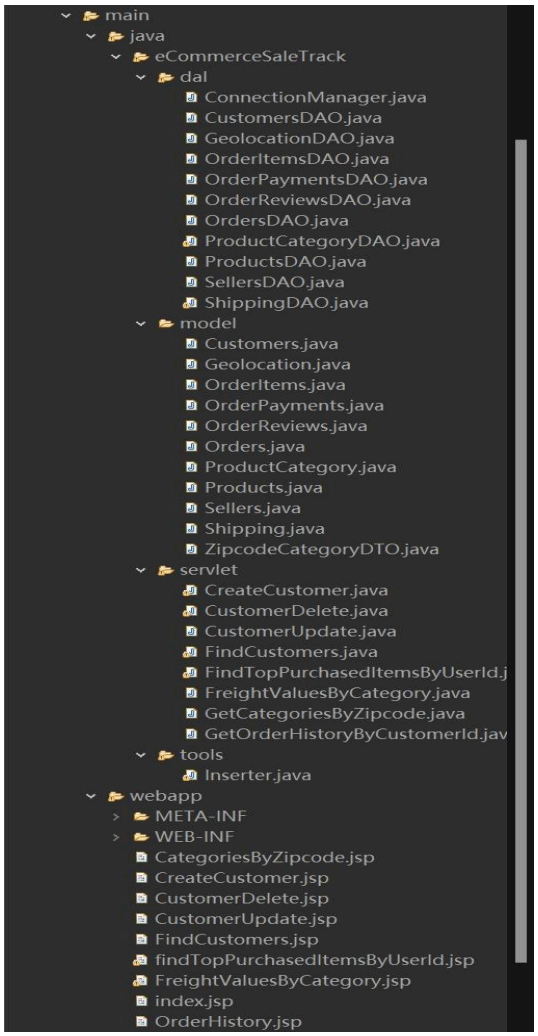
Milestone 4: Hello World

Team name: Team 8
Team Members: Chun-Cheng Liu, Yizhou Chen, Yimei Liang, Dixuan Zhao, Jianchang Li, David Kim, Minh nguyen, Yiwei Li
Project Name: E-commerce SaleTrack

- Use JDBC to build the data access layer for all classes. Use JSP to build the web application.
- At least 4 screenshots that demonstrate the ability to perform the following operations on your database: create, read, update, delete (CRUD). Put the screenshots in a single PDF file. For each screenshot, provide a quick narrative of the workflow.
- Screenshot and description of your advanced feature that you defined in Milestone 1.

Use JDBC to build the data access layer for all classes. Use JSP to build the web application.

Project structure



Screenshots that demonstrate the ability to perform the following operations on your database: create, read, update, delete (CRUD):

CREATE: create statement for ten classes

```java
// 1. Create Geolocation entries (needed by Customers and Sellers)
Geolocation geo1 = new Geolocation("12345", 40.7128f, -74.0060f, "New York", "NY");
geo1 = geolocationDao.create(geo1);
System.out.println("Created geolocation: " + geo1.getGeolocationZipCodePrefix());

Geolocation geo2 = new Geolocation("67890", 34.0522f, -118.2437f, "Los Angeles", "CA");
geo2 = geolocationDao.create(geo2);
System.out.println("Created geolocation: " + geo2.getGeolocationZipCodePrefix());

// 2. Create Customer using existing Geolocation
Customers customer1 = new Customers("cust123", "unique123", "12345", "New York", "NY");
customer1 = customersDao.create(customer1);
System.out.println("Created customer: " + customer1.getCustomerId());

// 3. Create Seller using existing Geolocation
Sellers seller1 = new Sellers("seller123", "12345", "New York", "NY");
seller1 = sellersDao.create(seller1);
System.out.println("Created seller: " + seller1.getSellerId());

// 4. Create ProductCategory (needed by Products)
ProductCategory category1 = new ProductCategory("Electronics", "Electronics");
category1 = productCategoryDao.create(category1);
System.out.println("Created product category: " + category1.getProductCategoryName());

// 5. Create Product using existing ProductCategory
Products product1 = new Products("prod123", "Electronics", 10, 30, 5, 1000, 15, 10, 5);
product1 = productsDao.create(product1);
System.out.println("Created product: " + product1.getProductId());

// 6. Create Order using existing Customer
Orders order1 = new Orders("order123", "cust123", Orders.OrderStatus.delivered, new Date(), new Date(), new Date(), new Date(), new Date());
order1 = ordersDao.create(order1);
System.out.println("Created order: " + order1.getOrderId());

// 7. Create OrderItem using existing Order, Product, and Seller
OrderItems orderItem1 = new OrderItems("order123", 1, "prod123", "seller123", new Date(), 29.99, 5.99);
orderItem1 = orderItemsDao.create(orderItem1);
System.out.println("Created order item: " + orderItem1.getOrderId());

// 8. Create OrderPayment using existing Order
OrderPayments orderPayment1 = new OrderPayments("order123", 1, OrderPayments.PaymentType.credit_card, 3, 99.99);
orderPayment1 = orderPaymentsDao.create(orderPayment1);
System.out.println("Created order payment: " + orderPayment1.getOrderId());

// 9. Create OrderReview using existing Order
OrderReviews review1 = new OrderReviews("review123", "order123", 5, "Excellent", "Great service", new Date(), new Date());
review1 = orderReviewsDao.create(review1);
System.out.println("Created order review: " + review1.getReviewId());

// 10. Create Shipping record using existing Order and Customer
Shipping shipping1 = new Shipping("order123", "cust123", "12345", "NY", 50.0, 500.0, 550.0, new Date());
shipping1 = shippingDao.create(shipping1);
System.out.println("Created shipping record with ShippingId: " + shipping1.getShippingId());
```

READ: read part of four classes

```java
// READ operations
String zipCodeToFetch = "12345";
Geolocation fetchedGeo = geolocationDao.getGeolocationByZipCodePrefix(zipCodeToFetch);
if (fetchedGeo != null) {
    System.out.format("Fetched geolocation: ZipCodePrefix=%s, City=%s, State=%s, Lat=%.4f, Lng=%.4f\n",
        fetchedGeo.getGeolocationZipCodePrefix(), fetchedGeo.getGeolocationCity(),
        fetchedGeo.getGeolocationState(), fetchedGeo.getGeolocationLat(),
        fetchedGeo.getGeolocationLng());
} else {
    System.out.println("Geolocation not found with ZipCodePrefix: " + zipCodeToFetch);
}

Customers fetchedCustomer = customersDao.getCustomerById("cust123");
if (fetchedCustomer != null) {
    System.out.format("Fetched customer: CustomerId=%s, City=%s, State=%s\n",
        fetchedCustomer.getCustomerId(), fetchedCustomer.getCustomerCity(), fetchedCustomer.getCustomerState());
}

OrderItems fetchedOrderItem = orderItemsDao.getOrderItemById("order123", 1);
if (fetchedOrderItem != null) {
    System.out.format("Fetched order item: OrderId=%s, ProductId=%s, Price=%.2f\n",
        fetchedOrderItem.getOrderId(), fetchedOrderItem.getProductId(), fetchedOrderItem.getPrice());
}

OrderPayments fetchedOrderPayment = orderPaymentsDao.getOrderPaymentById("order123", 1);
if (fetchedOrderPayment != null) {
    System.out.format("Fetched order payment: OrderId=%s, PaymentType=%s, Amount=%.2f\n",
        fetchedOrderPayment.getOrderId(), fetchedOrderPayment.getPaymentType(), fetchedOrderPayment.getPaymentValue());
}
```

UPDATE: update part of four classes

```java
// UPDATE operations
if (fetchedGeo != null) {
    Geolocation updatedGeo = geolocationDao.updateGeolocationCityAndState(fetchedGeo, "Buffalo", "NY");
    System.out.format("Updated geolocation: ZipCodePrefix=%s, New City=%s, New State=%s\n",
        updatedGeo.getGeolocationZipCodePrefix(), updatedGeo.getGeolocationCity(),
        updatedGeo.getGeolocationState());
}

if (fetchedCustomer != null) {
    Customers updatedCustomer = customersDao.updateCustomerCityAndState(fetchedCustomer, "Buffalo", "NY");
    System.out.format("Updated customer: CustomerId=%s, New City=%s, New State=%s\n",
        updatedCustomer.getCustomerId(), updatedCustomer.getCustomerCity(), updatedCustomer.getCustomerState());
}

if (fetchedOrderItem != null) {
    OrderItems updatedOrderItem = orderItemsDao.updatePriceAndFreightValue(fetchedOrderItem, 34.99, 6.99);
    System.out.format("Updated order item: OrderId=%s, New Price=%.2f, New Freight=%.2f\n",
        updatedOrderItem.getOrderId(), updatedOrderItem.getPrice(), updatedOrderItem.getFreightValue());
}

if (fetchedOrderPayment != null) {
    OrderPayments updatedOrderPayment = orderPaymentsDao.updatePaymentTypeAndValue(fetchedOrderPayment, OrderPayments.PaymentType.debit_card, 89.99);
    System.out.format("Updated order payment: OrderId=%s, New Type=%s, New Amount=%.2f\n",
        updatedOrderPayment.getOrderId(), updatedOrderPayment.getPaymentType(), updatedOrderPayment.getPaymentValue());
}
```

DELETE: delete part of four classes

```java
// DELETE operations
String zipCodeToDelete = "67890";
Geolocation deletedGeo = geolocationDao.delete(geo2);
System.out.println("Geolocation with ZipCodePrefix " + zipCodeToDelete + " deleted: " + (deletedGeo == null));

Customers deletedCustomer = customersDao.delete(customer1);
System.out.println("Customer deleted: " + (deletedCustomer == null));

OrderItems deletedOrderItem = orderItemsDao.delete(orderItem1);
System.out.println("Order item deleted: " + (deletedOrderItem == null));

OrderPayments deletedOrderPayment = orderPaymentsDao.delete(orderPayment1);
System.out.println("Order payment deleted: " + (deletedOrderPayment == null));
```

RESULT

eclipse interface

```
Created geolocation: 12345
Created geolocation: 67890
Created customer: cust123
Created seller: seller123
Created product category: Electronics
Created product: prod123
Created order: order123
Created order item: order123
Created order payment: order123
Created order review: review123
Created shipping record with ShippingId: 1
Fetched geolocation: ZipCodePrefix=12345, City=New York, State=NY, Lat=40.7128, Lng=-74.0060
Fetched customer: CustomerId=cust123, City=New York, State=NY
```

```
Fetched geolocation: ZipCodePrefix=12345, City=New York, State=NY, Lat=40.7128, Lng=-74.0060
Fetched customer: CustomerId=cust123, City=New York, State=NY
Fetched order item: OrderId=order123, ProductId=prod123, Price=29.99
Fetched order payment: OrderId=order123, PaymentType=credit_card, Amount=99.99
Updated geolocation: ZipCodePrefix=12345, New City=Buffalo, New State=NY
Updated customer: CustomerId=cust123, New City=Buffalo, New State=NY
Updated order item: OrderId=order123, New Price=34.99, New Freight=6.99
Updated order payment: OrderId=order123, New Type=debit_card, New Amount=89.99
Geolocation with ZipCodePrefix 67890 deleted: true
Customer deleted: true
Order item deleted: true
Order payment deleted: true
All done
```

database interface

```
1 •  select * from Geolocation
```

Result Grid | Filter Rows: | Edit: | Export/Import:

GeolocationZipCodePrefix	GeolocationLat	GeolocationLng	GeolocationCity	GeolocationState
12345	40.7128	-74.006	Buffalo	NY
NULL	NULL	NULL	NULL	NULL

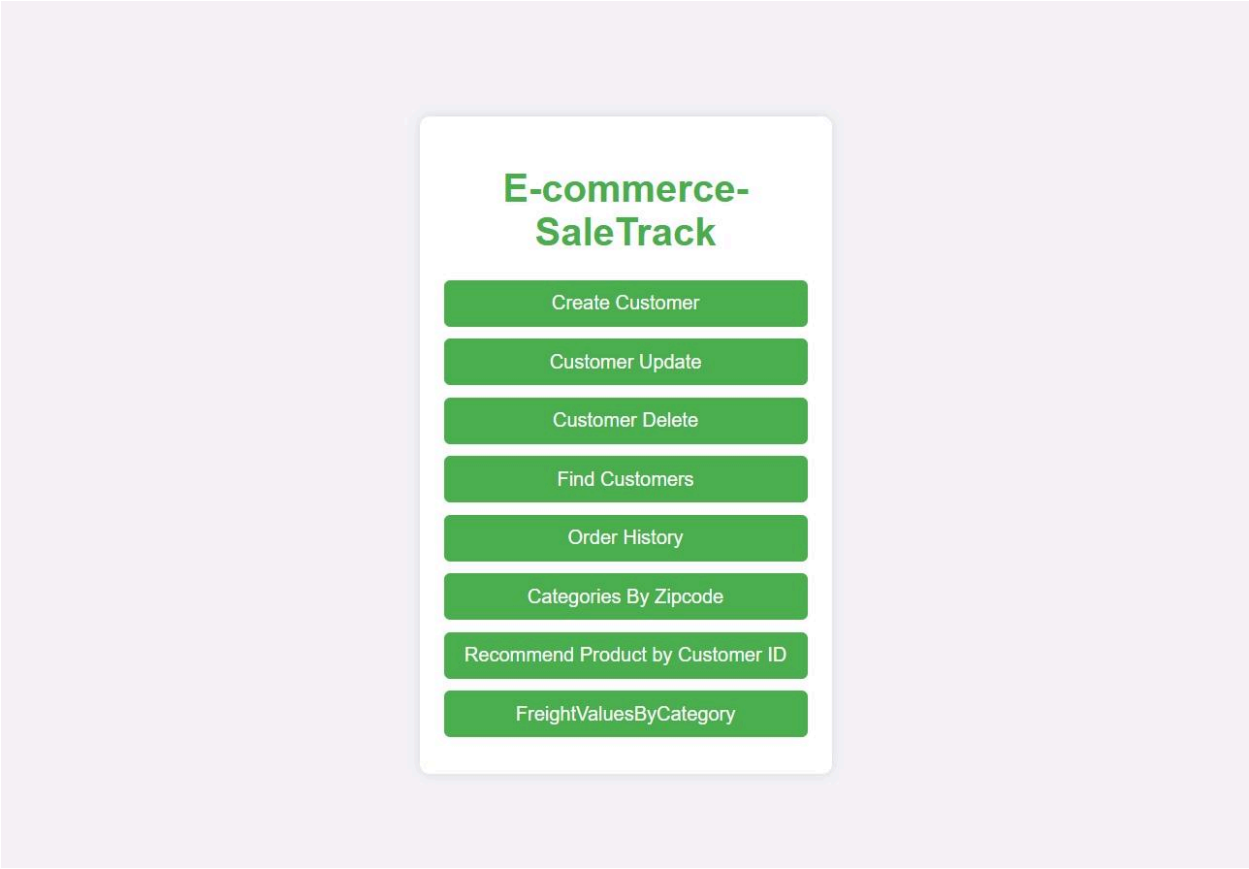
```
1 •  select * from Products
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

ProductId	ProductCategoryName	ProductNameLength	ProductDescriptionLength	ProductPhotosQty	ProductWeightG	ProductLengthCm	ProductHeightCm	P
prod123	Electronics	10	30	5	1000	15	10	5
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NU

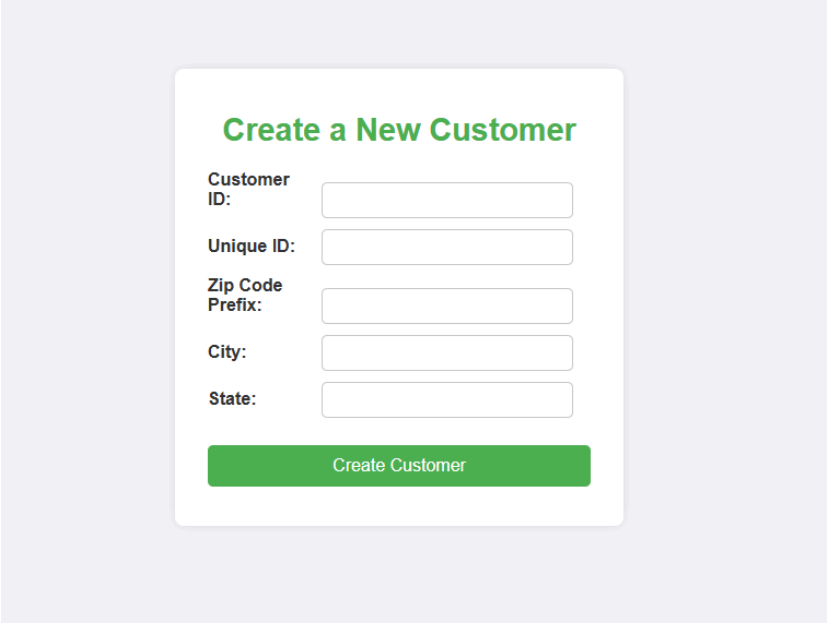
Screenshots and description of advanced feature

- Main menu

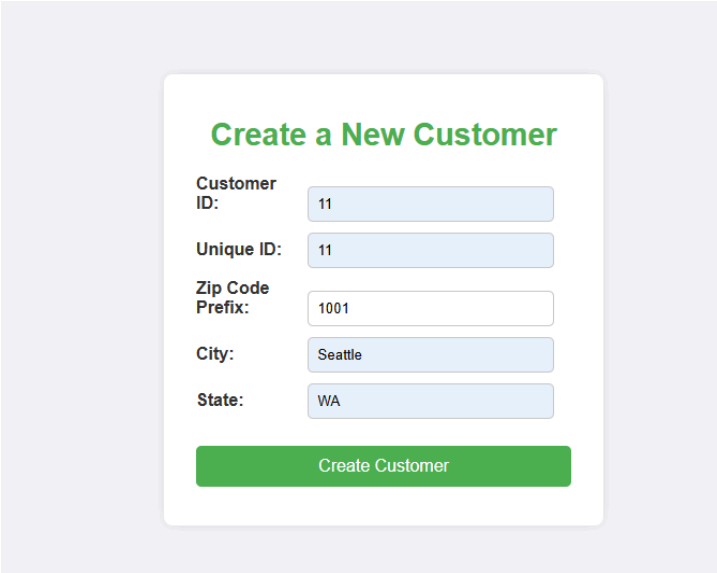


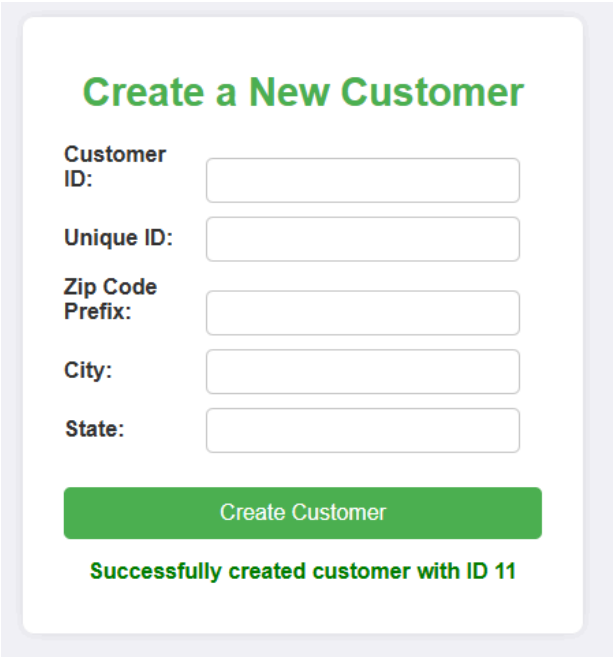
- Create customer

Workflow narrative: To create a new customer, the user provides all required customer details, including a valid zip code that exists in the Geolocation table. The INSERT query is executed, and a confirmation is displayed to ensure the customer was added successfully.



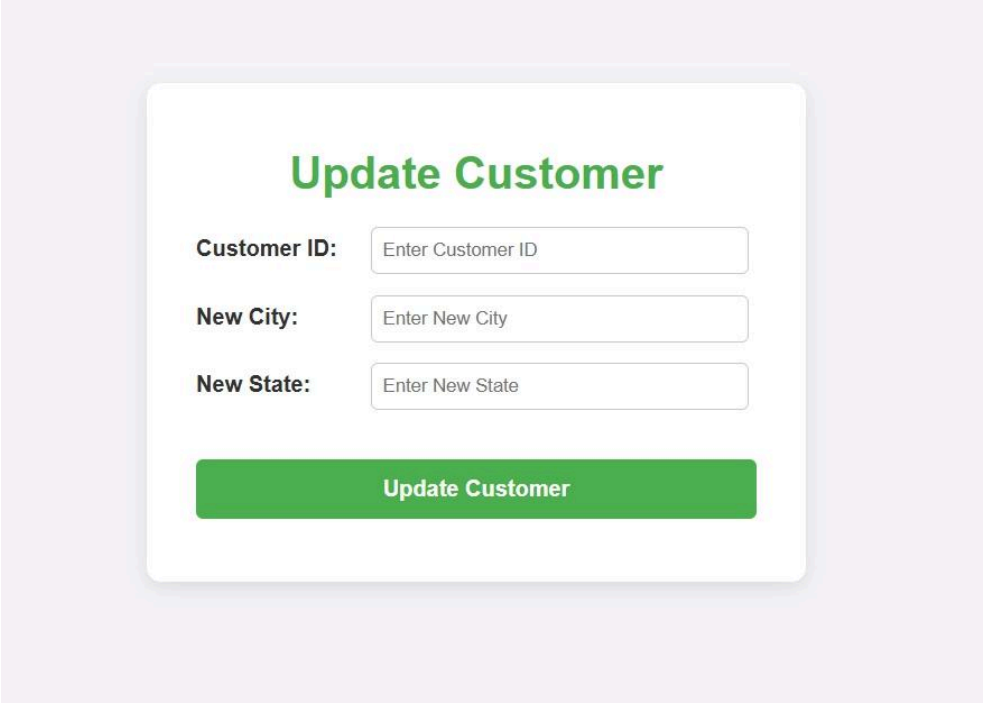
(Please enter a unique customer ID and a valid zip code. Ensure that the zip code exists in the referenced table. For example, the zip code ‘1001’ exists in the geolocation table. Otherwise, the webpage will display an error message: “Cannot add or update a child row: a foreign key constraint fails.”)



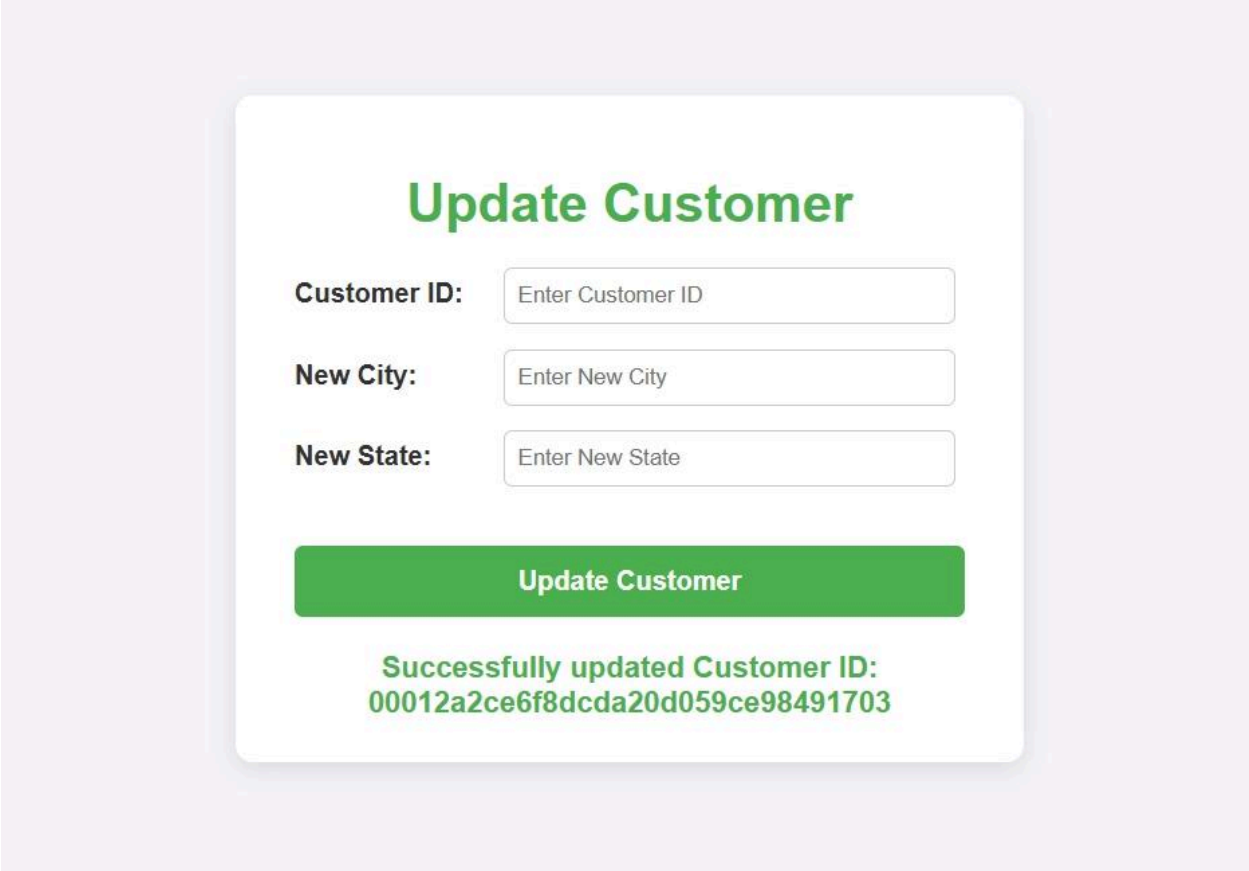


- Update customer

Workflow narrative: To update a customer's information, the user specifies the CustomerId and provides the new city and state values. The system runs an UPDATE query, modifying the existing record, and displays the updated information to confirm the change.

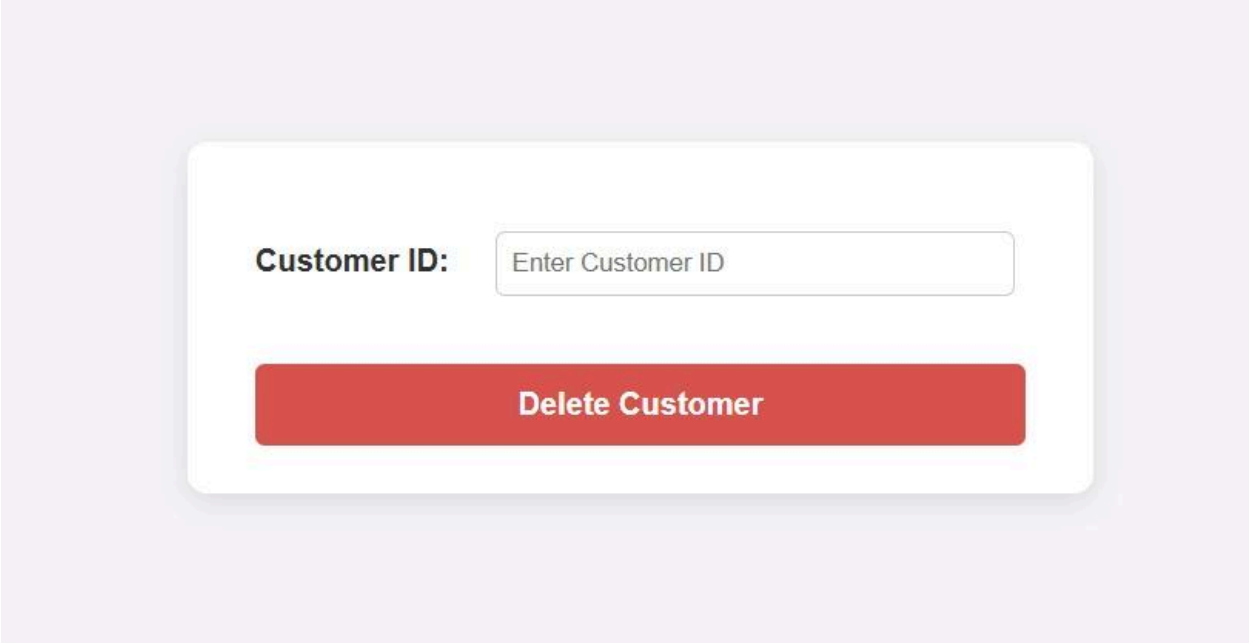


(Please enter an existing customer ID. If the provided customer ID does not exist, the webpage will display an error message indicating that Customer ID does not exist.)

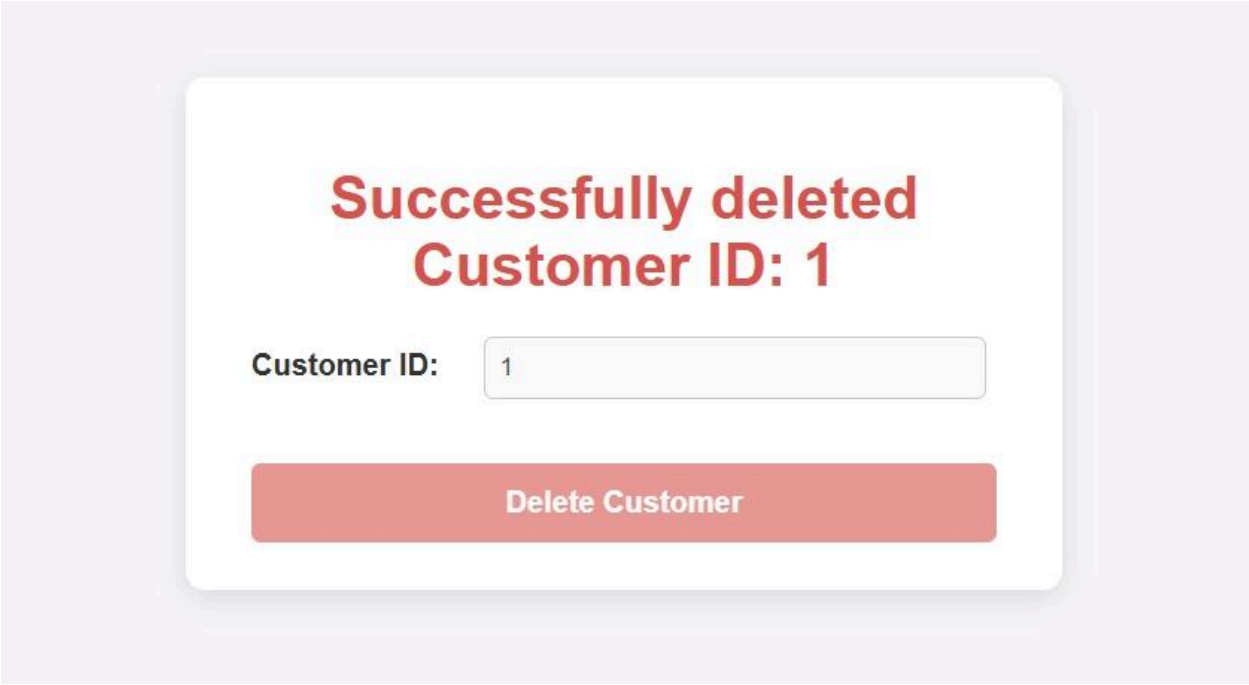


- Delete customer

Workflow narrative: To delete a customer, the user inputs the customer ID of the record to be removed. The system validates the ID, executes the DELETE query, and provides confirmation that the operation was successful.

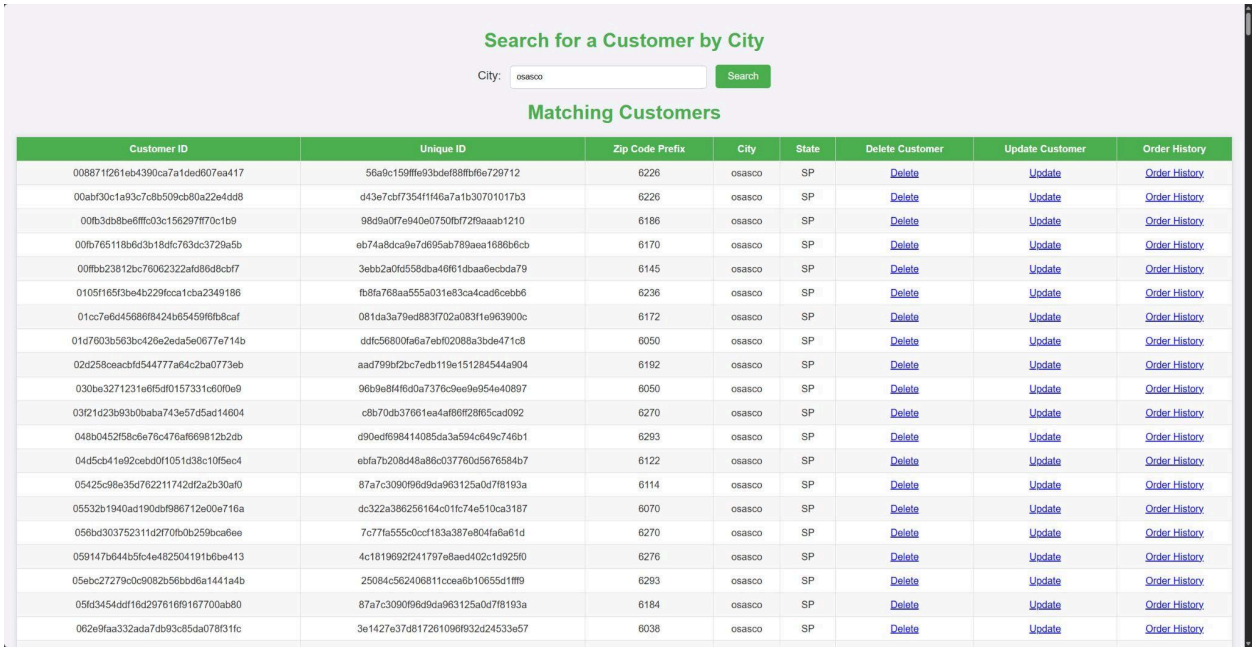


(Please enter an existing customer ID to delete. If the provided customer ID does not exist, the webpage will display an error message indicating, “Failed to delete Customer ID.”)



- Find customer

Workflow narrative: The user provides the customer ID to search for their details. The system executes a SELECT query to fetch the record, displaying it on the screen. This confirms that the customer exists and provides their information for further operations.



Search for a Customer by City

City: osasco Search

Matching Customers

Customer ID	Unique ID	Zip Code Prefix	City	State	Delete Customer	Update Customer	Order History
008871f261eb4390ca7a1ded607ea417	56a9c159fffe93bdef88ffbf6e729712	6226	osasco	SP	Delete	Update	Order History
00abf30c1a93c7c8b509cb80a22e4dd8	d43e7cbf7354f1f46a7a1b30701017b3	6226	osasco	SP	Delete	Update	Order History
00fb3db8be6fffc03c156297ff70c1b9	98d9a0f7e940e0750fbf72f9aaab1210	6186	osasco	SP	Delete	Update	Order History
00fb765118b6d3b18dfc763dc3729a5b	eb74a8dca9e7d695ab789aea1686b6cb	6170	osasco	SP	Delete	Update	Order History
00ffbb23812bc76062322afd86d8cbf7	3ebb2a0fd558dba46f61dbaa6ecbda79	6145	osasco	SP	Delete	Update	Order History
0105f165f3be4b229fcca1cba2349186	fb8fa768aa555a031e83ca4cad6cebb6	6236	osasco	SP	Delete	Update	Order History
01cc7e6d45686f8424b65459f6fb8caf	081da3a79ed883f702a083f1e963900c	6172	osasco	SP	Delete	Update	Order History
01d7603b563bc426e2eda5e0677e714b	ddfc56800fa6a7ebf02088a3bde471c8	6050	osasco	SP	Delete	Update	Order History
02d258ceacbfd544777a64c2ba0773eb	aad799bf2bc7edb119e151284544a904	6192	osasco	SP	Delete	Update	Order History
030be3271231e6f5df0157331c60f0e9	96b9e8f4f6d0a7376c9ee9e954e40897	6050	osasco	SP	Delete	Update	Order History
03f21d23b93b0baba743e57d5ad14604	c8b70db37661ea4af86ff28f65cad092	6270	osasco	SP	Delete	Update	Order History
048b0452f58c6e76c476af669812b2db	d90edf698414085da3a594c649c746b1	6293	osasco	SP	Delete	Update	Order History
04d5cb41e92cebd0f1051d38c10f5ec4	ebfa7b208d48a86c037760d5676584b7	6122	osasco	SP	Delete	Update	Order History
05425c98e35d762211742df2a2b30af0	87a7c3090f96d9da963125a0d7f8193a	6114	osasco	SP	Delete	Update	Order History
05532b1940ad190dbf986712e00e716a	dc322a386256164c01fc74e510ca3187	6070	osasco	SP	Delete	Update	Order History
056bd303752311d2f70fb0b259bca6ee	7c77fa555c0ccf183a387e804fa6a61d	6270	osasco	SP	Delete	Update	Order History
059147b644b5fc4e482504191b6be413	4c1819692f241797e8aed402c1d925f0	6276	osasco	SP	Delete	Update	Order History
05ebc27279c0c9082b56bbd6a1441a4b	25084c562406811ccea6b10655d1fff9	6293	osasco	SP	Delete	Update	Order History
05fd3454ddf16d297616f9167700ab80	87a7c3090f96d9da963125a0d7f8193a	6184	osasco	SP	Delete	Update	Order History
062e9faa332ada7db93c85da078f31fc	3e1427e37d817261096f932d24533e57	6038	osasco	SP	Delete	Update	Order History

(Please enter an existing city name to search for customers. If the provided city name does not exist in the database, the webpage will display an error message “No customers found for city: ”.)

- Get history order

Order History

Customer ID: Enter Customer ID Search

Displaying order history for Customer ID: 000161a058600d5901f007fab4c27140

Matching Orders

Order ID	Order Status	Purchase Timestamp	Approved At	Carrier Delivery Date	Customer Delivery Date	Estimated Delivery Date
a44895d095d7e0702b6a162fa2dbeced	delivered	2017-07-16 02:40:32	2017-07-16 02:55:12	2017-07-19 12:09:37	2017-07-25 11:57:33	2017-08-03 17:00:00

(Please enter an existing customer ID to retrieve the order history. If the provided customer ID does not exist, the webpage will display a message “No matching orders found ”.)

- Get seller data by zipcode

Categories by Zipcode

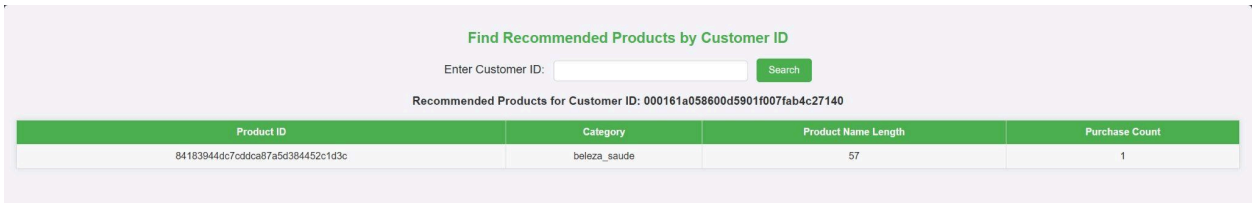
ZIP Code: Enter ZIP Code Search

Category Name	ZIP Code	Count
health_beauty	9080	9
furniture_decor	9080	6
pet_shop	9080	3
housewares	9080	3
sports_leisure	9080	2
computers_accessories	9080	2
audio	9080	1
auto	9080	1
bed_bath_table	9080	1
consoles_games	9080	1
construction_tools_safety	9080	1
electronics	9080	1
stationery	9080	1
perfumery	9080	1
telephony	9080	1

Displaying categories for ZIP code: 9080

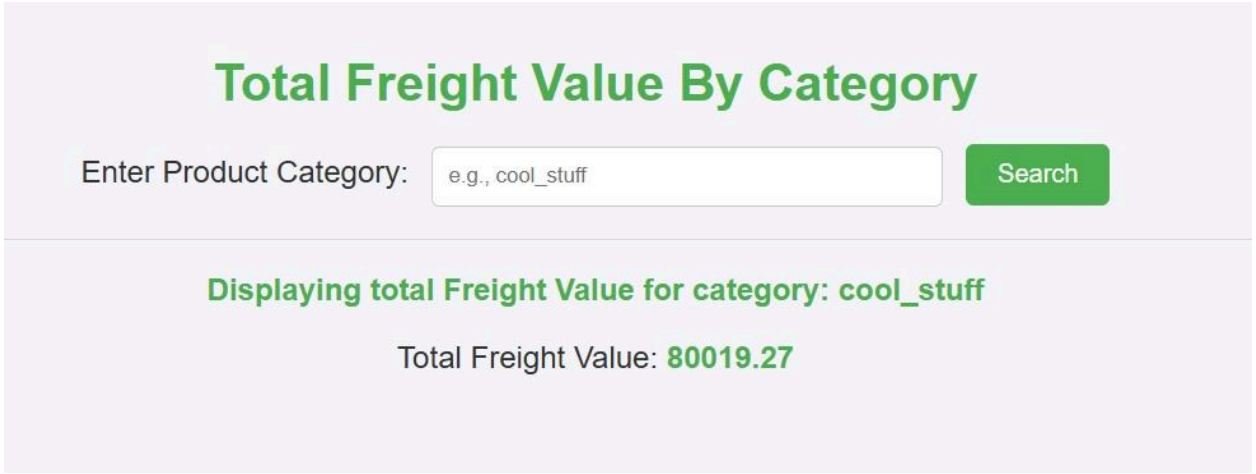
(Please enter a valid zip code to retrieve seller data. If the provided zip code does not exist in the database, the webpage will display an error message indicating, “No categories found for the provided ZIP code.”)

- Recommend product for customer



The feature recommends products for a customer based on their purchase history by identifying the most frequently purchased items using the customer's unique ID. Please enter a valid customer id that already exists in the Customers table.

- Get Freight Values by Category



This feature returns the freight value for each category. The freight value is the transportation cost to move goods from one place to another. In this way, business owners can analyze which goods are more expensive or cheaper to transport during their orders. The user enters their desired product category and the freight value is displayed.