Lesson 01 Demo 07

Updating Documents Using MongoDB CRUD Operations

Objective: To work with the CRUD operations and update the documents for existing and non-existing customers

Tools required: Eclipse IDE

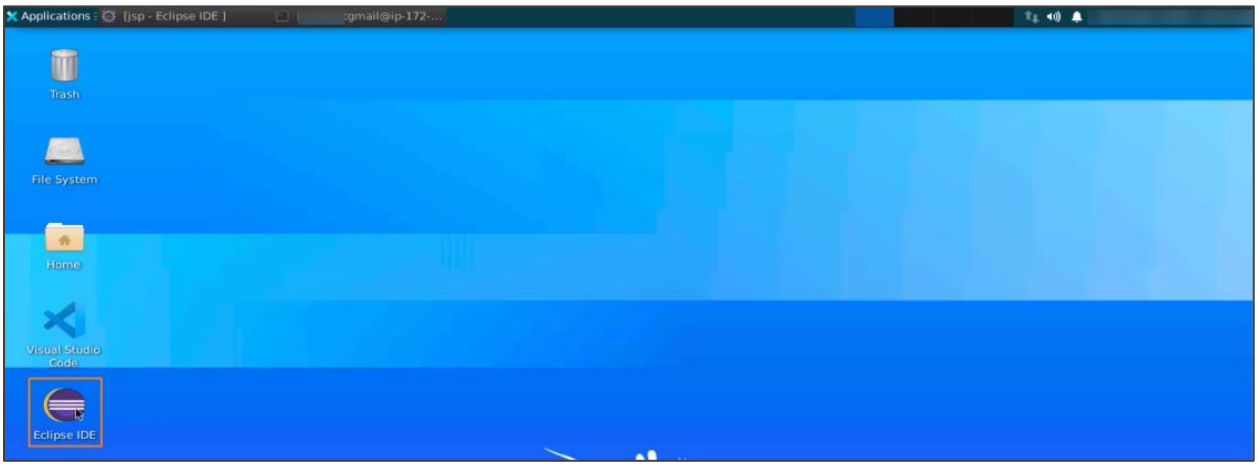
Prerequisites: None

Steps to be followed:

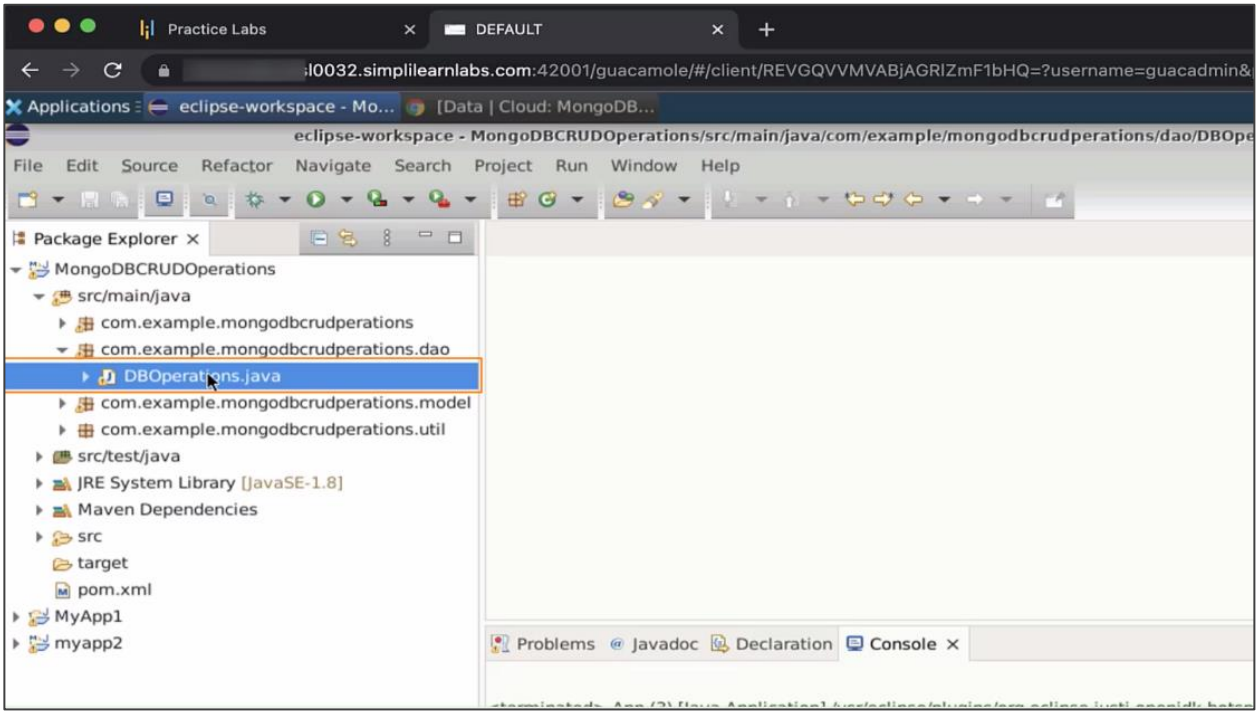
1. Update customer operations
2. Add try catch block
3. Execute another operation for an update
4. Create new function
5. Create more functions
6. Write a new method

Step 1: Update customer operations

1.1 Open the **Eclipse IDE**

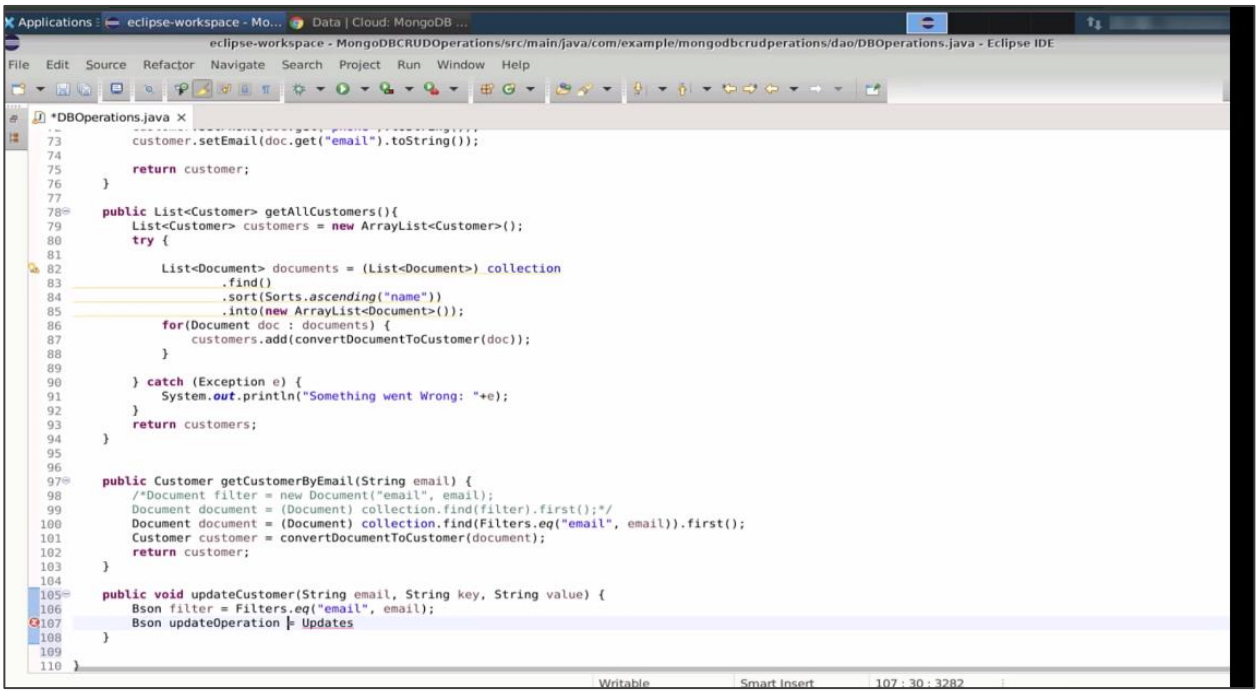


1.2 Navigate to the project **MongoDBCRUDOperations** and open the **DBOperations.java** file

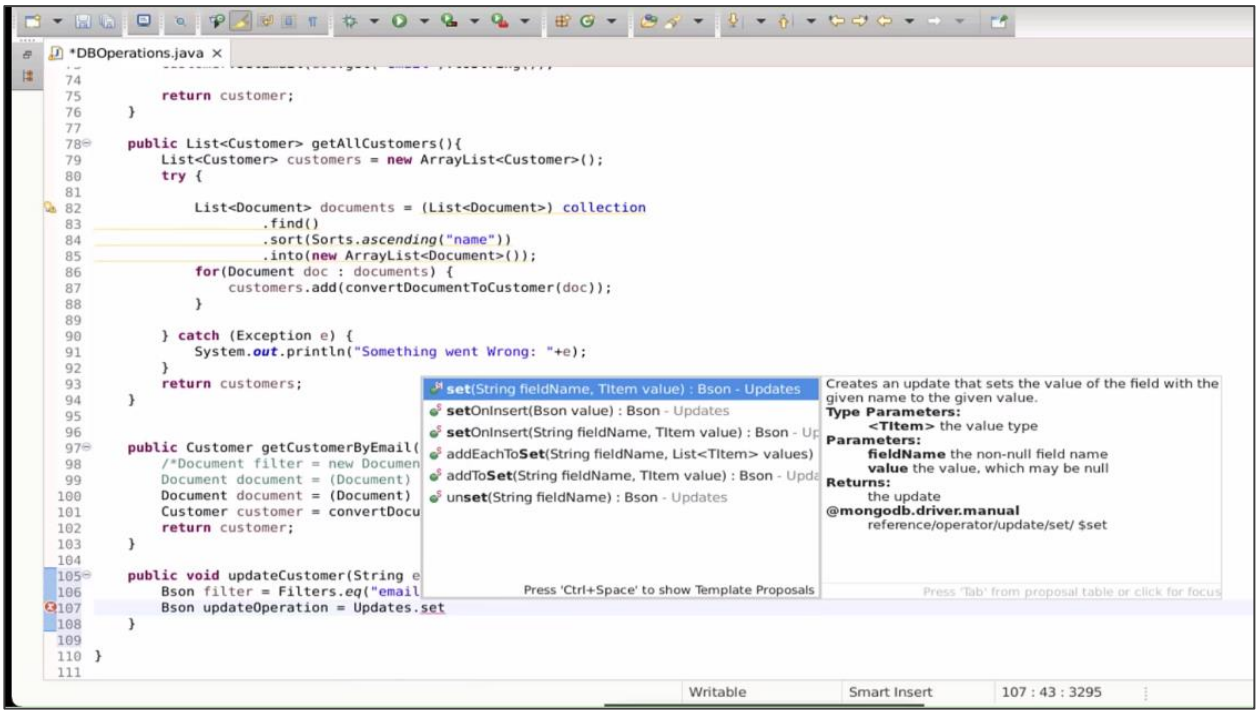


Note: Please refer to the previous demo on how to create the **MongoDBCRUDOperations** project

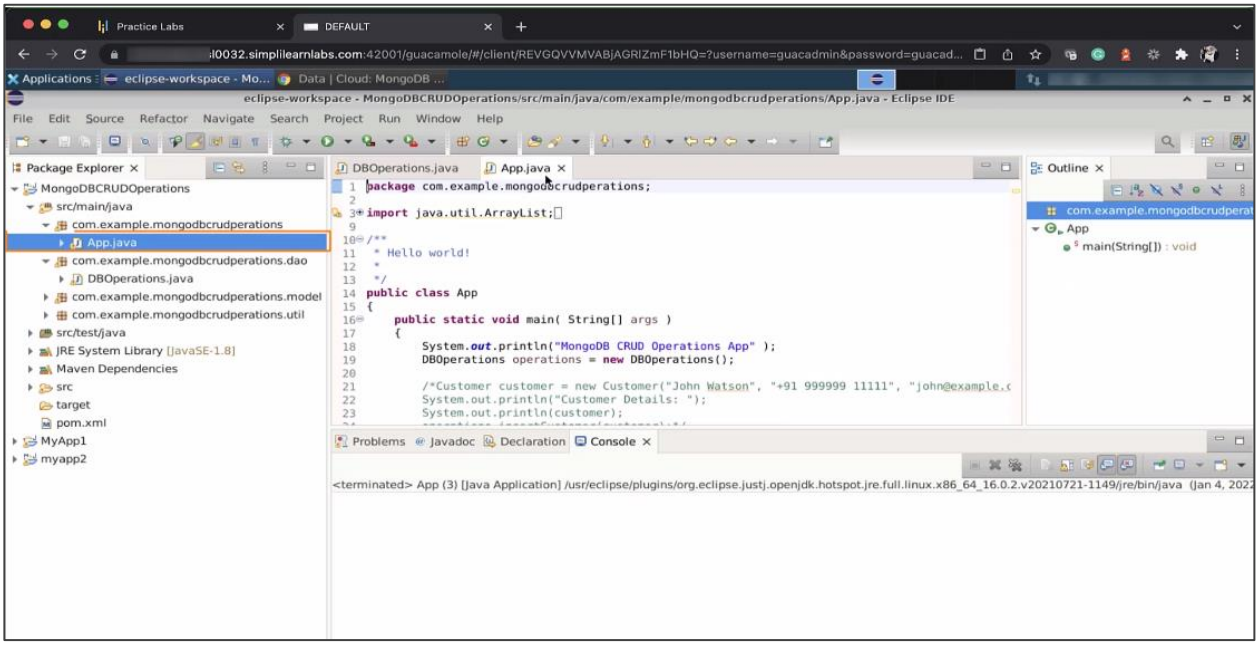
1.3 Write a new method to update the customer and add inputs to the method



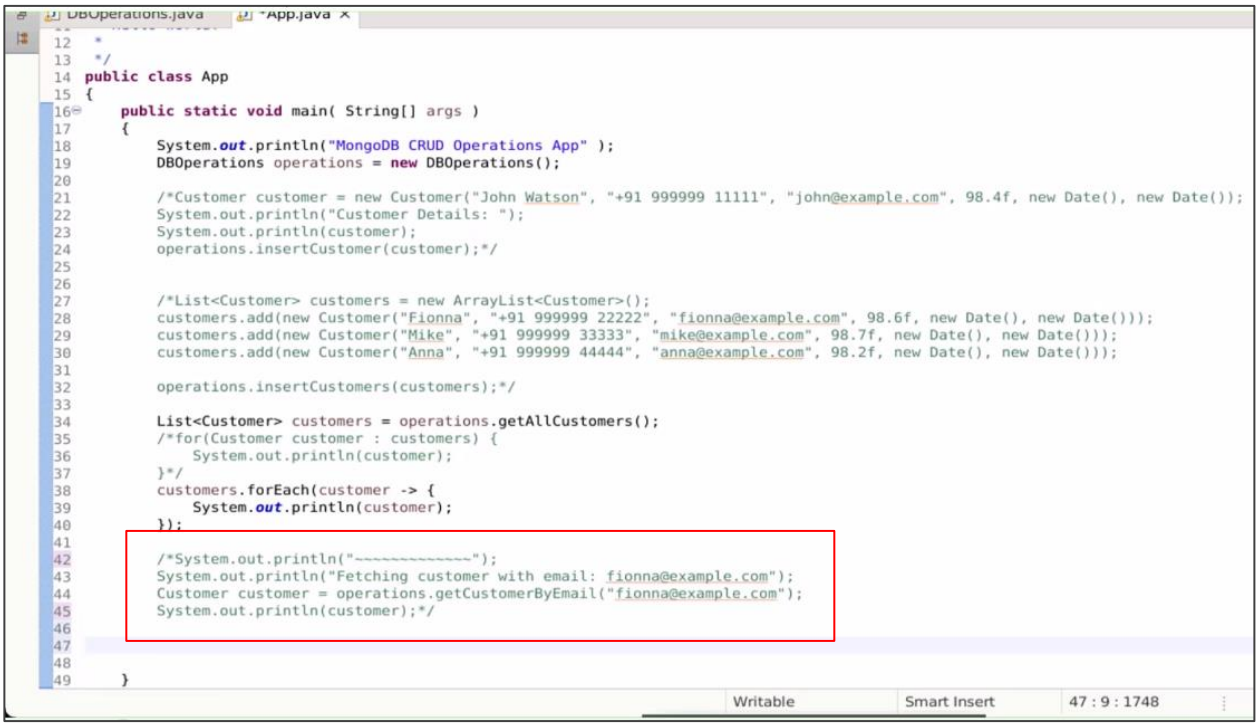
1.4 Select the **set** method to set the key-value pair to filter the data later



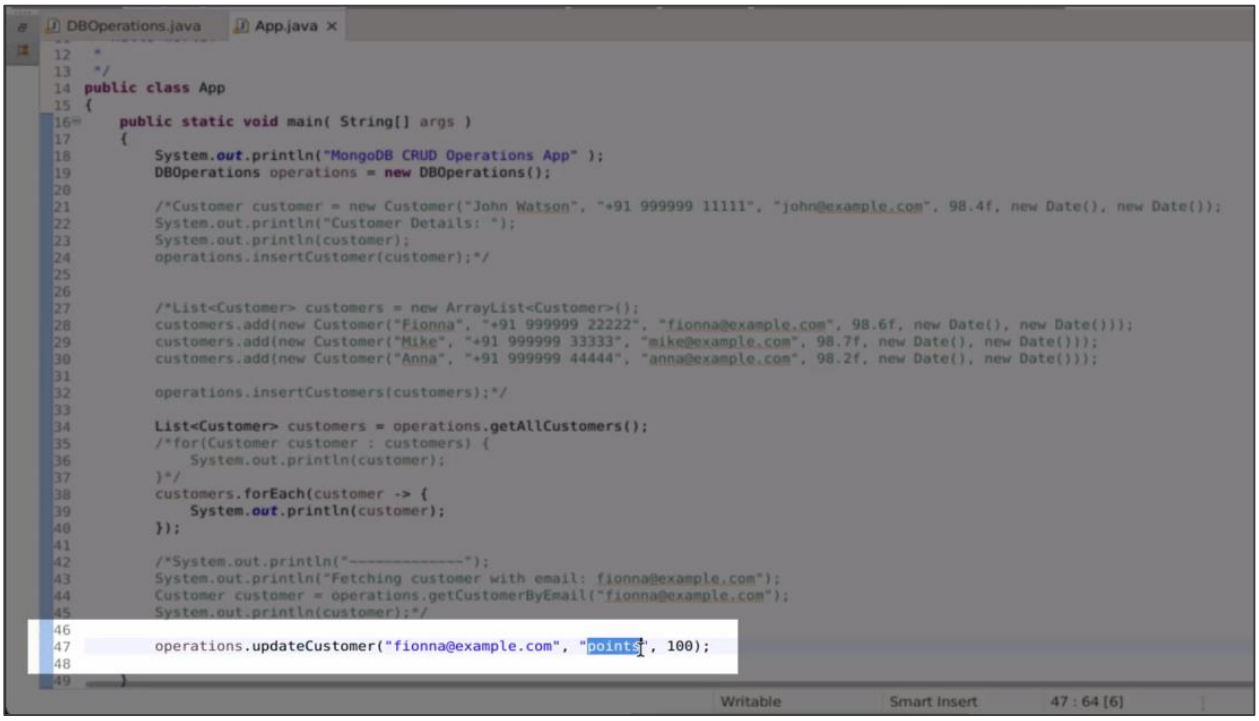
1.5 Navigate back to the **App.java** file



1.6 Comment out the print statements in the code



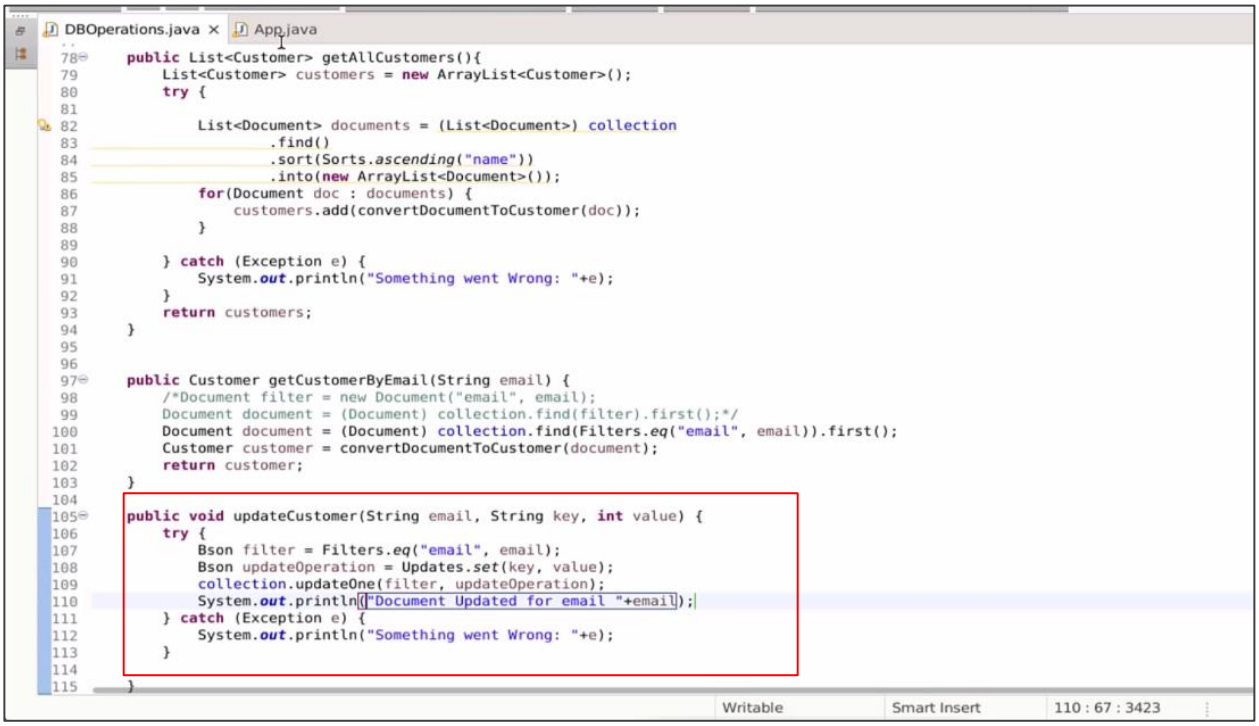
1.7 Use the **updateCustomer** function to update the details of the customer. Note that the key is **points** and the value is **100**



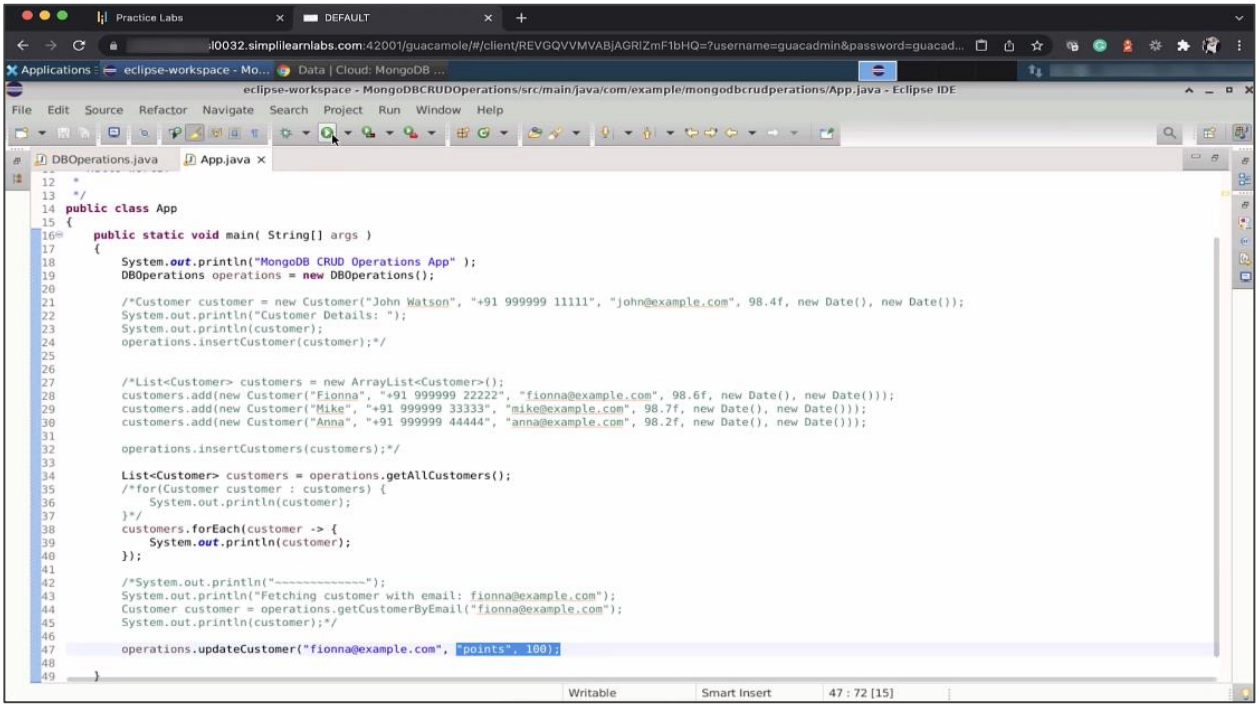
1.8 Go back to the **DBOperations.java**, add a collection and execute the **updateOne** function to perform a single update

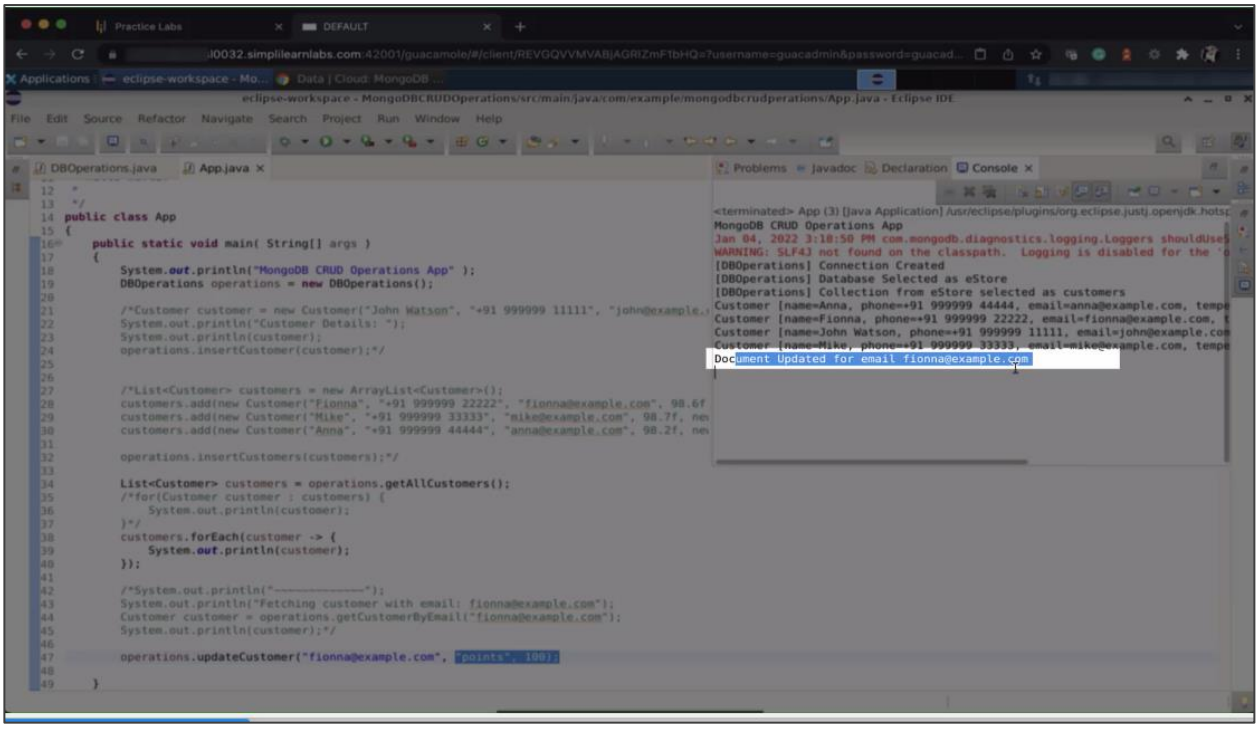
Step 2: Add try catch block

2.1 Add a try catch block in the **DBOperations.java** file for the update operation

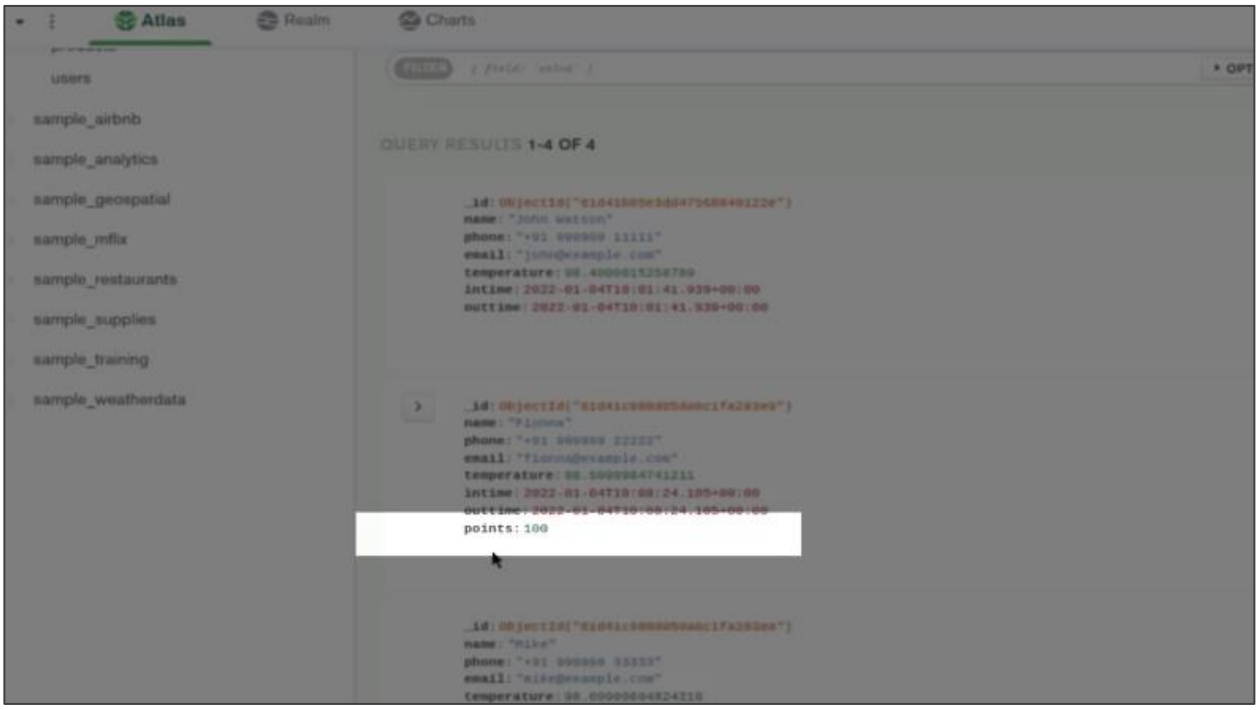


2.2 Go back to **App.java** and run the code





You can see the output as **Document Updated for email fionna@example.com**.



A new key-value pair will be added to the database.

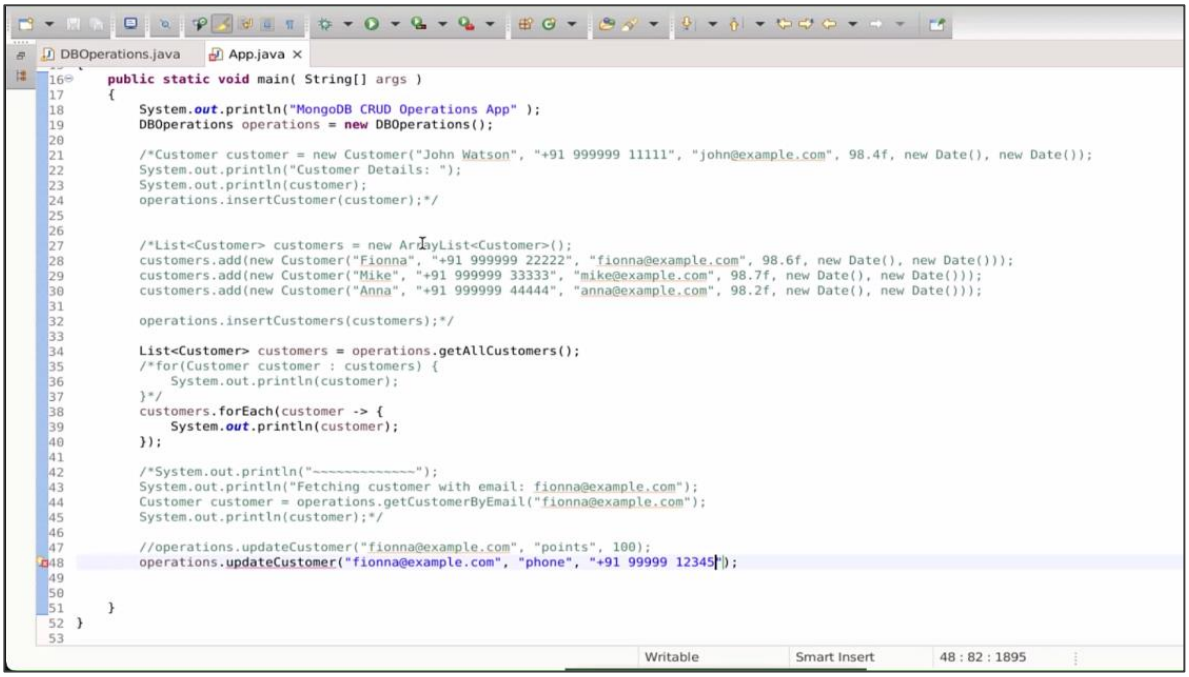
2.3 Comment out the **updateCustomer** operation

```
13  */
14 public class App
15 {
16     public static void main( String[] args )
17     {
18         System.out.println("MongoDB CRUD Operations App" );
19         DBOperations operations = new DBOperations();
20
21         /*Customer customer = new Customer("John Watson", "+91 999999 11111", "john@example.com", 98.4f, new Date(), new D
22         System.out.println("Customer Details: ");
23         System.out.println(customer);
24         operations.insertCustomer(customer);*/
25
26
27         /*List<Customer> customers = new ArrayList<Customer>();
28         customers.add(new Customer("Fionna", "+91 999999 22222", "fionna@example.com", 98.6f, new Date(), new Date()));
29         customers.add(new Customer("Mike", "+91 999999 33333", "mike@example.com", 98.7f, new Date(), new Date()));
30         customers.add(new Customer("Anna", "+91 999999 44444", "anna@example.com", 98.2f, new Date(), new Date()));
31
32         operations.insertCustomers(customers);*/
33
34         List<Customer> customers = operations.getAllCustomers();
35         /*for(Customer customer : customers) {
36             System.out.println(customer);
37         }*/
38         customers.forEach(customer -> {
39             System.out.println(customer);
40         });
41
42         /*System.out.println("~~~~~~~~~~~~~");
43         System.out.println("Fetching customer with email: fionna@example.com");
44         Customer customer = operations.getCustomerByEmail("fionna@example.com");
45         System.out.println(customer);*/
46
47         //operations.updateCustomer("fionna@example.com", "points", 100);
48
49     }
```

Writable | Smart Insert | 47 : 11 : 1750

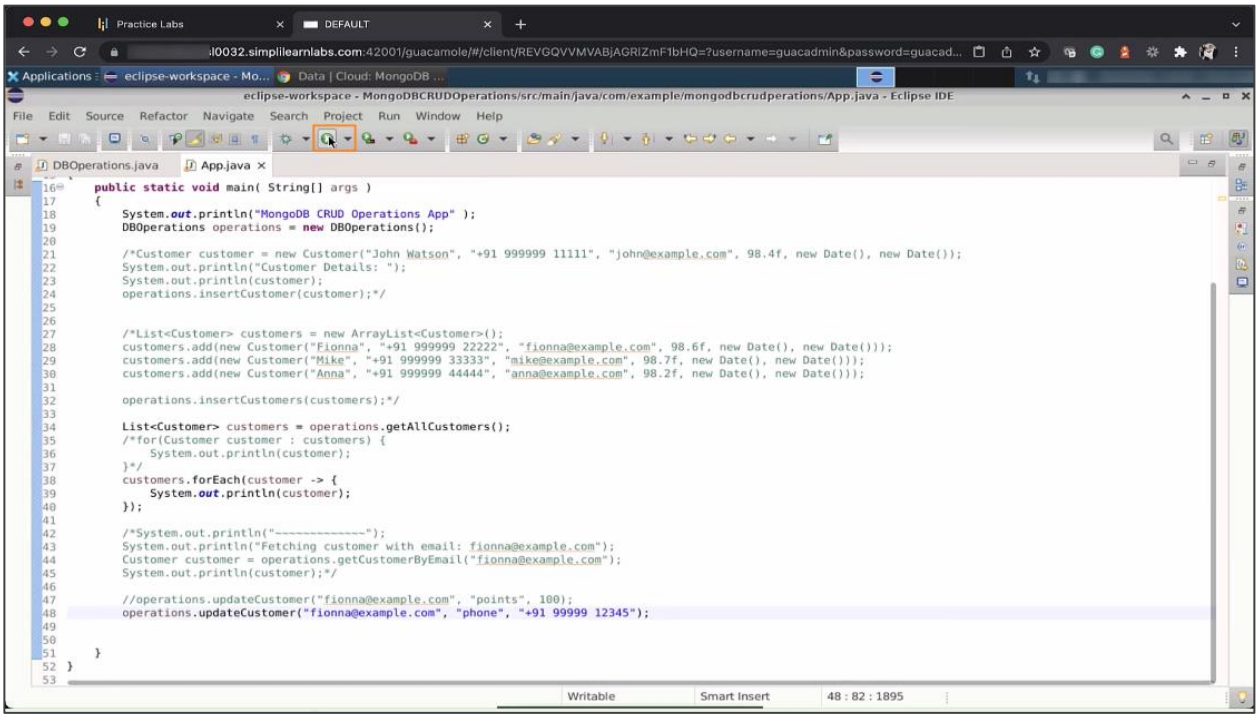
Step 3: Execute another operation for an update

3.1 Write another operation for **updateCustomer** to update the phone number of the customer

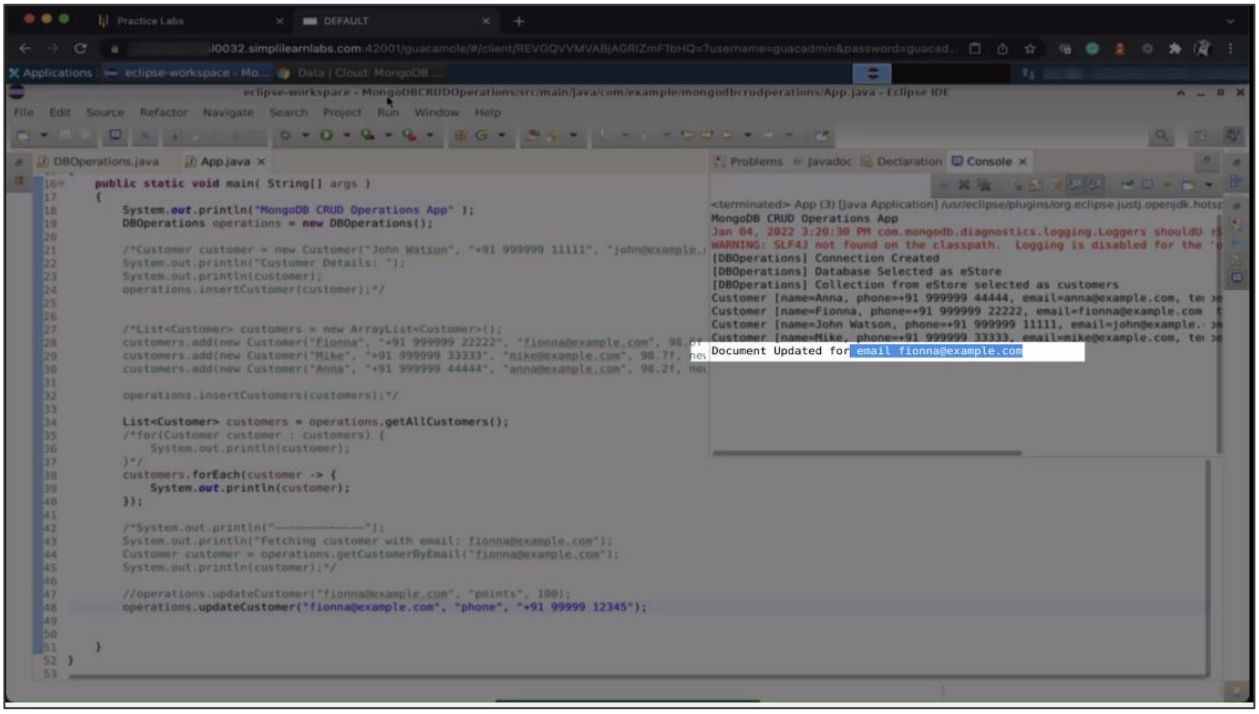


3.2 Go to **DBOperations.java** and change the value to a **String**. Save and run the code again.

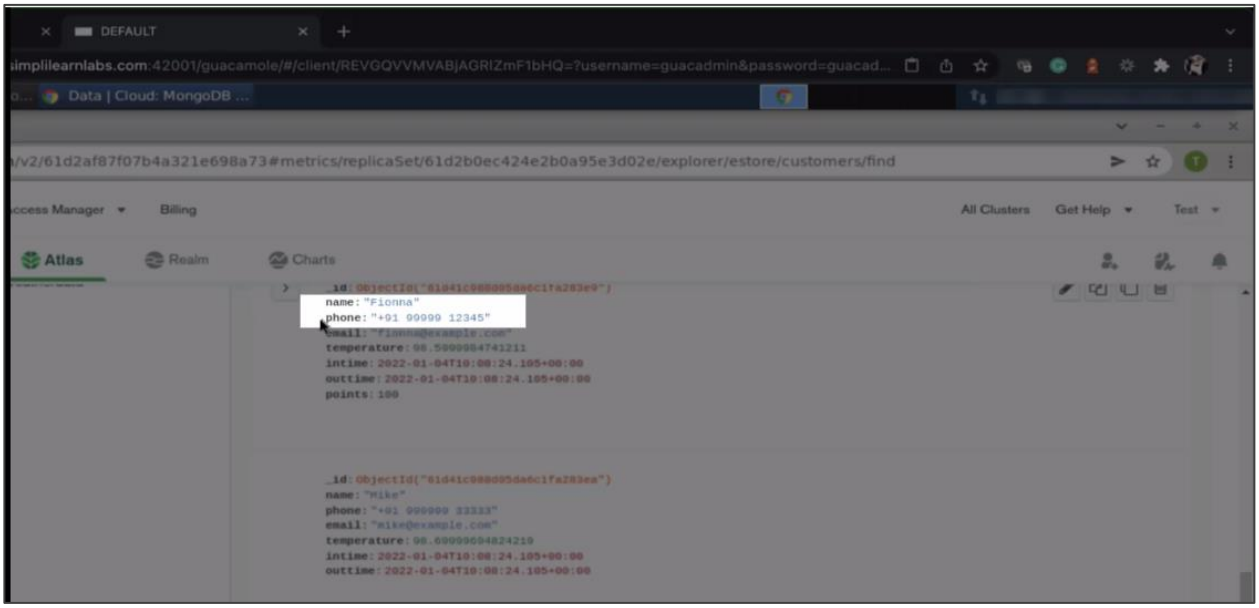
```java
*DBOperations.java × App.java
78    public List<Customer> getAllCustomers(){
79        List<Customer> customers = new ArrayList<Customer>();
80        try {
81
82            List<Document> documents = (List<Document>) collection
83                    .find()
84                    .sort(Sorts.ascending("name"))
85                    .into(new ArrayList<Document>());
86            for(Document doc : documents) {
87                customers.add(convertDocumentToCustomer(doc));
88            }
89
90        } catch (Exception e) {
91            System.out.println("Something went Wrong: "+e);
92        }
93        return customers;
94    }
95
96
97    public Customer getCustomerByEmail(String email) {
98        /*Document filter = new Document("email", email);
99        Document document = (Document) collection.find(filter).first();*/
100       Document document = (Document) collection.find(Filters.eq("email", email)).first();
101       Customer customer = convertDocumentToCustomer(document);
102       return customer;
103   }
104
105   public void updateCustomer(String email, String key, String value) {
106       try {
107           Bson filter = Filters.eq("email", email);
108           Bson updateOperation = Updates.set(key, value);
109           collection.updateOne(filter, updateOperation);
110           System.out.println("Document Updated for email "+email);
111       } catch (Exception e) {
112           System.out.println("Something went Wrong: "+e);
113       }
114
```



You can see the output as **Document Updated for email fionna@example.com**.



3.3 Go back and refresh the collection



The phone number for **Fionna** should now be updated.

3.4 Now, use another method **Updates.push** to see how you can update the data

```
        try {

            List<Document> documents = (List<Document>) collection
                    .find()
                    .sort(Sorts.ascending("name"))
                    .into(new ArrayList<Document>());
            for(Document doc : documents) {
                customers.add(convertDocumentToCustomer(doc));
            }

        } catch (Exception e) {
            System.out.println("Something went Wrong: "+e);
        }
        return customers;
    }


    public Customer getCustomerByEmail(String email) {
        /*Document filter = new Document("email", email);
        Document document = (Document) collection.find(filter).first();*/
        Document document = (Document) collection.find(Filters.eq("email", email)).first();
        Customer customer = convertDocumentToCustomer(document);
        return customer;
    }

    public void updateCustomer(String email, String key, String value) {
        try {
            Bson filter = Filters.eq("email", email);
            //Bson updateOperation = Updates.set(key, value);
            Bson updateOperation = Updates.push(key, value);
            collection.updateOne(filter, updateOperation);
            System.out.println("Document Updated for email "+email);
        } catch (Exception e) {
            System.out.println("Something went Wrong: "+e);
        }

                                                    Writable    Smart Insert    109 : 48 : 335
```

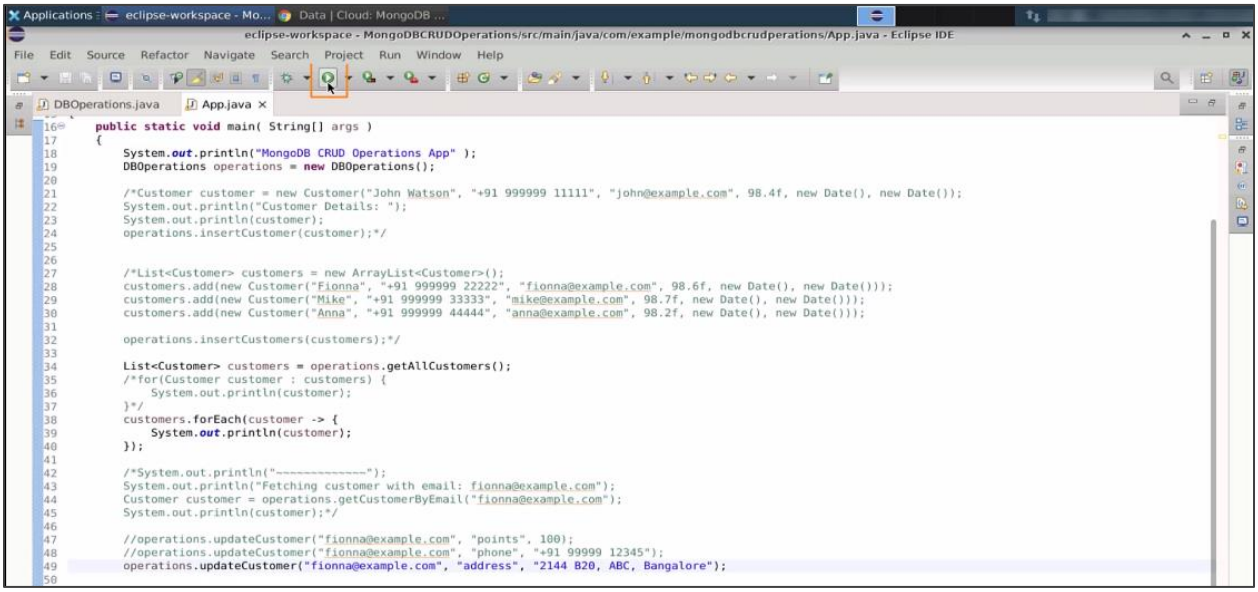
3.5 Go back to **App.java** and comment out the earlier part of the code

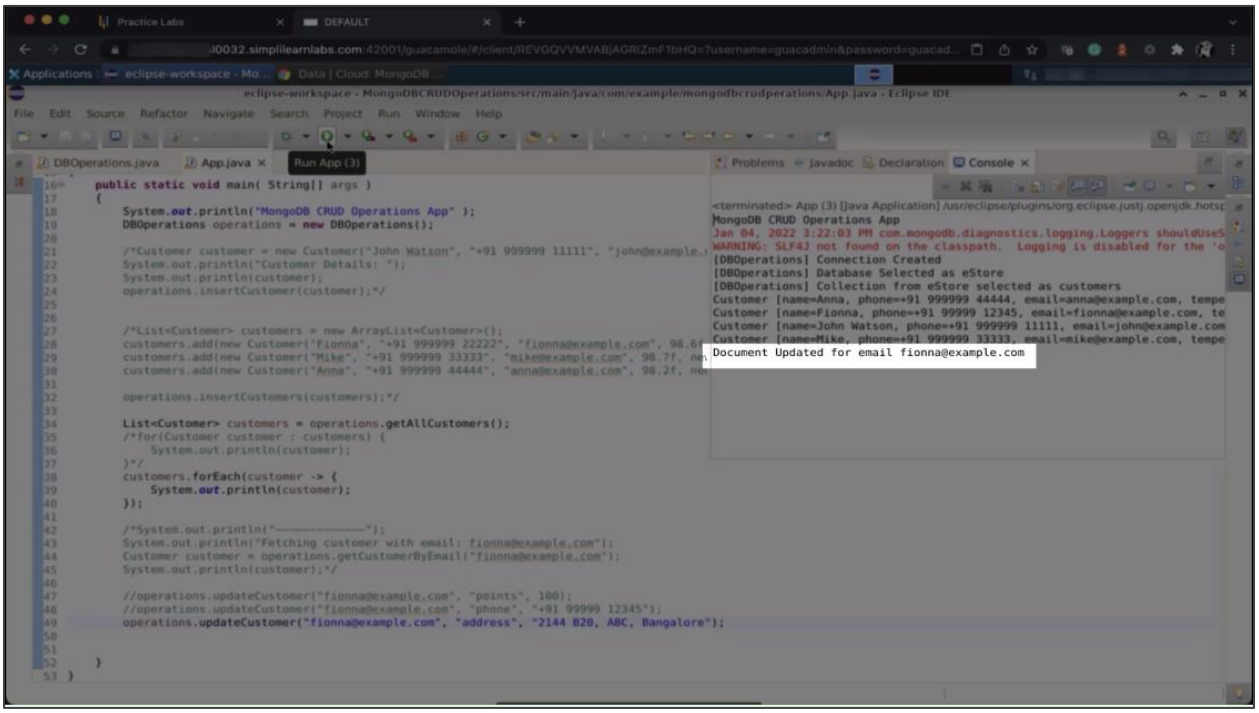
```
    {
        System.out.println("MongoDB CRUD Operations App" );
        DBOperations operations = new DBOperations();

        /*Customer customer = new Customer("John Watson", "+91 999999 11111", "john@example.com", 98.4f, new Date(), new
        System.out.println("Customer Details: ");
        System.out.println(customer);
        operations.insertCustomer(customer);*/


        /*List<Customer> customers = new ArrayList<Customer>();
        customers.add(new Customer("Fionna", "+91 999999 22222", "fionna@example.com", 98.6f, new Date(), new Date()));
        customers.add(new Customer("Mike", "+91 999999 33333", "mike@example.com", 98.7f, new Date(), new Date()));
        customers.add(new Customer("Anna", "+91 999999 44444", "anna@example.com", 98.2f, new Date(), new Date()));

        operations.insertCustomers(customers);*/

        List<Customer> customers = operations.getAllCustomers();
        /*for(Customer customer : customers) {
            System.out.println(customer);
        }*/
        customers.forEach(customer -> {
            System.out.println(customer);
        });

        /*System.out.println("~~~~~~~~~~~~~");
        System.out.println("Fetching customer with email: fionna@example.com");
        Customer customer = operations.getCustomerByEmail("fionna@example.com");
        System.out.println(customer);*/

        //operations.updateCustomer("fionna@example.com", "points", 100);
        //operations.updateCustomer("fionna@example.com", "phone", "+91 99999 12345");


    }
}

                                                    Writable    Smart Insert    48 : 11 : 1824
```

3.6 Use the **updateCustomer** method and enter a new address, for example, **2144 B20, ABC, Bangalore**

```
    {
        System.out.println("MongoDB CRUD Operations App" );
        DBOperations operations = new DBOperations();

        /*Customer customer = new Customer("John Watson", "+91 999999 11111", "john@example.com", 98.4f, new Date(), new
        System.out.println("Customer Details: ");
        System.out.println(customer);
        operations.insertCustomer(customer);*/


        /*List<Customer> customers = new ArrayList<Customer>();
        customers.add(new Customer("Fionna", "+91 999999 22222", "fionna@example.com", 98.6f, new Date(), new Date()));
        customers.add(new Customer("Mike", "+91 999999 33333", "mike@example.com", 98.7f, new Date(), new Date()));
        customers.add(new Customer("Anna", "+91 999999 44444", "anna@example.com", 98.2f, new Date(), new Date()));

        operations.insertCustomers(customers);*/

        List<Customer> customers = operations.getAllCustomers();
        /*for(Customer customer : customers) {
            System.out.println(customer);
        }*/
        customers.forEach(customer -> {
            System.out.println(customer);
        });

        /*System.out.println("~~~~~~~~~~~~~");
        System.out.println("Fetching customer with email: fionna@example.com");
        Customer customer = operations.getCustomerByEmail("fionna@example.com");
        System.out.println(customer);*/

        //operations.updateCustomer("fionna@example.com", "points", 100);
        //operations.updateCustomer("fionna@example.com", "phone", "+91 99999 12345");
        operations.updateCustomer("fionna@example.com", "address", "2144 B20, ABC, Bangalore");


    }
}
```

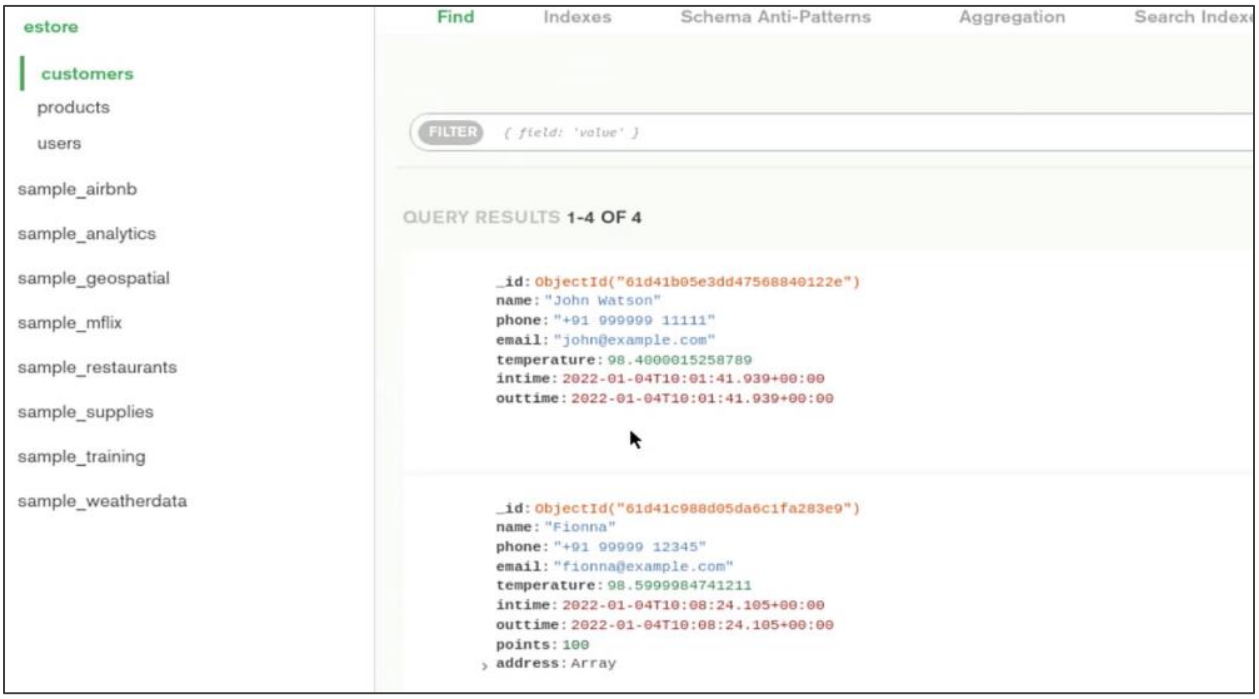
3.7 Save the file and run the code

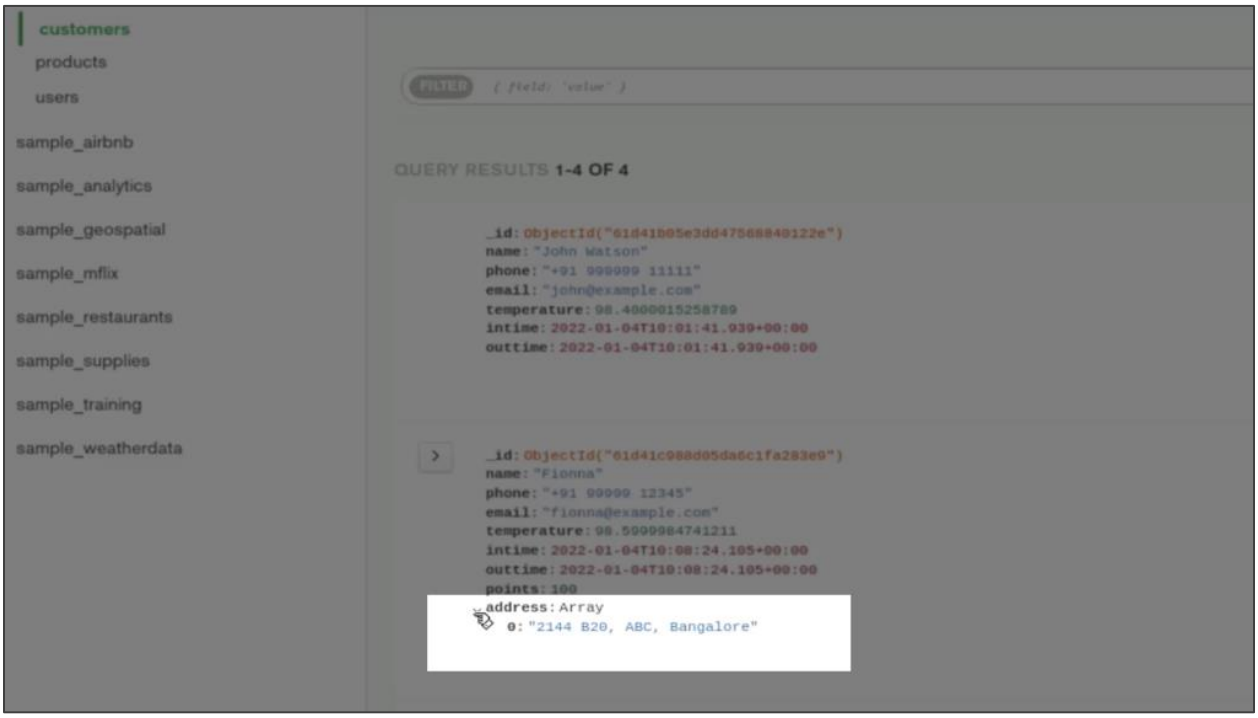




You can see the output as **Document Updated for email fionna@example.com.**

3.8 Go back to the **customers** collection and refresh the document list

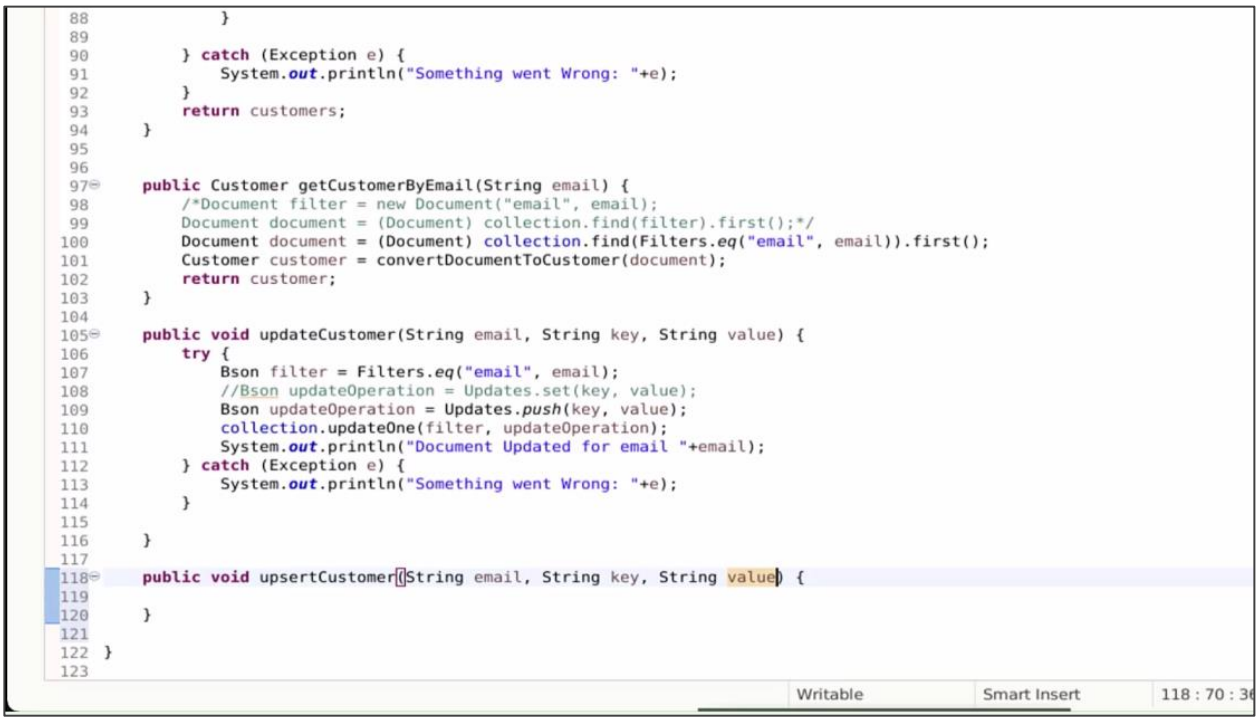




The push operation should have created an array of addresses.

Step 4: Create a new function

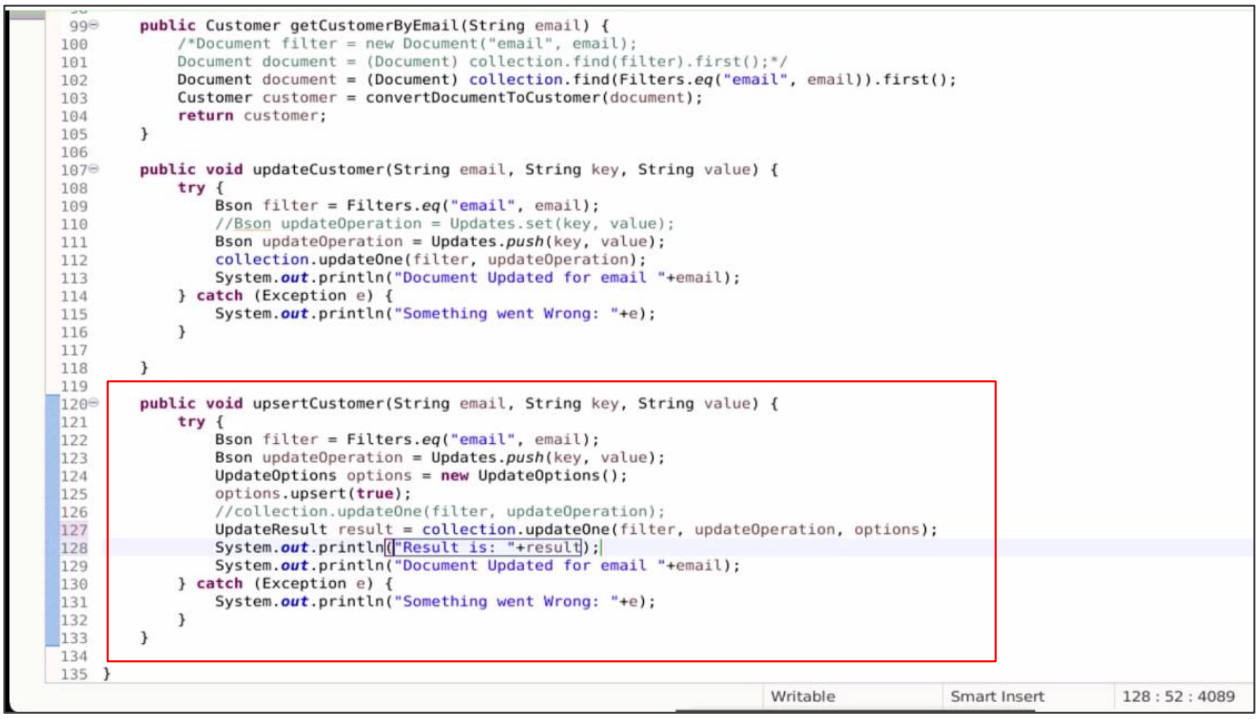
4.1 In the **DBOperations.java** file, create a new function called **upsertCustomer()** to update new records in the document



```
88              }
89
90          } catch (Exception e) {
91              System.out.println("Something went Wrong: "+e);
92          }
93          return customers;
94      }
95
96
97      public Customer getCustomerByEmail(String email) {
98          /*Document filter = new Document("email", email);
99          Document document = (Document) collection.find(filter).first();*/
100         Document document = (Document) collection.find(Filters.eq("email", email)).first();
101         Customer customer = convertDocumentToCustomer(document);
102         return customer;
103     }
104
105     public void updateCustomer(String email, String key, String value) {
106         try {
107             Bson filter = Filters.eq("email", email);
108             //Bson updateOperation = Updates.set(key, value);
109             Bson updateOperation = Updates.push(key, value);
110             collection.updateOne(filter, updateOperation);
111             System.out.println("Document Updated for email "+email);
112         } catch (Exception e) {
113             System.out.println("Something went Wrong: "+e);
114         }
115
116     }
117
118     public void upsertCustomer(String email, String key, String value) {
119
120     }
121
122 }
123
```

Writable | Smart Insert | 118 : 70 : 3

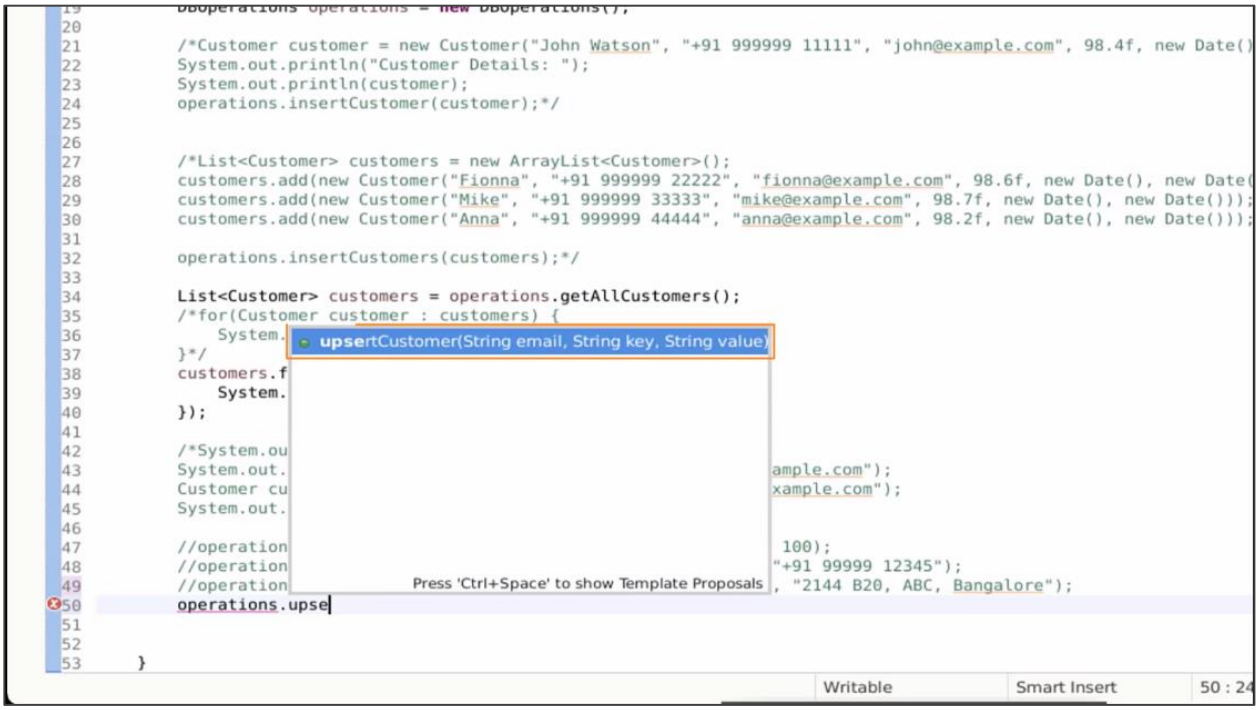
4.2 Create a try catch block, perform a push operation on the key-value pair, and use the **upsert** function to pass the value as **true**



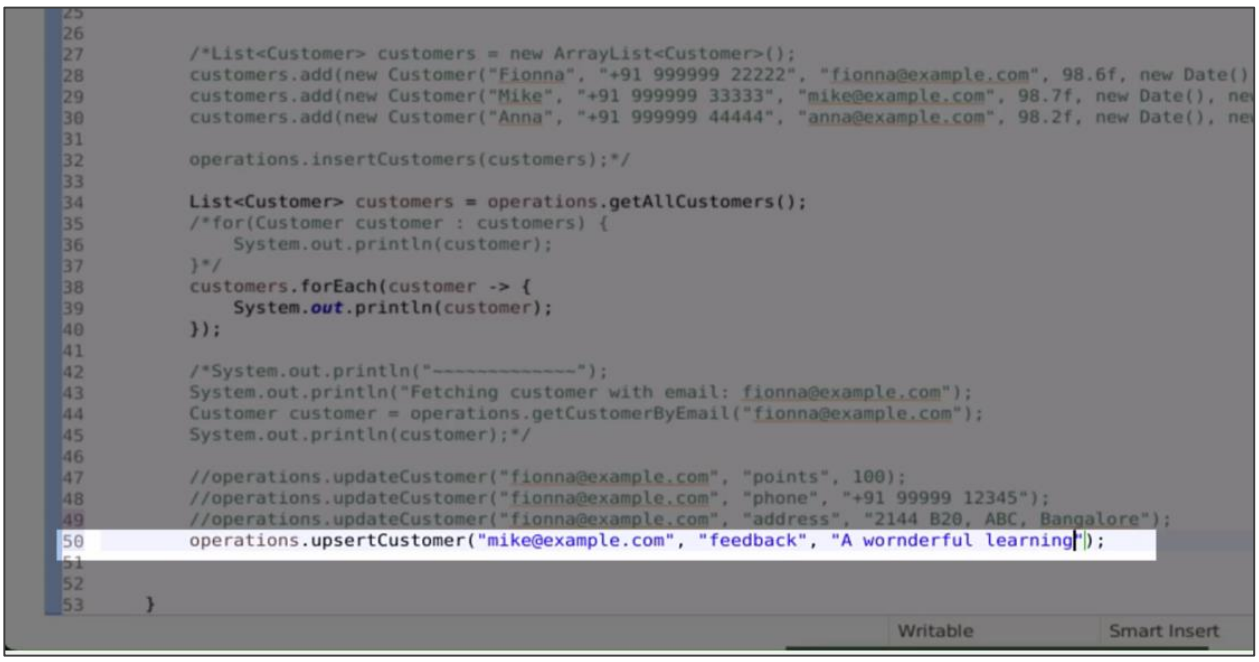
```
    public Customer getCustomerByEmail(String email) {
        /*Document filter = new Document("email", email);
        Document document = (Document) collection.find(filter).first();*/
        Document document = (Document) collection.find(Filters.eq("email", email)).first();
        Customer customer = convertDocumentToCustomer(document);
        return customer;
    }

    public void updateCustomer(String email, String key, String value) {
        try {
            Bson filter = Filters.eq("email", email);
            //Bson updateOperation = Updates.set(key, value);
            Bson updateOperation = Updates.push(key, value);
            collection.updateOne(filter, updateOperation);
            System.out.println("Document Updated for email "+email);
        } catch (Exception e) {
            System.out.println("Something went Wrong: "+e);
        }

    }

    public void upsertCustomer(String email, String key, String value) {
        try {
            Bson filter = Filters.eq("email", email);
            Bson updateOperation = Updates.push(key, value);
            UpdateOptions options = new UpdateOptions();
            options.upsert(true);
            //collection.updateOne(filter, updateOperation);
            UpdateResult result = collection.updateOne(filter, updateOperation, options);
            System.out.println("Result is: "+result);
            System.out.println("Document Updated for email "+email);
        } catch (Exception e) {
            System.out.println("Something went Wrong: "+e);
        }
    }
}
```

Writable | Smart Insert | 128 : 52 : 4089

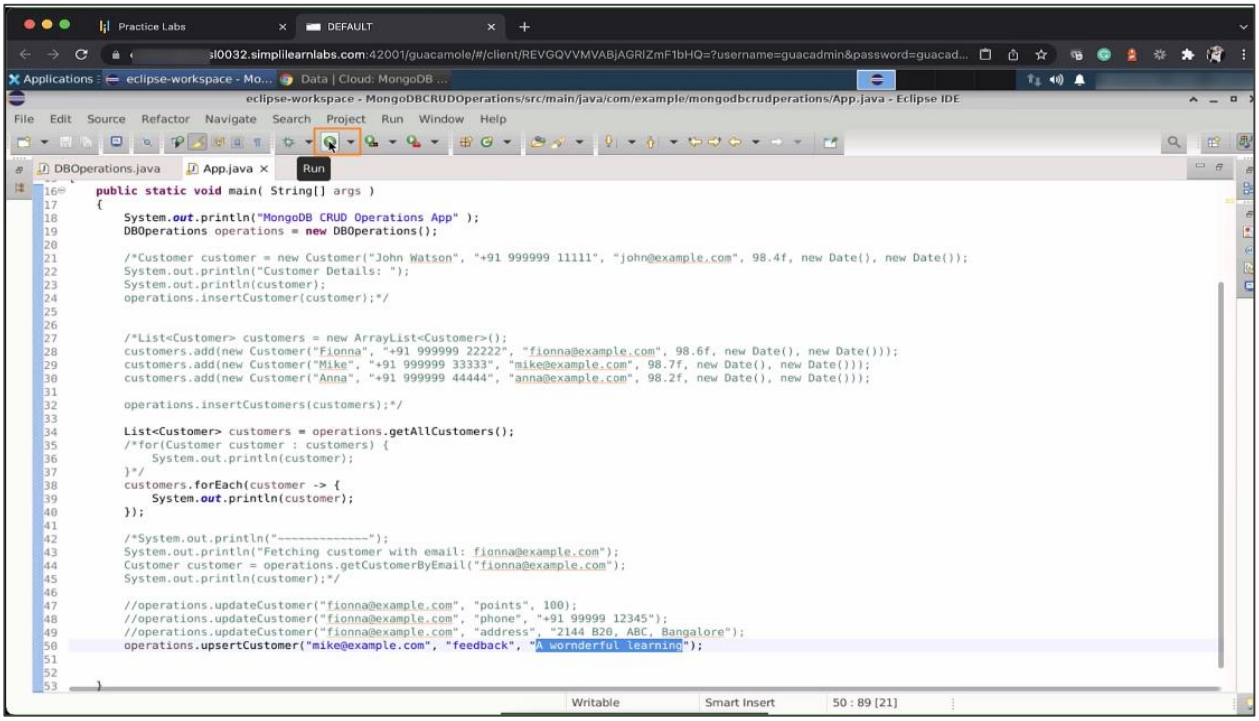
4.3 Go back to **App.java**, use the operation **upsertCustomer(String email, String key, String value)** to update an existing document

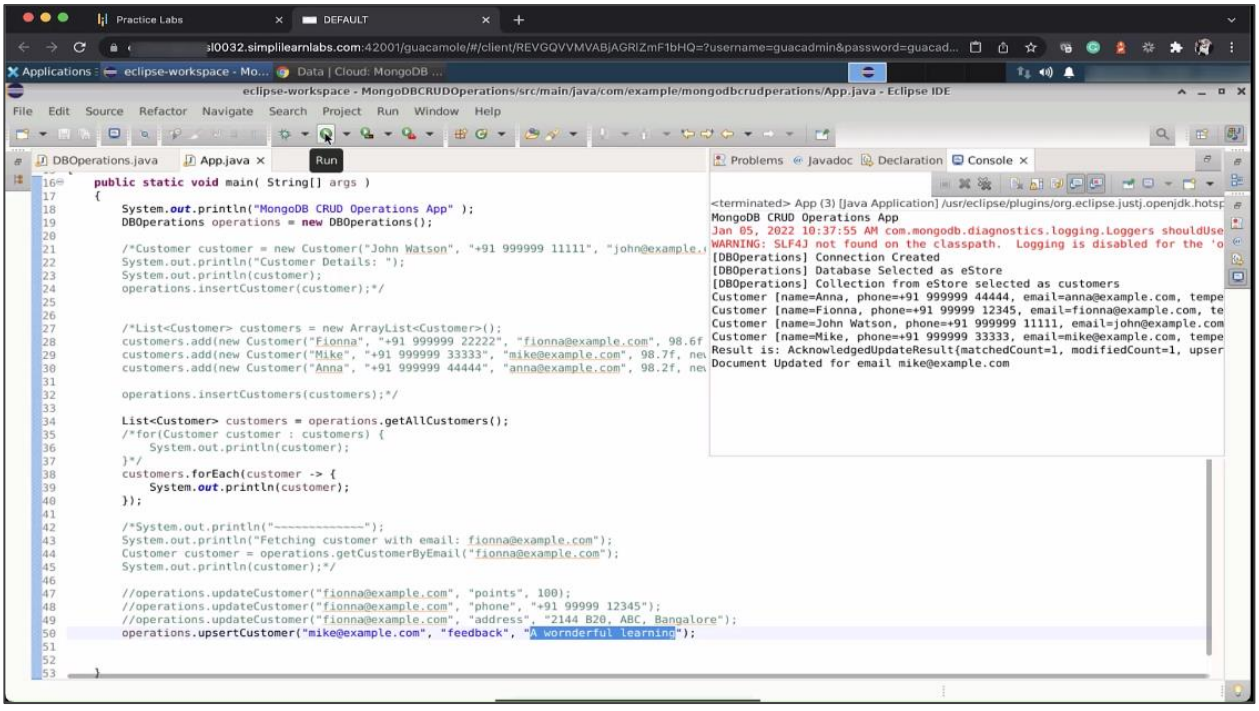


4.4 Add inputs to the key-value pair as **feedback** and **A wonderful learning**

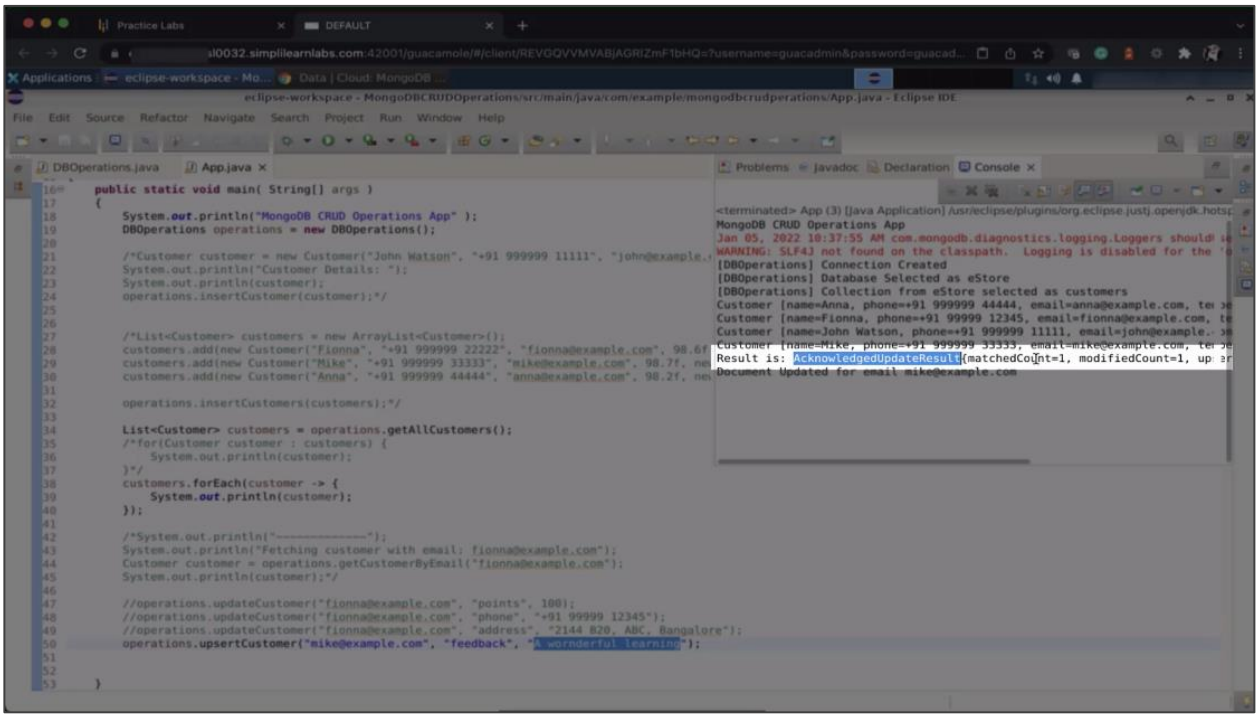


4.5 Save and run the code



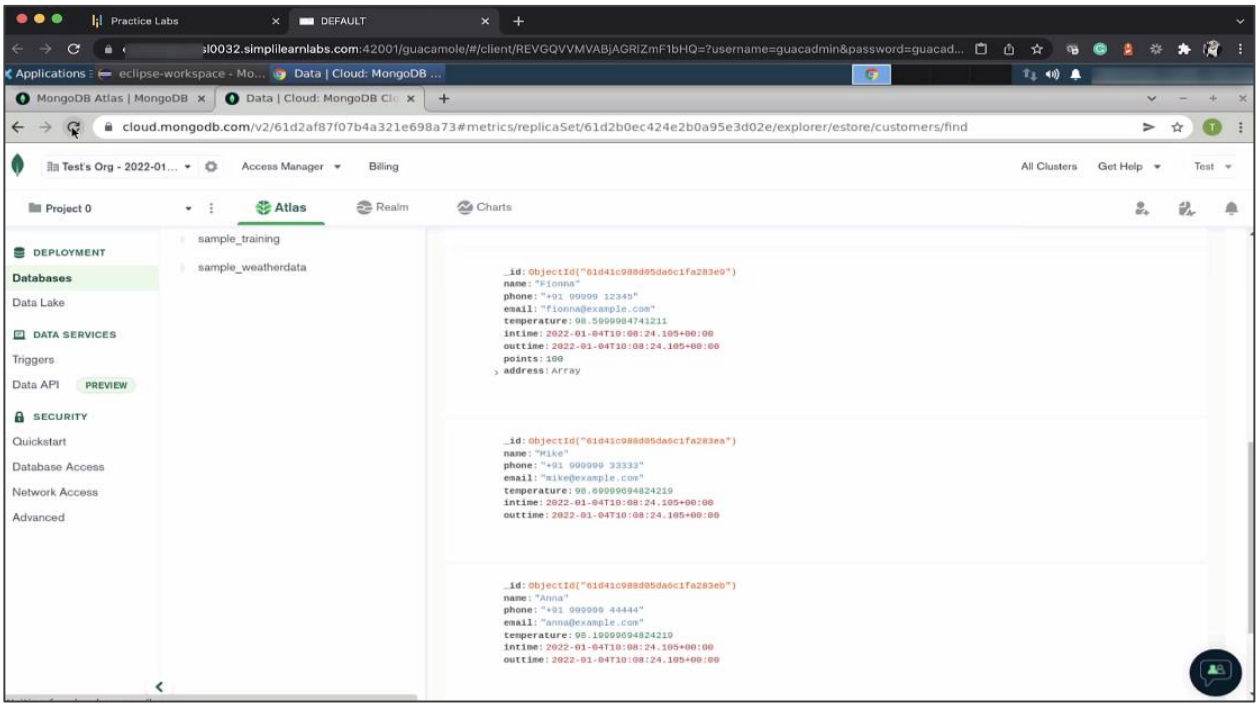


You can see the output as **Document Updated for email mike@example.com.**

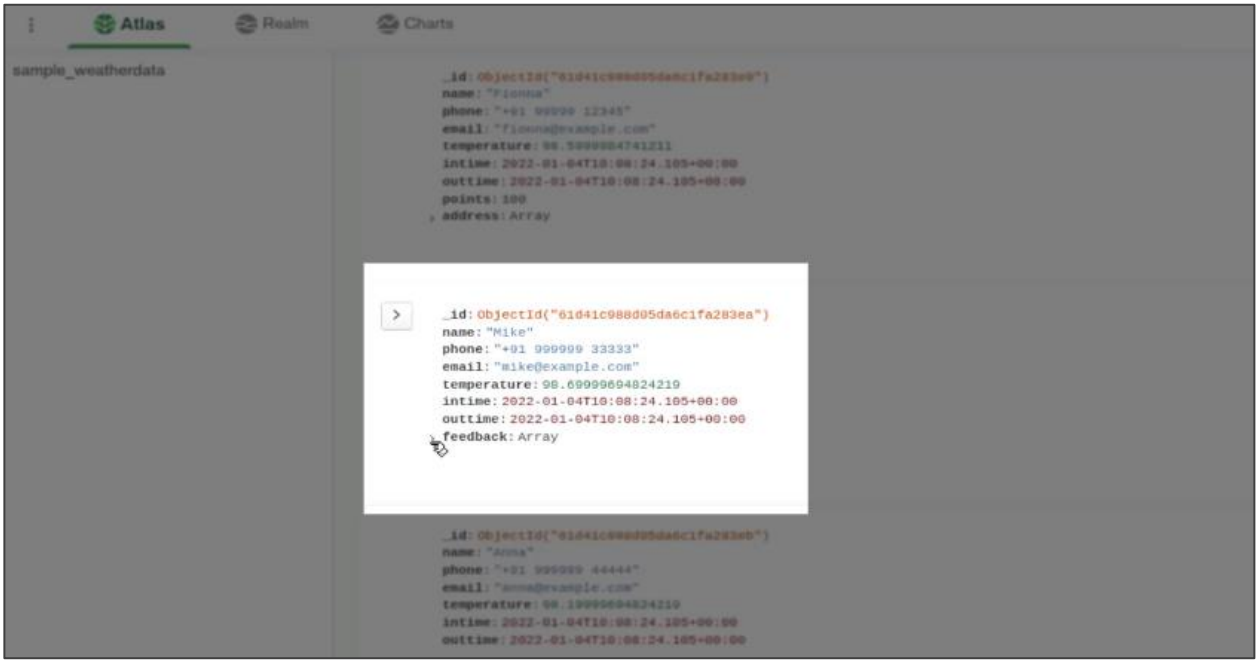


You can see the results state as **AcknowledgeUpdateResult**.

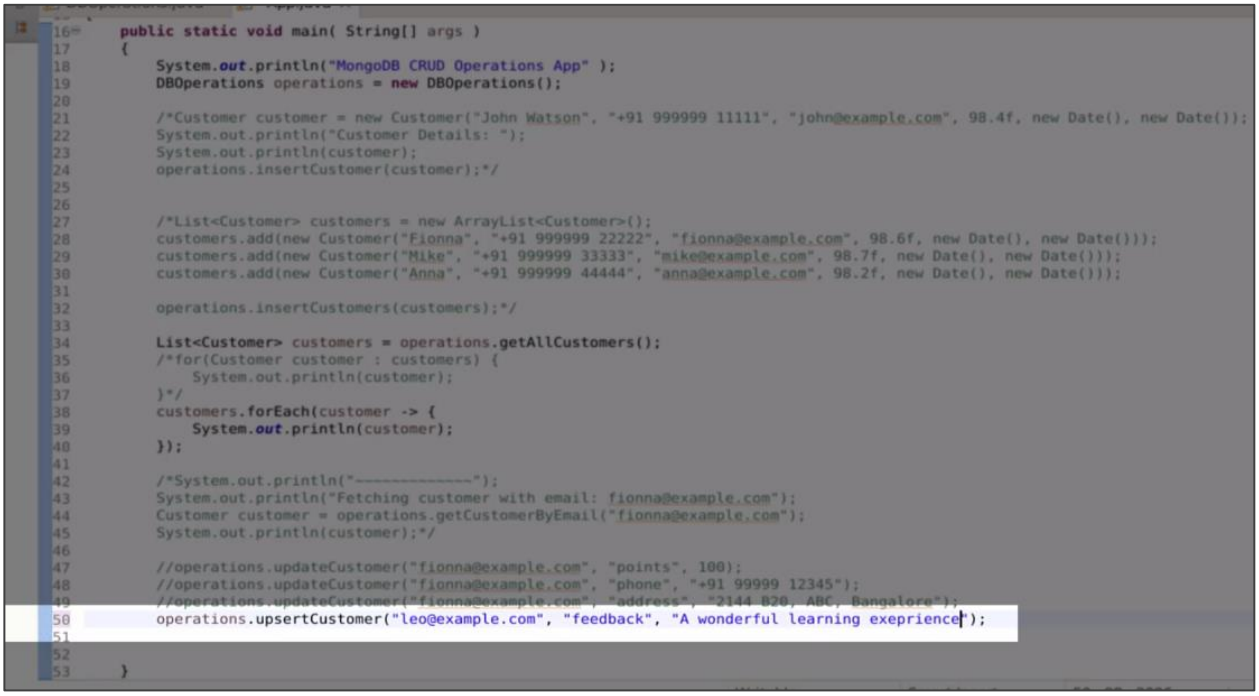
4.6 Go back to the database and refresh the screen



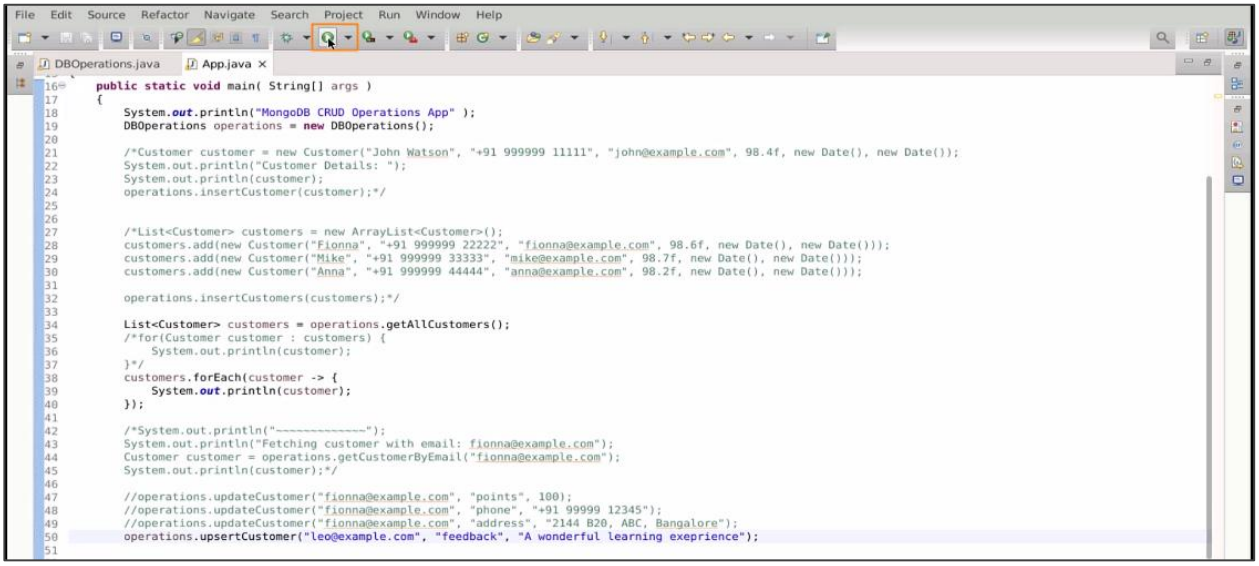
You can see the field is updated.

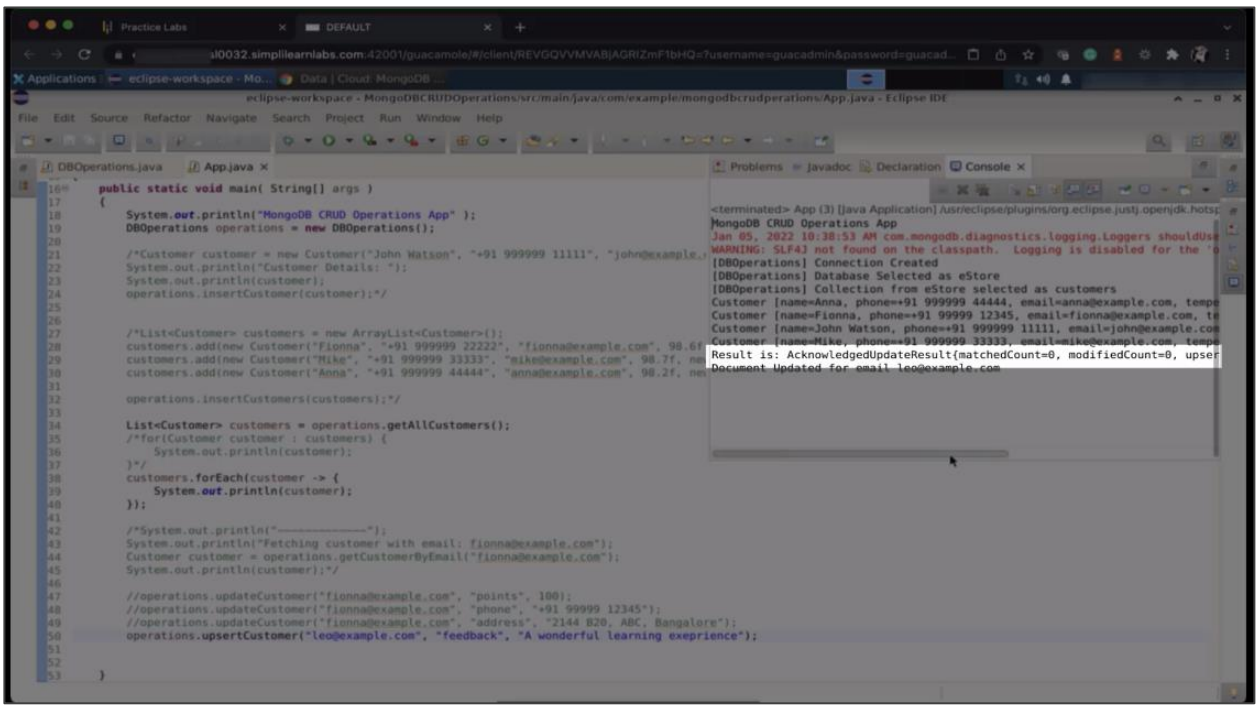


4.7 Go back to **App.java** and change the inputs to **leo@example.com,** which is non-existing data



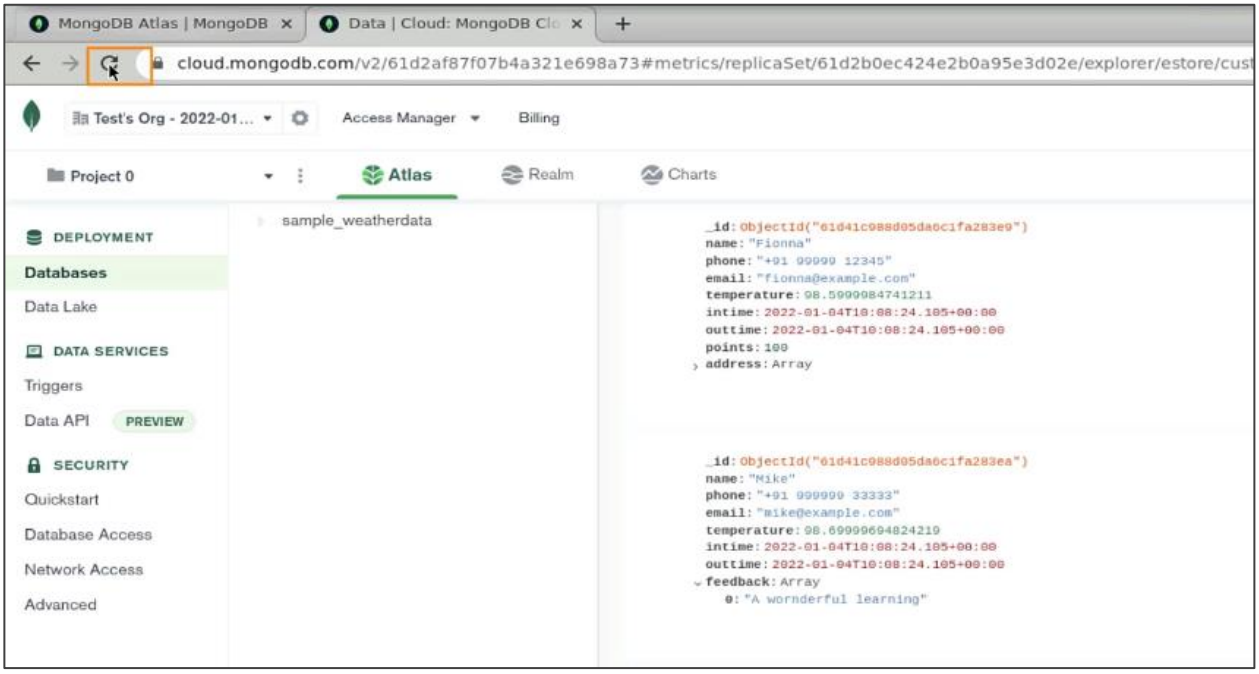
4.8 Save and run the code



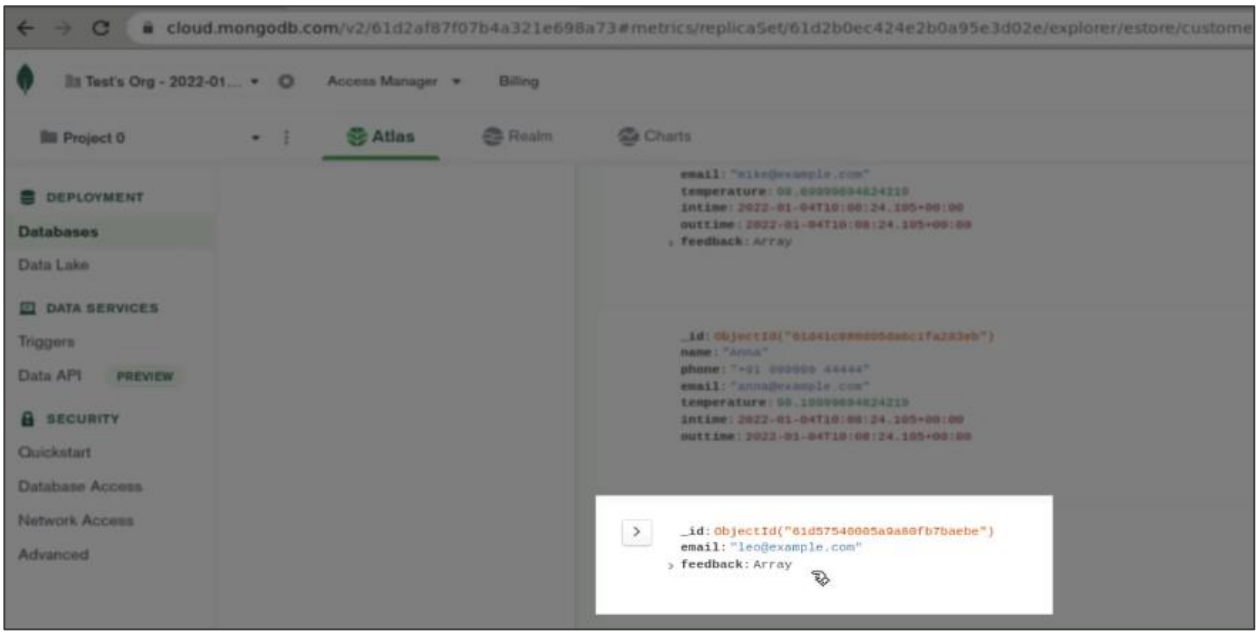


You can see that in the acknowledgement result, no match was found, that is, **modifiedCount=0**.

4.9 Go back to the database and refresh the screen



You can see the **upsertOperation** in action.

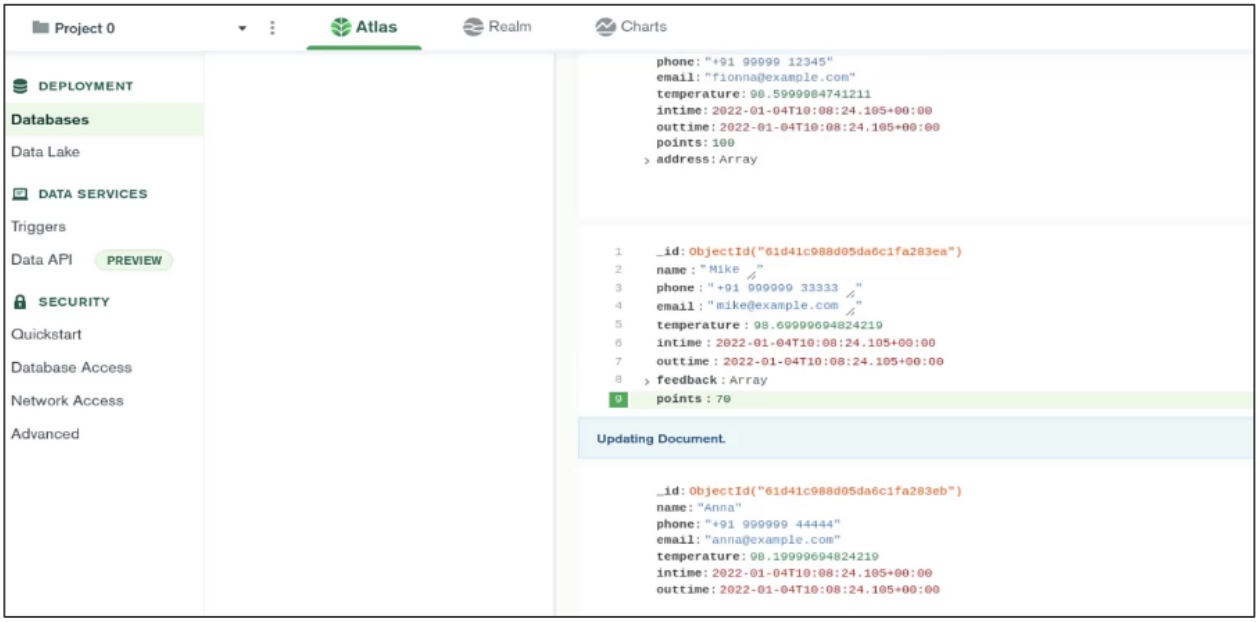


Step 5: Create more functions

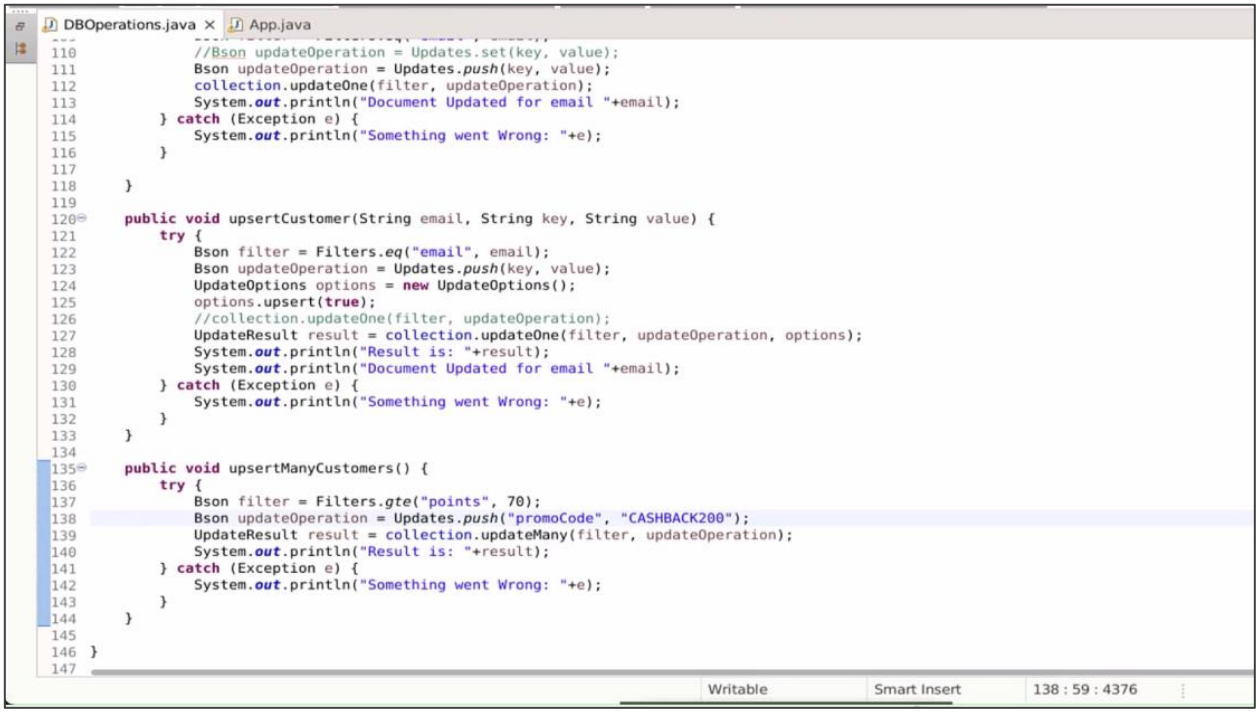
5.1 Create a new function **upsertManyCustomers(String email, String key, String value)** to update multiple documents

```
public void upsertCustomer(String email, String key, String value) {
    try {
        Bson filter = Filters.eq("email", email);
        Bson updateOperation = Updates.push(key, value);
        UpdateOptions options = new UpdateOptions();
        options.upsert(true);
        //collection.updateOne(filter, updateOperation);
        UpdateResult result = collection.updateOne(filter, updateOperation, options);
        System.out.println("Result is: "+result);
        System.out.println("Document Updated for email "+email);
    } catch (Exception e) {
        System.out.println("Something went Wrong: "+e);
    }
}
public void upsertManyCustomers() {
    try {
        Bson filter = Filters.gte("points", 50);
        Bson updateOperation = Updates.push("offer", "CASHBACK200");
        UpdateResult result = collection.updateMany(filter, updateOperation);
        System.out.println("Result is: "+result);
    } catch (Exception e) {
        System.out.println("Something went Wrong: "+e);
    }
}
```

5.2 Add the field points in the database

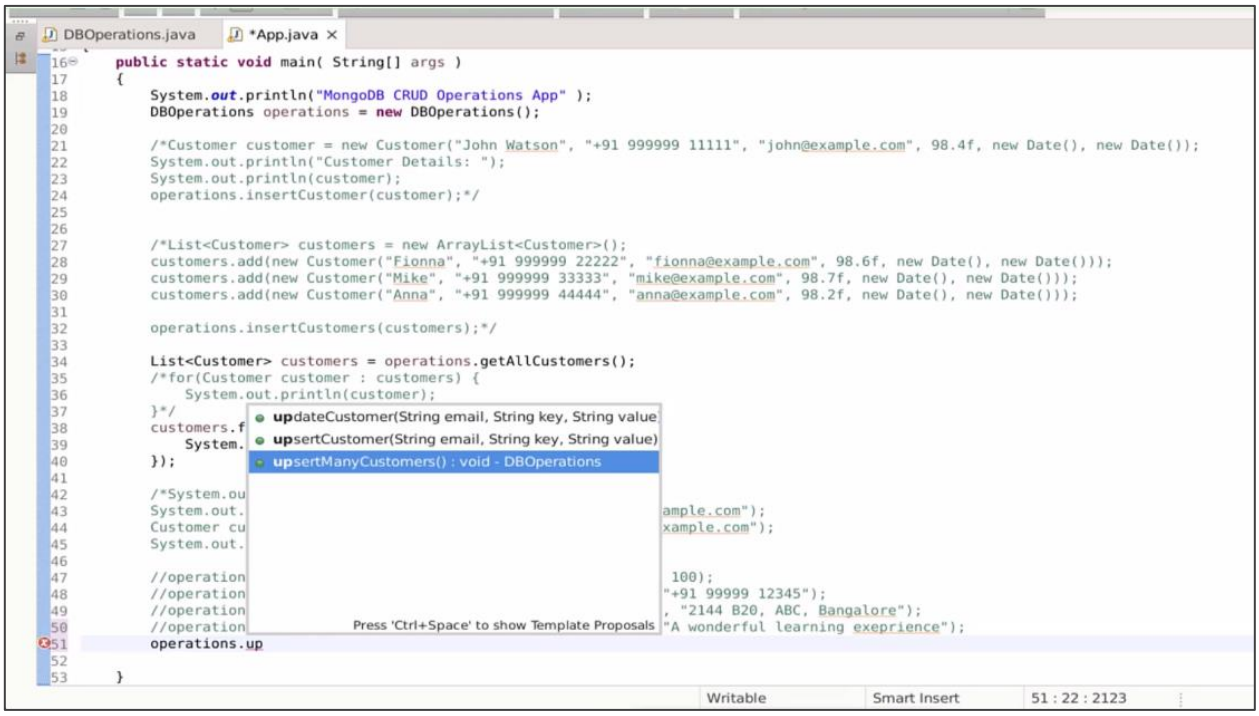


5.3 Update the try catch block with points as **70** and promocode as **CASHBACK200**



```
DBOperations.java × App.java
110             //Bson updateOperation = Updates.set(key, value);
111             Bson updateOperation = Updates.push(key, value);
112             collection.updateOne(filter, updateOperation);
113             System.out.println("Document Updated for email "+email);
114         } catch (Exception e) {
115             System.out.println("Something went Wrong: "+e);
116         }
117
118     }
119
120     public void upsertCustomer(String email, String key, String value) {
121         try {
122             Bson filter = Filters.eq("email", email);
123             Bson updateOperation = Updates.push(key, value);
124             UpdateOptions options = new UpdateOptions();
125             options.upsert(true);
126             //collection.updateOne(filter, updateOperation);
127             UpdateResult result = collection.updateOne(filter, updateOperation, options);
128             System.out.println("Result is: "+result);
129             System.out.println("Document Updated for email "+email);
130         } catch (Exception e) {
131             System.out.println("Something went Wrong: "+e);
132         }
133     }
134
135     public void upsertManyCustomers() {
136         try {
137             Bson filter = Filters.gte("points", 70);
138             Bson updateOperation = Updates.push("promoCode", "CASHBACK200");
139             UpdateResult result = collection.updateMany(filter, updateOperation);
140             System.out.println("Result is: "+result);
141         } catch (Exception e) {
142             System.out.println("Something went Wrong: "+e);
143         }
144     }
145
146 }
147
                                    Writable    Smart Insert    138 : 59 : 4376
```

5.4 Go back to **App.java** and use the **operation.upsertManyCustomer()** method

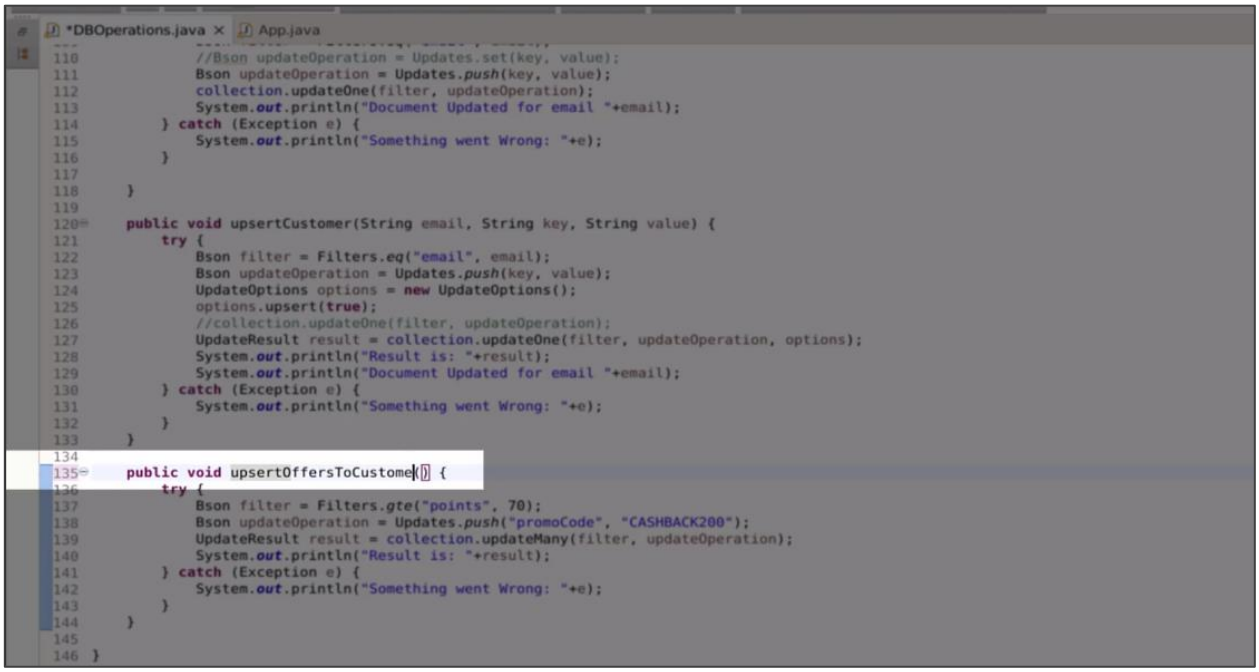


```
DBOperations.java  *App.java ×
16     public static void main( String[] args )
17     {
18         System.out.println("MongoDB CRUD Operations App" );
19         DBOperations operations = new DBOperations();
20
21         /*Customer customer = new Customer("John Watson", "+91 999999 11111", "john@example.com", 98.4f, new Date(), new Date());
22         System.out.println("Customer Details: ");
23         System.out.println(customer);
24         operations.insertCustomer(customer);*/
25
26
27         /*List<Customer> customers = new ArrayList<Customer>();
28         customers.add(new Customer("Fionna", "+91 999999 22222", "fionna@example.com", 98.6f, new Date(), new Date()));
29         customers.add(new Customer("Mike", "+91 999999 33333", "mike@example.com", 98.7f, new Date(), new Date()));
30         customers.add(new Customer("Anna", "+91 999999 44444", "anna@example.com", 98.2f, new Date(), new Date()));
31
32         operations.insertCustomers(customers);*/
33
34         List<Customer> customers = operations.getAllCustomers();
35         /*for(Customer customer : customers) {
36             System.out.println(customer);
37         }*/
38         customers.f
39             System.
40         });
41
42         /*System.ou
43         System.out.
44         Customer cu
45         System.out.
46
47         //operation
48         //operation
49         //operation
50         //operation
51         operations.up
52
53     }
                                    Writable    Smart Insert    51 : 22 : 2123
```

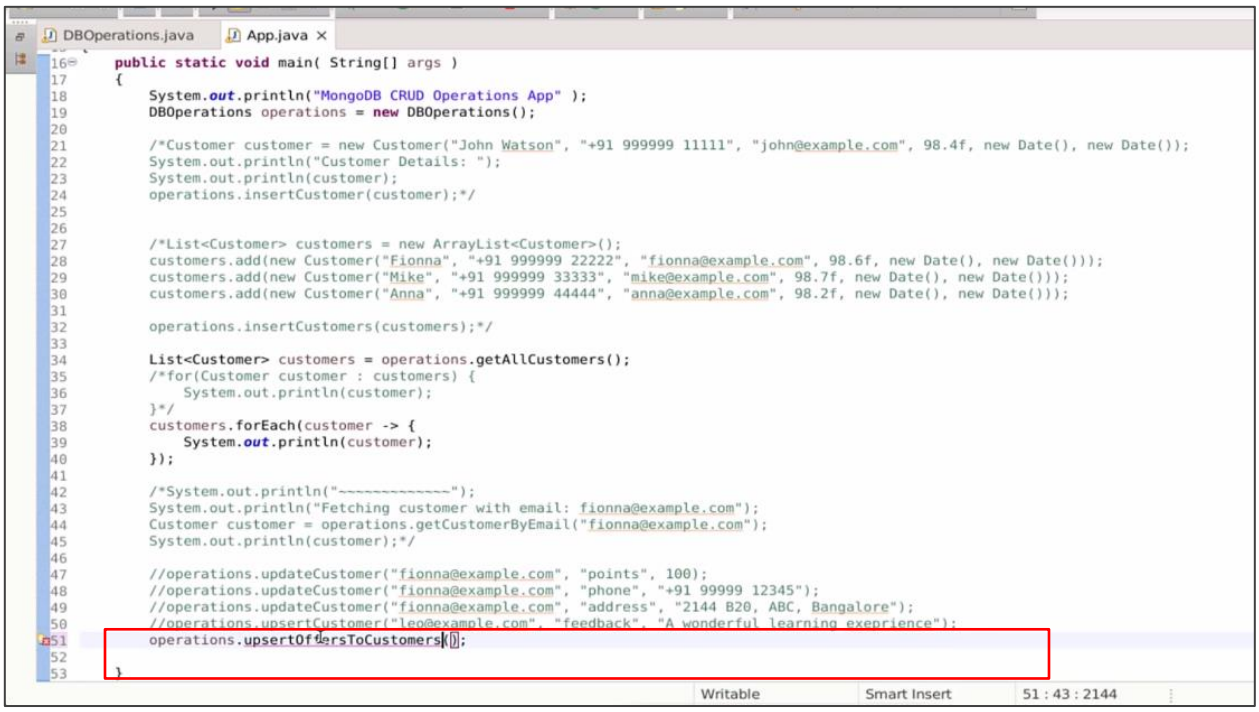
```
updateCustomer(String email, String key, String value
upsertCustomer(String email, String key, String value)
upsertManyCustomers() : void - DBOperations
Press 'Ctrl+Space' to show Template Proposals
```

```
ample.com");
xample.com");
100);
"+91 99999 12345");
, "2144 B20, ABC, Bangalore");
"A wonderful learning exeprience");
```

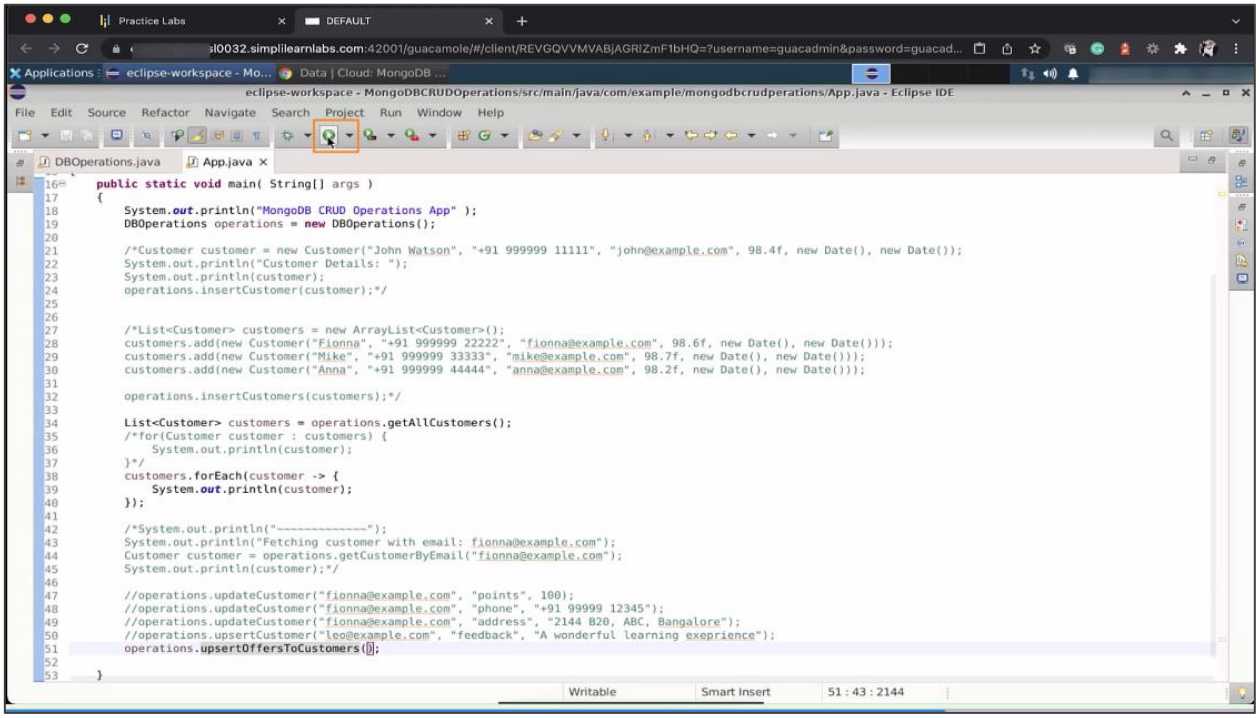
5.5 Go back to the **DBOperations.java** file, change the name of the method to **upsertOffersToCustomers()**, and this method will display the promo codes

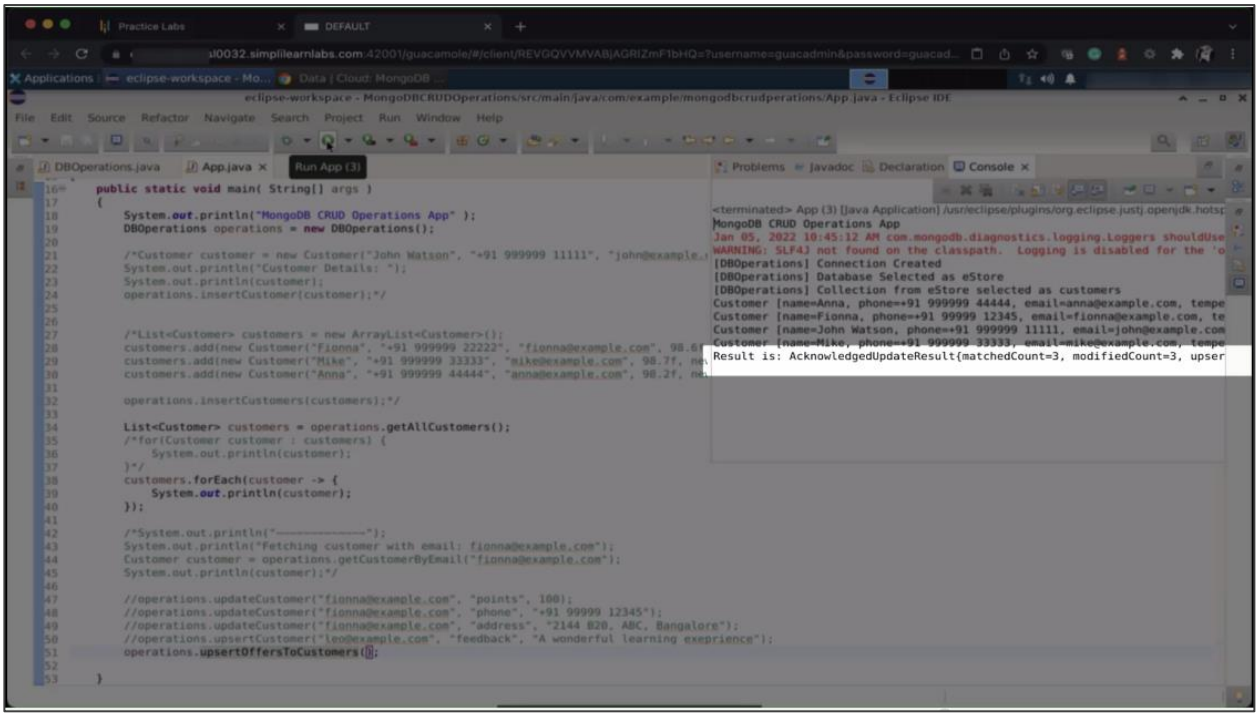


5.6 Go back to **App.java** and change the operation name to **upsertOffersToCustomers()**



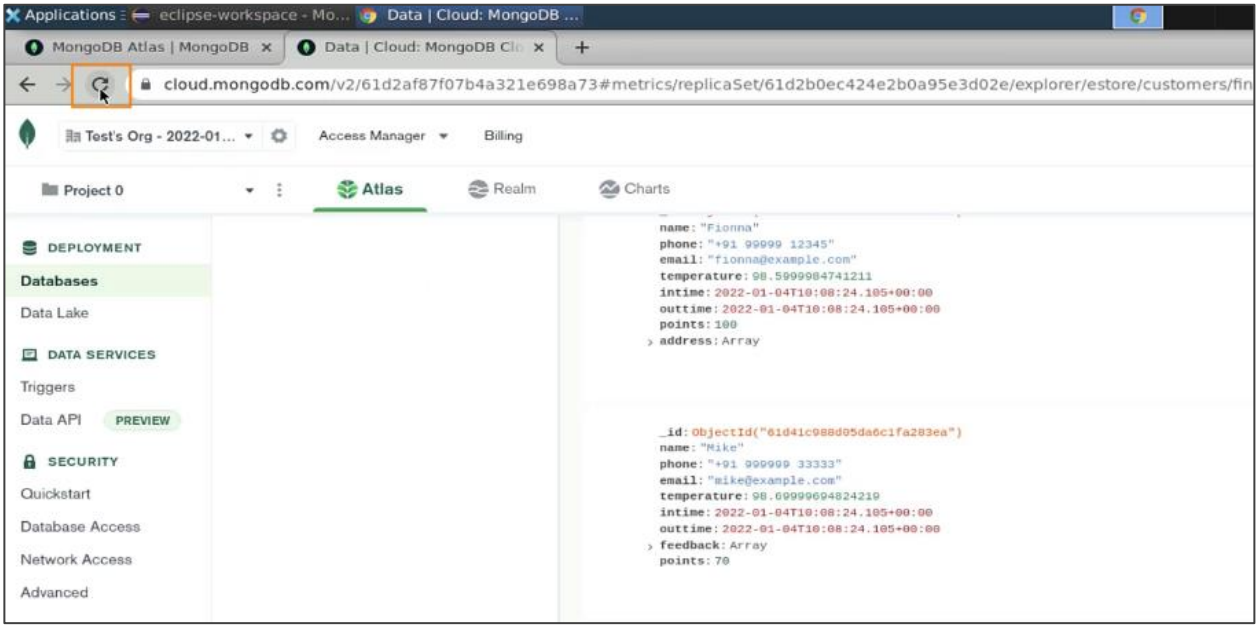
5.7 Save the file and run the code





You can see that three documents were matched, and three documents were modified.

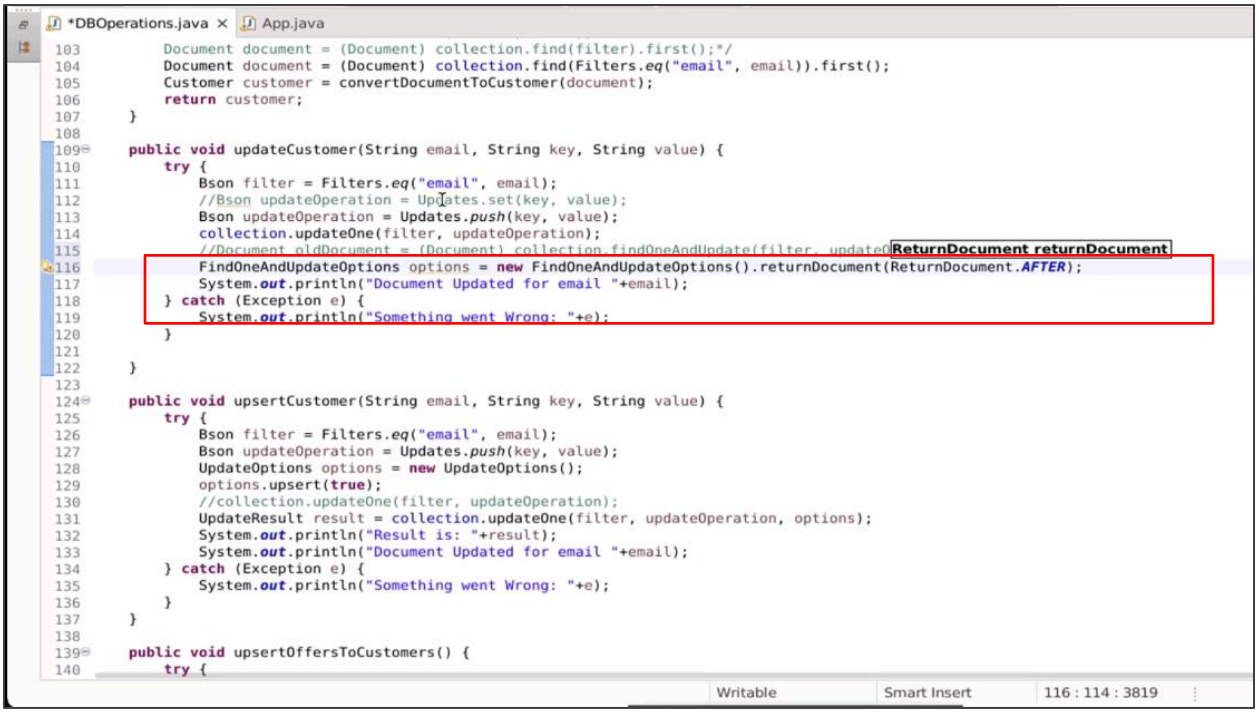
5.8 Go back to the database and refresh the screen



You can see a promo code for points **100** as the condition was greater than or equal to **70**.

Step 6: Write a new method

6.1 Go back to the **DBOperations.java**, create a new collection, and use the **findOneAndUpdate** method, which will take the update operation as input to update the document



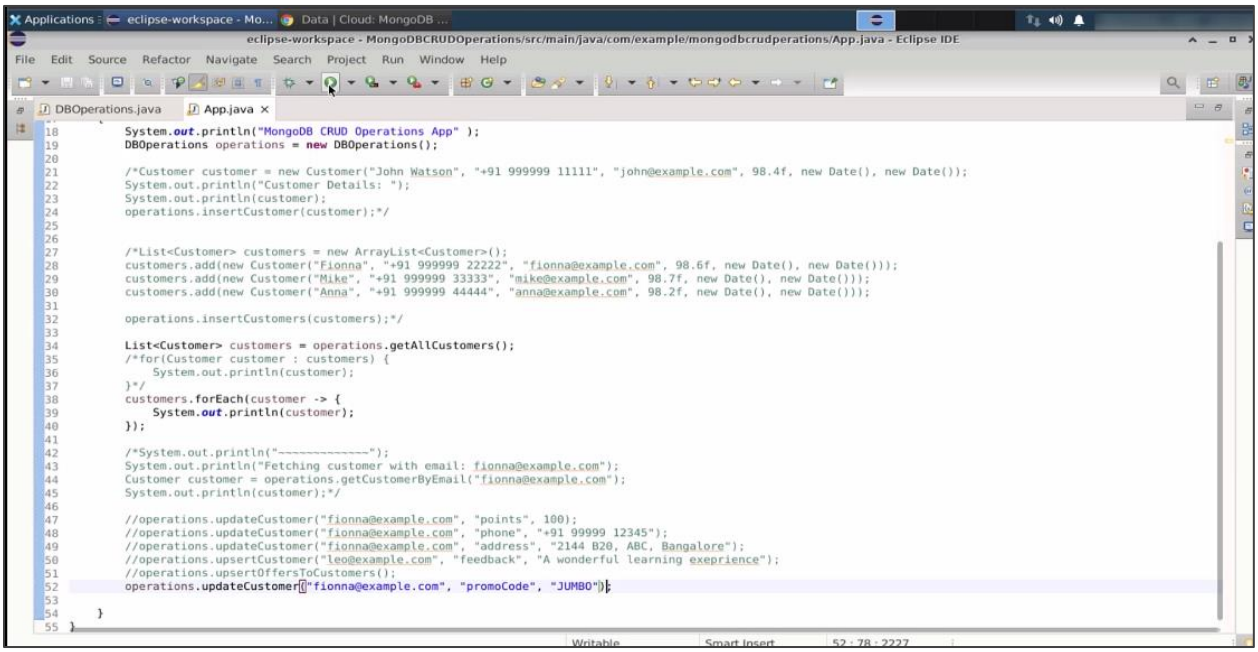
The final code appears as shown below:

```
DBOperations.java × App.java
103         Document document = (Document) collection.find(filter).first();*/
104         Document document = (Document) collection.find(Filters.eq("email", email)).first();
105         Customer customer = convertDocumentToCustomer(document);
106         return customer;
107     }
108
109     public void updateCustomer(String email, String key, String value) {
110         try {
111             Bson filter = Filters.eq("email", email);
112             //Bson updateOperation = Updates.set(key, value);
113             Bson updateOperation = Updates.push(key, value);
114             collection.updateOne(filter, updateOperation);
115             //Document oldDocument = (Document) collection.findOneAndUpdate(filter, updateOperation);
116             FindOneAndUpdateOptions options = new FindOneAndUpdateOptions().returnDocument(ReturnDocument.AFTER);
117             Document updatedDocument = (Document) collection.findOneAndUpdate(filter, updateOperation, options);
118             System.out.println("Document Updated for email "+email);
119             System.out.println("Updated Version of Document is: "+updatedDocument.toJson());
120         } catch (Exception e) {
121             System.out.println("Something went Wrong: "+e);
122         }
123
124     }
125
126     public void upsertCustomer(String email, String key, String value) {
127         try {
128             Bson filter = Filters.eq("email", email);
129             Bson updateOperation = Updates.push(key, value);
130             UpdateOptions options = new UpdateOptions();
131             options.upsert(true);
132             //collection.updateOne(filter, updateOperation);
133             UpdateResult result = collection.updateOne(filter, updateOperation, options);
134             System.out.println("Result is: "+result);
135             System.out.println("Document Updated for email "+email);
136         } catch (Exception e) {
137             System.out.println("Something went Wrong: "+e);
138         }
139     }
140
                                                    Writable    Smart Insert    119 : 93 : 4867
```

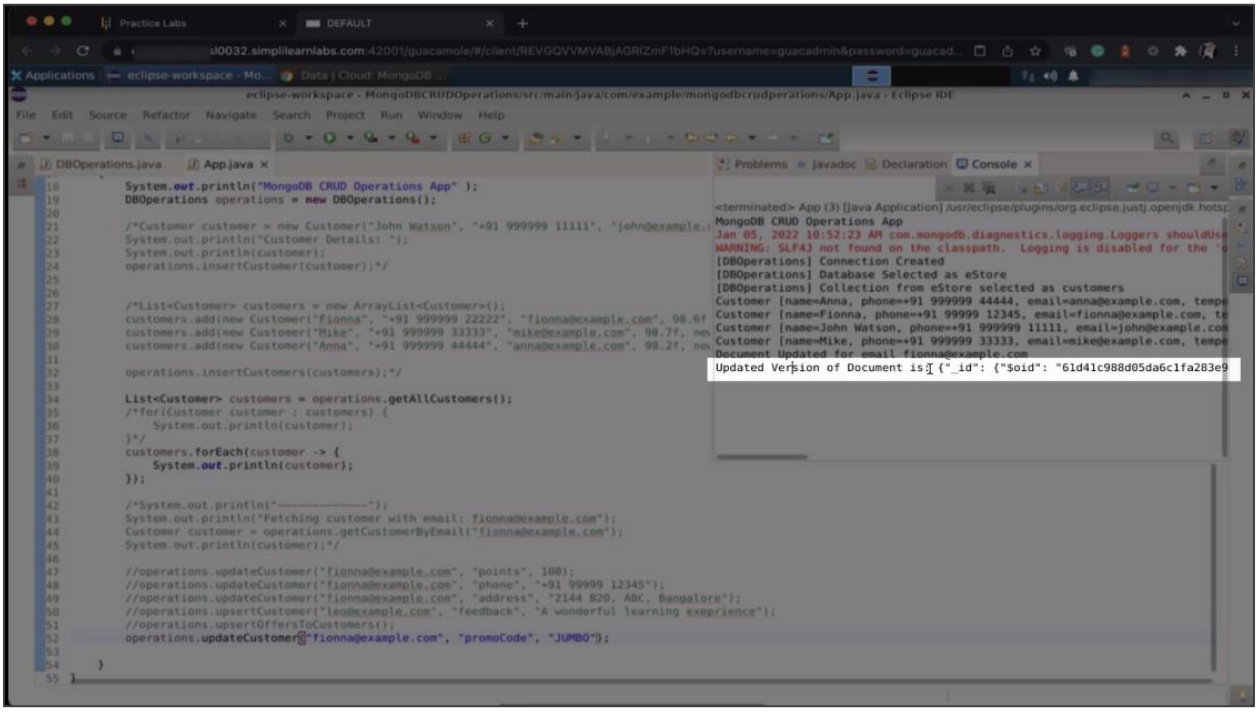
6.2 Go back to the **App.java** file and edit the promoCode as **JUMBO** in the **updateCustomer** method

```
DBOperations.java  App.java ×
18          System.out.println("MongoDB CRUD Operations App" );
19          DBOperations operations = new DBOperations();
20
21          /*Customer customer = new Customer("John Watson", "+91 999999 11111", "john@example.com", 98.4f, new Date(), new Date());
22          System.out.println("Customer Details: ");
23          System.out.println(customer);
24          operations.insertCustomer(customer);*/
25
26
27          /*List<Customer> customers = new ArrayList<Customer>();
28          customers.add(new Customer("Fionna", "+91 999999 22222", "fionna@example.com", 98.6f, new Date(), new Date()));
29          customers.add(new Customer("Mike", "+91 999999 33333", "mike@example.com", 98.7f, new Date(), new Date()));
30          customers.add(new Customer("Anna", "+91 999999 44444", "anna@example.com", 98.2f, new Date(), new Date()));
31
32          operations.insertCustomers(customers);*/
33
34          List<Customer> customers = operations.getAllCustomers();
35          /*for(Customer customer : customers) {
36              System.out.println(customer);
37          }*/
38          customers.forEach(customer -> {
39              System.out.println(customer);
40          });
41
42          /*System.out.println("~~~~~~~~~~~~~");
43          System.out.println("Fetching customer with email: fionna@example.com");
44          Customer customer = operations.getCustomerByEmail("fionna@example.com");
45          System.out.println(customer);*/
46
47          //operations.updateCustomer("fionna@example.com", "points", 100);
48          //operations.updateCustomer("fionna@example.com", "phone", "+91 99999 12345");
49          //operations.updateCustomer("fionna@example.com", "address", "2144 B20, ABC, Bangalore");
50          //operations.upsertCustomer("leo@example.com", "feedback", "A wonderful learning experience");
51          //operations.upsertOffersToCustomers();
52          operations.updateCustomer("fionna@example.com", "promoCode", "JUMBO");
53
54      }
55  }
                                                    Writable    Smart Insert    52 : 78 : 2227
```

6.3 Save the file and run the code



You can see the output as the **Updated Version of Document is** along with other details.



This is all about the update operations. You can perform an update on a single document using the **UpdateOne** function and an **upsert** operation which is a combination of update and insert. The **updateMany** operation can be used with some filters or conditions.

By following these steps, you have successfully worked with the CRUD operations and updated the documents for existing and non-existing customers.