Application code which implements the operations:

```
//create account of user and store in collection 1
    private static boolean createAccount(String username,String password,String
firstName,String lastName){

        Document d = new Document();
        d.append("_id", username);
        d.append("password", password);
        d.append("firstName", firstName);
        d.append("lastName", lastName);
        List<Document> order=new ArrayList<>();
        d.append("orders",order);
        List<Document> prate=new ArrayList<>();
        d.append("productrating",prate);
        try{
            col1.insertOne(d);
            return true;
        }catch(Exception e){
            return false;
        }
    }

    //Add products and their description in collection 2
    private static boolean addProduct(int productid, String name, String description, int price,int
initialStock) {
        Document d = new Document();
        d.append("_id",productid);
        d.append("name", name);
        d.append("description", description);
        d.append("price", price);
        d.append("initialStock", initialStock);
        List<Document> ldoc=new ArrayList<>();
        d.append("review",ldoc);

        try {
            col2.insertOne(d);
            return true;
        } catch (Exception e) {
            return false;
        }
    }

    //Authenticate the user,and Submit the order, add order details in collection 1. If it is not
initialisation state, then
    //decrease the stock level and dont place order if quantity available is less than required.
//In case of initialisation just submit the order without checking available stock. If init is false,
//then check available stock and reduce it by quantity of item required.
//If available stock is less than 0 then just return false.
//Query for authentication justs checks userid(index) so do not check whole data.
//Query for available stock check productid (index) so do not check whole data.
    private static boolean submitOrder(int oid, boolean init, LocalDate date, String username,
String password, Map<Integer,Integer> listOfProductsAndQuantities){
        AggregateIterable<Document> ot = col1.aggregate(Arrays.asList(new Document("$match",
                    new Document("_id", username)
                        .append("password", password)),
                new Document("$count", "count")));
        Iterator iter = ot.iterator();
        if (iter.hasNext()) {
            int productid = 0;
            int quantity = 0;
            List<Document> pq = new ArrayList<>();

            for (Map.Entry<Integer, Integer> entry : listOfProductsAndQuantities.entrySet()) {
                productid = entry.getKey();
                quantity = entry.getValue();

                if (!init) {
                    AggregateIterable<Document> ot1 = col2.aggregate(Arrays.asList(new
Document("$match",
                        new Document("_id", productid))));
                    Iterator iter1 = ot1.iterator();

                    Document doc2 = (Document) iter1.next();
                    int availablestock = doc2.getInteger("initialStock");
                    if (availablestock <= 0) {
                        return false;
                    }

                    Bson search = Filters.eq("_id", productid);
                    int dec = -quantity;
                    Bson update = Updates.inc("initialStock", dec);
                    col2.updateOne(search, update);
                    if(availablestock-quantity<0){
```

```
                    quantity=availablestock;
                }
            }

            Document docpq = new Document();
            docpq.append("product", productid);
            docpq.append("quantity", quantity);
            pq.add(docpq);
        }
        Document order = new Document("orderid", oid)
                .append("date", Date.valueOf(date))
                .append("cart", pq);
        Bson search = Filters.eq("_id", username);
        Bson update = Updates.push("orders", order);
        col1.updateOne(search, update);
    } else {
        return false;
    }
    return true;
}

//post review of product, add details in product collection (2). First authenticate user, then
check if
//a user has posted review for a product, if not then post review.
//Query for authentication justs checks userid(index) so do not check whole data.
//Query for checking if user is present in array of review, it finds productid(index) and fetches all
//user reviews present in it. So, does not scan whole data.

private static boolean postReview(String username,String password, int productID, float
rating, String reviewText){

    AggregateIterable<Document> ot = col1.aggregate(Arrays.asList(new Document("$match",
                new Document("_id", username)
                        .append("password", password)),
            new Document("$count", "count")));
    Iterator iter = ot.iterator();
    if (iter.hasNext()) {
        AggregateIterable<Document> ot1 = col2.aggregate(Arrays.asList(new
Document("$match",
                    new Document("_id", productID))));
        Iterator iter1 = ot1.iterator();

        Document doc2 = (Document)iter1.next();
```

```
List<Document> ll = doc2.get("review", List.class);
if (ll.isEmpty()) {
    //update
    java.util.Date date = new java.util.Date();
    java.sql.Date sqlDate = new java.sql.Date(date.getTime());

    Document newreview = new Document("user", username)
            .append("text", reviewText)
            .append("rating", rating)
            .append("Date", sqlDate);

    Bson search = Filters.eq("_id", productID);
    Bson update = Updates.push("review", newreview);
    col2.updateOne(search, update);

    Document pr = new Document("product", productID)
            .append("rating", rating);
    Bson search1 = Filters.eq("_id", username);
    Bson update1 = Updates.push("productrating", pr);
    col1.updateOne(search1, update1);


} else {
    for (Document doc : ll) {
        if (doc.get("user").equals(username)) {
            return false;
        }
    }
    //update
    java.util.Date date = new java.util.Date();
    java.sql.Date sqlDate = new java.sql.Date(date.getTime());

    Document newreview = new Document("user", username)
            .append("text", reviewText)
            .append("rating", rating)
            .append("Date", sqlDate);

    Bson search = Filters.eq("_id", productID);
    Bson update = Updates.push("review", newreview);
    col2.updateOne(search, update);

    Document pr = new Document("product", productID)
            .append("rating", rating);
            Bson search1 = Filters.eq("_id", username);
            Bson update1 = Updates.push("productrating", pr);
            col1.updateOne(search1, update1);

        }
    } else {
        return false;
    }
    return true;
}


//Adds new inventory associated with the product, adding to the current stock level.
private static void updateStockLevel(int productID, int itemCountToAdd){

    Bson search = Filters.eq("_id", productID);
    Bson update = Updates.inc("initialStock",itemCountToAdd);
    col2.updateOne(search, update);

}

//return the details about a product from collection 2
private static List<List<String>> getProductAndReviews(int productID){
    List<List<String>> ll=new ArrayList<>();
    List<String> l1=new ArrayList<>();
    AggregateIterable<Document> ot = col2.aggregate(Arrays.asList(new Document("$match",
                    new Document("_id", productID))));

    Iterator iter = ot.iterator();
    Document doc = (Document) iter.next();
    String name = doc.getString("name");
    String desc = doc.getString("description");
    int price = doc.getInteger("price");
    l1.add(name);
    l1.add(desc);
    l1.add(String.valueOf(price));
    ll.add(l1);

    List<Document> reviews = (List<Document>) doc.get("review");
    for(Document r:reviews){
        List<String> l2=new ArrayList<>();
        String u =r.getString("user");
        String t =r.getString("text");
        float rate = r.getDouble("rating").floatValue();
```

```
            l2.add(u);
            l2.add(t);
            l2.add(String.valueOf(rate));
            ll.add(l2);
        }

        return ll;
    }

    //Finds average of rating of all the products a user has rated from collection 1
//Query finds userid and gets average of all ratings of it.
    private static float getAverageUserRating(String username){
        AggregateIterable<Document> ot = col1.aggregate(Arrays.asList(new Document("$match",
                        new Document("_id", username)),
                new Document("$project",
                        new Document("avgrate",
                                new Document("$avg", "$productrating.rating")))));
        Iterator iter = ot.iterator();

        Document doc = (Document) iter.next();
        float avgrating = doc.getDouble("avgrate").floatValue();
        return avgrating;
    }
```