

- ✓ ReadManyFiles Will attempt to read and concatenate files using patterns:

```
### ReadManyFiles Result (Target Dir: `/home/ashwin/Node Basic Projects/E-commerce Project`)
```

```
Successfully read and concatenated content from **1 file(s)**.
```

```
**Processed Files:**
```

```
- `helpers/user-helpers.js`
```

> @helpers/user-helpers.js line 344 in place order, i want to iterate through products, group orders based on sellerId from products (maybe we need to use lookup) and then use insertMany. can you guide me? i will do it myself just teach

◆ Of course! I can guide you through that. Refactoring placeOrder to group orders by sellerId and use insertMany is a good approach for a multi-seller e-commerce site. It makes order management much cleaner.

Here's a step-by-step guide. You'll primarily be working in helpers/user-helpers.js and routes/users.js.

Step 1: Fetch Full Product Details

The products array currently passed to placeOrder only contains the product's item ID and quantity. To group by sellerId, you first need to get the full product details for each item.

In `routes/users.js`, find the /place-order route. Change how you get products:

```
* From: let products = await userHelpers.getCartProductList(req.body.userId)
* To: let products = await userHelpers.getCartProducts(req.body.userId)
```

getCartProducts will give you an array where each element contains the full product document, which I assume includes a sellerId.

Step 2: Refactor placeOrder in helpers/user-helpers.js

Now, let's modify the placeOrder function.

1. Group Products by Seller

Inside placeOrder, the first thing to do is group the products array by sellerId. You can use JavaScript's reduce() method for this. This will create an object where keys are seller IDs and values are arrays of their products.

```
1 // Inside placeOrder, after defining the 'status' variable
2
3 const productsBySeller = products.reduce((acc, cartItem) => {
4   // Assuming the seller's ID is on cartItem.product.sellerId
5   const sellerId = cartItem.product.sellerId.toString();
6   if (!acc[sellerId]) {
7     acc[sellerId] = [];
8   }
9   acc[sellerId].push(cartItem);
10  return acc;
11 }, {});
```

2. Create an Array of Order Objects

Instead of one orderObj, you will create one for each seller. It's good practice to link these "sub-orders" together for payment purposes.

Let's create a paymentGroupId for this.

```
1 const paymentGroupId = new ObjectId(); // This will link all sub-orders for this transaction.
2
3 const ordersToInsert = Object.keys(productsBySeller).map(sellerId => {
4   const sellerProducts = productsBySeller[sellerId];
5
6   // Calculate total amount for this specific seller's order
7   const sellerTotal = sellerProducts.reduce((sum, item) => {
8     return sum + (item.quantity * item.product.productPrice);
9   }, 0);
10
11  // Create the order object for this seller
12  return {
13    paymentGroupId: paymentGroupId, // Link to the payment group
14    sellerId: ObjectId(sellerId), // Store the seller's ID
15    userId: ObjectId(order.userId),
16    deliveryDetails: {
17      mobile: order.mobile,
18      address: order.address,
```

```

19     pincode:order.pincode,
20     trackOrder:{
21         ordered:true,
22         shipped:false,
23         outForDelivery:false,
24         delivered:false,
25         stage_od:true,
26         stage_ship:false,
27         stage_oad:false,
28         stage_del:false,
29     },
30 },
31 paymentMethod: order['payment-method'],
32 products: sellerProducts.map(p => ({ item: p.product._id, quantity: p.quantity })),
33 totalAmount: sellerTotal,
34 status: status, // 'placed' or 'pending'
35 userAction: cancelOrder,
36 productDelivered: productDelivered,
37 date: new Date()
38 };
39 });

```

Note: I've assumed `sellerId` is at `cartItem.product.sellerId` and price is `cartItem.product.productPrice`. Please adjust if your field names are different.

3. Use `insertMany` and Return `paymentGroupId`

Now, use insertMany to save all the new orders at once. You should resolve the promise with the paymentGroupId to track this group of orders.

```

1
db.get().collection(collections.ORDER_COLLECTION).insertMany(ordersToInsert).then((respon
se) => {
2   if (reOrderStatus) {
3     db.get().collection(collections.ORDER_COLLECTION).deleteOne({ _id:
objectId(reOrderId) });
4   } else {
5     db.get().collection(collections.CART_COLLECTION).deleteOne({ user:
objectId(order.userId) });
6   }
7   resolve(paymentGroupId); // Resolve with the group ID
8 });

```

This replaces the existing orderObj and insertOne call.

Step 3: Handle Payment and Status Updates

Because you are now working with a group of orders, a few other places need minor changes.

1. In `routes/users.js`

The orderId you get back from placeOrder is now a paymentGroupId. This ID is fine to use as a receipt for Razorpay, as it uniquely identifies the entire transaction.

```
1 userHelpers.placeOrder(req.body, products, totalPrice, reOrderStatus,  
reOrderId).then((paymentGroupId) => {  
2   if (req.body['payment-method'] === 'COD') {  
3     res.json({ codSuccess: true });  
4   } else {  
5     userHelpers.generateRazorpay(paymentGroupId.toString(), totalPrice).then((response)  
=> {  
6       res.json(response);  
7     });  
8   }  
9});
```

In the /verify-payment route, the order.receipt from the frontend will contain this paymentGroupId.

2. Modify `changePaymentStatus` in `helpers/user-helpers.js`

This function needs to update all orders in a payment group, not just one. Modify it to accept the paymentGroupId and use updateMany.

* From: changePaymentStatus:(orderId)=>{ ... updateOne({_id:objectId(orderId)}) ... }
* To:

```
1 changePaymentStatus:(paymentGroupId)=>{  
2   return new Promise((resolve,reject)=>{  
3     db.get().collection(collections.ORDER_COLLECTION)  
4       .updateMany({paymentGroupId:objectId(paymentGroupId)},  
5       {  
6         $set:{  
7           userAction: true,  
8           status:'placed'  
9         }  
10      }  
11    ).then(()=>{
```

```
12         resolve()
13     })
14   })
15 },
```

This approach will correctly create separate orders for each seller, handle payment for the entire cart as a single transaction, and update all related orders correctly after payment. Let me know if you have questions about any of the steps!

```
> foreach products checkig? [
  {
    "_id": "694144bf1b5afab7beadde26",
    "item": "69400b68b88739fdf6905e0b",
    "quantity": 1,
    "product": {
      "_id": "69400b68b88739fdf6905e0b",
      "productName": "Realme 9 pro",
      "Category": "smart phone",
      "productDescription": "good phone",
      "productPrice": "12000",
      "stockAmount": 44,
      "sellerId": "68a576c756d0e33fcd4cd8ef",
      "extsRender": [
        "jpg",
        "jpg",
        "jpg"
      ]
    }
  },
  {
    "_id": "694144bf1b5afab7beadde26",
    "item": "694115cacc239a266bde47bb",
    "quantity": 1,
    "product": {
      "_id": "694115cacc239a266bde47bb",
      "productName": "IPHONE 15",
      "Category": "Mobile",
      "productDescription": "good one",
      "productPrice": "120000",
      "stockAmount": 36,
      "sellerId": "694114bcc239a266bde47ba",
      "extsRender": [
        "png",
      ]
    }
  }
]
```

```
    "png",
    "png"
]
}
}
]
```

step one all good?

- ◆ Yes, that looks correct! The products array now contains the full product details, including the sellerId within each product object.

You've successfully completed Step 1. Now you can proceed with Step 2, which involves modifying the placeOrder function in