Part 1: Code Review and Debugging

**The given code has following list of problems**

1. No validation for uniqueness of SKU. Not checking if SKU is unique can lead to duplicate entries, which can cause redundancy
2. No validation of other fields. Data entered by user can be empty, which should be checked accordingly.
3. Commit operation of product and inventory happens seperatelty. This can cause data inconsistency.

   Above issues can lead to data redundancy and inconsisty. Proper validation must be performed to handle these problems.

**Solution:**

To solve the above issues, following is the list of solutions

1. Implement a proper validation for SKU. Check if record already exists for same SKU. If yes, display a message informing that.
2. Implement validations for empty fields or invalid format of data for other fields, which will reduce inconsistency. We can implement this using Global Exception Handling, which would handle all validation exceptions defined in every entity.
3. Use Transaction using the @Transactional annotation on the endpoint. @Transactional performs all operations. If one operation fails, all other operations are reversed, ensuring consistency

**E.g**

```
PostMapping('/api/products')

@Transactional

@RestController

public ResponseEntity<String> addProduct(AddProductDto addProductDto){

        @Autowired

        private InventoryRepository inventoryRepository;


          @Autowired

          private ProductRepository productRepository;


        if(productRepository.fetchBySku(addProductProductDto.sku)

                return ResponseEntity("Duplicate SKU id");
        Product product = new Product();
        product.name= addProductDto.name;
        product.sku= addProductDto.sku;
        product.price = addProductDto.price;
        product.warehouse_id= addProductDto.warehouse_id;

        productRepository.save(product);

        inventory inventory= new Inventory();
        inventory.product_id=addProductDto.id;
        inventory.warehouse_id= addProductDto.warehouse_id;
        inventory.initial_quantity=addProductDto.initial_quantity;

        inventoryRepository.save(inventory);

        return ResponseEntity.ok("Product added successfully");
    }
```

**Part 2: Database design**

This is a sample database design for for StockFlow. It showcases all the tables with their data columns.

| **Table** | **Columns (FKs)** |
|---|---|
| Companies | id, name |
| Warehouses | id, name, company_id (FK Companies.id) |
| Products | id, name, sku, price, type |
| Inventory | id, product_id (FK Products.id), warehouse_id (FK Warehouses.id), quantity |
| InventoryHistory | id, inventory_id (FK Inventory.id), change_type, quantity_changed, timestamp |
| Suppliers | id, name, contact_email |
| ProductSuppliers | product_id (FK Products.id), supplier_id (FK Suppliers.id) |
| Bundles | bundle_id (FK Products.id), product_id (FK Products.id) |

**Part 3: API endpoint to return low stock alerts for company**

```
@RestController
@RequestMapping("/api/companies")
public class AlertController {

    @Autowired
    private InventoryRepository inventoryRepo;

    @Autowired
    private SaleRepository saleRepo;

    @GetMapping("/{companyId}/alerts/low-stock")
    public Map<String, Object> getLowStockAlerts(@PathVariable Long companyId) {
        List<Map<String, Object>> alerts = new ArrayList<>();
        LocalDate thirtyDaysAgo = LocalDate.now().minusDays(30);

        inventoryRepo.findByWarehouse_Company_Id(companyId).forEach(inv -> {
            Product p = inv.getProduct();
            Integer recentSales = saleRepo.totalRecentSales(p.getId(), thirtyDaysAgo);
            if (recentSales != null && recentSales > 0 && inv.getQuantity() <= p.getLowStockThreshold()) {
                Map<String, Object> alert = new HashMap<>();
                alert.put("product_id", p.getId());
                alert.put("product_name", p.getName());
                alert.put("sku", p.getSku());
                alert.put("warehouse_id", inv.getWarehouse().getId());
                alert.put("warehouse_name", inv.getWarehouse().getName());
                alert.put("current_stock", inv.getQuantity());
                alert.put("threshold", p.getLowStockThreshold());
                alert.put("days_until_stockout", inv.getQuantity() / Math.max(recentSales,1));
            Supplier s = p.getSupplier();
            Map<String,Object> supplier = new HashMap<>();
            supplier.put("id", s.getId());
            supplier.put("name", s.getName());
            supplier.put("contact_email", s.getContactEmail());
            alert.put("supplier", supplier);

            alerts.add(alert);
          }
        });

        Map<String,Object> response = new HashMap<>();
        response.put("alerts", alerts);
        response.put("total_alerts", alerts.size());
        return response;
    }
}
```