University Of Tripoli

Mini ERP System

Course Name: Advanced Java

Student Info:

Name: Mohamed Masoud Alrmah **- ID: 2231809598**

Name: Ahmed Jamalalddin shlibek**- ID: 2231807964**

About The System

Purpose:

This project involves the development of a Mini-Enterprise Resource Planning (ERP) system, designed to manage core business processes, focusing on inventory and user management. Our ERP system centralizes data from various functions, such as product inventory, user roles, and order processing into a single, unified database. The primary goal is to improve efficiency, ensure data consistency across the organization that will use our system, and provide real-time reporting capabilities. We focused on a multi-layered architecture that securely manages data access and applies necessary business logic.

System Testing:

If you would like to test the System

Username: zenta **Password:**1234

System Architecture :

The Mini ERP System follows the standard 3-Layer Architecture. We chose this architecture to divide our system into three main groups, each with separate purposes and responsibilities, working together as one. This structure also helps us manage data flow more effectively. In addition, we selected this architecture to better plan our system and its functions.

1. **Presentation Layer (UI):**
   Using **Swing**, the system handles user interaction, collects input, and displays results. It manages the graphical interface elements and translates user actions into requests for the Service Layer. The Presentation layer communicates exclusively with the Service Layer and contains no business logic or direct database access code.

2. **Service Layer (business logic):**
   The Service Layer is the core of the ERP system, handling business logic and workflows. It takes requests from the presentation layer, applies rules, and coordinates with the data layer to retrieve or update information. This separation ensures consistent operations and keeps the interface free from complex database code.

3. **Data Access Layer (DAO):**
   The DAO Layer is responsible for managing all interactions with the database. It provides methods to create, read, update, and delete data (CRUD) while ensuring that the business logic layer does not directly access the database. By encapsulating database operations, the DAO layer helps seperate our logic m and allows changes to the underlying database structure without affecting other parts of the system.

UI Layer → Presentation Layer → DAO Layer → Database

Technologies and Tools Used :

- Git Hub
- Git
- Intelliji IDEA
- MySQL
- Avien cloud
- Gemini (but only a little 😄)
- Canva (for the report)

Key Features Implemented :

UUID :
The UUID (Universally Unique Identifier) feature ensures that every item in our system, like a Product or Order, gets a unique digital tag, similar to a serial number that no other item in the world will ever have. This unique tag guarantees that the system never confuses one record for another, making it easy to save, load, and track data (look for a product using its uuid) correctly without worrying about IDs accidentally overlapping or repeating.
We also chose to convert the uuid object from 128 bits to 16 bytes for more efficiency using to main functions **uuidToBytes** and **bytesToUUID**.

Product inventory management :
We chose to have a complete CRUD (Create, Read,Update,Delete) funcitonallity for our inventory items.
This Allowed us create new products, Update existing details (Price, Quantity) and maintain a centralized catalog for all of our available items (Read) and delete any categories and Products if needed to.

Real time stock tracking :
Our system always shows you the exact amount of stock you have right now. If a customer buys something or a new product arrives, the numbers update for everyone instantly. This is the simplest way to make sure you never accidentally sell an item you don't actually own, and it helps you know exactly when it's time to reorder.
Or for example when u want to order multiple pieces of the same item but the stock is not enough our System will inform u of the low Stock and also stop the user from completing the order.

Sample Screen Shots :

UUID :
conversion of UUID

```java
//this will convert a uuid object(128 bits) to 16byte for efficency
//return type : byte[]
//takes in a uuid type
private byte[] uuidToBytes(UUID uuid){  4 usages  m.alrmah
    if(uuid == null){
        return null;
    }
    //allocates 16 bytes for the uuid
    ByteBuffer bb = ByteBuffer.wrap(new byte[16]);

    //this basicaly gets us two 64 bits of the 128 -- 64 for msb and 64 lsb
    bb.putLong(uuid.getMostSignificantBits());
    bb.putLong(uuid.getLeastSignificantBits());

    //we dont have to write return bb.byte[] cuz we already declared it
    return bb.array();
}

//from the 16 byte to original 128
//return type: UUID
// takes in byte[] type
private UUID bytesToUUID(byte[] bytes){  1 usage  m.alrmah
    if (bytes == null || bytes.length < 16){
        return null;
    }
    ByteBuffer bb = ByteBuffer.wrap(bytes);

    long firstLong = bb.getLong();
    long secondLong = bb.getLong();

    return new UUID(firstLong , secondLong);
}
```

Product inventory management :
SQL code from Product DAO Impl class showing how we implemented the CRUD idea

```java
private static final String INSERT_PRODUCT_SQL = "INSERT INTO products (product_id , name ,category_id, price,quantity ) V
private static final String UPDATE_PRODUCT_SQL = "UPDATE products SET name = ?, category_id = ?, price = ? ,quantity =? WH
private static final String DELETE_PRODUCT_SQL = "DELETE FROM products where product_id = ?";  1 usage
private static final String FIND_BY_PRODUCTID_SQL ="SELECT * FROM products WHERE product_id =?";  1 usage
private static final String FIND_BY_PRODUCTNAME_SQL = "SELECT * FROM products WHERE name=?";  1 usage
private static final String FIND_ALL_SQL = "SELECT * FROM products";  1 usage
private static final String FIND_LOW_STOCK_SQL = "SELECT * FROM products WHERE quantity < ?";  1 usage
private static final String UPDATE_STOCK_SQL = "UPDATE products SET quantity = quantity + ? WHERE product_id = ?";  1 usage
```

The save method first connects to our database. It then calls our SQL statement, the INSERT_PRODUCT_SQL command. Next, it uses our getter methods to retrieve necessary information, such as the product's name, ID, category, and so on. Finally, the method executes the compiled code and returns the result to our service layer.

```
@Override  m.alrmah +1
public Product save(Product product) throws SQLException {
    if(product.getProductId() == null){
        product.setProductId(UUID.randomUUID());
    }
    try(Connection conn = DBConnection.getConnection();
        PreparedStatement ps = conn.prepareStatement(INSERT_PRODUCT_SQL)) {

        byte[] productIdBytes = uuidToByte(product.getProductId());
        byte[] categoryIdBytes = uuidToByte(product.getCategoryId());

        ps.setBytes( parameterIndex: 1,productIdBytes);
        ps.setString( parameterIndex: 2,product.getName());
        ps.setBytes( parameterIndex: 3,categoryIdBytes);
        ps.setLong( parameterIndex: 4,product.getPrice());
        ps.setInt( parameterIndex: 5,product.getQuantity());

        ps.executeUpdate();

    }

    return product;
}
```

Product Panel:

This is a sample of our product panel and how it uses different methods to successfully work and how we used different controllers to communicate with the UI layer and Service layer.

```java
public ProductPanel(InventoryController inventoryController, CategoryController categoryController) {  1 usage  ahmed +1
    this.inventoryController = inventoryController;
    this.categoryController = categoryController; // حقن التبعية

    setLayout(new BorderLayout( hgap: 10,  vgap: 10)); // تنسيق إطار المنتج

    // 1. جلب الأصناف أولاً
    //we get the categories first
    loadCategories();

    // 2. تهيئة الجدول
    initializeTable();

    // 3. إضافة الجدول إلى لوحة Scrolling
    //we make the table scrollable if needed
    JScrollPane scrollPane = new JScrollPane(productTable);
    add(scrollPane, BorderLayout.CENTER);

    // 4. إضافة لوحة التحكم بالأزرار
    add(createControlPanel(), BorderLayout.SOUTH);

    // تحميل البيانات الأولية
    loadProducts();
}
```

```java
private void loadCategories() {  4 usages  ahmed +1
    try {
        List<Category> categories = categoryController.getAllCategories(); // استخدام CategoryController
        categoryMap.clear();
        for (Category c : categories) {
            categoryMap.put(c.getCategoryId(), c.getName());
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog( parentComponent: this,  message: "Failed to load Categories " + e.getMessage(),
                title: "Error in Data Base", JOptionPane.ERROR_MESSAGE);
    }
}

private void initializeTable() {  1 usage  ahmed +1
    // أعمدة الجدول
    String[] columnNames = {"ID", "Name", "Category", "Price", "Quantity"};
    tableModel = new DefaultTableModel(columnNames,  rowCount: 0) {  ahmed
        @Override  ahmed
        public boolean isCellEditable(int row, int column) {
            // جعل جميع الخلايا غير قابلة للتعديل مباشرة
            return false;
        }
    };
    productTable = new JTable(tableModel);
}
```

Buttons and Action event:

We made different buttons and each button has its own action event , when clicked on the button it does a certain function. We used something called action listeners and Lambda (e ->) instead of manually typing the action performed function which will make our code longer and more complicated.

```
// الأزرار الإدارية تظهر للمدير فقط
if ("ADMIN".equalsIgnoreCase(SessionUtil.getCurrentUser().getRole())) {

    // زر إضافة صنف جديد (حل المشكلة)
    JButton addCategoryButton = new JButton( text: "Add New Category");
    addCategoryButton.addActionListener( ActionEvent e -> showAddCategoryDialog());
    panel.add(addCategoryButton);

    JButton addButton = new JButton( text: "Add New Item");
    addButton.addActionListener( ActionEvent e -> showAddProductDialog());
    panel.add(addButton);

    JButton updateStockButton = new JButton( text: "Update Inventory");
    updateStockButton.addActionListener( ActionEvent e -> showUpdateStockDialog());
    panel.add(updateStockButton);

    //here we created a button and when clicked on it does the deleteCategoryButton function
    //we use the action listener
    JButton deleteCategoryButton = new JButton( text: "Delete Category");
    deleteCategoryButton.addActionListener( ActionEvent e -> showDeleteCategoryDialog());
    panel.add(deleteCategoryButton);

    JButton deleteButton = new JButton( text: "Delete Item");
    deleteButton.addActionListener( ActionEvent e ->showDeleteDialog());
    panel.add(deleteButton);

}
```

Explanation of how each required topic was implemented :

1. **Exception Handling:** We used try, catch, and Throw blocks throughout the system to prevent crashes. This was important for handling unexpected problems, especially when talking to the database. If the database connection failed, for instance, the catch block would manage the error, and the finally block would ensure we always closed the connection safely so the system didn't fail.
   We also used try catch in our UI layer functions, we used an if statement, anf if the condition was fulfilled we used try to complete the function and catch to produce an exception message to inform us of the issue. EX:-

```
} catch (Exception e) {
    JOptionPane.showMessageDialog(this,
        "Failed to add Category: " + e.getMessage(),
        "Error",
        JOptionPane.ERROR_MESSAGE);
}
```

2. **Collections Framework:** The Java Collections (like ArrayList and HashMap) were used as temporary storage containers inside the application. We used lists (ArrayList) to hold groups of items, like all the products in inventory, after fetching them from the database.

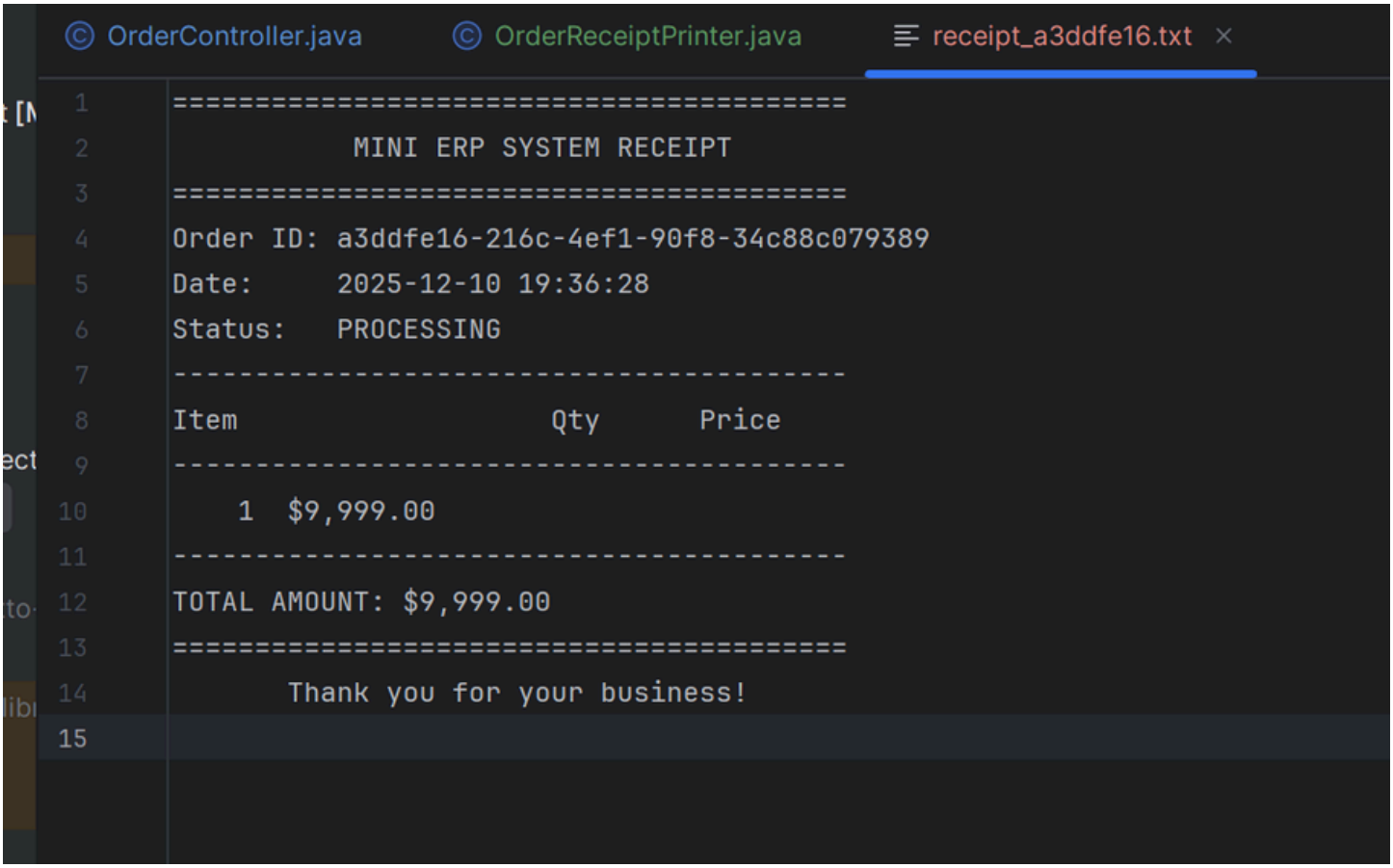
We used maps (HashMap) to quickly look up a specific item using its unique ID (UUID) without having to search through the entire list.

3. **JDBC - Java Database Connectivity:** JDBC was our system's bridge to the database. We built specific functions to handle the four main database actions: Create, Read, Update, and Delete. This allowed our ERP to manage all the data reliably. The JDBC happened in our DAO Implement layer.

4. **GUI Using Swing :** We built the entire user interface using **Java Swing**. To keep the layout clean, we used **tabs** to separate major screens (like Product and Orders) and we organized buttons by putting one panel inside another panel to neatly group our controls.

   Event Handling: We used event handling to manage all interactions within our system. This involved having ActionListener connected with our Buttons. When an event occurred, the handler method was triggered to execute the necessary application logic, like updating a products price and quantity.

5. **I/O Streams:** We used I/O Stream in this feature where, upon order completion, the system automatically writes a detailed receipt. This functionality ensures that every transaction is recorded and provides the admins a file to review the complete sales history.



6. **Multithreading:**
   Our system now uses Multithreading to handle the writing order receipts and reports to files (I/O operations) task. By running these file-writing processes on a separate

background thread, the application prevents the main User Interface from freezing, ensuring the user gets an instant response and a smoother overall experience.

Task distribution among team members :

Planning : Ahmed - Mohamed
File Organization: Ahmed
Data Base : Ahmed
DAO Layer: Ahmed - Mohamed
Service Layer: Ahmed - Mohamed
UI Layer: Ahmed - Mohamed
Report: Mohamed

Challenges and lessons learned :

We had a couple of Challenges that faced us during our Project

1. The first challenge we faced was communicating between our selves. Since the start of the project, neither of us was able to meet in person due to our differing personal responsibilities. To solve this issue, we thought of remote collaboration to ensure we could complete our system on time. Our solution was to use GitHub for uploading and merging code , alongside to being available online to communicate, discuss ideas, and provide assistance to each other when needed.

2. The data base was also a challenge at first we did not know how to have a working data base we both could use at the same time, we ended up using anvia cloud to host our data base . This did come with some performance problems which made our system slower but changing the data base to local makes it fast and with no performance issues.

3. Complexity of Project: At first we wanted to make a full ERP system but then realized that we did not have enough time to make an ERP system, so we chose to make a Mini ERP system.

4. Time was also another challenge we had ,and unfortunately we were not able to implement all the ideas we first had.