INVENTRA - INTELLIGENT INVENTORY MANAGEMENT SYSTEM

MILESTONE-3

PRESENTED BY
RISHWANA SIRAJUTHEEN

WHAT IS INVENTRA?

System

🧠 System Definition

Inventra is a rule-driven inventory monitoring system implemented in Java, designed to detect stock threshold violations and manage alert generation efficiently by preventing redundant notifications using stack-based state tracking.

🎯 Technical Problem Addressed

In conventional inventory monitoring systems:

- Stock status is evaluated at fixed intervals
- Threshold conditions remain unchanged over time
- Identical alerts are generated repeatedly
- Leads to alert flooding, reduced system signal-to-noise ratio, and delayed human response
- Inventra addresses this using state-aware alert validation.

MODULE 1: AUTHENTICATION MODULE

What is the Authentication Module?

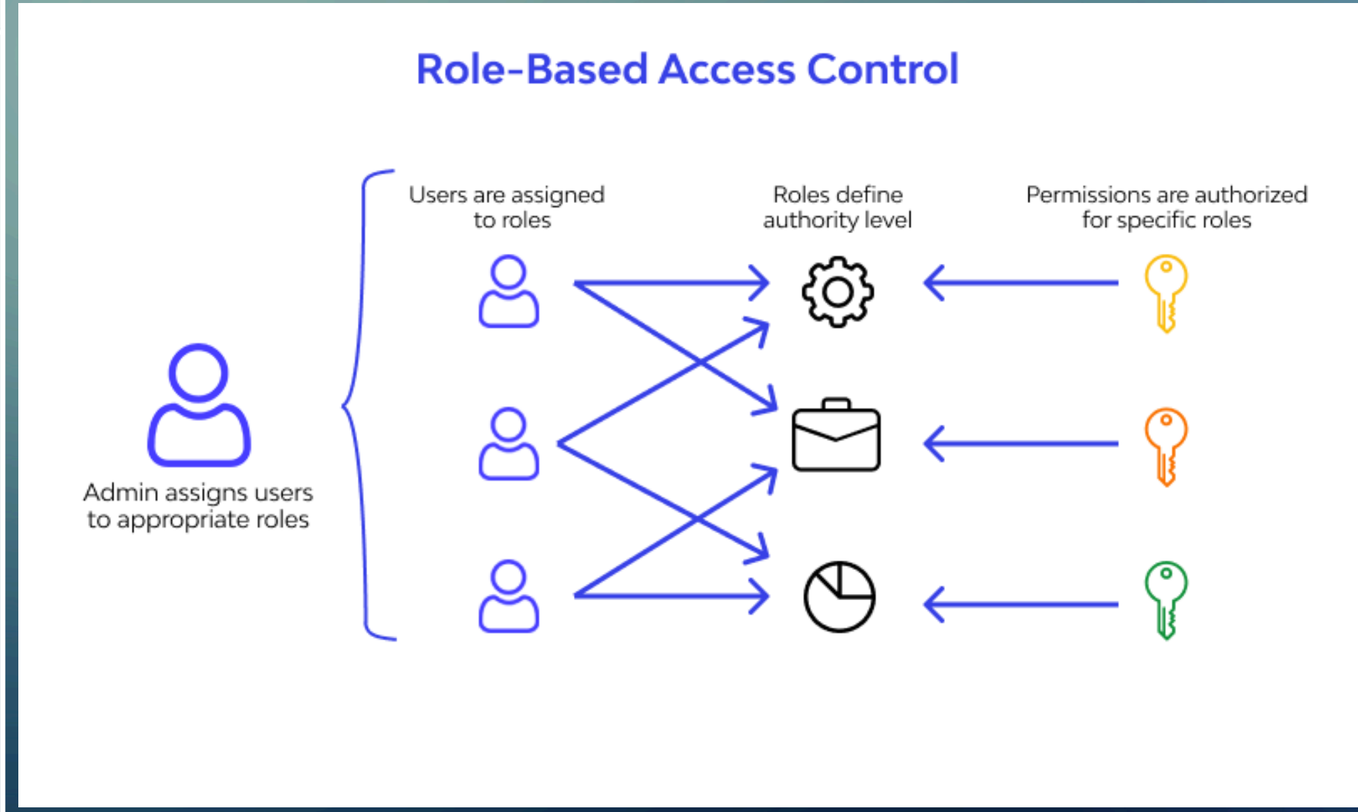
The Authentication Module is responsible for verifying user identity and controlling access to the inventory system.

In Inventra, this module ensures that:

- Only authorized users can log in
- Each user accesses features based on their role
- Inventory data remains secure and protected

Why Authentication is Needed (Problem Statement)

- Without authentication:
- Anyone can access inventory data
- Unauthorized stock changes may occur
- No accountability for actions



User Roles Implemented

Role	Access Level
Admin	Full system control
Manager	Inventory & alerts
Staff	View & update stock

Authentication Workflow (Step-by-Step)

- User enters username & password
- System checks if user exists in database
- Password is validated
- User role is fetched
- Access is granted based on role

Authentication Logic (Pseudocode)

```
START
INPUT username, password
FETCH user from database

IF user exists THEN
    IF password matches THEN
        LOGIN success
        LOAD role permissions
    ELSE
        DISPLAY "Invalid password"
    ENDIF
ELSE
    DISPLAY "User not found"
ENDIF
END
```

1. Architecture Used (Very Important to Say)

The Authentication Module follows Layered Architecture:

UI → Controller → Service → Repository → Database

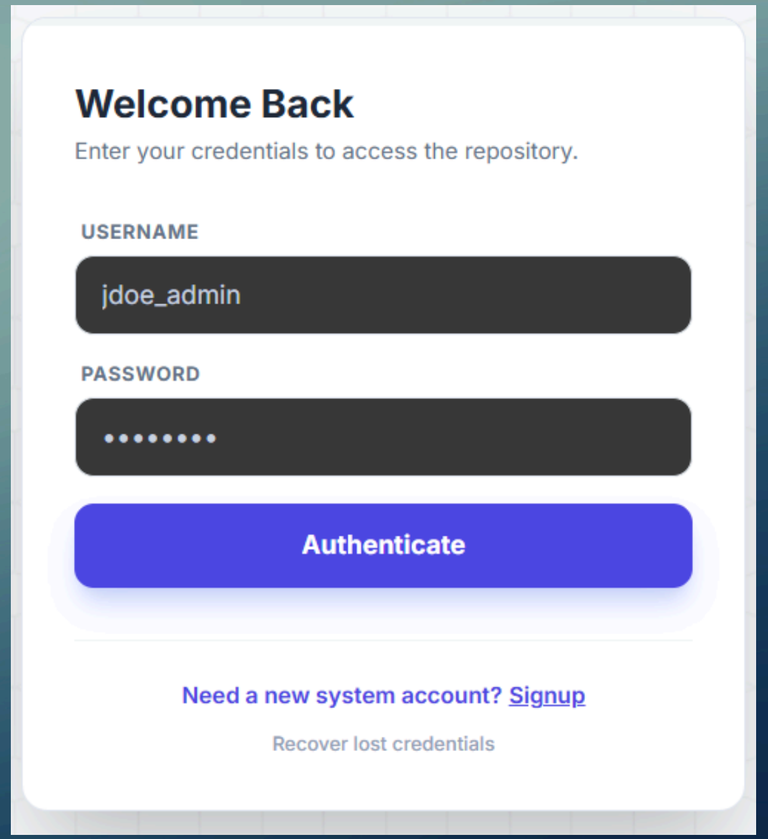
Each layer has a single responsibility, which improves:

- Security
- Maintainability
- Scalability

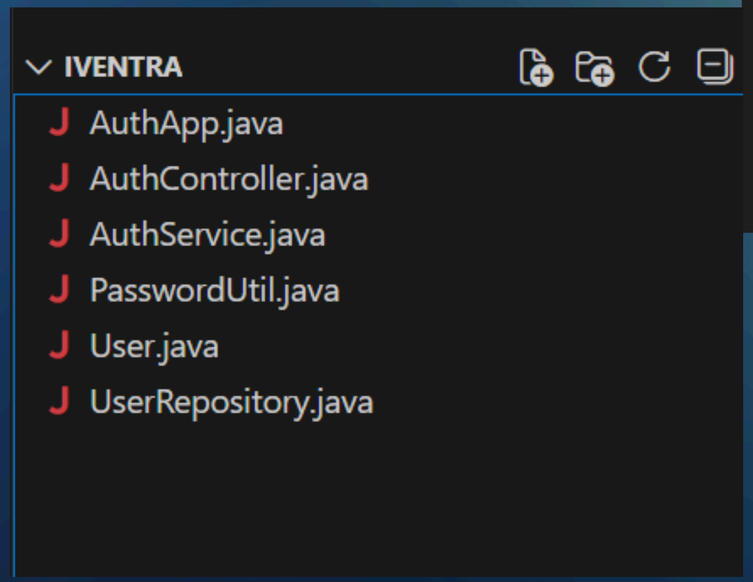
2. Database Management (Module 1)

USERS Table Structure

Class	Responsibility
AuthController	Handles user requests
AuthService	Business logic
User	User entity
UserRepository	Database operations
JWTUtility	Token generation
EmailService	Password reset emails



code screenshots



User.java > ...

```java
public class User {
    private String username;
    private String password; // encrypted
    private String role;
    private String email;

    public User(String username, String password, String role, String email) {
        this.username = username;
        this.password = password;
        this.role = role;
        this.email = email;
    }

    public String getUsername() { return username; }
    public String getPassword() { return password; }
    public String getRole() { return role; }
    public String getEmail() { return email; }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

UserRepository.java > ...

```java
import java.util.HashMap;
import java.util.Map;

public class UserRepository {

    private Map<String, User> users = new HashMap<>();

    public User findByUsername(String username) {
        return users.get(username);
    }

    public User findByEmail(String email) {
        for (User user : users.values()) {
            if (user.getEmail().equals(email)) {
                return user;
            }
        }
        return null;
    }

    public void save(User user) {
        users.put(user.getUsername(), user);
    }
}
```

AuthController.java > ...

```java
public class AuthController {

    private AuthService service = new AuthService();

    public void signup(String username, String password, String role, String email) {
        System.out.println(service.registerUser(username, password, role, email));
    }

    public void signin(String username, String password) {
        System.out.println(service.authenticate(username, password));
    }

    public void forgotPassword(String email, String newPassword) {
        System.out.println(service.resetPassword(email, newPassword));
    }
}

```

PasswordUtil.java > ...

```java
import java.util.Base64;

public class PasswordUtil {

    public static String encrypt(String password) {
        return Base64.getEncoder().encodeToString(password.getBytes());
    }

    public static boolean matches(String rawPassword, String encryptedPassword) {
        return encrypt(rawPassword).equals(encryptedPassword);
    }
}

```

AuthService.java > ...

```java
public class AuthService {

    private UserRepository repository = new UserRepository();

    // SIGN-UP
    public String registerUser(String username, String password, String role, String email) {
        if (repository.findByUsername(username) != null) {
            return "User already exists";
        }

        String encryptedPassword = PasswordUtil.encrypt(password);
        User user = new User(username, encryptedPassword, role, email);
        repository.save(user);

        return "Signup successful";
    }

    // SIGN-IN
    public String authenticate(String username, String password) {
        User user = repository.findByUsername(username);

        if (user == null) {
            return "Invalid username";
        }

        if (PasswordUtil.matches(password, user.getPassword())) {
            return "Login successful | Role: " + user.getRole();
        } else {
            return "Invalid password";
        }
    }

    // FORGOT PASSWORD
    public String resetPassword(String email, String newPassword) {
        User user = repository.findByEmail(email);

        if (user == null) {
            return "Email not registered";
        }

        user.setPassword(PasswordUtil.encrypt(newPassword));
        return "Password reset successful";
    }
}
```

AuthApp.java > ...

```java
public class AuthApp {

    Run | Debug
    public static void main(String[] args) {

        AuthController controller = new AuthController();

        // SIGN-UP
        controller.signup(username: "admin", password: "admin123", role: "ADMIN", email: "admin@mail.cc
        controller.signup(username: "staff1", password: "staff123", role: "STAFF", email: "staff@mail.c

        // LOGIN
        controller.signin(username: "admin", password: "admin123");
        controller.signin(username: "staff1", password: "wrongpass");

        // FORGOT PASSWORD
        controller.forgotPassword(email: "staff@mail.com", newPassword: "newpass123");

        // LOGIN AFTER RESET
        controller.signin(username: "staff1", password: "newpass123");
    }
}

```

```
Login successful | Role: STAFF
PS D:\iventra> ^C
PS D:\iventra>
PS D:\iventra>  d:; cd 'd:\iventra'; & 'C:\P
s' '-cp' 'C:\Users\richu\AppData\Roaming\Cod
entra_ee79eb4\bin' 'AuthApp'
Signup successful
Signup successful
Login successful | Role: ADMIN
Invalid password
Password reset successful
Login successful | Role: STAFF
PS D:\iventra>
```

MODULE 2: PRODUCT / INVENTORY MANAGEMENT

Module 2 is the core operational module of the Inventra – Intelligent Inventory Management System. It is responsible for managing products and maintaining accurate stock levels in real time.

- 🔹 Purpose of Module 2
- The main goal of this module is to:
- Store product details in a structured manner
- Track stock availability continuously
- Handle real-world inventory operations like stock-in and stock-out
- Prevent stock mismatch and negative inventory

- 🔹 Architecture Followed
- This module follows a layered architecture:
- Controller Layer: Receives product and stock requests from the user interface
- Service Layer: Applies all inventory business rules
- Repository Layer: Handles database read/write operations
- This separation ensures scalability, maintainability, and clean design.

🔹 Product Management

Each product in the system has:

- A unique SKU to avoid duplication
- Basic details like name, category, supplier, and unit price
- Stock-related fields such as total quantity and minimum stock level
- Before adding a product, the system checks whether the SKU already exists to prevent duplicate entries.

PSEUDOCODE

Product Entity

```
CLASS Product
    productId
    sku
    name
    category
    supplier
    unitPrice
    stockQuantity
    minStockLevel
END CLASS
```

Add Product

```
FUNCTION addProduct(productData)
    IF product SKU does not exist THEN
        save product
        RETURN "Product added"
    ELSE
        RETURN "Duplicate product"
    END IF
END FUNCTION
```

Stock In

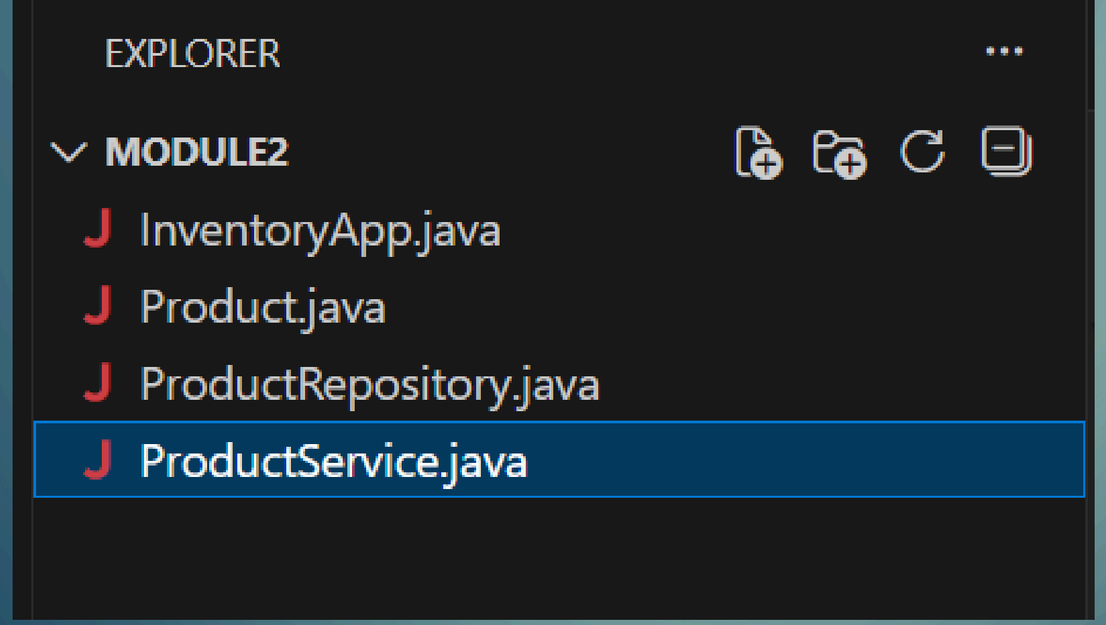
```
FUNCTION stockIn(productId, quantity)
    product = find product by productId
    product.stockQuantity += quantity
    save product
    log STOCK_IN transaction
    check low stock
END FUNCTION
```

Stock Out

```
FUNCTION stockOut(productId, quantity)
    product = find product by productId

    IF product.stockQuantity >= quantity THEN
        product.stockQuantity -= quantity
        save product
        log STOCK_OUT transaction
        check low stock
    ELSE
        RETURN "Insufficient stock"
    END IF
END FUNCTION
```

code screenshots



Product.java > ...

```java
ublic class Product {

    int productId;
    String sku;
    String name;
    int stockQuantity;
    int minStockLevel;

    public Product(int productId, String sku, String name, int stockQuantity, int minStockLevel) {
        this.productId = productId;
        this.sku = sku;
        this.name = name;
        this.stockQuantity = stockQuantity;
        this.minStockLevel = minStockLevel;
    }


```

ProductRepository.java > ...

```java
import java.util.HashMap;
import java.util.Map;

public class ProductRepository {

    Map<Integer, Product> productDB = new HashMap<>();

    public Product findById(int productId) {
        return productDB.get(productId);
    }

    public Product findBySKU(String sku) {
        for (Product p : productDB.values()) {
            if (p.sku.equals(sku)) {
                return p;
            }
        }
        return null;
    }

    public void save(Product product) {
        productDB.put(product.productId, product);
    }
}

```

ProductService.java > ...

```java
public class ProductService {
    public void stockIn(int productId, int quantity) {
        checkLowStock(product);
    }

    public String stockOut(int productId, int quantity) {
        Product product = repository.findById(productId);

        if (product.stockQuantity < quantity) {
            return "Insufficient stock";
        }

        product.stockQuantity -= quantity;
        System.out.println("Stock OUT: " + quantity);
        checkLowStock(product);
        return "Stock updated";
    }

    private void checkLowStock(Product product) {
        if (product.stockQuantity < product.minStockLevel) {
            System.out.println("⚠ ALERT: Low stock for " + product.name);
        }
    }
}

```

InventoryApp.java > ...

```java
public class InventoryApp {

    Run | Debug
    public static void main(String[] args) {

        ProductService service = new ProductService();

        Product p1 = new Product(productId: 1, sku: "SKU101", name: "Popcorn", stockQuantity: 50, minSt

        System.out.println(service.addProduct(p1));

        service.stockIn(productId: 1, quantity: 30);
        System.out.println(service.stockOut(productId: 1, quantity: 40));
        System.out.println(service.stockOut(productId: 1, quantity: 30)); // triggers low stock
    }
}

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\module2>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-
chu\AppData\Roaming\Code\User\workspaceStorage\f83f010f7ff904ab4
InventoryApp'
Product added successfully
Stock IN: 30
Stock OUT: 40
Stock updated
Stock OUT: 30
? ALERT: Low stock for Popcorn
Stock updated
PS D:\module2>
```

MODULE 3: INVENTORY ALERTS MODULE

Module 3 is the Inventory Alerts Module, which adds intelligence and automation to the Inventra – Intelligent Inventory Management System.

Its primary role is to monitor stock levels continuously and notify responsible users before a stock-out situation occurs.

Purpose of Module 3

- The main objectives of this module are to:
- Detect low-stock conditions in real time
- Notify users at the right time
- Avoid repeated or unnecessary alerts
- Improve inventory decision-making

Problem Addressed

In traditional inventory systems:

- The same low-stock alert is generated repeatedly
- Users get alert fatigue
- Important alerts may be ignored
- This module solves that problem by using controlled alert generation.

Alert Trigger Mechanism

Module 3 is automatically triggered after every stock update:

- Stock-in
- Stock-out

If the stock quantity falls below the minimum stock level, the alert logic is activated.

🔹 Intelligent Alert Handling

To avoid duplicate alerts:

- Alerts are stored using a Stack data structure
- Before adding a new alert, the system checks the top of the stack
- If the alert already exists, it is not repeated
- This ensures only meaningful alerts are shown.

.

🔹 Alert Flow

Inventory quantity is updated

System checks minimum stock threshold

If stock is low:

- Alert message is generated
- Duplicate alerts are prevented

Alert is displayed to the user

🔹 Integration with Other Modules

Works directly with Module 2 (Inventory Management)

Can be extended to:

- Email notifications
- Dashboard warnings
- SMS alerts

PSEUDOCODE

🔹 Alert Logic Using Stack

```
CLASS AlertManager
    CREATE Stack alertStack
END CLASS
```

Check Low Stock

```
FUNCTION checkLowStock(product)
    IF product.stockQuantity < product.minStockLevel THEN
        alertMessage = "Low stock for " + product.name
        addAlert(alertMessage)
    END IF
END FUNCTION
```

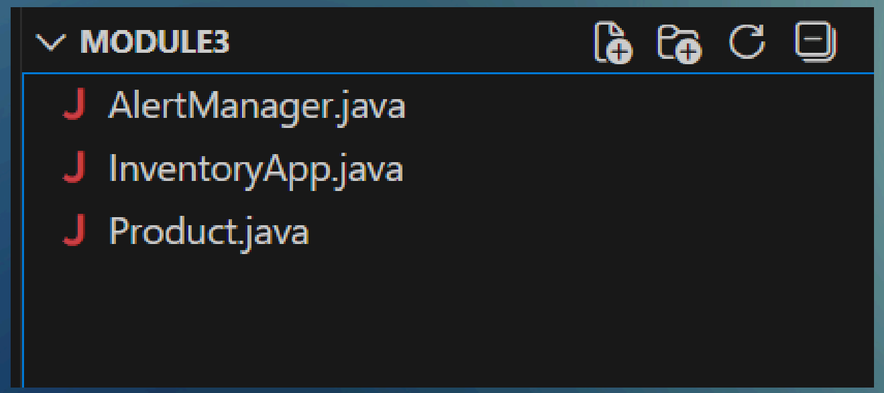
Add Alert (Avoid Duplicates)

```
FUNCTION addAlert(alertMessage)
    IF alertStack is empty OR top of stack != alertMessage THEN
        PUSH alertMessage into stack
        DISPLAY alertMessage
    END IF
END FUNCTION
```

View Latest Alert

```
FUNCTION getLatestAlert()
    IF stack not empty THEN
        RETURN top alert
    ELSE
        RETURN "No alerts"
    END IF
END FUNCTION
```

code screenshots



Product.java > ...

```java
public class Product {

    int productId;
    String name;
    int stockQuantity;
    int minStockLevel;

    public Product(int productId, String name, int stockQuantity, int minStockLevel) {
        this.productId = productId;
        this.name = name;
        this.stockQuantity = stockQuantity;
        this.minStockLevel = minStockLevel;
    }
}
```

AlertManager.java > ...

```java
import java.util.Stack;

public class AlertManager {

    private Stack<String> alertStack = new Stack<>();

    public void checkLowStock(Product product) {

        if (product.stockQuantity < product.minStockLevel) {
            String alertMessage = "⚠ LOW STOCK ALERT: " + product.name +
                                  " | Current Stock: " + product.stockQuantity;
            addAlert(alertMessage);
        }
    }

    private void addAlert(String alertMessage) {

        if (alertStack.isEmpty() || !alertStack.peek().equals(alertMessage)) {
            alertStack.push(alertMessage);
            System.out.println(alertMessage);
        }
    }

    public String getLatestAlert() {

        if (!alertStack.isEmpty()) {
            return alertStack.peek();
        }
        return "No alerts available";
    }
```

InventoryApp.java > ...

```java
public class InventoryApp {

    Run | Debug
    public static void main(String[] args) {

        AlertManager alertManager = new AlertManager();

        Product p1 = new Product(productId: 1, name: "Popcorn", stockQuantity: 50, minStockLevel: 20);

        // Simulate stock usage
        p1.stockQuantity = 25;
        alertManager.checkLowStock(p1); // No alert

        p1.stockQuantity = 15;
        alertManager.checkLowStock(p1); // Alert generated

        p1.stockQuantity = 10;
        alertManager.checkLowStock(p1); // Duplicate avoided

        System.out.println("Latest Alert: " + alertManager.getLatestAlert());
    }
}
```

```
PS D:\module3>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDeta
chu\AppData\Roaming\Code\User\workspaceStorage\dc58451fed23108212d06f1c64dfec56\
InventoryApp'
? LOW STOCK ALERT: Popcorn | Current Stock: 15
? LOW STOCK ALERT: Popcorn | Current Stock: 10
Latest Alert: ? LOW STOCK ALERT: Popcorn | Current Stock: 10
PS D:\module3>
```

MODULE 1: AUTHENTICATION – DATABASE MANAGEMENT

Column Name	Data Type	Description
user_id	INT (PK)	Unique user identifier
username	VARCHAR	Unique login name
password	VARCHAR	Encrypted password
role	VARCHAR	ADMIN / STAFF
email	VARCHAR	Used for password reset
created_at	TIMESTAMP	Account creation time

```sql
CREATE TABLE users (
    user_id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(100) NOT NULL,
    role VARCHAR(20) NOT NULL,
    email VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- How Module 1 Uses the DB
- Signup → INSERT new user
- Login → SELECT username & password
- Role validation → SELECT role
- Forgot password → SELECT by email

✓ Ensures security, authentication, and authorization

MODULE 2: PRODUCT / INVENTORY MANAGEMENT – DATABASE MANAGEMENT

Column Name	Data Type	Description
product_id	INT (PK)	Unique product ID
sku	VARCHAR	Unique product code
name	VARCHAR	Product name
category	VARCHAR	Category
stock_quantity	INT	Current stock
min_stock_level	INT	Alert threshold

```sql
CREATE TABLE products (
    product_id INT PRIMARY KEY AUTO_INCREMENT,
    sku VARCHAR(50) UNIQUE,
    name VARCHAR(100),
    category VARCHAR(50),
    stock_quantity INT,
    min_stock_level INT
);
```

- PRODUCTS Table

Column	Type	Description
transaction_id	INT (PK)	Unique transaction
product_id	INT (FK)	Related product
type	VARCHAR	STOCK_IN / STOCK_OUT
quantity	INT	Quantity changed
transaction_date	TIMESTAMP	When it happened

```sql
CREATE TABLE inventory_transactions (
    transaction_id INT PRIMARY KEY AUTO_INCREMENT,
    product_id INT,
    type VARCHAR(20),
    quantity INT,
    transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

- INVENTORY_TRANSACTIONS Table

- How Module 2 Uses the DB

Add product → INSERT into products

Stock in/out → UPDATE stock_quantity

Audit trail → INSERT into inventory_transactions

MODULE 3: INVENTORY ALERTS – DATABASE MANAGEMENT

Column Name	Data Type	Description
alert_id	INT (PK)	Unique alert
product_id	INT (FK)	Related product
alert_message	VARCHAR	Alert text
alert_time	TIMESTAMP	Alert timestamp
status	VARCHAR	ACTIVE / RESOLVED

```sql
CREATE TABLE alerts (
    alert_id INT PRIMARY KEY AUTO_INCREMENT,
    product_id INT,
    alert_message VARCHAR(255),
    alert_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    status VARCHAR(20),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

- How Module 3 Uses the DB
  - Low stock detected → INSERT alert
  - Duplicate prevention → CHECK last alert
  - Alert resolution → UPDATE status

THANK YOU