Inventra - Intelligent Inventory Management System

Presented by
Rishwana Sirajutheen
9123106061@kcgcollege.com

ABOUT

- Definition: Inventra is an intelligent repository management system.
- Purpose: Helps businesses efficiently manage stock, suppliers, purchase orders, and sales analysis.
- Key benefit: Reduces manual effort and improves inventory accuracy.

FEATURES OF INVENTRA

- Stock Tracking: Real-time monitoring of inventory levels.
- Supplier Management: Stores supplier details and manages communication.
- Purchase Order Monitoring: Tracks orders and ensures availability.
- Sales Analysis: Analyzes sales to predict demand.
- Automation: Updates inventory, sends alerts, and generates reports automatically.
- User-Friendly Interface: Easy navigation for non-technical users.

EXISTING PROBLEM

1. Manual Tracking Errors

Keeping records manually leads to mistakes in stock counts.

Hard to track real-time inventory levels.

2. Overstocking & Stockouts

Excess stock ties up capital.

Shortages affect sales and customer satisfaction.

3. Inefficient Supplier Management

Difficulty in tracking orders from multiple suppliers.

Communication delays or missed deliveries.

4. Lack of Analytics

No data-driven insights for predicting demand.

Hard to make informed purchase or sales decisions.

5. Time-Consuming Processes

Manual updating of stock, purchase orders, and reports consumes resources.

6. Security & Access Issues

No control over who can access inventory data.

Risk of data manipulation or unauthorized access.

PROPOSED SOLUTION – INVENTRA

1. Automated Inventory Management

Real-time stock tracking with minimal manual intervention.

Automatic updates on stock levels after sales or purchases.

2. Smart Stock Control

Alerts for low stock and overstock situations.

Prevents stockouts and reduces excess inventory.

3. Efficient Supplier Management

Stores supplier details and tracks all purchase orders.

Improves communication and reduces missed deliveries.

4. Analytics and Reporting

Generates sales and inventory reports automatically.

Helps in forecasting demand and making data-driven decisions.

5. Secure Access & Authentication

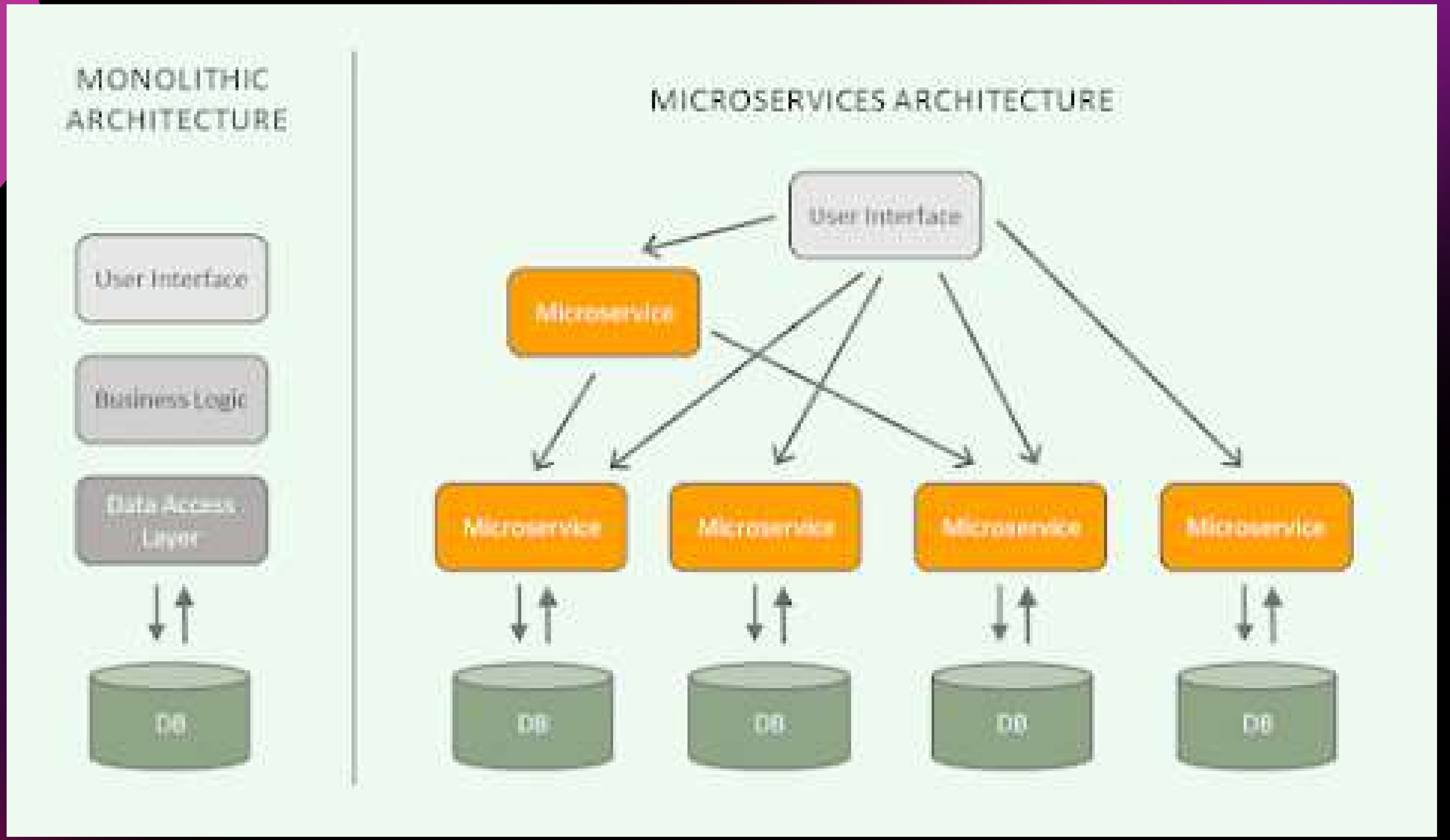
Role-based access for Admin, Staff, and Viewers.

Password protection and user verification to prevent unauthorized access.

6. User-Friendly Interface

Intuitive design for easy navigation.

Reduces training time for employees.


MONOLITHIC ARCHITECTURE
User Interface
Business Logic
Data Access Layer
DB
MICROSERVICES ARCHITECTURE
User Interface
Microservice
Microservice
Microservice
Microservice
Microservice
DB
DB
DB
DB

THE TOOLS USED LIKE THIS:

- Programming language: Java 17
- Backend framework: Spring Boot (Spring Web, Spring Data JPA, Spring Security)
- Frontend: React or Angular or HTML/CSS/JavaScript
- Database: MySQL or PostgreSQL
- Authentication: JWT (JSON Web Token)
- IDE: IntelliJ IDEA
- Build tool: Maven (or Gradle)
- API testing: Postman / REST Client
- Version control: Git + GitHub/GitLab/Bitbucket

SYSTEM ARCHITECTURE – 1.PRESENTATION LAYER:

The system has controllers like AuthController, ProductController, AlertController, and ReportController, which only handle incoming requests from the UI and forward them to the Service layer, without containing business logic.

1. AuthController

Use: Handles user authentication like login, logout, and role verification.

2. ProductController

Use: Manages inventory items—adding, updating, viewing, or deleting products.

3. AlertController

Use: Sends notifications or alerts, e.g., low stock warnings or system messages.

4. ReportController

Use: Generates reports and analytics for sales, stock levels, and supplier activity.

2.SERVICE LAYER (BUSINESS LOGIC)

The Service Layer contains the business logic, processes requests from controllers, and interacts with the database via repositories.

Each Service and Its Use:

1. **AuthService** – Handles authentication and user-related logic.
2. **ProductService** – Manages inventory operations like add, update, or delete products.
3. **AlertService –** Processes alerts and notifications for low stock or system events.
4. **ReportService** – Generates and processes reports for sales, stock, and suppliers.

Key Point:

Each service uses a repository to fetch or store data in the database.

Keeps business logic separate from UI and request handling

3. REPOSITORY LAYER

The Repository Layer handles direct database access and executes queries to fetch or store data.

Each Repository and Its Use:

UserRepository – Manages database operations for users (e.g., login, role info).

ProductRepository – Handles inventory item data in the database.

AlertRepository – Stores and retrieves alerts or notifications.

ReportRepository – Manages report data for sales, stock, and supplier analytics.

Key Point:

These are interfaces in Spring Boot that allow automatic SQL query execution.

Keeps database operations separate from business logic and UI.

4.Model Layer (Data Classes / Entities)

Models represent database tables and define the structure of data used in the system.

Each Model and Its Use:

User – Represents user information like login credentials and roles.

Product – Represents inventory items with details like name, quantity, and price.

Alert – Represents alerts or notifications for events like low stock.

Report – Represents report data for sales, stock, or supplier analysis.

ReportRequest – Represents parameters or requests for generating specific reports.

Key Point:

Models act as a bridge between the database and the Service layer, ensuring structured data is passed and stored correctly.

Each model corresponds to a table in the database.

Module 1: Authentication Module

Classes Involved:

User – Model representing user data.
AuthController – Receives login/logout requests from the UI.
AuthService – Handles authentication logic like password verification and role checking.
UserRepository – Accesses the User table in the database.

Flow:

UI → AuthController → AuthService → UserRepository → Database

(User table)

Explanation:

AuthController gets the user request.

AuthService processes login/logout logic.

UserRepository fetches user data from the database.

User table stores all user credentials and roles.

One-line summary:

“This module ensures secure user authentication by connecting the UI, service logic, and database.”

DB schema

Field Name	Data Type	Description
id	INT (Primary Key, Auto Increment)	Unique user ID
username	VARCHAR(50)	User login name
email	VARCHAR(100)	User email address
password	VARCHAR(255)	Hashed password
role	VARCHAR(20)	User role (Admin, Staff, Viewer)
created_at	TIMESTAMP	Account creation time
updated_at	TIMESTAMP	Last update time

1. User.java (Model / Entity)

```
import jakarta.persistence.*;
import java.time.LocalDateTime;

@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(nullable = false, unique = true, length = 50)
    private String username;

    @Column(nullable = false, unique = true, length = 100)
    private String email;

    @Column(nullable = false, length = 255)
    private String password;

    @Column(nullable = false, length = 20)
    private String role;

    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;

    // Getters and Setters
    // Constructor(s)
}
```

2. UserRepository.java (Repository Interface)

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends JpaRepository<User, Integer> {
    User findByUsername(String username);
}
```

3. AuthService.java (Service Layer)

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Service
public class AuthService {

    @Autowired
    private UserRepository userRepository;

    private BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();

    // Authenticate user
    public boolean authenticate(String username, String password) {
        User user = userRepository.findByUsername(username);
        if (user != null) {
            return passwordEncoder.matches(password, user.getPassword());
        }
        return false;
    }

    // Other user-related business logic
}
```

4. AuthController.java (Presentation Layer)

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/auth")
public class AuthController {

    @Autowired
    private AuthService authService;

    @PostMapping("/login")
    public String login(@RequestParam String username,
@RequestParam String password) {
        boolean isAuthenticated = authService.authenticate(username,
password);
        if (isAuthenticated) {
            return "Login Successful";
        } else {
            return "Invalid Username or Password";
        }
    }
}
```

Explanation:

User → Represents the database table.

UserRepository → Handles database operations.

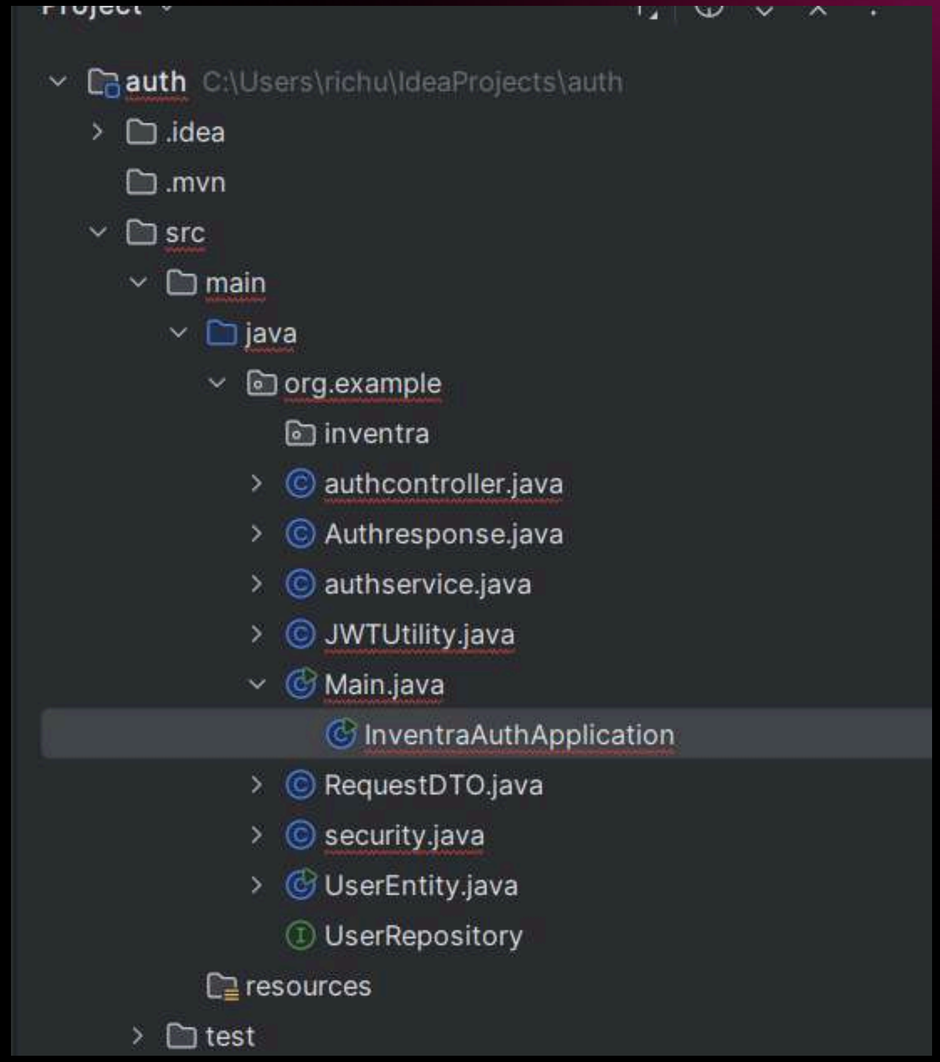
AuthService → Contains the business logic for authentication.

AuthController → Receives login requests and returns responses.

Auth DTOs

DTO Class	Purpose	Fields	Used for
LoginRequest	Captures the credentials sent by a client trying to log in.	username, password	The POST /api/auth/login endpoint.
RegisterRequest	Captures all necessary data for creating a new user account.	username, email, password, role (optional)	The POST /api/auth/register endpoint.
ResetPasswordRequest	Captures the information needed to securely update a forgotten password.	email, newPassword	The POST /api/auth/reset-password endpoint.
AuthResponse	The standardized structure for the server's response after a successful login or registration.	username, role, token (the generated JWT)	Returned by the login and register endpoints.

main classes



Main.java | UserEntity.java | UserRepository.java | Authresponse.java | authservice.java | JWT

```java
package com.inventra.entity;

import jakarta.persistence.*;
import lombok.*;
import org.hibernate.annotations.CreationTimestamp;

import java.time.Instant;

@Entity  9 usages  2 related problems
@Table(name = "users")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class User {

    @Id  no usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "user_id")
    private Long userId;

    @Column(unique = true, nullable = false, length = 50)  no usages
    private String username;

    @Column(unique = true, nullable = false, length = 100)  no usages
    private String email;

    @Column(nullable = false)  no usages
    private String password;

    @Column(length = 30)  no usages
    private String role; // ADMIN, STAFF, MANAGER
```

```java
package com.inventra.repository;

import com.inventra.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository  no usages  1 related problem
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);  no usages  1 related problem
    Optional<User> findByEmail(String email);  no usages  1 related problem
    boolean existsByUsername(String username);  no usages  1 related problem
    boolean existsByEmail(String email);  no us
}
```

Main.java | UserEntity.java | UserRepository.java | Authresponse.java | authservice.java | JWTUtility.java

```java
package com.inventra;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class InventraAuthApplication {

    public static void main(String[] args) {
        SpringApplication.run(InventraAuthApplication.class, args);
    }

}
```

Main.java | UserEntity.java | UserRepository.java | Authresponse.java | authservice.java | JWTUtility.java

```java
import com.inventra.entity.User;
import com.inventra.repository.UserRepository;
import com.inventra.util.JwtUtil;
import lombok.RequiredArgsConstructor;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

@Service  no usages  1 related problem
@RequiredArgsConstructor
public class AuthService {

    private final UserRepository userRepository;  6 usages
    private final PasswordEncoder passwordEncoder;  3 usages
    private final JwtUtil jwtUtil;  2 usages

    public AuthResponse login(LoginRequest request) {  no usages  2 related problems
        User user = userRepository.findByUsername(request.getUsername())
                .orElseThrow(() -> new RuntimeException("User not found"));

        if (!user.getIsActive()) {
            throw new RuntimeException("Account is inactive");
        }

        if (!passwordEncoder.matches(request.getPassword(), user.getPassword())) {
            throw new RuntimeException("Invalid credentials");
        }

        String token = jwtUtil.generateToken(user.getUsername(), user.getRole());

        return AuthResponse.builder()
                .username(user.getUsername())
                .role(user.getRole())
```

Main.java | UserEntity.java | UserRepository.java | Authresponse.java | authservice.java | JWTUtility.java

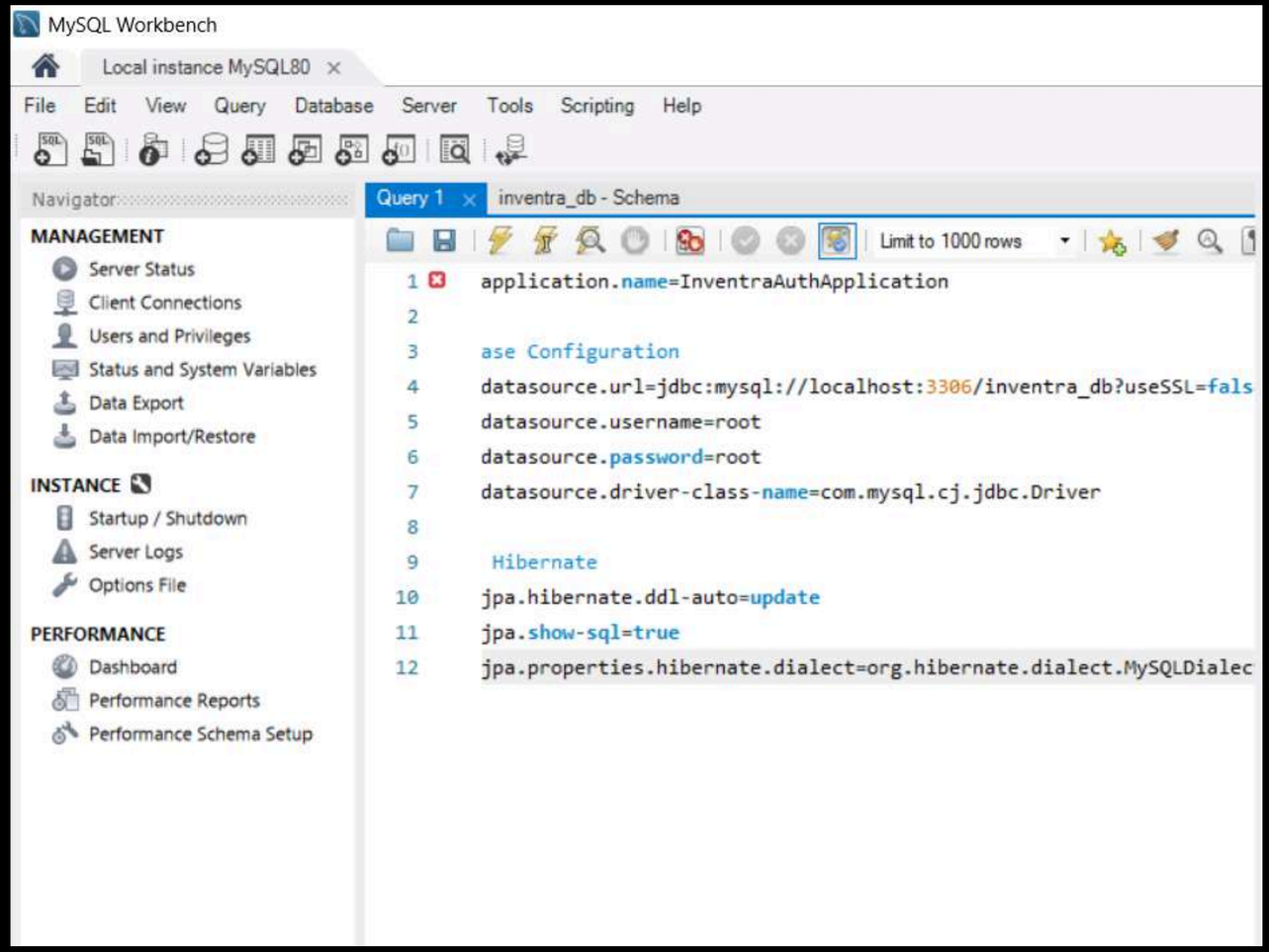
```java
package com.inventra.util;

import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.io.Decoders;
import io.jsonwebtoken.security.Keys;
import org.springframework.stereotype.Component;

import java.security.Key;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

@Component  no usages  1 related problem
public class JwtUtil {

    // 256-bit HEX secret key for HS256 algorithm
    public static final String SECRET = "5367566B597033733367639792F423F4528482B4D62516554685765D5A71347437";  1 usag

    // 15 minutes in milliseconds
    private static final long EXPIRATION_TIME = 1000 * 60 * 15;  1 usage

    public String generateToken(String username, String role) {  no usages  2 related problems
        Map<String, Object> claims = new HashMap<>();
        claims.put("role", role);
        return createToken(claims, username);
    }

    private String createToken(Map<String, Object> claims, String subject) {  1 usage
        return Jwts.builder()
                .setClaims(claims)
                .setSubject(subject)
```

```java
    // 15 minutes in milliseconds
    private static final long EXPIRATION_TIME = 1000 * 60 * 15;  1 usage

    public String generateToken(String username, String role) {  no usages  2 related problems
        Map<String, Object> claims = new HashMap<>();
        claims.put("role", role);
        return createToken(claims, username);
    }

    private String createToken(Map<String, Object> claims, String subject) {  1 usage
        return Jwts.builder()
                .setClaims(claims)
                .setSubject(subject)
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + EXPIRATION_TIME))
                .signWith(getSignKey(), SignatureAlgorithm.HS256)
                .compact();
    }
```

ity.java | UserRepository.java | Authresponse.java | authservice.java | JWTUtility.java | RequestDTO.java

```java
package com.inventra.dto;

import lombok.Data;

// Containing multiple DTOs in one file for concise copying;
// in production, these can be split into separate files.

@Data  no usages
public class AuthRequests {
    // Placeholder if needed
}

@Data  no usages  2 related problems
class LoginRequest {
    private String username;  no usages
    private String password;  no usages
}

@Data  no usages  2 related problems
class RegisterRequest {
    private String username;  no usages
    private String email;  no usages
    private String password;  no usages
    private String role;  no usages
}

@Data  no usages  2 related problems
class ResetPasswordRequest {
    private String email;  no usages
    private String newPassword;  no usages
}
```

my SQL


MySQL Workbench
Local instance MySQL80
File Edit View Query Database Server Tools Scripting Help
Navigator
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Performance Schema Setup
Query 1
inventra_db - Schema
Limit to 1000 rows
1 application.name=InventraAuthApplication
2
3 ase Configuration
4 datasource.url=jdbc:mysql://localhost:3306/inventra_db?useSSL=fals
5 datasource.username=root
6 datasource.password=root
7 datasource.driver-class-name=com.mysql.cj.jdbc.Driver
8
9 Hibernate
10 jpa.hibernate.ddl-auto=update
11 jpa.show-sql=true
12 jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialec

THANKYOU