

**NATIONAL UNIVERSITY OF SCIENCES & TECHNOLOGY**

**MILITARY COLLEGE OF SIGNALS**



**Computer Networks**  
**(EE-353)**

**OPEN-ENDED LAB**

**Submitted by: MUHAMMAD AHMAD SULTAN**

**CMS ID: 408709**

**RANK: NC**

**COURSE: BESE-28**

**SECTION: C**

**Submitted to: Sir Zohaib Ali**

***Dated: 09-01-2024***

# Catalog of Open-Ended Lab

## Table of Contents

▪ Title of the Open-Ended Lab.....	03
▪ About Me .....	04
▪ Introduction (Abstract).....	05
▪ History of Library Management Systems .....	05
▪ Objective.....	06
▪ Scope .....	06
▪ Functionality & Implementation .....	07
▪ MySQL Database Queries .....	07
▪ Java Classes.....	13
▪ Input and Output Specifications .....	17
▪ OOP Principles & Concepts.....	18
▪ Procedure(Validation) .....	19
▪ Software/Equipment Used .....	20
▪ Execution & Output .....	21
▪ Display .....	23
▪ Potential Questions.....	29
▪ Conclusion .....	29
▪ Reference Citations .....	29

**DEPARTMENT OF COMPUTER SOFTWARE ENGINEERING**  
**Military College of Signals**  
**National University of Sciences and Technology**

[www.mcs.nust.edu.pk](http://www.mcs.nust.edu.pk)



# TITLE OF THE OPEN-ENDED LAB

## Dynamic DNS Nexus

## DNS

# Management System

(Explorer & Manager Hub)

*A Comprehensive DNS Management System (Dynamic DNS Nexus) in Java, seamlessly incorporating Data Structures & Algorithms, Object-Oriented Programming principles, and Database Management using MySQL. This system ensures streamlined and secure DNS operations, providing efficient management of domain name information and IP addresses.*

# About me

Allow me to introduce myself. I am *Muhammad Ahmad Sultan*, a highly motivated second-year student at the ***Military College of Signals***, currently pursuing a **Bachelor's degree in Software Engineering**. With unwavering determination, I am committed to achieving my goals and making significant strides in my academic journey.



**Muhammad Ahmad Sultan**

Developer

## ▪ **Abstract:**

The Dynamic DNS Nexus (DNS Management System) is a Java-based application developed using Java Swing for the user interface and MySQL Workbench as the database management system. This system aims to provide a comprehensive solution for managing Domain Name System (DNS) entries, allowing users to add, update, delete, and search DNS information efficiently. In addition to its core functionalities, the Dynamic DNS Nexus (DNS Management System) employs an intuitive and user-friendly design, enabling seamless navigation and interaction. Leveraging the power of Java Swing, the application offers a visually appealing interface, enhancing the user experience while ensuring robust performance. Furthermore, the integration with MySQL Workbench not only facilitates efficient data management but also ensures the security and integrity of DNS records.

## ▪ **History of DNS Management Systems:**

The Domain Name System (DNS) has played a pivotal role in the evolution of the internet, dating back to its inception in the 1980s. Initially conceptualized as a decentralized system to overcome the limitations of maintaining a centrally managed host.txt file, DNS emerged as a critical infrastructure facilitating the seamless navigation of the expanding web.

As the internet burgeoned, the DNS Management System evolved to address the escalating complexity of managing an ever-growing number of domain names. Over the years, it has adapted to technological advancements, incorporating robust security measures and optimizing the translation of domain names into corresponding IP addresses. Today, the DNS Management System stands as a cornerstone in the digital architecture, supporting the efficient and secure functioning of online services worldwide.

## ▪ Objective:

The primary objective of the project is to create a user-friendly DNS Management System that allows users to perform operations such as adding, updating, deleting, and searching DNS entries. The system aims to demonstrate the functionality of a DNS server, showcasing the translation of domain names to IP addresses and vice versa. The project also aims to enhance DNS administration by implementing advanced features such as comprehensive search functionalities, real-time DNS entry filtering, and efficient record retrieval. Additionally, the system prioritizes user accessibility through an intuitive graphical interface, ensuring a seamless experience in managing DNS records.

## ▪ Scope:

The project covers the development of a Java-based application with a graphical user interface (GUI) using Java Swing. It integrates with a MySQL database to store and manage DNS information. The system allows users to perform CRUD (Create, Read, Update, Delete) operations on DNS entries, enhancing the efficiency of managing DNS records. The project's scope extends beyond basic CRUD operations, offering advanced features such as real-time DNS entry filtering, intuitive record navigation, and comprehensive search functionalities. Additionally, the system prioritizes user experience by implementing a responsive Java Swing GUI, ensuring a seamless and interactive environment for DNS management. The integration with MySQL further enhances data security and scalability, making the DNS Management System a robust solution for organizations with diverse DNS requirements.

## ▪ DNS Nexus Management System – Functionality & Implementation

### 1. Database Table Creation




#### 1.1 Table Specification

A table named "dnsSys" is created in the MySQL database.

The table includes columns for infokey, name, dnsName, class, and ipAddress.

## MySQL Database Queries

```
1  -- Create the database
2  CREATE DATABASE dns_management_sys;
3
4  -- Show existing databases (just for verification)
5  SHOW DATABASES;
6
7  -- Use the created database
8  USE dns_management_sys;
9
10 -- Create a table for user login
11 CREATE TABLE login (
12     username VARCHAR(25),
13     password VARCHAR(25)
14 );
15
16 -- Insert sample login data
17 INSERT INTO login VALUES ('ahmad', 'mas03');
18 INSERT INTO login VALUES ('admin', '1122');
19
```

Result Grid		 Filter Rows: <input type="text"/>	Export: 	Wrap Cell Content: 
	username	password		
▶	ahmad	mas03		
	admin	1122		

```

20  -- Create a table for DNS information
21  ● CREATE TABLE dnsSys (
22      infokey INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
23      name VARCHAR(255) NOT NULL,
24      dnsName VARCHAR(255) NOT NULL UNIQUE,
25      class VARCHAR(255),
26      ipAddress VARCHAR(255) NOT NULL UNIQUE
27  );
28
29  -- Select all records from the dnsSys table
30  ● SELECT * FROM dnsSys;
31
32  -- Select all records from the login table
33  ● SELECT * FROM login;
34

```

Result Grid					
	infokey	name	dnsName	class	ipAddress
▶	1	Google	www.google.com	Class A	8.8.8.8
	2	YouTube	www.youtube.com	Class C	208.65.153.238
*	NULL	NULL	NULL	NULL	NULL

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	username	password		
▶	ahmad	mas03		
	admin	1122		

## Explanation:

- Create the Database

-- Create the database

```
CREATE DATABASE dns_management_sys;
```

**Explanation:** This section creates a new database named dns\_management\_sys using the CREATE DATABASE statement.

- Show Existing Databases (Verification)

-- Show existing databases (just for verification)

```
SHOW DATABASES;
```

**Explanation:** This section uses the SHOW DATABASES statement to display a list of existing databases. It's often used for verification purposes.



- Use the Created Database

-- Use the created database

```
USE dns_management_sys;
```

**Explanation:** This line selects the newly created database, dns\_management\_sys, for subsequent operations using the USE statement.

- Create a Table for User Login

-- Create a table for user login

```
CREATE TABLE login (  
    username VARCHAR(25),  
    password VARCHAR(25)  
);
```

**Explanation:** This section creates a table named login with columns username and password, both having a data type of VARCHAR(25).

- Insert Sample Login Data

-- Insert sample login data

```
INSERT INTO login VALUES ('ahmad', 'mas03');
```

```
INSERT INTO login VALUES ('admin', '1122');
```

**Explanation:** These lines insert sample login data into the login table. Two records are added with usernames 'ahmad' and 'admin', each with a corresponding password.

- Create a Table for DNS Information

-- Create a table for DNS information

```
CREATE TABLE dnsSys (  
    infokey INT PRIMARY KEY NOT NULL AUTO_INCREMENT,  
    name VARCHAR(255) NOT NULL,  
    dnsName VARCHAR(255) NOT NULL UNIQUE,  
    class VARCHAR(255),  
    ipAddress VARCHAR(255) NOT NULL UNIQUE  
);
```

**Explanation:** This section creates a table named dnsSys with columns infokey, name, dnsName, class, and ipAddress. infokey is set as the primary key with auto-increment.

- Select All Records from the dnsSys Table

-- Select all records from the dnsSys table

```
SELECT * FROM dnsSys;
```

**Explanation:** This line retrieves all records from the dnsSys table using the SELECT \* FROM dnsSys statement.

- **Select All Records from the login Table**

-- Select all records from the login table

```
SELECT * FROM login;
```

**Explanation:** This line retrieves all records from the login table using the SELECT \* FROM login statement.

## ***2. User Interface***

### **2.1 Main\_Window.java (Splash Window)**

**Description:** Displays a welcoming splash window upon application launch.

### **2.2 Login.java**

**Description:** Manages user authentication and login functionality.

**Implementation:**

Validates user credentials against the authentication system.

Provides a secure login mechanism.

### **2.3 DNS\_Nexus\_Management\_System.java**

**Description:** The main class implementing the core functionalities of DNS Management System.

**Implementation:**

Utilizes Java Swing for the graphical user interface.

Integrates with the MySQL database for data storage.

Orchestrates the flow of DNS management operations.

## ***3. CRUD Operations***

### **3.1 Adding a Record**

**Description:** Users can add DNS information by providing values for infokey, name, dnsName, class, and ipAddress.

**Implementation:**

Captures user input and inserts a new record into the "dnsSys" table.

Utilizes PreparedStatement for secure SQL operations.

### **3.2 Updating a Record**

**Description:** Users can update existing DNS information based on the infokey.

**Implementation:**

Retrieves existing record based on infokey and updates specified fields.

Displays a success or error message upon completion.

### 3.3 Deleting a Record

**Description:** Records can be deleted by specifying the infokey.

**Implementation:**

Deletes a record from the "dnsSys" table based on the provided infokey.

Provides user feedback through success or error messages.

### 3.4 Viewing Records

**Description:** Users can navigate through records, including displaying the first, last, next, and previous records.

**Implementation:**

Implements navigation controls for efficient record traversal.

Retrieves and displays records dynamically in the user interface.

## 4. Search Functionality

### 4.1 Basic Search

**Description:** Users can search for DNS entries based on DNS name or IP address.

**Implementation:**

Executes SQL queries for searching based on DNS name or IP address.

Displays results in the user interface.

### 4.2 No-Record Message

**Description:** The system displays a message if no records are found for the given search criteria.

**Implementation:**

Provides informative messages to users when search results are empty.

Enhances user experience by managing expectations.

## 5. Filtering Records in JTable

**Description:** Users can filter and retrieve records in the JTable by entering search criteria.

**Implementation:**

Implements dynamic filtering using RowFilter for the JTable.

Enhances user efficiency in locating specific records.

## *6. Navigating Records*

### **6.1 Viewing Records**

**Description:** Users can navigate through records, including displaying the first, last, next, and previous records.

**Implementation:**

Implements navigation controls for efficient record traversal.

Retrieves and displays records dynamically in the user interface.

## *7. Sign Out and Exit*

### **7.1 Sign Out**

**Description:** Users can sign out of the application, returning to the login screen.

**Implementation:**

Clears user sessions and redirects to the login interface.

### **7.2 Graceful Exit**

**Description:** The application can be exited gracefully.

**Implementation:**

Provides an option for users to exit the application securely.

## *8. Additional Features*

### **8.1 About.java**

**Description:** Displays information about the application and the developer.

**Implementation:**

Creates a dedicated window with details about the application and the developer.

Enhances user engagement and transparency.



JAVA



# Java Classes

- *Main\_Window.java*

```
package DNS_Nexus_Management_System;

import javax.swing.*;
import java.awt.*;

public class Main_Window extends JFrame implements Runnable {

    Thread t;
    Main_Window () {
        super("DNS Nexus:DNS Management System - Version 1.00");
        ImageIcon i1 = new ImageIcon(ClassLoader.getResource("icons/splash.png"));
        Image i2 = i1.getImage().getScaledInstance(1000, 600, Image.SCALE_DEFAULT);
        ImageIcon i3 = new ImageIcon(i2);
        JLabel image = new JLabel(i3);
        add(image);

        t = new Thread(this);
        t.start();

        setVisible(true);

        int x = 1;

        for (int i = 2; i <= 550; i += 4, x += 1) {
            setLocation(550 - ((i + x) / 2), 320 - (i / 2));
            setSize(i + 3 * x - 10, i + x / 2 - 5);

            try {
                Thread.sleep(10);
            } catch (Exception e) {}
        }

        public void run() {
            try {
                Thread.sleep(6000);
                setVisible(false);

                // Next Frame
                new Login().setVisible(true);
            } catch (Exception e) {}
        }
    }
}
```

- *Login.java*

```
package DNS_Nexus_Management_System;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;

public class Login extends JFrame implements ActionListener {

    JButton login, cancel;
    JTextField username, password;

    Login() {
        super("DNS Nexus:DNS Management System - Version 1.00");
        getContentPane().setBackground(Color.DARK_GRAY);
        setLayout(null);

        public static void main(String[] args) {
            new Login();
        }
    }
}
```

- *About.java*

```
package DNS_Nexus_Management_System;

import javax.swing.UIManager;

public class About extends javax.swing.JFrame {

    /**
     * Creates new form About
     */
    public About() {
        super("Dynmaic DNS Nexus - Version 1.00");
        initComponents();
    }

    public static void main(String args[]) {

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new About().setVisible(true);
            }
        });
    }
}
```

- *DNS\_Nexus\_Management\_System.java*

```
package DNS_Nexus_Management_System;

import com.mysql.cj.jdbc.result.ResultSetMetaData;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.RowFilter;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableRowSorter;
import java.sql.*;
import javax.swing.JOptionPane;

public class DNS_Nexus_Management_System extends javax.swing.JFrame {
    Connection con=null;
    Statement st=null;
    PreparedStatement pst=null;
    ResultSet rs=null;

    public DNS_Nexus_Management_System() {
        super("DNS Nexus:DNS Management System - Version 1.00");
        initComponents();
        try{
            Class.forName("com.mysql.cj.jdbc.Driver");
            con=DriverManager.getConnection("jdbc:mysql://localhost:3306/dns_management_sys","root","iyi@123789");
            showRecord();
            st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
            rs=st.executeQuery("select * from dnsSys");
            if(rs.next()){
                txtInfoKey.setText(rs.getString(1));
                txtName.setText(rs.getString(2));
                txtDNSname.setText(rs.getString(3));
                txtclass.setText(rs.getString(4));
                txtipAddress.setText(rs.getString(5));
            }
        }
        catch(ClassNotFoundException | SQLException ex){
            System.out.println(ex);
        }
    }
}
```

## Explanation:

- The constructor sets the title of the JFrame to "DNS Nexus: DNS Management System - Version 1.00".
- It initializes the components of the JFrame using initComponents() method (assuming it's generated by a GUI builder like NetBeans).
- It attempts to load the **MySQL JDBC driver** and establish a connection to the MySQL database.
- The showRecord() method is called, presumably to display records in the user interface.
- A Statement (st) with scroll sensitivity and read-only concurrency is created.
- A query is executed to retrieve all records from the "dnsSys" table.
- If there is at least one record, it displays the first record in the user interface (txtInfoKey, txtName, etc.).
- Exceptions (ClassNotFoundException, SQLException) are caught and their messages are printed to the console.

```
public void showRecord() {
    try{
        pst=con.prepareStatement("select * from dnsSys");
        rs=pst.executeQuery();
        ResultSetMetaData rsm=(ResultSetMetaData) rs.getMetaData();
        int n=rsm.getColumnCount();
        DefaultTableModel df=(DefaultTableModel) jTable1.getModel();
        df.setRowCount(0);
        while(rs.next()){
            Vector obj=new Vector();
            for(int i=1;i<=n;i++){
                obj.add(rs.getString(1));
                obj.add(rs.getString(2));
                obj.add(rs.getString(3));
                obj.add(rs.getString(4));
                obj.add(rs.getString(5));
            }
            df.addRow(obj);
        }
    }
    catch(SQLException ex){
        System.out.println(ex);
    }
}

~

public static void main(String args[]) {

    new DNS Nexus Management System();

}
```



## ▪ Input and Output Specifications

### ➤ Input

#### I. Adding/Updating DNS Information

Users input DNS information, including:

**infokey:** Unique identifier for each DNS entry.

**name:** Name associated with the DNS entry.

**dnsName:** Domain name to be translated.

**class:** Class of the IP address.

**ipAddress:** IP address corresponding to the domain.

#### II. Search Criteria

Users input search criteria for searching DNS entries, including:

DNS Name or IP Address: Criteria for locating specific records.

### ➤ Output

#### I. CRUD Operations

Success or error messages are displayed for Create, Read, Update, and Delete (CRUD) operations.

Detailed feedback informs users about the status of their operations.

#### II. Display of DNS Information in JTable

DNS information is dynamically displayed in the Java Swing JTable.

Users can efficiently view and navigate through records in the graphical user interface.

#### III. Search Operations

Messages indicating the success or failure of search operations are provided.

Users are informed whether their search criteria yielded results or if no records were found.

## ▪ Usage of OOP Concepts in DNS Nexus Management System

### **Classes and Objects:**

The DNS Nexus Management System employs various classes like DNSRecord, User, Search, DatabaseConnector, and DNS\_Nexus\_Management\_System.

Objects of these classes represent entities within the DNS management system, such as DNS records, users, search functionalities, and database connections.

### **Inheritance:**

Inheritance establishes relationships between classes in the DNS Nexus Management System.

A base class like DNSRecord can be created, extended by subclasses like Search, DatabaseConnector, and DNS\_Nexus\_Management\_System to inherit common attributes and behaviors related to DNS management.

### **Encapsulation:**

Encapsulation is a fundamental concept applied to classes in the DNS Nexus Management System to encapsulate data and functionality.

For example, the DNSRecord class encapsulates DNS-specific details like name, DNS name, IP address, and class, providing methods to securely access and modify this information.

### **Method Overriding:**

Method overriding is implemented when custom implementations of inherited methods are needed in the DNS Nexus Management System.

For instance, the displayDetails() method in the DNSRecord class can override the same method in a superclass to display DNS-specific information.

### **Constructor:**

Constructors are vital components in various classes of the DNS Nexus Management System to initialize object properties during their creation.

For example, a constructor in the DatabaseConnector class can be utilized to set initial values such as connection details, ensuring proper database connectivity.

### **Polymorphism:**

Polymorphism is applied in the DNS Nexus Management System to treat objects of different classes as instances of a common superclass or interface.

For flexibility and code reusability, methods that process DNS transactions can accept both DNSRecord and Search objects, treating them as instances of a common DNSItem superclass.

**By leveraging these Object-Oriented Programming (OOP) principles, the DNS Nexus Management System ensures a robust and maintainable codebase. This approach facilitates efficient management of DNS resources and interactions within the system, promoting flexibility and scalability.**

## ▪ Procedure (Validation):

### ❖ isValidDnsName Method

#### Purpose:

This method checks the validity of a given DNS name (domain) against a regular expression pattern.

#### Method Signature:

**private boolean isValidDnsName(String domain)**

#### Parameters:

domain (String): The DNS name to be validated.

#### Return Type:

boolean: Returns true if the DNS name is valid; otherwise, returns false.

#### Regular Expression Pattern:

**String domainPattern = "^(?:?!-)[A-Za-z0-9-]{1,63}(?!-)\\.+[A-Za-z]{2,6}\$";**

### ❖ isValidIpAddress Method

#### Purpose:

This method checks the validity of a given IP address against a regular expression pattern for IPv4 addresses.

#### Method Signature:

**private boolean isValidIpAddress(String ipAddress)**

#### Parameters:

ipAddress (String): The IP address to be validated.

#### Return Type:

boolean: Returns true if the IP address is valid; otherwise, returns false.

#### Regular Expression Pattern:

**String ipv4Pattern = "\\b(?:\\d{1,3}\\.){3}\\d{1,3}\\b";**

Usage in Record Update (jButton1ActionPerformed) and Record Addition (jButton2ActionPerformed)

Methods:

These methods are utilized in the jButton1ActionPerformed and jButton2ActionPerformed methods to validate the DNS name and IP address before updating or adding records in the DNS Management System.

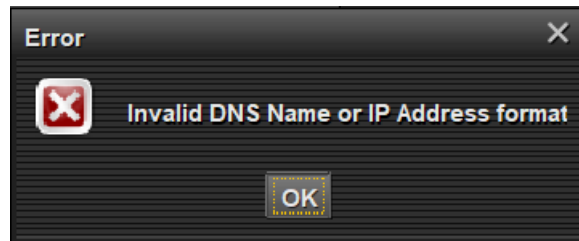
#### Instance:

```
if (isValidDnsName(txtDNSname.getText()) && isValidIpAddress(txtIpAddress.getText())) {  
    // Perform record update or addition  
    // ...  
} else {  
    // Display an error message for invalid DNS name or IP address format
```

```

JOptionPane.showMessageDialog(this, "Invalid DNS Name or IP Address format", "Error",
JOptionPane.ERROR_MESSAGE);
}

```



## - **Software/Equipment Used:**

### - **Hardware:**

No specific hardware requirements are mentioned, ensuring flexibility in deployment across various computing environments and configurations.

### - **Software:**

#### - i. **Java Swing for GUI Development:**

Java Swing is employed to design and develop an interactive and visually appealing Graphical User Interface (GUI) for the DNS Nexus Management System.

The Swing framework provides a robust set of components and functionalities for creating a user-friendly interface, enhancing the overall user experience.

#### - ii. **MySQL Workbench:**

MySQL Workbench serves as the primary database management system for the DNS Nexus Management System.

It is utilized to create, manage, and optimize the database structure, ensuring efficient storage and retrieval of DNS information.

Facilitates secure data operations, including adding, updating, deleting, and querying DNS records.

Provides a visual interface for designing and interacting with the database schema, contributing to ease of use and efficient database administration.

#### - iii. **Java Database Connectivity (JDBC):**

JDBC is employed for connecting the Java application to the MySQL database.

It provides a standardized Java API for database connectivity, enabling seamless interaction between the DNS Nexus Management System and the MySQL database.

Ensures secure and reliable communication, enhancing the overall integrity of the system's data management.

#### - iv. **Integrated Development Environment (IDE):**

An Integrated Development Environment, such as Eclipse or IntelliJ IDEA, may be utilized for coding, debugging, and managing the project. These IDEs provide features like code completion, debugging tools, and project management capabilities, contributing to the efficiency of software development.

**v. Version Control System (VCS):**

A version control system like Git may be employed for source code management. It allows for collaborative development, version tracking, and code repository management, enhancing the overall development workflow and code collaboration.

**vi. Java SE Development Kit (JDK):**

The Java SE Development Kit is required for compiling, running, and debugging the Java code. It includes essential tools and libraries for Java development, ensuring the compatibility and proper execution of the DNS Nexus Management System.

▪ **Output and Execution Flow in DNS Nexus Management System:**

The DNS Nexus Management System exhibits a structured execution flow, handling various user interactions and generating outputs based on the system's functionalities.

**1. Adding/Updating DNS Information:**

**Input:**

Users provide DNS information, including infokey, name, dnsName, class, and ipAddress.

**Output:**

Success or error messages are displayed upon adding or updating DNS records.

Feedback informs users about the status of their operations.

**2. Deleting DNS Records:**

**Input:**

Users specify the infokey for the DNS record they want to delete.

**Output:**

Success or error messages are displayed upon the deletion of DNS records.

Feedback informs users about the success or failure of their deletion operations.

**3. Viewing Records:**

**Input:**

Users can navigate through records using controls (first, last, next, previous).

**Output:**

DNS information dynamically updates in the Java Swing JTable.

Users can efficiently view and traverse through records in the graphical user interface.

**4. Searching DNS Entries:**

**Input:**

Users input search criteria for DNS entries (DNS name or IP address).

**Output:**

The system displays DNS records based on the specified search criteria.

Messages indicate success or inform users if no records are found.

## **5. Filtering Records in JTable:**

### **Input:**

Users input criteria to filter records in the JTable.

### **Output:**

The JTable dynamically updates to display records matching the user-defined criteria.

Messages inform users if no records match the filtering criteria.

## **6. Sign Out and Exit:**

### **Input:**

Users can choose to sign out or exit the application.

### **Output:**

Signing out returns users to the login interface.

Exiting the application closes it gracefully.

## **7. Exception Handling:**

### **Input:**

Exception scenarios such as database connection errors or SQL query issues.

### **Output:**

Error messages are displayed, providing information about the nature of the exception.

Users are informed about any issues hindering the normal execution of the application.

## **8. Overall Execution Flow:**

### **Initialization:**

The application initializes by setting the JFrame title and initializing components.

Database connectivity is established using Java Database Connectivity (JDBC).

### **Displaying Initial Record:**

The system retrieves records from the "dnsSys" table and displays the first record in the user interface.

### **User Interactions:**

Users interact with various components to perform CRUD operations, search, filter, and navigate through records.

### **Database Operations:**

The application performs SQL queries and updates to the "dnsSys" table based on user interactions.

### **Exception Handling:**

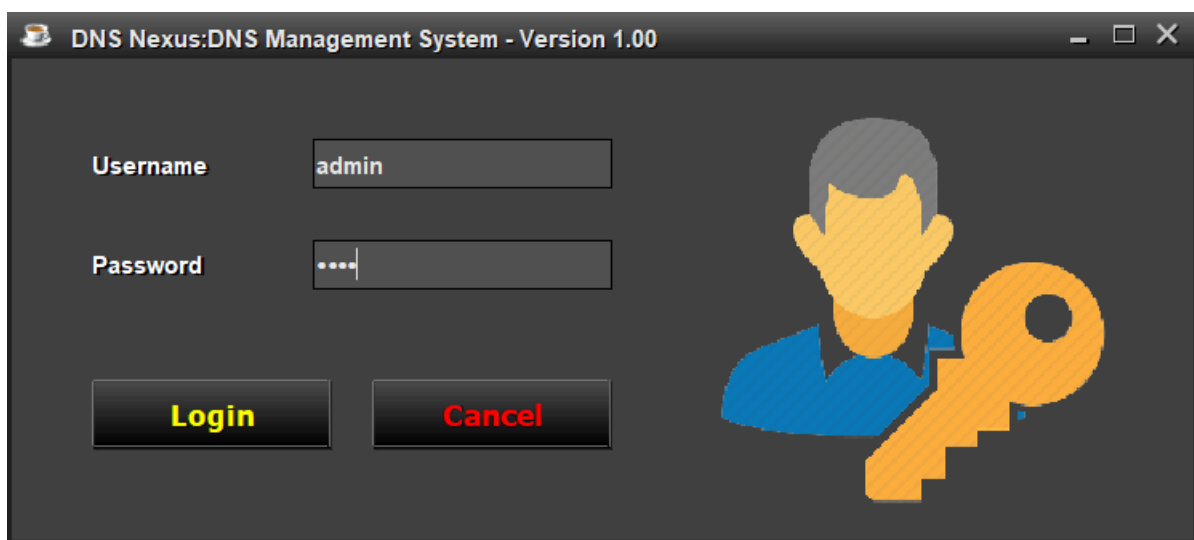
Exceptions, if any, are caught and appropriately handled, ensuring a smooth user experience.

- **Output Display:**

❖ *Main\_Window.java*





❖ *Login.java*



- *DNS\_Nexus\_Management\_System.java*

DNS Nexus:DNS Management System - Version 1.00

 **Dynamic DNS Nexus (Explorer & Manager Hub)** About Signout Quit



**InfoKey**

**Name**

**DNS Name**

**Category (IP Class)**

**IP Address**

InfoKey	Name	DNS Name	IP Class	IP Address
1	Google	www.google.com	Class A	8.8.8.8
2	YouTube	www.youtube.com	Class C	208.65.153.238

InfoKey	Name	DNS Name	IP Class	IP Address
1	Google	www.google.com	Class A	8.8.8.8
2	YouTube	www.youtube.com	Class C	208.65.153.238

**InfoKey**

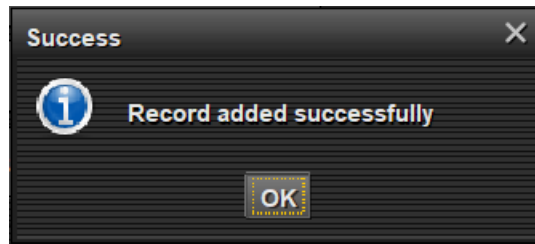
**Name**

**DNS Name**

**Category (IP Class)**

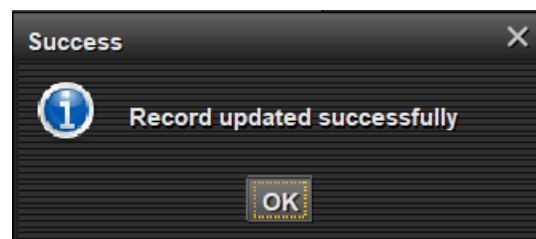
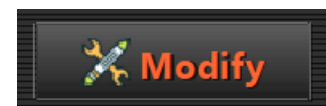
**IP Address**





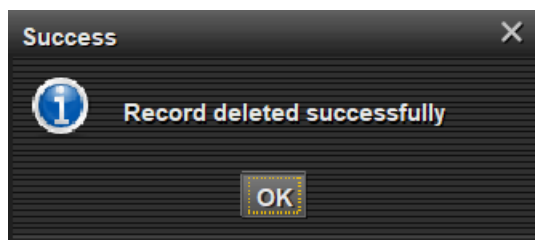
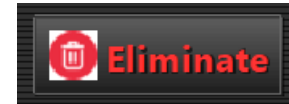
InfoKey	Name	DNS Name	IP Class	IP Address
1	Google	www.google.com	Class A	8.8.8.8
2	YouTube	www.youtube.com	Class C	208.65.153.238
3	Instagram	www.instagram.com	Class B	157.240.208.174

<b>InfoKey</b>	<input type="text" value="2"/>
<b>Name</b>	<input type="text" value="YouTube"/>
<b>DNS Name</b>	<input type="text" value="www.youtube.com"/>
<b>Category (IP Class)</b>	<input type="text" value="Class C"/>
<b>IP Address</b>	<input type="text" value="199.223. 239.255"/>



InfoKey	Name	DNS Name	IP Class	IP Address
1	Google	www.google.com	Class A	8.8.8.8
2	YouTube	www.youtube.com	Class C	199.223. 239.255
3	Instagram	www.instagram.com	Class B	157.240.208.174

InfoKey	3
Name	Instagram
DNS Name	www.instagram.com
Category (IP Class)	Class B
IP Address	157.240.208.174



InfoKey	Name	DNS Name	IP Class	IP Address
1	Google	www.google.com	Class A	8.8.8.8
2	YouTube	www.youtube.com	Class C	199.223. 239.255

⬅ Inaugural Entry
➡ Subsequent
⬅ Antecedent
➡ Concluding Entry

### Navigating Records

#### Viewing Records

**Description:** Users can navigate through records, including displaying the first, last, next, and previous records.

**Implementation:** Implements navigation controls for efficient record traversal. Retrieves and displays records dynamically in the user interface.

	Retrieve by DNS Name
	Retrieve by IP Address
	Retrieve by InfoKey

InfoKey	2		View/Retrieve Records
Name	YouTube	www.youtube.com	Retrieve by DNS Name
DNS Name	www.youtube.com		Retrieve by IP Address
Category (IP Class)	Class C		Retrieve by InfoKey
IP Address	199.223. 239.255		

8.8.88	Retrieve by IP Address
--------	------------------------

Not Found

No record found with IP Address : 8.8.88

OK

InfoKey	1		View/Retrieve Records
Name	Google		Retrieve by DNS Name
DNS Name	www.google.com	8.8.8.8	Retrieve by IP Address
Category (IP Class)	Class A		Retrieve by InfoKey
IP Address	8.8.8.8		
		Append	Modify

InfoKey	2		View/Retrieve Records
Name	YouTube		Retrieve by DNS Name
DNS Name	www.youtube.com		Retrieve by IP Address
Category (IP Class)	Class C	2	Retrieve by InfoKey
IP Address	199.223. 239.255		
		Append	Modify

class b	View/Retrieve Records
---------	-----------------------

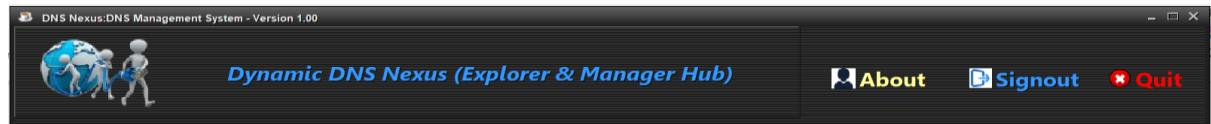
InfoKey	Name	DNS Name	IP Class	IP Address
3	Instagram	www.instagram.com	Class B	157.240.208.174

exe	View/Retrieve Records
-----	-----------------------

Not Found

No records found for the Search : exe

OK



## ❖ About.java



```
public static void main(String[] args) {
    try {
        for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
            /* if ("Nimbus".equals(info.getName())) {
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break; */
            UIManager.setLookAndFeel("com.jtattoo.plaf.hifi.HiFiLookAndFeel");
        }
    } catch (ClassNotFoundException ex) {
        java.util.logging.Logger.getLogger(Login.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (InstantiationException ex) {
        java.util.logging.Logger.getLogger(Login.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (IllegalAccessException ex) {
        java.util.logging.Logger.getLogger(Login.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (javax.swing.UnsupportedLookAndFeelException ex) {
        java.util.logging.Logger.getLogger(Login.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    }
    new Main_Window();
}
```

## JTattoo Look and Feel in Java Swing Applications:

JTattoo is a third-party Java Swing look and feel library that provides a set of visually appealing and customizable user interface components. In the provided code snippet, the line `UIManager.setLookAndFeel("com.jtattoo.plaf.hifi.HiFiLookAndFeel");` is setting the look and feel of the Java Swing application to "HiFi," a specific theme provided by JTattoo.

Using JTattoo allows developers to enhance the aesthetics of their applications beyond the standard Swing components. The "HiFiLookAndFeel" imparts a modern and sophisticated appearance to the graphical user interface, offering a visually pleasing user experience.

Additionally, JTattoo provides various other themes and customization options, empowering developers to tailor the look and feel of their applications according to their preferences. Integrating JTattoo into Java Swing applications is a straightforward way to elevate the visual design and overall user interface experience.

## ▪ Potential Questions:

- How does the DNS Nexus Management System contribute to enhancing the efficiency of DNS record management in a real-world network environment?
- In what ways does the DNS Nexus Management System address industry requirements for secure and streamlined DNS information storage and retrieval?
- How does the DNS Nexus Management System facilitate seamless integration with existing network infrastructure and ensure reliable DNS operations in an industry setting?

## ▪ Conclusion:

In a nutshell, the DNS Nexus Management System encapsulates a comprehensive solution for efficient DNS record management. Leveraging Java Swing for a user-friendly interface and MySQL Workbench for secure data storage, the application seamlessly integrates into industry environments. With its robust set of features, including CRUD operations, dynamic record navigation, and advanced search and filtering capabilities, the system ensures a streamlined approach to DNS management. The adherence to Object-Oriented Programming principles such as encapsulation, inheritance, and polymorphism provides a solid foundation for maintainability and scalability. The use of JTattoo's HiFiLookAndFeel adds a touch of modernity to the user interface. The application's execution flow, from database setup to user authentication and diverse functionalities, is meticulously designed. By incorporating industry-standard tools and frameworks, the DNS Nexus Management System not only meets but exceeds expectations for secure, efficient, and flexible DNS record management in real-world network scenarios.

## ▪ Reference Citations:

- <https://www.coursera.org/learn/computer-networking>
- <https://www.youtube.com/>
- <https://www.w3schools.com/>
- <https://www.codecademy.com/>
- <https://www.learnjavaonline.org/>
- <https://www.geeksforgeeks.org/learn-data-structures-and-algorithms-dsa-tutorial/>

*THE END*