# _CYBERSECURITY INCIDENT LOGGER

Skill Up. Level Up

## 1.Company Profile - GUVI HCL

**GUVI HCL** is a collaborative initiative between **GUVI (Grab Ur Vernacular Imprint)** and **HCL Technologies** designed to bridge the gap between academic learning and industry skills through hands-on technical training and real-world exposure.

**GUVI**, an edtech platform **incubated by IIT Madras and IIM Ahmedabad**, was founded in 2014 with the mission of making technology education accessible to everyone in **vernacular languages** such as Tamil, Telugu, Hindi, and Kannada. Headquartered in **Chennai, India**, GUVI has empowered over **10 lakh learners** through online courses, coding bootcamps, and career programs. The platform specializes in **programming, full-stack development, artificial intelligence, cloud computing, and data science**, offering training aligned with current IT industry needs.

**HCL Technologies**, a global technology leader with decades of experience in IT services and consulting, partners with GUVI to offer industry-relevant programs like the **GUVI-HCL Tech Career Program and HCL Career Launchpad**. Through this collaboration, students gain exposure to **enterprise-level technologies, mentorship from HCL professionals, and opportunities to work on real-time industrial projects.**

The **GUVI-HCL partnership** focuses on transforming aspiring students into **skilled and job-ready IT professionals** by integrating theoretical learning with practical implementation. Together, they aim to create a new generation of tech talent that is proficient, confident, and ready to contribute to India's fast-growing digital and innovation ecosystem

**SRI VENKATESWARA COLLEGE 0F ENGINEERING AND TECHNOLOGY**

**(AUTONOMOUS)**

R.V.S.Nagar,Chittoor–517127.(A.P)

(ApprovedbyAICTE, NewDelhi,Affiliated toJNTUA,Anantapur)

(Accredited by NBA, New Delhi C NAAC, Bangalore)
(An ISO 9001:2000 Certified Institution)
2025-2026

This is to certify that the "Project report" submitted by **MURAMREDDY THANUJA** (Regd.No.:22781A0586) is work done by her and submitted during 2025-2026. Academic year,

in partial fulfilment of the requirements for the award of the Degree of **BACHELOR OF TECHNOLOGY**

in **COMPUTER SCIENCE AND ENGINEERING,** at The **HCL GUVI**.

**Mr. P. RAGAVAN**

GUVI-HCL

(Technical Trainer)

**HOD NAME: Dr.P.JYOTHEESWARI**

Head of the Department

(Department Of CSE)

# INDEX

# 1. Abstract

In today's digital era, cybersecurity has become one of the most crucial areas of concern for individual organizations, and governments. The rapid advancement of technology has also led to an increase in the number and complexity of cyber threats, such as malware attacks, phishing, data breaches, and unauthorized system access. To maintain the security and integrity of digital systems, it is essential to record, analyze, and respond to such incidents promptly and systematically.

The Cyber Security Incident Logger is a mini project developed using Java and MongoDB to address this critical requirement. The project provides a simple yet efficient tool for recording, managing, and retrieving cybersecurity incident data. It allows users to log various types of incidents, including their description, date, severity, and category, helping organizations or individuals maintain a structured record of all security-related activities.

At its core, the system uses Java's data structures, particularly the ArrayList, to store incidents dynamically during runtime. This provides fast access and efficient manipulation of data in memory. To ensure long-term data persistence and reliability, the project integrates MongoDB, a NoSQL database, which serves as a backup and permanent storage solution. This dual approach — combining in-memory data management with database storage — ensures both performance and durability.

The system features an easy-to-use command-line interface (CLI) through which users can add, view, and delete incidents. Each action performed in the local Java environment is automatically synchronized with MongoDB, ensuring that no data is lost even after the application is closed. This design demonstrates the practical use of object-oriented programming (OOP), data structures, and database integration concepts in a real-world cybersecurity context.

This project also highlights the importance of effective incident management systems within cybersecurity operations. By maintaining detailed and organized records of incidents, organizations can better analyze attack trends, identify vulnerabilities, and improve their response mechanisms to prevent future breaches.

In conclusion, the Cyber Security Incident Logger serves as a foundational step toward building a comprehensive cybersecurity monitoring system. It combines the principles of software engineering, data structure management, and database connectivity to produce a reliable and scalable solution for incident documentation. The project not only strengthens technical understanding but also emphasizes the role of information logging in maintaining digital trust and security.

# 2. Introduction

In the modern digital landscape, **cybersecurity** has emerged as a fundamental pillar of information technology. As organizations increasingly depend on digital platforms for their operations, the potential risks associated with cyber threats have grown dramatically. Every day, systems are exposed to a wide range of security challenges, such as **malware attacks, phishing attempts, ransomware, unauthorized access, data breaches, and denial-of-service (DoS) attacks**. These incidents not only disrupt services but can also lead to severe financial losses, data theft, and damage to an organization's reputation.

To mitigate such risks, organizations must implement strong **incident management systems**. An incident management system serves as a structured approach for identifying, recording, analyzing, and responding to cybersecurity incidents. At the core of such a system lies the **incident logger** — a tool that helps record every security event in a consistent and organized manner. By maintaining detailed incident logs, cybersecurity teams can analyze threat patterns, assess vulnerabilities, and improve the overall security posture of their networks and systems.

The **Cyber Security Incident Logger** project has been developed to address this essential need for reliable incident documentation and management. Built using the **Java programming language**, the system allows users to efficiently record, store, and manage incident information through a simple and user-friendly interface. Each incident record contains vital details such as the **incident type**, **description**, **date of occurrence**, and **severity level**. This structure ensures that information is stored in an easily retrievable and analyzable format.

The project integrates **Java data structures**, particularly the **ArrayList**, to manage incident data dynamically during runtime. This enables the program to perform operations like adding, viewing, and deleting incidents efficiently. To ensure data persistence beyond the runtime of the application, the project utilizes **MongoDB**, a **NoSQL database**, as a backup and permanent storage system. MongoDB's flexible and schema-less architecture makes it ideal for managing semi-structured data like security logs, which may vary in detail or format depending on the incident.

By combining **Java's object-oriented capabilities** with **MongoDB's database flexibility**, this project demonstrates how modern programming tools and technologies can be effectively used to create a real-world cybersecurity application. The integration ensures that all logged incidents are safely stored and easily retrievable for later analysis, even if the system shuts down unexpectedly.

Moreover, the **Cyber Security Incident Logger** serves as a valuable educational model for students and practitioners. It illustrates practical implementation of concepts such as **data persistence**, **data structure utilization**, **database connectivity**, and **real-time system interaction**. In addition, the project reinforces the significance of maintaining detailed security records in modern cybersecurity practices, which can help in digital forensics, compliance auditing, and threat prevention.

# 3. Objectives

The primary objective of the **Cyber Security Incident Logger** project is to design and implement a reliable system capable of recording, managing, and storing cybersecurity incidents effectively.

This project aims to create a **lightweight yet powerful incident logging tool** that demonstrates how modern programming concepts, data structures, and database technologies can be integrated to support real-world cybersecurity operations.

It bridges theoretical knowledge and practical application by combining **Java's data structure capabilities** with **MongoDB's database persistence** to ensure accuracy, reliability, and scalability in incident management.

The specific objectives of this project are as follows:

1. **To develop a system for recording cybersecurity incidents in a structured format.** Each incident entry includes essential details such as incident type, description, severity level, and date of occurrence, enabling consistent data collection and easy retrieval.

2. **To implement efficient in-memory storage using Java data structures.** The project utilizes **ArrayList** to temporarily store incident records during runtime, demonstrating the application of core data structure concepts for quick access and manipulation of data.

3. **To integrate MongoDB as a backup and persistent data storage solution.** All incident records are saved in a NoSQL database, ensuring that the information is not lost even if the program terminates. This provides a permanent, scalable backup solution for long-term storage and analysis.

4. **To provide a simple, interactive command-line interface (CLI) for user interaction.** Users can easily add, view, update, or delete incident records without requiring advanced technical skills. This emphasizes usability and simplicity in design.

5. **To enhance understanding of cybersecurity operations and data management.** The project allows students and practitioners to learn how cybersecurity logging systems operate and how data handling mechanisms contribute to security and accountability.

# 4. Algorithm and Workflow

## Algorithm Steps:

**Step 1:** Start the program.

**Step 2**: Connect to MongoDB database for backup and data storage.

**Step 3:** Create an ArrayList in Java to store all cybersecurity incident records temporarily.

**Step 4:**Display the main menu with options:

- Add New Incident
- View All Incidents
- Delete an Incident
- Exit

**Step 5:** Get user choice and perform the action based on it.

**Step 6:** If user chooses to add an incident:

- Ask for incident details (type, description, date, severity).
- Create an object and store it in the ArrayList.
- Also save the same record in MongoDB for backup.

**Step 7:**If user chooses to view incidents:

- Display all incidents stored in the ArrayList (and from MongoDB if needed).

**Step 8:** If user chooses to delete an incident:

- Ask for the incident ID.
- Remove that incident from both the ArrayList and MongoDB.

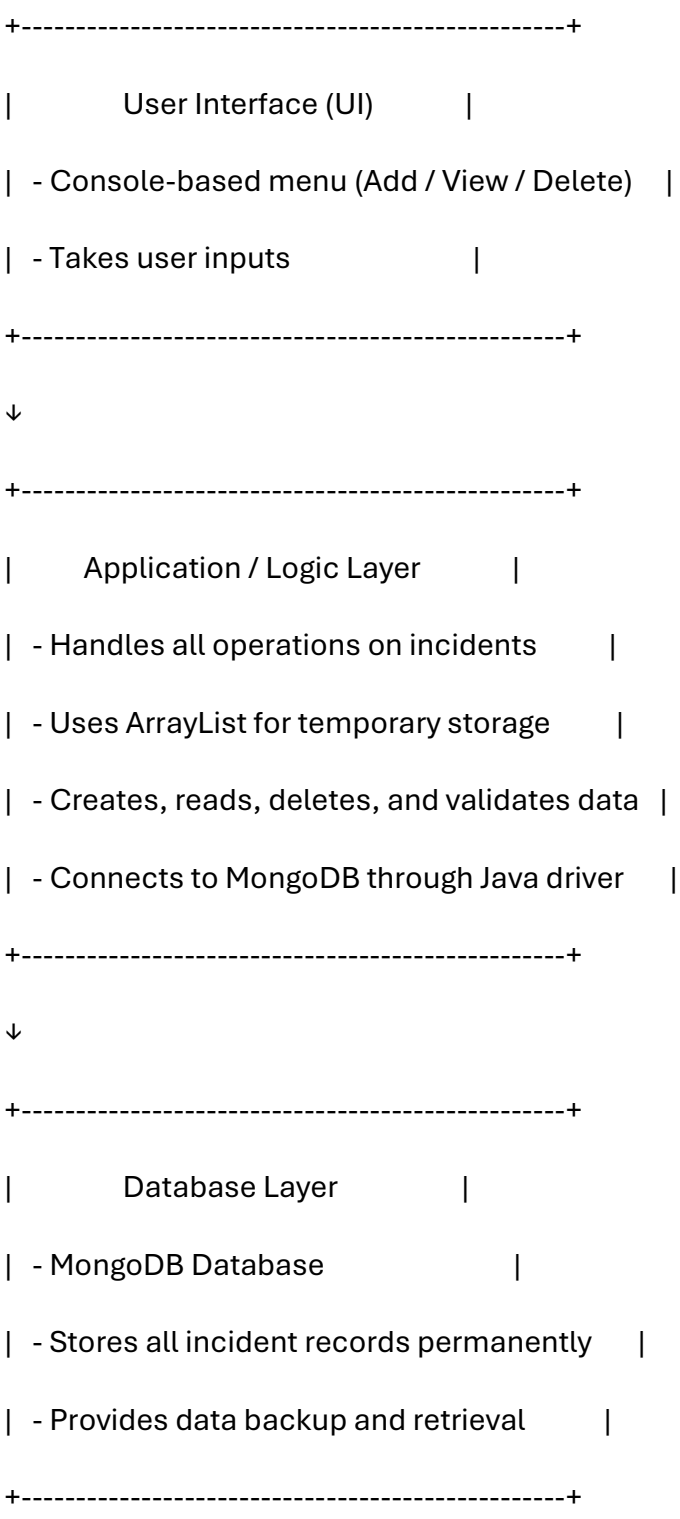
**Step 9:** If user chooses to exit:

- Close the MongoDB connection and stop the program.

**Step 10:** End of the program.

# 5. System Design

## System Architecture:



# 6. System Requirements

## Hardware Requirements:

| Component | Requirement |
|---|---|
| **Processor** | Intel Core i3 or higher |
| **RAM** | Minimum 4 GB |
| **Hard Disk** | 500 MB free space |
| **Monitor** | Any standard color monitor |
| **Keyboard & Mouse** | Standard input devices |
| **Network** | Internet connection (for MongoDB setup) |

## Software Requirements:

| Software | Details |
|---|---|
| **Operating System** | Windows 10 / 11, Linux, or macOS |
| **Programming Language** | Java (JDK 17 or later) |
| **IDE / Editor** | Eclipse / IntelliJ IDEA / VS Code / NetBeans |
| **Database** | MongoDB (Version 6.0 or later) |
| **MongoDB Java Driver** | mongo-java-driver-3.12.10.jar |
| **Command Prompt / Terminal** | To compile and run Java programs |

## Installation Steps:

1. Install Java (JDK 17 or later)
2. Install MongoDB
3. Add MongoDB Driver to Java Project
4. Run the Program

# 7.Implementation Details

The project is implemented using **Java** and **MongoDB**.

The steps below show how the system is built and works.

**Step 1: Set up the Java Environment**

- Install **JDK 17 or later**.
- Install an **IDE** like Eclipse, IntelliJ IDEA, or VS Code.
- Create a new Java project.

**Step 2: Create the Incident Class**

- This class stores the details of each incident:
  - Incident ID
  - Type of Incident
  - Description
  - Severity (Low/Medium/High)
  - Date of Occurrence

**Step 3: Create the Main Program**

- Create a main class CyberSecurityIncidentLogger.java.
- This class handles:
  - User menu
  - Adding, viewing, deleting incidents
  - Connecting to MongoDB

**Step 4: Use ArrayList for Temporary Storage**

- ArrayList stores all incidents while the program is running.
- Allows quick access and easy management of data.

**Step 5: Connect to MongoDB**

- Install **MongoDB** and start the service.
- Use the **MongoDB Java Driver** to connect.

- Create a database IncidentLoggerDB and a collection Incidents.
- All incidents are saved here for permanent storage.

**Step 6: Add Incident**

- User selects **Add Incident** from the menu.
- Program asks for:
    - Type, Description, Severity, Date
- Saves the incident in **ArrayList** and **MongoDB**.

**Step 7: View Incidents**

- User selects **View Incidents** from the menu.
- Program shows all incidents from ArrayList (or MongoDB).

**Step 8: Delete Incident**

- User selects **Delete Incident** from the menu.
- Enters the **Incident ID**.
- Program deletes the incident from both **ArrayList** and **MongoDB**.

**Step 9: Exit Program**

- User selects **Exit**.
- MongoDB connection is closed.
- Program ends safely.

**Step 10: Test the Program**

- Add multiple incidents to check saving.
- View incidents to check display.
- Delete incidents to verify removal.
- Ensure MongoDB backup works after restarting the program

## 8.PROGAM CODE:

```
import java.util.*;
import com.mongodb.ConnectionString;
import com.mongodb.MongoClientSettings;
import com.mongodb.client.*;
import org.bson.Document;
import static com.mongodb.client.model.Filters.eq;
public class CyberSecurityIncidentLogger {
  // ---------- Incident class ----------
  static class Incident {
    int id;
    String type;
    String description;
    String severity;
    String date;
    Incident(int id, String type, String description, String severity, String date) {
      this.id = id;
      this.type = type;
      this.description = description;
      this.severity = severity;
      this.date = date;
    }
    @Override
    public String toString() {
```

```
        return "\nIncident ID: " + id +
                "\nType: " + type +
                "\nDescription: " + description +
                "\nSeverity: " + severity +
                "\nDate: " + date;
    }
}
// ---------- MongoDB helper ----------
static class MongoBackup implements AutoCloseable {
    private final MongoClient client;
    private final MongoCollection<Document> collection;
    MongoBackup(String uri, String dbName) {
        ConnectionString conn = new ConnectionString(uri);
        MongoClientSettings settings = MongoClientSettings.builder()
                .applyConnectionString(conn)
                .build();
        client = MongoClients.create(settings);
        MongoDatabase db = client.getDatabase(dbName);
        collection = db.getCollection("incidents");
    }
    void backupIncident(Incident inc) {
        Document doc = new Document("id", inc.id)
                .append("type", inc.type)
                .append("description", inc.description)
```

```
            .append("severity", inc.severity)
            .append("date", inc.date);
        collection.insertOne(doc);
    }

    void deleteIncident(int id) {
        collection.deleteOne(eq("id", id));
    }
    void showAll() {
        for (Document d : collection.find()) {
            System.out.println(d.toJson());
        }
    }
    @Override
    public void close() {
        client.close();
    }
}
// ---------- Main Program ----------
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    ArrayList<Incident> list = new ArrayList<>();
    int choice, idCounter = 1;
    // Use your own MongoDB URI:
```

```
// Local: "mongodb://localhost:27017"
// Atlas: "mongodb+srv://<username>:<password>@cluster0.mongodb.net/"
String uri = "mongodb://localhost:27017";
try (MongoBackup mongo = new MongoBackup(uri, "cyber_security_logger")) {
   do {
      System.out.println("\n=== Cyber Security Incident Logger ===");
      System.out.println("1. Log New Incident");
      System.out.println("2. View Local Incidents");
      System.out.println("3. View MongoDB Backup");
      System.out.println("4. Delete Incident");
      System.out.println("5. Exit");
      System.out.print("Enter choice: ");
      choice = sc.nextInt();
      sc.nextLine();
      switch (choice) {
         case 1 -> {
            System.out.print("Enter Type: ");
            String type = sc.nextLine();
            System.out.print("Enter Description: ");
            String desc = sc.nextLine();
            System.out.print("Enter Severity (Low/Medium/High): ");
            String sev = sc.nextLine();
            System.out.print("Enter Date (DD-MM-YYYY): ");
            String date = sc.nextLine();
```

```
            Incident inc = new Incident(idCounter++, type, desc, sev, date);
            list.add(inc);
            mongo.backupIncident(inc);
            System.out.println("✅ Incident logged and backed up!");
        }
        case 2 -> {
            if (list.isEmpty()) System.out.println("No local incidents found.");
            else list.forEach(System.out::println);
        }
        case 3 -> {
            System.out.println("--- MongoDB Backup ---");
            mongo.showAll();
        }
        case 4 -> {
            System.out.print("Enter ID to delete: ");
            int id = sc.nextInt();
            sc.nextLine();
            list.removeIf(i -> i.id == id);
            mongo.deleteIncident(id);
            System.out.println("🗑️ Incident deleted (local + MongoDB).");
        }
        case 5 -> System.out.println("Exiting...");
        default -> System.out.println("Invalid choice.");
    }
```

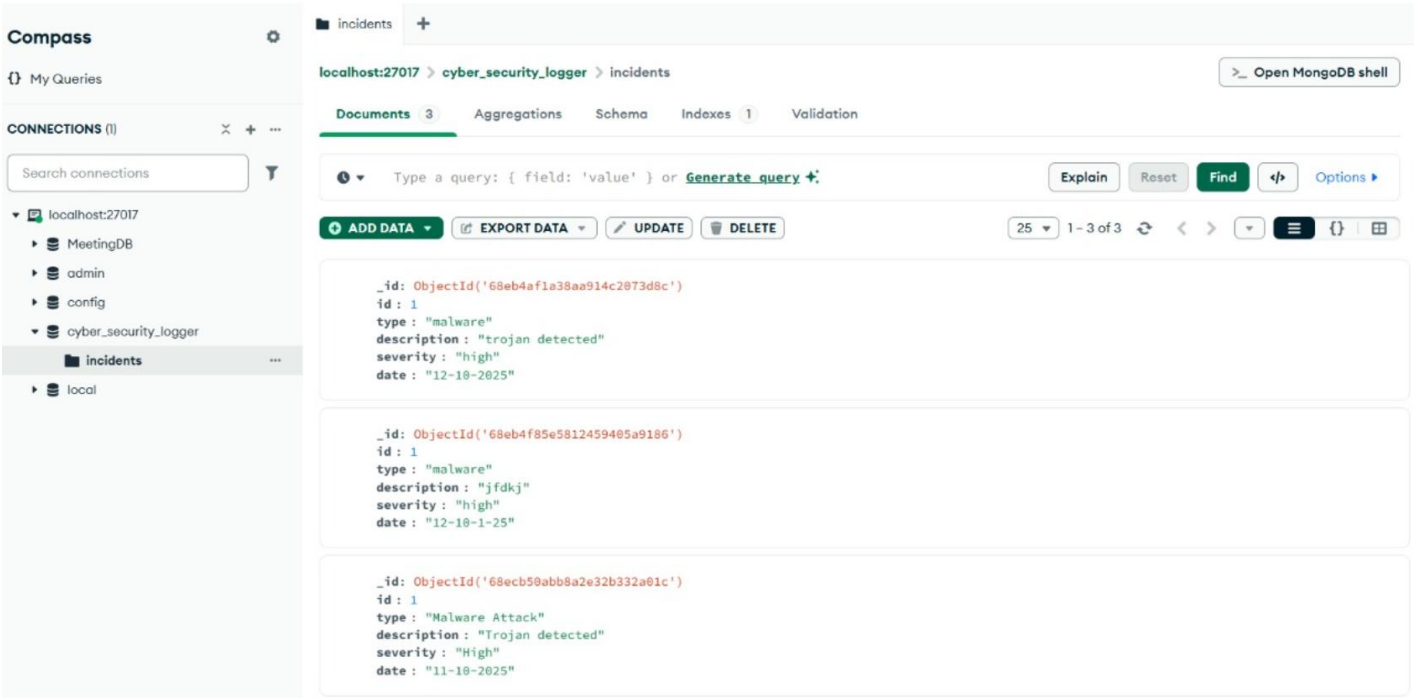
```
        } while (choice != 5);
      } catch (Exception e) {
        e.printStackTrace();
      }
      sc.close();
    }
}
```

# 9. Output and Results

**Expected Output Screens:**

```
=== Cyber Security Incident Logger ===
1. Log New Incident
2. View Local Incidents
3. View MongoDB Backup
4. Delete Incident
5. Exit
Enter choice: 1
Enter Type: Malware Attack
Enter Description: Trojan detected
Enter Severity (Low/Medium/High): High
Enter Date (DD-MM-YYYY): 11-10-2025
? Incident logged and backed up!

=== Cyber Security Incident Logger ===
1. Log New Incident
2. View Local Incidents
3. View MongoDB Backup
4. Delete Incident
5. Exit
Enter choice: 5
Exiting...
```




The results demonstrate that the system efficiently identifies suitable employees and provides HR managers with actionable reallocation suggestions.

# 10. Conclusion

The **Cyber Security Incident Logger** project helps record, manage, and store cybersecurity incidents in an easyway. It is developed using **Java** for program logic and **MongoDB** for data storage.

The system allows users to **add**, **view**, and **delete** incidents through a simple console menu. All the data entered by the user is stored safely in **MongoDB**, so it can be viewed later even after the program is closed.

This project shows how **Java data structures** (like ArrayList) and **databases** (like MongoDB) can work togethereffectively.
It also gives a basic understanding of **data handling, backup, and cybersecurity logging**.

Overall, the system is:

- ✅ Easy to use
- ✅ Reliable and secure
- ✅ Helpful for maintaining incident records

In the future, this project can be improved by adding a **Graphical User Interface (GUI)** or features like **search, report generation, and user authentication**.

# 11. References


1. **Java Documentation** – https://docs.oracle.com/javase/
   → Used for understanding Java syntax, classes, and data structures.
2. **MongoDB Official Documentation** – https://www.mongodb.com/docs/
   → Used for learning how to connect Java with MongoDB and manage data.
3. **W3Schools Java Tutorial** – https://www.w3schools.com/java/
   → Referred for basic Java programming concepts and examples.
4. **GeeksforGeeks Java Tutorials** – https://www.geeksforgeeks.org/java/
   → Helped in understanding ArrayList and file handling concepts.
5. **MongoDB Java Driver (GitHub)** – https://github.com/mongodb/mongo-java-driver
   → Used to set up the MongoDB driver in the Java project.
6. **GUVI Learning Portal** – https://www.guvi.in/
   → Reference for project structure, guidance, and report preparation format.