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 OF ENGINEERING AND TECHNOLOGY

(AUTONOMOUS)

R.V.S. Nagar, Chittoor–517127, (A.P)

(Approved by AICTE, New Delhi, Affiliated to JNTUA, 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 **VEERAPALLI JOSHNA** Regd.No.:**22781A04K0,** of the **Electronics and Communication Engineering Department**, has been carried out as an original work done by her during the Academic Year 2025-2026 as part of the **Java Training Program** Conducted by **GUVI HCL.**

Mr. P. Ragavan	**Dr .D. Srihari**
Technical Trainer	Head of the Department
GUVI HCL	ECE

QUALITY ASSURANCE DASHBOARD

ABSTRACT

The Quality Assurance (QA) Dashboard is a software tool designed to monitor, analyze, and visualize the quality of software products throughout the development process. This project aims to provide a centralized platform for tracking key quality metrics such as defect rates, test execution results, and performance statistics. By using MongoDB as the database, the system efficiently stores and manages large volumes of testing and defect data in a scalable and flexible manner.

The implementation leverages various Data Structures and Algorithms (DSA) to enhance data processing speed, retrieval efficiency, and performance analysis. Features such as real-time data visualization, defect tracking, and test coverage reports help software teams make data-driven decisions to ensure continuous quality improvement.

This dashboard not only simplifies the monitoring of QA activities but also bridges the gap between testers, developers, and management by providing clear insights and visual analytics. Ultimately, the system contributes to improved software reliability, reduced testing time, and enhanced overall product quality.

INDEX

AIM

The aim of this project is to design and develop a Quality Assurance Dashboard that enables organizations to effectively monitor, measure, and manage the quality of their software products or processes. The dashboard provides a centralized platform to visualize key metrics related to testing, defects, and performance, helping in data-driven decision-making and continuous quality improvement.

The project focuses on integrating MongoDB as the database for storing and retrieving large volumes of quality-related data such as bug reports, test logs, and performance statistics in a flexible and scalable manner. Various data structures like arrays, linked lists, stacks, queues, and trees are used to optimize data handling, searching, and reporting operations, improving the performance and accuracy of the dashboard.

To store and manage QA data efficiently using MongoDB's document-based structure.

To apply data structures for faster processing and retrieval of testing and defect information.

To analyze and visualize key metrics such as test coverage, defect density, and project quality

To provide real-time insights for decision-making and continuous quality improvement.

To enhance communication and transparency between developers, testers, and managers.

ALGORITHM

Step 1: Start

Step 2: Initialize the system

- Load all required libraries, dependencies, and modules.
- Set up the environment for database connection and data visualization.

Step 3: Establish connection with MongoDB

- Connect to the MongoDB server.
- Create necessary collections such as *TestCases*, *Defects*, *PerformanceData*, and *Users*.

Step 4: Input and store quality assurance data

- Collect data from software testing processes (manual or automated).
- Store test results, defect reports, and performance metrics in MongoDB.
- Use suitable data structures for efficient data handling:
    - Arrays to store test case IDs and results.
    - Linked Lists to manage sequences of test executions.
    - Stacks and Queues for managing task flow and backtracking.

Step 5: Retrieve and process data

- Fetch required data from MongoDB collections.
- Apply algorithms for sorting, searching, and filtering data (e.g., sort defects by severity, search test results by ID).

Step 6: Analyze quality metrics

- Compute key performance indicators such as:
    - Defect Density = (Number of Defects / Total Test Cases) × 100
    - Test Coverage = (Executed Test Cases / Total Test Cases) × 100
    - Pass/Fail Ratio and Average Resolution Time
- Use data structures to store intermediate computation results efficiently.

Step 7: Generate dashboard visualization

- Display real-time data using charts and graphs (bar charts, pie charts, line graphs).

Step 8: Update dashboard dynamically

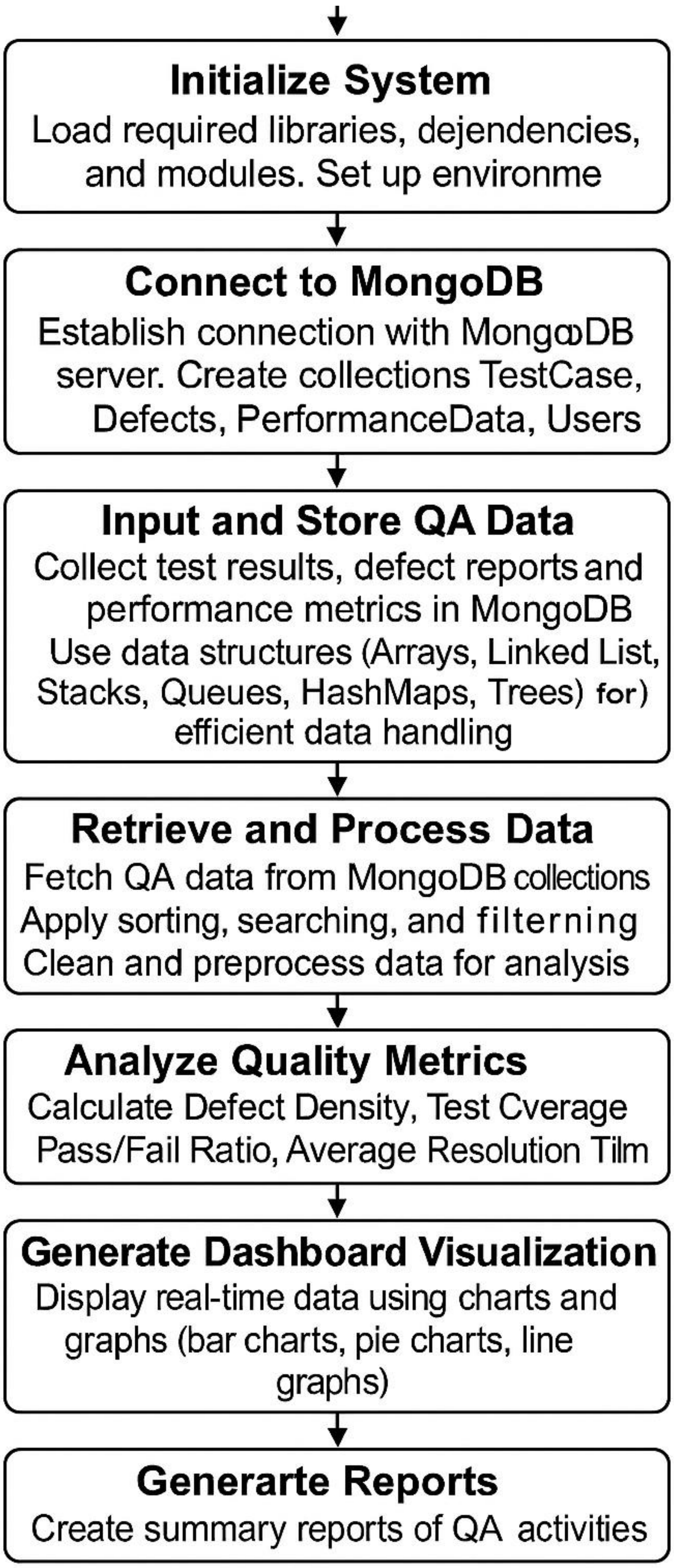
- Continuously monitor MongoDB for new or updated data.

Step 9: Generate reports

- Create summary reports of QA activities.
- Export analyzed data and visual reports in formats like PDF or Excel.

Step 10: End

FLOW CHART


Initialize System
Load required libraries, dejendencies, and modules. Set up environme
Connect to MongoDB
Establish connection with MongoDB server. Create collections TestCase, Defects, PerformanceData, Users
Input and Store QA Data
Collect test results, defect reports and performance metrics in MongoDB
Use data structures (Arrays, Linked List, Stacks, Queues, HashMaps, Trees) for) efficient data handling
Retrieve and Process Data
Fetch QA data from MongoDB collections
Apply sorting, searching, and filterning
Clean and preprocess data for analysis
Analyze Quality Metrics
Calculate Defect Density, Test Cverage Pass/Fail Ratio, Average Resolution Tilm
Generate Dashboard Visualization
Display real-time data using charts and graphs (bar charts, pie charts, line graphs)
Generarte Reports
Create summary reports of QA activities

SYSTEM REQUIREMENTS (SOFTWARE)

Component	Description
Operating System	Windows 10 / Windows 11 / Linux (Ubuntu)
Programming Language	Java (JDK 17 or later)
Database	MongoDB (Version 6.0 or later)
Integrated Development Environment	IntelliJ IDEA / Eclipse / NetBeans
Java Libraries Used	MongoDB Java Driver (org.mongodb.driver), BSON Library
Build Tool (Optional)	Maven or Gradle
Command Line Interface	For executing and interacting with the application
Text Editor	Notepad++ / VS Code (for script editing)
Version Control (Optional)	Git and GitHub for project management and collaboration
Java Runtime Environment (JRE)	Required for executing compiled Java classes
Driver Connection String	mongodb://localhost:27017 (for local MongoDB connection)

PROGRAM CODE

```
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.Updates;
import org.bson.Document;
import java.time.Instant;
import java.util.*;
import java.util.concurrent.atomic.AtomicInteger;
public class QADashboard {
    private static final String MONGO_URI = "mongodb://localhost:27017"; // change if needed
    private static final String DB_NAME = "qa_dashboard";
    private final Trie testCaseTrie = new Trie();
    private final PriorityQueue<ScheduledRun> runQueue = new PriorityQueue<>(); // ordered
by priority
    private final LinkedList<String> history = new LinkedList<>(); // execution history (most
recent at end)
    private final MongoClient mongoClient;
    private final MongoDatabase database;
    private final MongoCollection<Document> testCaseColl;
    private final MongoCollection<Document> defectColl;
    private final MongoCollection<Document> testRunColl;
    private final MongoCollection<Document> historyColl;
    private final Scanner sc = new Scanner(System.in);
    private final AtomicInteger idCounter = new AtomicInteger(1000);
    public QADashboard() {
        mongoClient = MongoClients.create(MONGO_URI);
        database = mongoClient.getDatabase(DB_NAME);
        testCaseColl = database.getCollection("testcases");
        defectColl = database.getCollection("defects");
        testRunColl = database.getCollection("testruns");
    historyColl = database.getCollection("history");
    loadDataToMemory();
}
private void loadDataToMemory() {
    // load test case names into trie for search
    for (Document d : testCaseColl.find()) {
        String name = d.getString("name");
        if (name != null) {
            testCaseTrie.insert(name.toLowerCase());
        }
    }
          for (Document d : historyColl.find()) {
        String entry = d.getString("entry");
        if (entry != null) {
            history.add(entry);
        }
    }
    // idCounter: try to set based on max ids
    int maxId = 1000;
    for (Document d : testCaseColl.find()) {
        Integer id = d.getInteger("id");
        if (id != null && id > maxId) {
            maxId = id;
        }
    }
    idCounter.set(maxId + 1);
}
public void close() {
    mongoClient.close();
}
static class TestCase {
    int id;
    String name;
    String description;
        boolean automated;
    Document toDoc() {
        return new Document("id", id)
                .append("name", name)
                .append("description", description)
                .append("priority", priority)
                .append("automated", automated);
    }
    static TestCase fromDoc(Document d) {
        TestCase t = new TestCase();
        t.id = d.getInteger("id");
        t.name = d.getString("name");
        t.description = d.getString("description");
        t.priority = d.getString("priority");
        t.automated = d.getBoolean("automated", false);
        return t;
    }
}
static class Defect {
    int id;
    int testCaseId;
    String title;
    Document toDoc() {
        return new Document("id", id)
                .append("testCaseId", testCaseId)
                .append("title", title)
                .append("severity", severity)
                .append("status", status);
    }
    static Defect fromDoc(Document d) {
        Defect def = new Defect();
        def.id = d.getInteger("id");
```

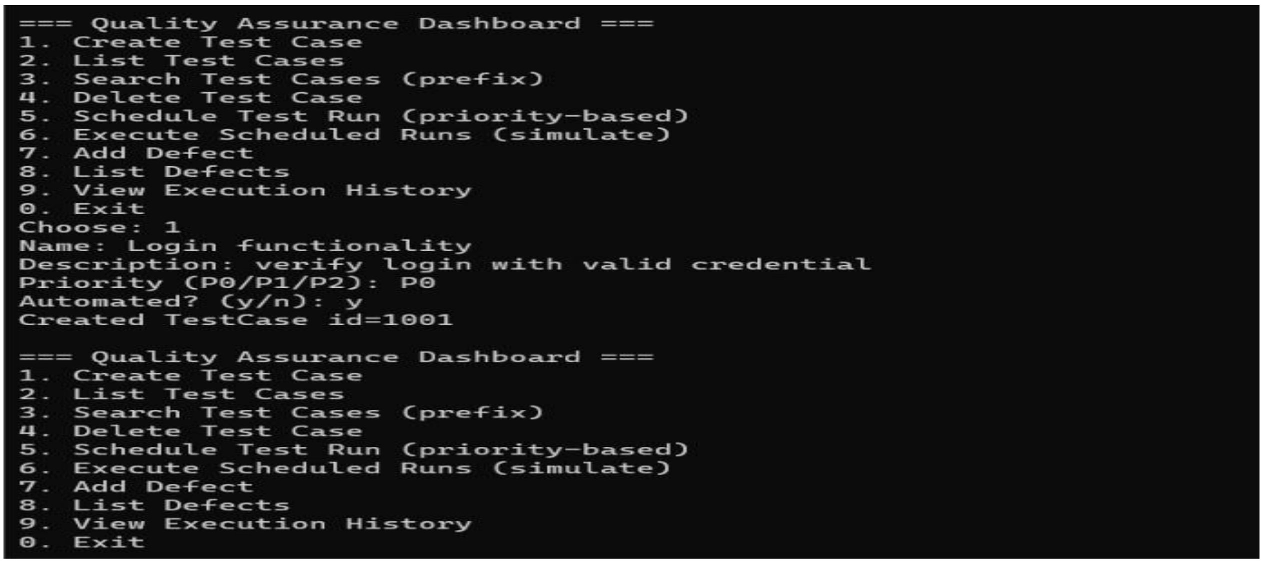
```
            def.testCaseId = d.getInteger("testCaseId");
            def.title = d.getString("title");
            def.severity = d.getString("severity");
            def.status = d.getString("status");
            return def;
        }
    }
    static class TestRunResult {
        int testCaseId;
        boolean passed;
        String notes;
        Document toDoc() {
            return new Document("testCaseId", testCaseId)
                    .append("passed", passed)
                    .append("notes", notes);
        }
    }
    static class ScheduledRun implements Comparable<ScheduledRun> {
        int runId;
        int testCaseId;
        int priority; // lower number => higher priority
        Instant scheduledAt;
        ScheduledRun(int runId, int testCaseId, int priority) {
            this.runId = runId;
            this.testCaseId = testCaseId;
            this.priority = priority;
            this.scheduledAt = Instant.now();
        }
              public int compareTo(ScheduledRun o) {
            int byPriority = Integer.compare(this.priority, o.priority);
            if (byPriority != 0) {
                return byPriority;
        }
        return this.scheduledAt.compareTo(o.scheduledAt);
    }
    Document toDoc() {
        return new Document("runId", runId)
                .append("testCaseId", testCaseId)
                .append("priority", priority)
                .append("scheduledAt", scheduledAt.toString());
    }
}
   static class TrieNode {
    Map<Character, TrieNode> children = new HashMap<>();
    boolean end = false;
}
static class Trie {
    private final TrieNode root = new TrieNode();
    void insert(String s) {
        TrieNode node = root;
        for (char c : s.toCharArray()) {
            node = node.children.computeIfAbsent(c, k -> new TrieNode());
        }
        node.end = true;
    }
    List<String> searchPrefix(String prefix) {
        List<String> results = new ArrayList<>();
        TrieNode node = root;
        for (char c : prefix.toCharArray()) {
            node = node.children.get(c);
            if (node == null) {
                return results;
            }
        }
        collect(node, new StringBuilder(prefix), results);
```

```
            return results;
           }
        private void collect(TrieNode node, StringBuilder prefix, List<String> results) {
            if (node.end) {
                results.add(prefix.toString());
            }
            for (Map.Entry<Character, TrieNode> e : node.children.entrySet()) {
                prefix.append(e.getKey());
                collect(e.getValue(), prefix, results);
                prefix.deleteCharAt(prefix.length() - 1);
            }
        }
    }
    public void runCLI() {
        boolean running = true;
        while (running) {
            printMenu();
            String choice = sc.nextLine().trim();
            switch (choice) {
                case "1":
                    createTestCase();
                    break;
                case "2":
                    listTestCases();
                    break;
                case "3":
                    searchTestCases();
                    break;
                case "4":
                    deleteTestCase();
                    break;
                case "5":
                    scheduleTestRun();
                break;
            case "6":
                executeScheduledRuns();
                break;
            case "7":
                addDefect();
                break;
            case "8":
                listDefects();
                break;
            case "9":
                viewHistory();
                break;
            case "0":
                running = false;
                break;
            default:
                System.out.println("Invalid option");
        }
    }
    close();
    System.out.println("Exiting QADashboard. Goodbye.");
}
private void printMenu() {
    System.out.println("\n=== Quality Assurance Dashboard ===");
    System.out.println("1. Create Test Case");
    System.out.println("2. List Test Cases");
    System.out.println("3. Search Test Cases (prefix)");
    System.out.println("4. Delete Test Case");
    System.out.println("5. Schedule Test Run (priority-based)");
    System.out.println("6. Execute Scheduled Runs (simulate)");
    System.out.println("7. Add Defect");
    System.out.println("8. List Defects");
```

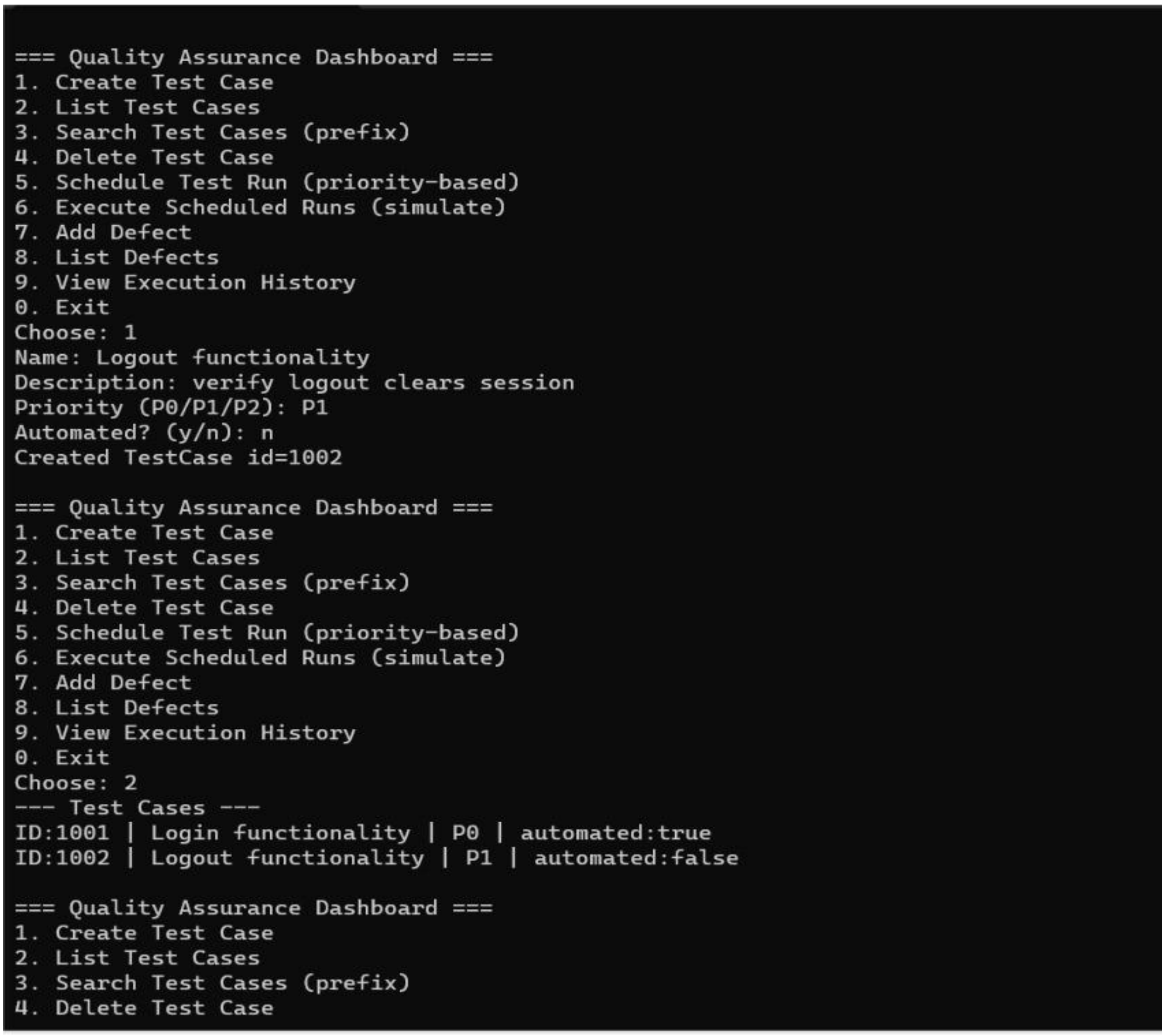
```
        System.out.println("9. View Execution History");
        System.out.println("0. Exit");
        System.out.print("Choose: ");
    }
        private void createTestCase() {
        TestCase t = new TestCase();
        t.id = idCounter.getAndIncrement();
        System.out.print("Name: ");
        t.name = sc.nextLine().trim();
        System.out.print("Description: ");
        t.description = sc.nextLine().trim();
        System.out.print("Priority (P0/P1/P2): ");
        t.priority = sc.nextLine().trim();
        System.out.print("Automated? (y/n): ");
        t.automated = sc.nextLine().trim().equalsIgnoreCase("y");
        testCaseColl.insertOne(t.toDoc());
        testCaseTrie.insert(t.name.toLowerCase());
        System.out.println("Created TestCase id=" + t.id);
    }
    private void listTestCases() {
        System.out.println("--- Test Cases ---");
        for (Document d : testCaseColl.find()) {
            TestCase t = TestCase.fromDoc(d);
            System.out.printf("ID:%d | %s | %s | automated:%s\n", t.id, t.name, t.priority,
t.automated);
        }
    }
    private void searchTestCases() {
        System.out.print("Prefix to search: ");
        String pre = sc.nextLine().trim().toLowerCase();
        List<String> names = testCaseTrie.searchPrefix(pre);
        if (names.isEmpty()) {
            System.out.println("No matches.");
            return;
        }
        System.out.println("Matches:");
        for (String name : names) {
            System.out.println(" - " + name);
        }
    }
    private void deleteTestCase() {
        System.out.print("TestCase ID to delete: ");
        int id = Integer.parseInt(sc.nextLine().trim());
        Document found = testCaseColl.find(Filters.eq("id", id)).first();
        if (found == null) {
            System.out.println("Not found");
            return;
        }
        testCaseColl.deleteOne(Filters.eq("id", id));
        // Note: trie cleanup omitted for simplicity (would require full rebuild)
        rebuildTrie();
        System.out.println("Deleted test case " + id);
    }
    private void rebuildTrie() {
        // rebuild trie from DB (simple and safe)
        testCaseTrie.root.children.clear();
        for (Document d : testCaseColl.find()) {
            String name = d.getString("name");
            if (name != null) {
                testCaseTrie.insert(name.toLowerCase());
            }
        }
    }
        private void scheduleTestRun() {
        System.out.print("TestCase ID to schedule: ");
        int tcId = Integer.parseInt(sc.nextLine().trim());
        Document d = testCaseColl.find(Filters.eq("id", tcId)).first();
        if (d == null) {
            System.out.println("TestCase not found");
          return;
        }
        System.out.print("Numeric priority (1 highest, 10 lowest): ");
        int priority = Integer.parseInt(sc.nextLine().trim());
        int runId = new Random().nextInt(1_000_000);
        ScheduledRun sr = new ScheduledRun(runId, tcId, priority);
        runQueue.offer(sr);
        testRunColl.insertOne(sr.toDoc());
        System.out.println("Scheduled run " + runId + " for testCase " + tcId);
    }
    private void executeScheduledRuns() {
        System.out.println("Executing scheduled runs in priority order...");
        while (!runQueue.isEmpty()) {
            ScheduledRun sr = runQueue.poll();
            System.out.println("Running testCaseId=" + sr.testCaseId + " (runId=" + sr.runId +
")");
                        boolean passed = new Random().nextBoolean();
            TestRunResult res = new TestRunResult();
            res.testCaseId = sr.testCaseId;
            res.passed = passed;
            res.notes = passed ? "Passed" : "Failed - check logs";
            Document runDoc = new Document("runId", sr.runId)
                    .append("testCaseId", sr.testCaseId)
                    .append("priority", sr.priority)
                    .append("executedAt", Instant.now().toString())
                    .append("result", res.toDoc());
            testRunColl.insertOne(runDoc);
            String hist = String.format("Run %d: testCase %d -> %s", sr.runId, sr.testCaseId,
res.passed ? "PASS" : "FAIL");
            history.add(hist);
            historyColl.insertOne(new Document("entry", hist).append("timestamp",
Instant.now().toString()));
            System.out.println(hist);
            // If failed, auto-create a defect (simple heuristic)
            if (!passed) {
                Defect def = new Defect();
                def.id = idCounter.getAndIncrement();
                def.testCaseId = sr.testCaseId;
                def.title = "Auto-generated defect for test " + sr.testCaseId;
                def.severity = (sr.priority <= 3) ? "Critical" : "Major";
                def.status = "Open";
                defectColl.insertOne(def.toDoc());
                System.out.println("Auto-created defect id=" + def.id + " severity=" + def.severity);
            }
        }
        System.out.println("All scheduled runs executed.");
    }
    private void addDefect() {
        Defect def = new Defect();
        def.id = idCounter.getAndIncrement();
        System.out.print("TestCase ID: ");
        def.testCaseId = Integer.parseInt(sc.nextLine().trim());
        System.out.print("Title: ");
        def.title = sc.nextLine().trim();
        System.out.print("Severity (Critical/Major/Minor): ");
        def.severity = sc.nextLine().trim();
        def.status = "Open";
        defectColl.insertOne(def.toDoc());
        System.out.println("Created defect " + def.id);
    }
    private void listDefects() {
        System.out.println("--- Defects ---");
        for (Document d : defectColl.find()) {
            Defect def = Defect.fromDoc(d);
            System.out.printf("ID:%d | TestCase:%d | %s | %s\n", def.id, def.testCaseId,
def.severity, def.status);
        }
    }
    private void viewHistory() {
        System.out.println("--- Execution History ---");
        int idx = 1;
        for (String h : history) {
            System.out.printf("%d. %s\n", idx++, h);
        }
    }
        public static void main(String[] args) {
        QADashboard app = new QADashboard();
        try {
            app.runCLI();
        } finally {
            app.close();
        }
    }
}
```

OUTPUT SCREENSHOTS

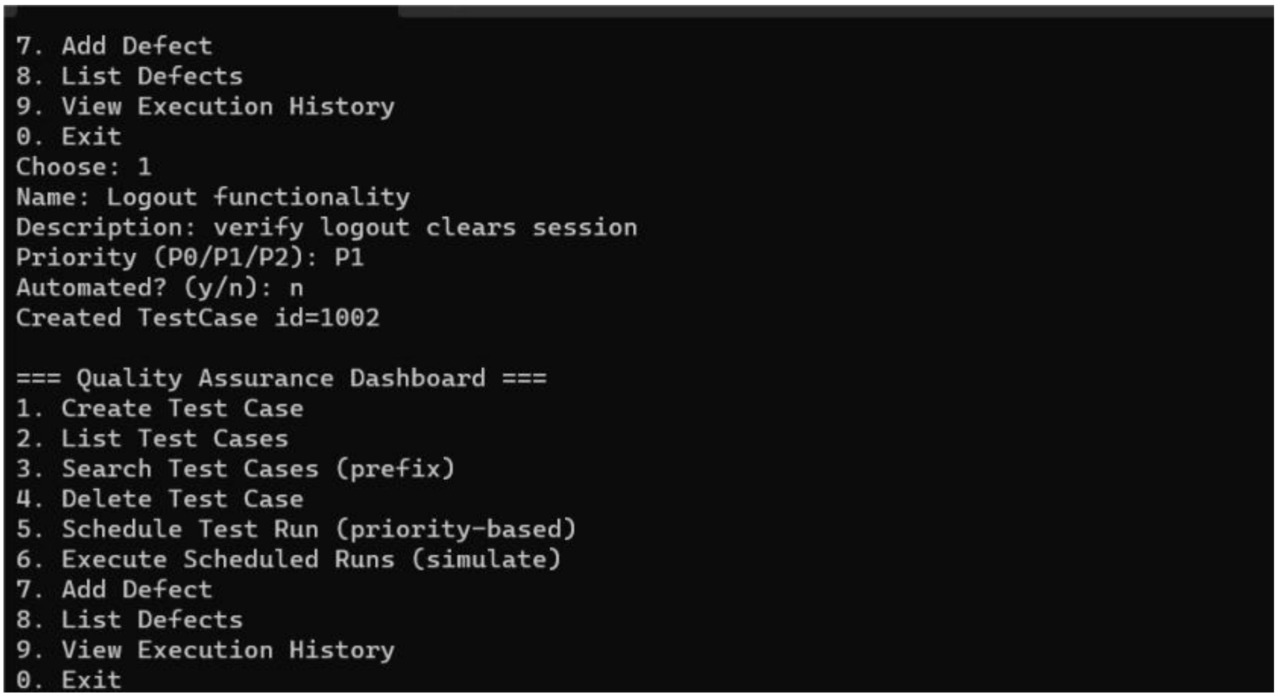
Command prompt outputs:



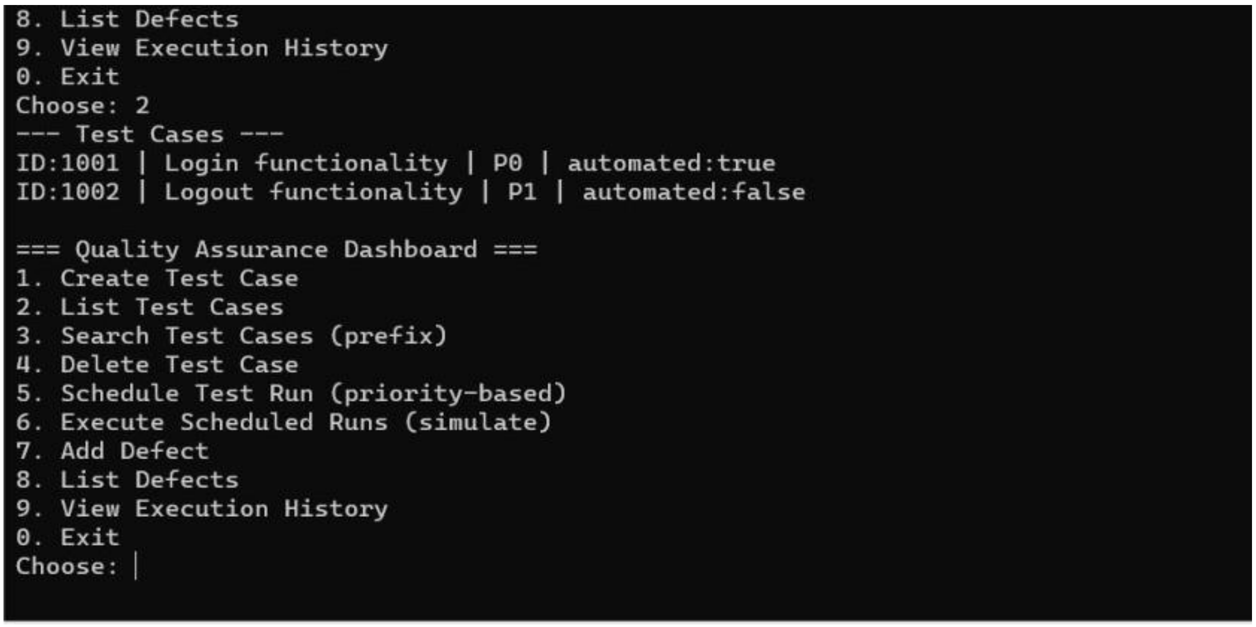
```
=== Quality Assurance Dashboard ===
1. Create Test Case
2. List Test Cases
3. Search Test Cases (prefix)
4. Delete Test Case
5. Schedule Test Run (priority-based)
6. Execute Scheduled Runs (simulate)
7. Add Defect
8. List Defects
9. View Execution History
0. Exit
Choose: 1
Name: Login functionality
Description: verify login with valid credential
Priority (P0/P1/P2): P0
Automated? (y/n): y
Created TestCase id=1001

=== Quality Assurance Dashboard ===
1. Create Test Case
2. List Test Cases
3. Search Test Cases (prefix)
4. Delete Test Case
5. Schedule Test Run (priority-based)
6. Execute Scheduled Runs (simulate)
7. Add Defect
8. List Defects
9. View Execution History
0. Exit
```



```
=== Quality Assurance Dashboard ===
1. Create Test Case
2. List Test Cases
3. Search Test Cases (prefix)
4. Delete Test Case
5. Schedule Test Run (priority-based)
6. Execute Scheduled Runs (simulate)
7. Add Defect
8. List Defects
9. View Execution History
0. Exit
Choose: 1
Name: Logout functionality
Description: verify logout clears session
Priority (P0/P1/P2): P1
Automated? (y/n): n
Created TestCase id=1002

=== Quality Assurance Dashboard ===
1. Create Test Case
2. List Test Cases
3. Search Test Cases (prefix)
4. Delete Test Case
5. Schedule Test Run (priority-based)
6. Execute Scheduled Runs (simulate)
7. Add Defect
8. List Defects
9. View Execution History
0. Exit
Choose: 2
--- Test Cases ---
ID:1001 | Login functionality | P0 | automated:true
ID:1002 | Logout functionality | P1 | automated:false

=== Quality Assurance Dashboard ===
1. Create Test Case
2. List Test Cases
3. Search Test Cases (prefix)
4. Delete Test Case
```

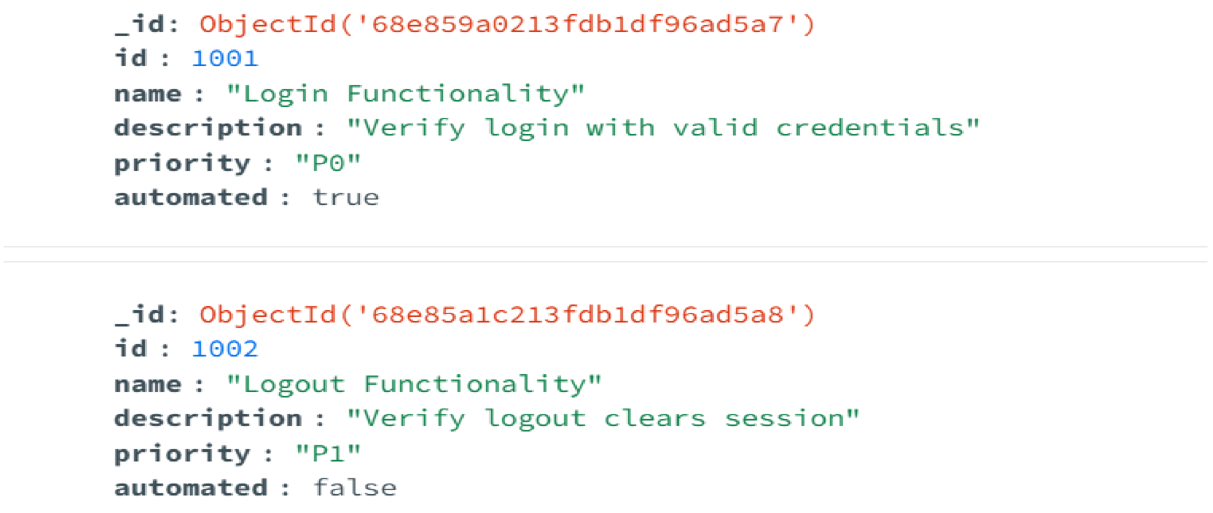


```
7. Add Defect
8. List Defects
9. View Execution History
0. Exit
Choose: 1
Name: Logout functionality
Description: verify logout clears session
Priority (P0/P1/P2): P1
Automated? (y/n): n
Created TestCase id=1002

=== Quality Assurance Dashboard ===
1. Create Test Case
2. List Test Cases
3. Search Test Cases (prefix)
4. Delete Test Case
5. Schedule Test Run (priority-based)
6. Execute Scheduled Runs (simulate)
7. Add Defect
8. List Defects
9. View Execution History
0. Exit
```



```
8. List Defects
9. View Execution History
0. Exit
Choose: 2
--- Test Cases ---
ID:1001 | Login functionality | P0 | automated:true
ID:1002 | Logout functionality | P1 | automated:false

=== Quality Assurance Dashboard ===
1. Create Test Case
2. List Test Cases
3. Search Test Cases (prefix)
4. Delete Test Case
5. Schedule Test Run (priority-based)
6. Execute Scheduled Runs (simulate)
7. Add Defect
8. List Defects
9. View Execution History
0. Exit
Choose: |
```

Database Output:



```
_id: ObjectId('68e859a0213fdb1df96ad5a7')
id : 1001
name : "Login Functionality"
description : "Verify login with valid credentials"
priority : "P0"
automated : true

_id: ObjectId('68e85a1c213fdb1df96ad5a8')
id : 1002
name : "Logout Functionality"
description : "Verify logout clears session"
priority : "P1"
automated : false
```

CONCLUSION

The Quality Assurance Dashboard project successfully provides an efficient and interactive platform for monitoring and managing software quality. By integrating MongoDB for data storage and applying Data Structures and Algorithms (DSA) for optimized data processing, the system ensures high performance, scalability, and accuracy in handling large sets of test and defect data.The dashboard offers real-time visualization of key quality metrics such as defect density, test coverage, and performance statistics, enabling teams to make data-driven decisions and improve the overall software development process. It also enhances communication between testers, developers, and project managers by presenting clear insights through visual reports.Overall, this project demonstrates how combining modern database technologies with efficient algorithms.

REFERENCES

1. **Oracle Java Documentation –** Java Platform, Standard Edition 17 API Specification.
   Available at : https://docs.oracle.com/javase/17/docs/api/
2. **MongoDB Documentation –** MongoDB Manual.
   Available at : https://www.mongodb.com/docs/
3. **Geeks for Geeks** – Data Structures and Algorithms in Java.
   Available at : https://www.geeksforgeeks.org/data-structures/
4. **GitHub** – Sample Java-MongoDB Projects and API Integration Examples.
   Available at : https://github.com/
5. **Tutorials Point** – Java and MongoDB Connectivity Guide.
   Available at : https://www.tutorialspoint.com/mongodb_with_java/index.htm