University of Science and Technology (USTC)

Department of Computer Science and Engineering (CSE)

Lab Project-Volunteer Management System

Course Title : **Object-Oriented Programming Lab**

Course Code: **CSE 124**

Submitted to:

Debabarata Mallick

Lecturer

CSE, FSET, USTC

Submitted By:

Name- Nargis Sultana Reshma

Roll-24070173

Registration:1214

Batch & Semester -43(B), 2nd.

Dept.- B.Sc(CSE)

Table of Contents

List of Figures

Proposal

The project aims to create a console-based Volunteer Management System that helps organizations efficiently manage volunteers and events. The system provides secure login capabilities for both administrators and volunteers, and allows volunteers to track tasks, give feedback, and view their event history. It is designed to simplify coordination and enhance communication between the organization and its volunteers.

Objective

- To develop a Java-based Volunteer Management System that enables efficient administration of volunteers, event planning, task assignments, and feedback collection, using Object-Oriented Programming principles.

Scope

The system will allow an admin to :

- Register volunteers
- Create and manage events
- Assign task
- View reports and logs

Volunteers will be able to:

- Log in securely
-
- View assigned tasks and event history
-
- Submit feedback
-

The system will be command-line-based with file storage, and designed for future expansion into GUI or web interfaces.

Problem Statement

- Organizations often face challenges in manually managing volunteers, tracking tasks, maintaining records, and organizing events. Paper-based or unstructured systems result in inefficiencies, errors, and data loss. A centralized software system can streamline the entire process.

Methodology

- The project is implemented using Java and follows Object-Oriented Programming (OOP) principles. Key classes include Admin, Volunteer, Event, and User. Secure login is handled with SHA-256 encryption. File handling is used to store and load persistent data. The system is modular, scalable, and test-driven.

System Design:

Class Name	Description	Key Methods/Features
User	Handles authentication and role identity	authenticate(), hashPassword()
Admin	Admin user who manages the system	registerVolunteer(), createEvent(), assignTask(), generateReport()

Volunteer	Stores volunteer data and actions	viewProfile(), viewTasks(), giveFeedback(), viewEventHistory()
Event	Holds event details and participants	addParticipant(), showEventDetails()

Figure 1

Implementation

Class Diagram

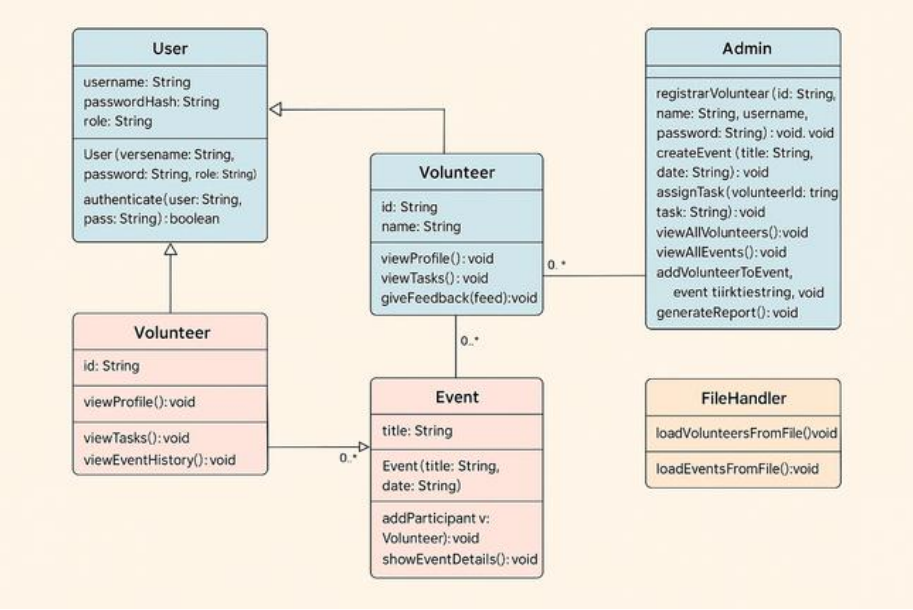

Figure 2

User Class

```
class User { String username, passwordHash,
role;

User(String username, String password,
String role) {
   this.username = username;
   this.passwordHash =
hashPassword(password);
   this.role = role;
}

boolean authenticate(String user, String
pass) {
   return username.equals(user) &&
passwordHash.equals(hashPassword(pass));
}

String hashPassword(String pass) {
   try {
      MessageDigest md = MessageDigest.
getInstance("SHA-256");
      byte[] hash = md.digest(pass.getBytes());
      StringBuilder hex = new StringBuilder();
      for (byte b : hash) {
         hex.append(String.format("%02x", b));
      }
      return hex.toString();
   } catch (NoSuchAlgorithmException e) {
      throw new RuntimeException(e);
   }
}
}
```

Figure 3

Output

```
--- Volunteer Management System ---
Enter username: admin
Enter password: 1234
Welcome, admin!
```

Figure 4

Volunteer Class

```
class Volunteer extends User {
    String name, id;
    List<String> tasks = new ArrayList<>();
List<String> eventHistory = new ArrayList<>();

Volunteer(String id, String name, String
username, String password) {
    super(username, password, "volunteer");
    this.id = id;
    this.name = name;
}

void viewProfile() {
    System.out.println("Volunteer ID: " + id);
    System.out.println("Name: " + name);
}

void viewTasks() {
    System.out.println("Tasks for " + name +
":");
    for (String task : tasks) System.out.println("-
 " + task);
}

void viewEventHistory() {
    System.out.println("Event History for " +
name + ":");
    for (String e : eventHistory) System.out.
println("- " + e);
}
```

+

```
class Volunteer extends User {
    String name, id;
    List<String> tasks = new ArrayList<>();
List<String> eventHistory = new ArrayList<>();

Volunteer(String id, String name, String
username, String password) {
    super(username, password, "volunteer");
    this.id = id;
    this.name = name;
}

void viewProfile() {
    System.out.println("Volunteer ID: " + id);
    System.out.println("Name: " + name);
}

void viewTasks() {
    System.out.println("Tasks for " + name +
":");
    for (String task : tasks) System.out.println("-
 " + task);
}

void viewEventHistory() {
    System.out.println("Event History for " +
name + ":");
    for (String e : eventHistory) System.out.
println("- " + e);
}
```

Figure 5

Output

Figure 6

Event Class

```
class Event {
String title, date;
List<String> participants = new ArrayList<>();

Event(String title, String date) {
   this.title = title;
   this.date = date;
}

void addParticipant(Volunteer v) {
   participants.add(v.name);
   v.eventHistory.add(title);
}

void showEventDetails() {
   System.out.println("Event: " + title + " | Date:
" + date);
   System.out.println("Participants:");
   for (String p : participants) System.out.
println(" - " + p);
}

void saveToFile() {
   try (FileWriter fw = new FileWriter("events.
txt", true)) {
      fw.write(title + "," + date + "\n");
   } catch (IOException e) {
      System.out.println("Error saving event: " +
 e);
   }
}
}
```

Figure 7

Output

```
Admin Dashboard:
1. Register Volunteer
2. View All Volunteers
3. Create Event
4. View All Events
5. Assign Task to Volunteer
6. Add Volunteer to Event
7. Generate Report
8. Logout
Select: 4
Event: Tree Plantation | Date: 02-06-2025
Participants:
- Tahrim
```

Figure 8

Volunteer Management System (Main Class)

```
public class VolunteerManagementSystem {
static Scanner scanner = new
Scanner(System.in); static Admin
defaultAdmin = new Admin("admin", "1234");

public static void main(String[] args) {
   Admin.users.add(defaultAdmin);
   loadVolunteersFromFile();
   loadEventsFromFile();

   while (true) {
      System.out.println("\n— Volunteer
Management System —");
      System.out.print("Enter username: ");
      String user = scanner.next();
      System.out.print("Enter password: ");
      String pass = scanner.next();

      User loginUser = null;
      for (User u : Admin.users) {
         if (u.authenticate(user, pass)) {
            loginUser = u;
            break;
         }
      }

      if (loginUser == null) {
         System.out.println("Invalid credentials.
");
         continue;
      }
```

```
      System.out.println("Welcome, " +
loginUser.username + "!");
      if (loginUser.role.equals("admin")) {
         adminMenu((Admin) loginUser);
      } else if (loginUser.role.
equals("volunteer")) {
         volunteerMenu((Volunteer) loginUser);
      }

      System.out.println("Logged out
successfully.");
   }
}
```

Figure 9

Output

Figure 10

Figure 11

Technologies Used

- Programming Language: Java (JDK 17)
- Framework: None (Core Java with OOP principles)
- Database: Text file storage (for now); planned upgrade to MySQL or SQLite

Use of OOP Concepts

- Encapsulation: Each class (e.g., Volunteer, Event) has its own data and methods, hiding internal details.
- Inheritance: Admin and Volunteer classes inherit from the base User class.
- Polymorphism: Login is handled through the User reference, allowing different behaviors based on role.
- Abstraction: Core functionalities (like authentication, logging, feedback) are implemented through well-defined classes and methods, hiding internal complexity.

Project Timeline:

Week 1: Requirement analysis and planning

Week 2: Class structure and user roles design

Week 3: Implementation of Admin and Volunteer classes

Week 4: File handling and secure login

Week 5: Feedback and report generation

Week 6: Testing, debugging, documentation

Main Functionalities:

- Admin login with SHA-256 password security
- Volunteer registration and authentication
- Create and view events
- Assign tasks to volunteers
- View all volunteers and events
- Volunteer dashboard: profile, tasks, event history, feedback
- Feedback submission stored in file
- Report generation summarizing system status

Requirements /Future Scope

Current Requirements:

- Java JDK 17
- IDE (VS Code or IntelliJ)
- CLI or terminal access
- File system access for saving/loading

Future Scope:

- GUI interface with JavaFX/Swing
- SQL database integration
- Email notification features
- Analytics dashboard

Conclusion

The Volunteer Management System demonstrates a complete Java-based CLI application that uses OOP, secure login, file I/O, and admin/volunteer roles. It meets core goals and is ready for GUI expansion.

Future Work

- Add a GUI using JavaFX or Swing.
- Use a database (e.g., MySQL) instead of text files.
- Develop a web or mobile version.
- Include email/SMS notifications.
- Add an analytics dashboard for tracking.
- Support admin password reset and multi-admin roles.
- Track volunteer performance and feedback.

References

- Oracle Java Documentation - https://docs.oracle.com/javase
- GeeksforGeeks Java OOP - https://www.geeksforgeeks.org/oops-concepts-in-java/
- W3Schools Java Tutorials - https://www.w3schools.com/java/
- ChatGPT - for content and technical enhancements
- DeepSeek AI - for class diagram formatting

Appendix

1. **Tools:** Java, VS Code, CLI, File I/O
2. **Concepts:** OOP, File Handling, Secure Login
3. **Data:** Volunteers, Events, Logs
4. **Limitations:** No GUI, text files only
5. **Future:** GUI, SQL, Web/Mobile version

Github link- https://github.com/R3shm/Volunteer-Management-System-