HOSTEL COMPLAINT MANAGEMENT SYSTEM

A PROJECT REPORT

Submitted by

NIVETHITHA S	**731122205036**
RAKAVI R	**731122205040**
RASIGA M	**731122205042**
SOWMIYA SK	**731122205053**

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

GOVERNMENT COLLEGE OF ENGINEERING

ERODE – 638316

ANNA UNIVERSITY: CHENNAI 600 025

OCTOBER 2025

BONAFIDE CERTIFICATE

Certified that this project report **"COMPLAINT MANAGEMENT SYSTEM"** is the bonafide work of **" Ms. NIVETHITHA S (731122205036), Ms. RASIGA M (731122205042), Ms. RAKAVI R (731122205040), SOWMIYA SK (731122205053)"** who carried out the project under my supervision.

SIGNATURE OF HOD	**SIGNATURE OF SUPERVISOR**
Dr I Bhuvaneshwari, M.E., Ph.D.,	**Dr SATHYAKALA M, M.E, Ph.D**
Assistant Professor (Senior)	Assistant Professor
HEAD OF THE DEPARTMENT	SUPERVISOR
Department of IT,	Department of IT,
Government College of Engineering,	Government College of Engineering,
Erode -638 316	Erode - 638 316

Submitted for the University Practical Examination on _______________ at the Government College of Engineering, Erode**.**

Internal Examiner **External Examiner**

TABLE OF CONTENTS

ABSTRACT

The Hostel Complaint Management System (HCMS) is a desktop-based application developed to streamline the process of reporting and resolving hostel-related issues. Built using Java and JavaFX for the frontend, with SQLite serving as the backend database and Maven as the build automation tool, the system offers a robust and user-friendly interface for students and administrators. It provides a digital platform for students to register complaints, monitor their progress, and receive timely updates, thereby reducing manual paperwork and enhancing overall efficiency. The primary objective of HCMS is to bridge the communication gap between students, hostel administrators, and maintenance staff, ensuring transparent and prompt resolution of issues related to maintenance, cleanliness, utilities, and other hostel facilities. Through this system, students can submit complaints via an interactive interface, while administrators can efficiently assign, track, and update the status of each complaint. Workers are provided with a dedicated dashboard and notification mechanism to manage assigned tasks and update progress in real time

The project was developed collaboratively by four students, each handling specific modules including complaint submission and interface design, administrative processing, worker dashboard and notifications, and database management with authentication and DevOps integration. A DevOps pipeline using GitHub Actions, Jenkins, and Docker automates build, testing, and deployment, ensuring continuous integration and consistent software quality. Overall, HCMS demonstrates the complete Software Development Life Cycle (SDLC) from analysis and design to implementation and deployment. The system provides an efficient, scalable, and maintainable solution for managing hostel operations, improving responsiveness, transparency, and accountability for all stakeholders.

CHAPTER - 1

1.1 Introduction

Hostel management involves overseeing student accommodations, facilities, and services, including the handling of complaints and maintenance requests. Traditionally, these complaints are managed manually through paper forms or informal communication, which often results in delays, inefficiencies, and lack of transparency. Manual tracking makes it difficult for administrators to prioritize tasks, monitor progress, and ensure accountability, while students may experience slow resolution of their issues. The challenges of paper-based systems highlight the need for a more structured and efficient approach to managing hostel complaints.

The Hostel Complaint Management System (HCMS) addresses these issues by providing a digital platform that allows students to submit complaints online, track their status, and receive notifications regarding progress. Administrators can assign tasks, monitor resolutions, and generate reports, while workers can update the status of their assigned tasks through a dedicated dashboard. By digitizing the complaint management process, HCMS improves efficiency, enhances transparency, and ensures better communication between students, administrators, and maintenance staff. The system's use of modern technologies such as Java, JavaFX, and SQLite, along with automated build and deployment pipelines, makes it scalable, robust, and maintainable, offering a practical solution for modern hostel operations.

1.2 Objectives

- Automate the submission and tracking of hostel complaints.
- Provide a simple and user-friendly interface using JavaFX.
- Enable administrators to efficiently assign complaints to workers.
- Maintain real-time updates of complaint status for students, administrators, and workers.
- Integrate DevOps tools such as GitHub Actions, Jenkins, and Docker for automated build, testing, and deployment.
- Improve overall transparency, accountability, and efficiency in hostel management operations.

1.3 Scope

- Designed for hostels within educational institutions.
- Includes separate modules for students, administrators, and workers.
- Students can submit complaints and track their progress.
- Administrators can assign tasks and monitor complaint resolutions.
- Workers can update complaint status through a dedicated dashboard.
- Modular and scalable design allows future extensions such as web or mobile interfaces.
- Improves communication, transparency, and overall efficiency inhostel management.

1.4 Tools & Technologies

- Frontend: JavaFX
- Backend: Core Java
- Database: SQLite
- Build Tool: Maven
- Version Control: Git & GitHub
- DevOps: Jenkins, Docker, GitHub Actions

CHAPTER – 2

LITERATURE REVIEW

2.1 Existing systems

- Hostel complaints are often managed using manual registers, paper forms, or spreadsheets.
- Lack of structured tracking leads to delays and mismanagement.
- Data integrity is frequently compromised, and reports are difficult to generate.
- Students are often unaware of complaint progress, causing frustration.
- Overall, existing systems are inefficient, unreliable, and time-consuming.

2.2 Proposed System Advantages

- Enables efficient complaint handling with real-time tracking.
- Ensures secure storage of complaints using SQLite.
- Provides role-based functionality for students, administrators, and workers.
- Integrates DevOps practices with automated build, test, and deployment via CI/CD pipelines.
- Improves communication, accountability, and operational efficiency in hostel management.

2.3 Technologies Chosen

- JavaFX for cross-platform, responsive graphical user interface.
- SQLite as a lightweight, local database for secure and easy data storage.
- Maven for dependency management and build automation.
- Ensures a robust, scalable, and maintainable system suitable for educational institutions.

CHAPTER – 3

SYSTEM ANALYSIS

3.1 System Requirements

Hardware:

- Minimum: 4GB RAM, 2GHz Processor, 200MB storage
- Recommended: 8GB RAM, SSD, Java-compatible system

Software:

- Java 17, Maven, SQLite, Jenkins, Docker, GitHub

3.2 Functional Requirements

- **Students can submit complaints:** Students can log in, submit complaints related to hostel facilities, and track their status.
- **Admins can assign and update status:** Admins can assign complaints to workers, update status, and monitor resolution progress.
- **Workers can view assigned complaints:** Workers can see complaints assigned to them, update progress, and mark tasks as completed.

3.3 Non-functional Requirements

- **Reliability and scalability:** The system should work consistently for multiple users and support future growth.
- **Easy deployment and maintenance:** The system should be simple to install, configure, and update, with automated build/deployment.
- **Secure data access:** Role-based access ensures students, admins, and workers access only relevant data, maintaining integrity and privacy.

CHAPTER – 4

SYSTEM ARCHITECTURE

MVC Pattern: separates UI, logic, and data.

Database Layer: Contains tables such as Complaint and User, storing all necessary data securely in SQLite.

Controller Layer: Handles logic and validation.

UI Layer: Built using JavaFX with FXML forms to provide an interactive and user-friendly interface.

4. Database Schema (SQLite)

Column	**Type**	**Description**
id	INTEGER	Primary Key
name	TEXT	Student Name
room	TEXT	Room Number
category	TEXT	Complaint Category
description	TEXT	Complaint Description
status	TEXT	Complaint Status
worker	TEXT	Worker Name

CHAPTER – 5

IMPLEMENTATION

5.1 Module Distribution

Student	Module	Responsibility
A	Complaint Submission	UI & Student view
B	Complaint Processing	Admin dashboard
C	Worker Module	Status update, logs
D	DevOps & DB	CI/CD, Schema, Auth

5.2 Sample Code Explanation

The **MainApp.java** file acts as the entry point of the application. It initializes the JavaFX framework, sets up the primary stage, and establishes a connection to the SQLite database. The **ComplaintController.java** file manages user interactions such as form submissions and button clicks. It validates the input data, communicates with the data layer, and updates the interface accordingly. The **ComplaintDAO.java** class handles all database operations, including inserting, retrieving, and updating complaint records. It ensures a clean separation between the business logic and data management.

5.3 Database Initialization

When the application runs for the first time, it automatically checks for the existence of the `hcms.db` file. If the file is not found, the system creates it along with the required tables for storing complaints and user details. This process ensures that the database is always ready for use without any manual setup, making deployment simpler and more efficient.

CHAPTER – 6
TESTING AND EVALUATION

6.1 Testing Approach

The testing phase ensured the reliability and stability of the Hostel Complaint Management System. **Unit testing** was carried out using **JUnit** to verify the functionality of the DAO (Data Access Object) layer, ensuring all database operations such as insert, update, and retrieval worked correctly. In addition, **manual testing** was performed for JavaFX user interface workflows to check the proper navigation, input validation, and overall user experience.

6.2 Sample Test Cases

Test Case ID	Description	Expected Result
TC01	Submit valid complaint	Complaint saved successfully
TC02	Empty fields	Validation message displayed
TC03	Database connection error	Error handled gracefully without crash

6.3 Evaluation

The system was thoroughly tested to meet all **functional and non-functional requirements**. The complaint submission, tracking, and status update features worked smoothly across multiple test scenarios. CRUD operations in the DAO layer were verified for correctness and consistency.The user interface was evaluated for usability, responsiveness, and error handling. The system handled invalid inputs gracefully, ensuring no data corruption or application crashes. Data persistence in SQLite was tested by checking whether complaints and status updates remained intact after application restarts.Overall, the testing results confirmed that the HCMS is stable, reliable, and ready for deployment. The combination of JUnit testing and manual UI verification provided complete coverage for the application's workflow.

CHAPTER – 7

DEVOPS INTEGRATION

7.1 CI/CD Pipeline

To streamline the development and deployment process, a **Continuous Integration and Continuous Deployment (CI/CD)** pipeline was implemented using **GitHub Actions** and **Jenkins**. This automation ensures that every code change is tested, built, and deployed seamlessly, reducing human error and improving delivery speed.

The CI/CD pipeline is triggered automatically whenever a **push or pull request** is made to the GitHub repository. The pipeline executes a sequence of stages:

- **Build Stage:** Maven compiles the Java code and runs unit tests.
- **Test Stage:** JUnit test results are analyzed to ensure that all modules function as expected.
- **Package Stage:** A JAR file is generated and stored as a build artifact.

STEPS:

Step 1: Prerequisites

1. **Jenkins Installed** (locally or on a server)
   - Make sure Jenkins is running and you can access it in your browser (e.g., `http://localhost:8080`).
2. **GitHub Repository**

- Repo with 3 branches (e.g., `dev`, `test`, `prod`).

3.Jenkins Plugins

Make sure these plugins are installed:

- GitHub Integration Plugin
- Pipeline Plugin
- Git Plugin

Step 2: Create gitHub webhook

- Go to your repository → **Settings** → **Webhooks** → **Add webhook**.
- Payload URL: `http://<your-jenkins-server>:8080/github-webhook/`

  Example: `http://localhost:8080/github-webhook/` (replace localhost with your Jenkins IP if remote).
- Content type: `application/json`
- Trigger: **Just the push event** (or optionally pull request events).
- Save.

Note: `github-webhook` is a default Jenkins endpoint; no extra config is needed for the URL. Jenkins will listen here.

Step 3: Configure Jenkins job

Single Multibranch Pipeline:

- Go to Jenkins → **New Item** → **Multibranch Pipeline**.
- Name it: e.g., `project-multi`.
- Branch Sources → Add Source → GitHub

  Repository URL + credentials

- Jenkins will automatically detect branches (`dev`, `test`, `prod`)
- **Scan Repository Triggers**: set a schedule (e.g., `* * * * *` for every minute) or rely on GitHub webhook
- Save

Jenkins will automatically create jobs for each branch.

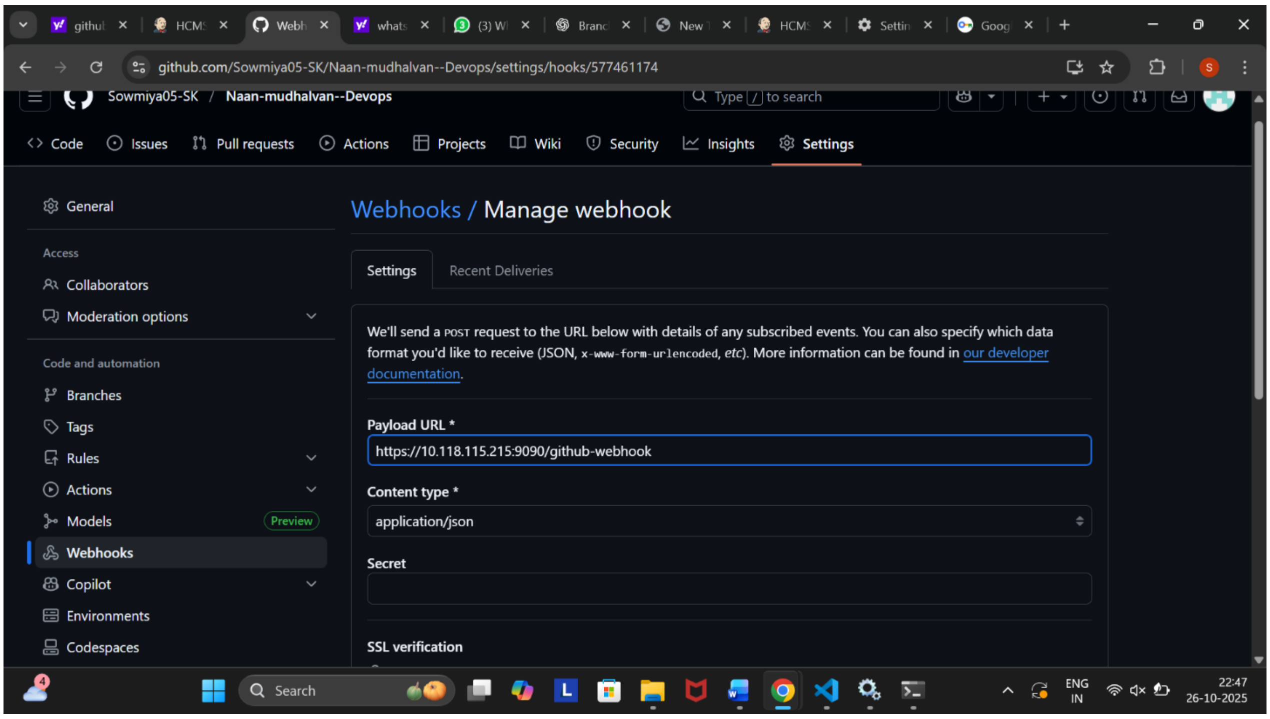
Step 4: Create jenkinsfile for each branch

Inside your repo, create a **Jenkinsfile** in the root directory (or per branch if needed). Example:

```
        pipeline {
 agent any
stages {
       stage('Checkout') {
          steps {
              git branch: "${env.BRANCH_NAME}", url: 'https://github.com/<your-
repo>.git'
          } }
        stage('Build') {
          steps {
              // Example: Python project
              sh 'python -m pip install -r requirements.txt'
              sh 'pytest'
          }  }
        stage('Deploy') {
          steps {
              echo "Deploying branch ${env.BRANCH_NAME}"
              // Add your deployment script here (Docker, server copy, etc.)
          } }}
    post {
       success {
          echo 'Build and deployment successful!'
       }
       failure {
          echo 'Build or deployment failed!'
       }}}
```

Step 5: Test the CI/CD Pipeline

GitHub Actions



Webhooks / Manage webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, *etc*). More information can be found in our developer documentation.

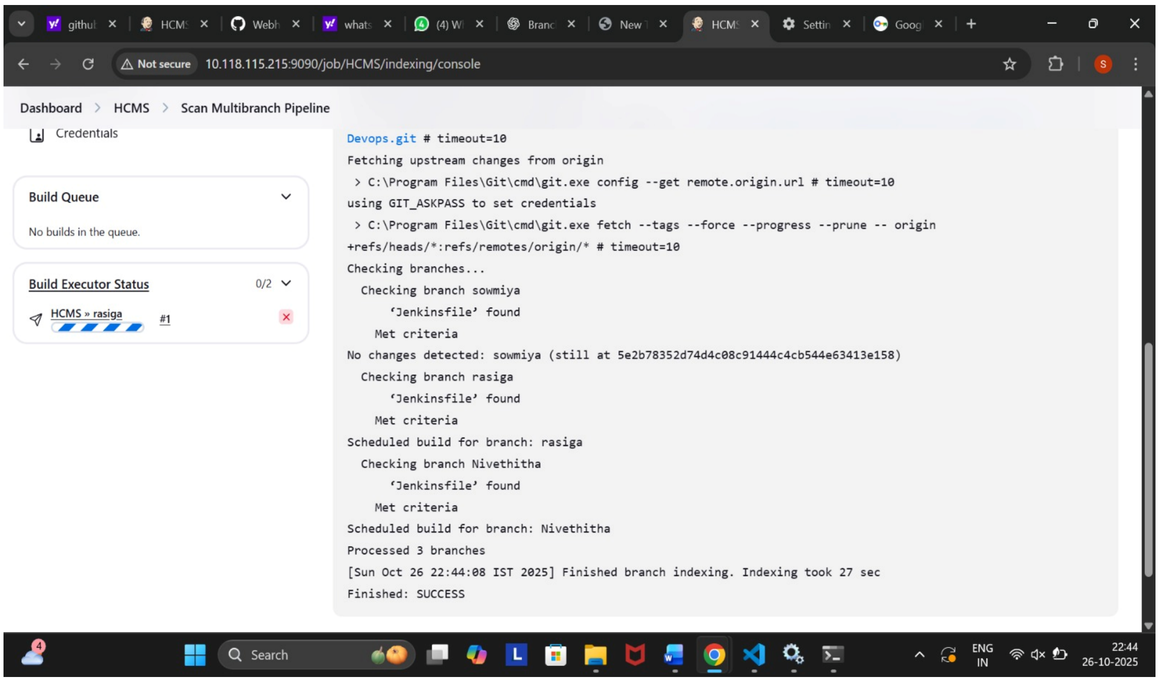
Payload URL *
https://10.118.115.215:9090/github-webhook

Content type *
application/json

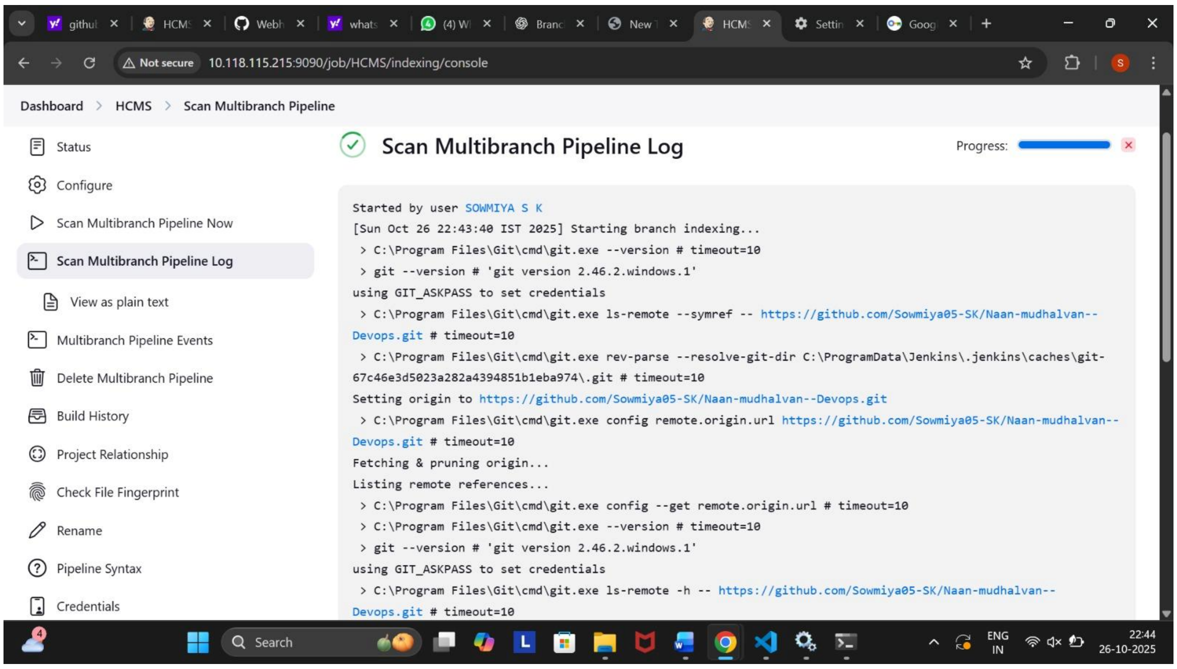
Secret

SSL verification

Jenkins Pipeline



```
Devops.git # timeout=10
Fetching upstream changes from origin
 > C:\Program Files\Git\cmd\git.exe config --get remote.origin.url # timeout=10
using GIT_ASKPASS to set credentials
 > C:\Program Files\Git\cmd\git.exe fetch --tags --force --progress --prune -- origin
+refs/heads/*:refs/remotes/origin/* # timeout=10
Checking branches...
  Checking branch sowmiya
      'Jenkinsfile' found
    Met criteria
No changes detected: sowmiya (still at 5e2b78352d74d4c08c91444c4cb544e63413e158)
  Checking branch rasiga
      'Jenkinsfile' found
    Met criteria
Scheduled build for branch: rasiga
  Checking branch Nivethitha
      'Jenkinsfile' found
    Met criteria
Scheduled build for branch: Nivethitha
Processed 3 branches
[Sun Oct 26 22:44:08 IST 2025] Finished branch indexing. Indexing took 27 sec
Finished: SUCCESS
```



Scan Multibranch Pipeline Log

```
Started by user SOWMIYA S K
[Sun Oct 26 22:43:40 IST 2025] Starting branch indexing...
 > C:\Program Files\Git\cmd\git.exe --version # timeout=10
 > git --version # 'git version 2.46.2.windows.1'
using GIT_ASKPASS to set credentials
 > C:\Program Files\Git\cmd\git.exe ls-remote --symref -- https://github.com/Sowmiya05-SK/Naan-mudhalvan--
Devops.git # timeout=10
 > C:\Program Files\Git\cmd\git.exe rev-parse --resolve-git-dir C:\ProgramData\Jenkins\.jenkins\caches\git-
67c46e3d5023a282a4394851b1eba974\.git # timeout=10
Setting origin to https://github.com/Sowmiya05-SK/Naan-mudhalvan--Devops.git
 > C:\Program Files\Git\cmd\git.exe config remote.origin.url https://github.com/Sowmiya05-SK/Naan-mudhalvan--
Devops.git # timeout=10
Fetching & pruning origin...
Listing remote references...
 > C:\Program Files\Git\cmd\git.exe config --get remote.origin.url # timeout=10
 > C:\Program Files\Git\cmd\git.exe --version # timeout=10
 > git --version # 'git version 2.46.2.windows.1'
using GIT_ASKPASS to set credentials
 > C:\Program Files\Git\cmd\git.exe ls-remote -h -- https://github.com/Sowmiya05-SK/Naan-mudhalvan--
Devops.git # timeout=10
```

7.2 Maven Integration

Maven is used as the build automation and dependency management tool in this project. It simplifies the process of compiling, packaging, testing, and running the HCMS modules (backend, UI, and database integration) by managing all required libraries and plugins automatically.

Key dependencies:

Dependency	**Purpose**
`org.xerial:sqlite-jdbc`	SQLite database connectivity
`org.openjfx:javafx-controls`	JavaFX UI components
`org.openjfx:javafx-fxml`	FXML-based UI layout
`org.projectlombok:lombok`	Reduces boilerplate code (getters, setters)
`junit:junit`	Unit testing framework

Build and Run commands

1.Clean and Build

Removes previously compiled files and rebuilds the project:

- mvn clean install

2. Run the Application (JavaFX UI)

- mvn javafx:run

This uses the **JavaFX Maven Plugin** to compile and launch the application directly.

3. Package the Application

To create a JAR file:

- mvn package

The generated `.jar` file will be available in the `target/` directory.

7.3 Docker Integration

To simplify deployment and ensure platform independence, **Docker** was used to containerize the application. The **Dockerfile** packages the HCMS Java application with **JRE 17**, dependencies, and environment configurations into a single portable container.

This allows the backend service to be deployed easily on any system supporting Docker, without the need to configure dependencies manually. It eliminates environment-specific issues that often occur during development and deployment. The Docker image can be pulled and run on different environments—development, testing, or production—ensuring consistent performance.

The integration of Docker with Jenkins and GitHub Actions completes the DevOps workflow, making the project fully automated, portable, and ready for real-world deployment.

CHAPTER – 8

RESULTS AND DISCUSSION

The **Hostel Complaint Management System (HCMS)** was successfully implemented and tested to ensure smooth operation across all user roles. The results confirm that the system meets the defined functional and performance objectives.

Students can easily **log complaints** through the JavaFX interface, providing details such as category, description, and room number. The complaints are stored securely in the **SQLite database**, ensuring that no data is lost between sessions. Administrators can **view all submitted complaints**, assign them to appropriate workers, and update their status as progress is made. Workers can access their assigned tasks, mark them as completed, and submit feedback once the issue is resolved.

All complaint records are **persistently stored** in the database, enabling reliable tracking and future reference. The integrated **DevOps pipeline**, built using GitHub Actions and Jenkins, automates the build, testing, and deployment process. This reduces manual effort and ensures consistent delivery across environments.

During performance testing, the system demonstrated **fast database access**, smooth navigation, and **minimal latency** in user interface operations. The overall results indicate that the system is stable, responsive, and efficient for daily hostel management operations.

CHAPTER – 9

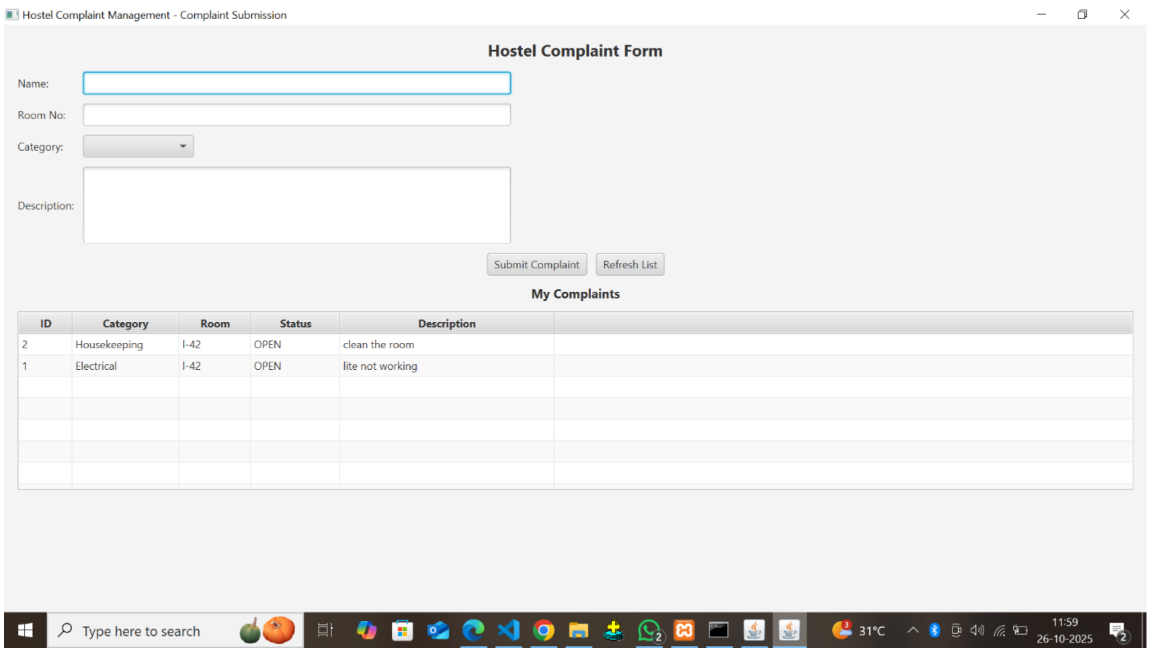
CODE AND OUTPUT SNAPSHOTS

GitHub Repository:

https://github.com/Sowmiya05-SK /Naan-mudhalvan--Devops.git

Output Snapshots:

Hostel Complaint Form


Hostel Complaint Management - Complaint Submission

Hostel Complaint Form

Name:

Room No:

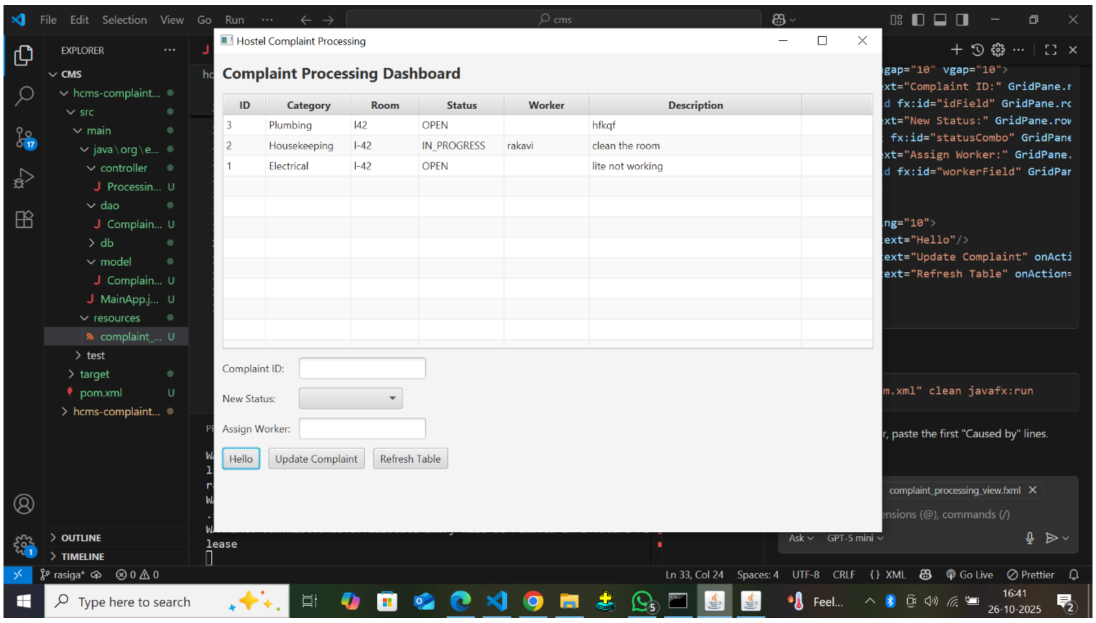
Category:

Description:

My Complaints

ID	Category	Room	Status	Description
2	Housekeeping	I-42	OPEN	clean the room
1	Electrical	I-42	OPEN	lite not working

Complaint Processing Dashboard


Hostel Complaint Processing

Complaint Processing Dashboard

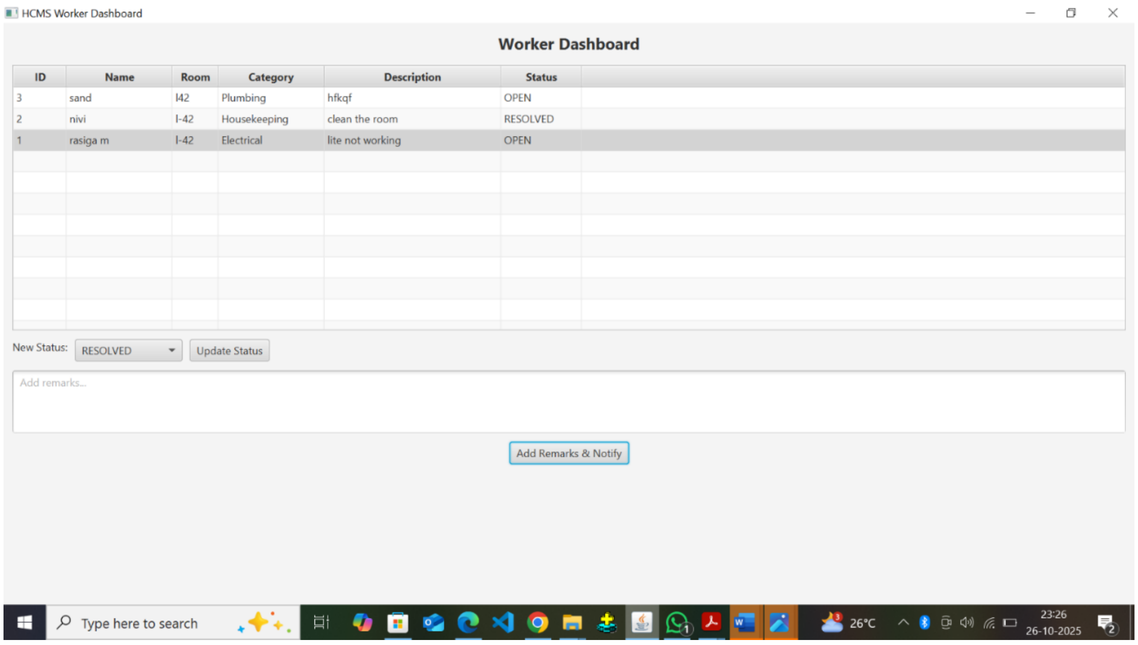
ID	Category	Room	Status	Worker	Description
3	Plumbing	I42	OPEN		hfkqf
2	Housekeeping	I-42	IN_PROGRESS	rakavi	clean the room
1	Electrical	I-42	OPEN		lite not working

Complaint ID:

New Status:

Assign Worker:

Worker Dashboard


HCMS Worker Dashboard

Worker Dashboard

ID	Name	Room	Category	Description	Status
3	sand	I42	Plumbing	hfkqf	OPEN
2	nivi	I-42	Housekeeping	clean the room	RESOLVED
1	rasiga m	I-42	Electrical	lite not working	OPEN

New Status: RESOLVED

Add remarks...

CHAPTER – 10

CONCLUSION AND FUTURE SCOPE

Conclusion

The Hostel Complaint Management System (HCMS) provides an effective digital solution for managing hostel-related complaints. It replaces manual processes with an automated system that ensures quick submission, tracking, and resolution of complaints. Developed using JavaFX and SQLite, the system offers a simple, user-friendly interface with secure data storage.

The implementation of the **MVC architecture** improves modularity and maintainability, while the integration of **DevOps tools** such as GitHub Actions, Jenkins, and Docker automates building, testing, and deployment. The system was thoroughly tested and met all major functional and performance requirements, ensuring reliability and user satisfaction. Overall, the project successfully demonstrates a complete software development lifecycle from design to deployment.

Future Enhancement

The system can be further enhanced in future versions by adding:

- **Web or mobile support** for better accessibility.
- **Email or SMS notifications** to alert users of complaint updates.
- **Cloud database integration** for scalability and remote access.
- **Analytics dashboard** for administrators to track performance.

These improvements will help transform the HCMS into a more efficient, scalable, and user-centric application suitable for real-time hostel management.