**A PROJECT REPORT ON**

# "REPORT-MATE"

**Submitted to the Mangalore University in partial fulfillment of the requirement for the award of the Degree of**

## BACHELOR OF COMPUTER APPLICATION

**Submitted by**


## PRANAMYA

## Register Number: U05SD22S0109


**Under the valuable guidance of**

## MRS. SHASHIKALA SHETTY


**SHRI DHARMASTHALA MANJUNATESHWARA COLLEGE OF BUSINESS MANAGEMENT**
**MANGALURU – 575003**


**2024–2025**

# Shri Dharmasthala Manjunatheshwara College of Business Management, Mangaluru - 575003

(Affiliated to Mangalore University)

## CERTIFICATE

This is to certify that the project report entitles **"REPORT-MATE"** is an authenticated record of the project work carried out by **Ms. Pranamya** bearing Register Number **U05SD22S0109** in partial fulfillment of the requirement for the award of **Bachelor's Degree in Computer Application of Mangalore University** under the guidance and supervision during the year 2024-2025.

**Forwarded to Principal for Approval**

Place: Mangaluru (Mrs. Shashikala Shetty)

Date: May 17, 2025 Project guide

**Approved and forwarded to Mangalore University**

Place: Mangaluru (Prof. Aruna.P. Kamath)

Date: May 17, 2025 Principal

**Signature of the Examiners:**

**1……** **2……..**

# DECLARATION

I hereby declare that the project report titled as **"REPORT-MATE"** has been prepared by me during the year 2024– 2025 under the valuable guidance and supervision of **Mrs. Shashikala Shetty**, Associate Professor and project guide, SDM College of Business Management, Mangaluru, in partial fulfillment of the requirement for the award of degree in Bachelor of Computer Application from Mangalore University for the academic year 2024-2025.

I also declare that this project is the result of my own effort and has not been submitted to any other University for the award of any degree or diploma.

Place: Mangaluru

Date: May 17, 2025

.....

Ms. Pranamya

Reg No : U05SD22S0109

# ACKNOWLEDGEMENT

Whatever we are able to put forward in terms of outwork, is only due to few people from whom we have learnt. Thus, it is a great pleasure in mentioning their names.

Working on the live project was very interesting and really enhanced our knowledge.

We would like to express our sincere gratitude to our beloved **Principal Mrs. Aruna P.Kamath** without whose permission we would not be able to do our project and for taking keen interest for the students of BCA in providing useful guidelines and giving all the necessary facilities.

We extend thanks to our guide **Mrs. Shashikala Shetty,** Associate Professor of Computer Application Department for her valuable guidance and constant encouragement, which helped us in successfully completing our project.

Finally, we extend thanks to our parents and friends who were directly or indirectly involved in the completion of our project.

Place: Mangaluru .....

Date: May 17, 2025 Ms. Pranamya

Reg No: U05SD22S0109

# MANGALORE SMART CITY LIMITED

Lalbagh, Mangaluru

**TO WHOMSOEVER IT MAY CONCERN**

This is to certify that Ms. Pranamya , bearing U05SD22S0109, a student of SDM College Of Business Management, Mangalore has successfully completed her project titled "REPORT MATE" for the purpose of partial requirements for the degree of Bachelors of Computer Application (BCA) of Mangalore University during the academic year 2024-2025.

The project was carried out under the guidance of official, Shamsuddin, Corporator of Kudroli (Ward No.43) at Mangaluru Smart City Limited (MSCL), which focuses on enabling citizens to report issues in real-time enhancing efficient responsiveness.

She has collected the necessary data and completed the project work to our satisfaction during the period from January 2025 to April 2025.

During this tenure her conduct and character was found good. We wish her success in all her future endeavors.

SHAMSUDDIN
Corporator
Mangaluru City Corporation

Thanking You,

(Name & Signature)

Place: Mangalore

Date: 12-05-2025

# Contents

# Chapter 1

# SYNOPSIS

## 1.1.Title of the Project

## ‘Report-Mate’

## 1.2.Introduction

Report-Mate is a smart issue-reporting web application designed to bridge the gap between citizens and civic authorities. It enables people to report problems they see around them like damaged roads, street lights not working, garbage issues, or water leakage. Anyone can use it to quickly submit a report by filling a form with the issue details, their location, and even upload a photo. The submitted issues are categorized, stored, and mapped with suitable workers who can resolve them .The admin can approve or decline reports, and also assign available workers who can fix the problems. The system shows workers based on their issue and location, so the right person is chosen for each task. `

## 1.2.1 Objectives and scope of the Project.

The main objective of Report-Mate is to provide a simple and effective way for citizens to report public issues such as road damage, garbage accumulation, streetlight failures, and water leakage. The application allows users to fill out a report form with the issue type, description, location, and an optional image. These reports are securely stored in a MySQL database and made available to the admin through a dedicated panel. The admin can view, approve, or decline reports and assign suitable workers based on their skills and the reported location. The system aims to improve communication between the public and civic authorities, making issue resolution faster and more organized. ReportMate is suitable for urban or semi-urban areas and can be scaled to support mobile apps, real-time alerts, and advanced worker tracking in the future. For instance, if a user notices a pothole near their home, they can visit the ReportMate site, select “Road Damage,” enter the location and description, and submit the report. The admin receives this report, approves it, and assigns a road repair worker nearby. The user can later check the status and confirm when the issue is resolved. This simple process ensures that everyday civic problems are reported and addressed quickly and efficiently.

## 1.3.Project category:

Web Application for public utility.

## 1.4. Tools / Platform

- **Visual Studio Code**

One of the key features of VS Code is its extensive marketplace of extensions, which allows developers to customize and extend the functionality of the editor. There are thousands of extensions available, covering everything from syntax highlighting and code formatting to version control and testing frameworks.

- **Python with Flask framework**

Python is a powerful, easy-to-learn programming language widely used in web development, data science, automation, and artificial intelligence. It is known for its simple syntax, large community support, and extensive library ecosystem. In the ReportMate project, Python is used as the backend language to process form submissions, manage database interactions, and control application logic. Flask is a lightweight and flexible web framework built with Python. It allows developers to create secure and scalable web applications with minimal setup. Flask supports route handling, session management, form processing, and integration with databases such as MySQL.

- **JavaScript**

JavaScript is a high-level, interpreted programming language primarily used for creating dynamic and interactive content on websites. It enables developers to add functionalities such as user interaction, animations, form validation, and asynchronous data loading. JavaScript is often used in combination with HTML and CSS to build web applications.

- **My SQL**

MySQL is the world's most widely used open source Relational Database Management System (RDBMS) that runs as a server providing multi-user access to a number of databases. The MySQL development project has made it source code available under the terms of the GNU General Public License, as well as under a variety of proprietary agreements.

### 1.4.1 Hardware Requirements

- RAM : 8 GB or above
- Hard disk : 10 GB or above
- Processor : Intel i3 2.4 GHZ or above

### 1.4.2 Software Requirements

- Front end : HTML, CSS Java Script
- Back end : Python (Flask framework)
- Database : MySQL
- Server: Flask local server
- Code Editor : Visual Studio Code
- Browser : Chrome, Firefox, or Edge (latest)

## 1.5. Module description

- **Authentication**: is the process of verifying the identity of a user to ensure that they are who they claim to be. It typically involves the use of credentials such as usernames and passwords. Once authenticated, the user is granted access to resources or functionalities within an application.
  - **Admin:** Admins can log in using their own credentials, often with more stringent requirements (e.g., strong passwords, MFA).
  - **Citizen:** The citizen has to enter their credentials (username and password) into a login form. The new user must register first giving the credentials only after that they can log in.
  - **Worker:** Workers can log in to see the assigned task using their username and password.
- **Report**: This module enables users to report problems in their surroundings—such as non-functional street lights, potholes, broken benches, or garbage accumulation. It helps local authorities respond efficiently to public complaints by collecting detailed issue reports from citizens.
  - **Issue:** It allows user to select the issue or describe it.
  - **Location:** Users can specify their location.
  - **Worker availability:** Before submitting the report, users can view whether any maintenance workers are available or assigned nearby, providing transparency on response expectations.

- **Media upload:** Users can attach photos or videos of the issue to give authorities a clear view of the problem, aiding faster diagnosis and resolution.

- **Dashboard**: It is a key feature of the system that allows to manage and monitor all respective works.
  - **Citizen Dashboard:** It allows user to navigate towards the links that directs citizen to report an issue or to track their report.
  - **Worker Dashboard:** It contains assigned tasks to particular workers they can see the assigned task after they successfully log in using their username and password.
  - **Admin Dashboard:** It allows the admin to see the interaction between the citizen and the worker.
  - **Worker Response Dashboard:** It shows how the worker responses to the issues reported by the user.
- **Track Report status**: This feature allows users to track the status of issues they have submitted. It provides real-time updates on whether an issue has been addressed by the admin or is still awaiting action.

## 1.6. Limitation of the project

- The project relies on an internet connection, limiting its functionality in offline scenarios.

## 1.7. Future scope and further enhancement of the project.

- Mobile App Integration (with GPS Support)**:** A dedicated app allows real-time issue reporting with automatic location capture. It will also boost user engagement and improves the accuracy of reports through GPS tagging.
- Real-Time Notifications (Email/SMS): It keeps users informed when their report is submitted, reviewed, or resolved. It also helps in enhancing trust, improving transparency, and will encourage repeated usage.
- Developing an offline mode to allow users to access the application even without a stable connection because people in remote areas may find it difficult to submit the report. This feature will ensure that no report is lost due to poor connection as it can be stored it local storage and is automatically synced once the connection is established.

# Chapter 2

# DATABASE DESIGN

## 2.1 Introduction:

Database description describes all the databases used in the software to store all the records. The database in turn is further described in detail giving all the fields used with their data type, constraints available to them and description. Constraints include primary key, foreign key, etc., which allow the entities to be uniquely identified

In this database description we describe all databases which are used to store all the records of the Cuisine Connect Catering System.

## 2.2 Purpose:

- Database description describes the entire database used in the software to store all the records.
- Database design is the process of producing a detailed data model of a database.

Database design is a collection of related data.

- This document describes standards to use when designing and developing the database.

## 2.3 Scope:

A good database is one that is simple to understand and well planned. The database doesn't have redundant tables. One can use ER Diagram (Entity Relationship Diagrams) in order to make a good database. This database design is used to understand the software hotel KOT and billing

- Organize the system into modules
- Organize sub-modules for each module
- Allocate tasks to processors
- Choose an approach to manage data store
- Handle access to global resources
- Choose implementation logic.

## 2.4 Normalized Tables:

**Users Table: It stores the user credentials in the database**

| Column Name | Data Type | Length | Constraints |
|---|---|---|---|
| id | Int | 11 | Primary Key |
| username | Varchar | 45 | Not null |
| email | Varchar | 45 | Not null |
| password | Varchar | 10 | Not null |
| first_name | Varchar | 255 | Not null |
| last_name | Varchar | 255 | Not null |
| phone | Varchar | 20 | Not null |

**Reports Table: It stores the reports in the database**

| Column Name | Data Type | Length | Constraints |
|---|---|---|---|
| id | Int | 11 | Primary Key |
| issue_type | Varchar | 50 | Foreign Key |
| description | Text | 25 | Not null |
| image_path | Varchar | 15 | Not null |
| timestamp | Datetime | | Not null |
| status | Date | | Not null |
| username | Varchar | 11 | Not null |
| assigned_workers | Varchar | 45 | Not null |

**Workers Table: It stores the worker details in the database**

| Column Name | Data Type | Length | Constraints |
|---|---|---|---|
| id | Int | 10 | Primary Key |
| name | Varchar | 45 | Not null |
| location | Text | 100 | Not null |
| username | Varchar | 255 | Not null |
| password | Varchar | 255 | Not null |
| issue_type | Varchar | 255 | Not null |

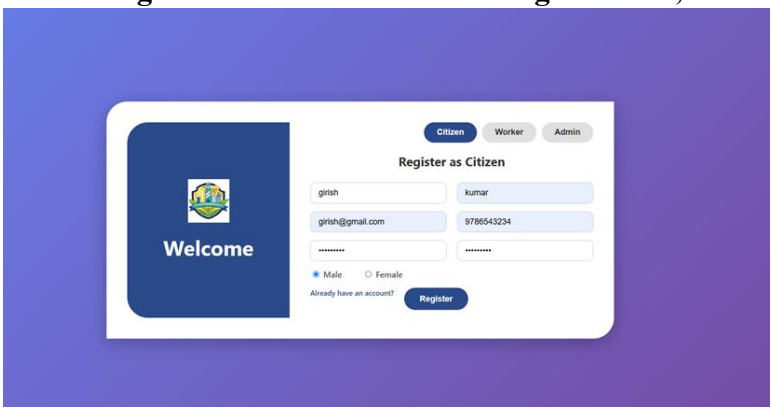
# Chapter 3

# IMPLEMENTATION

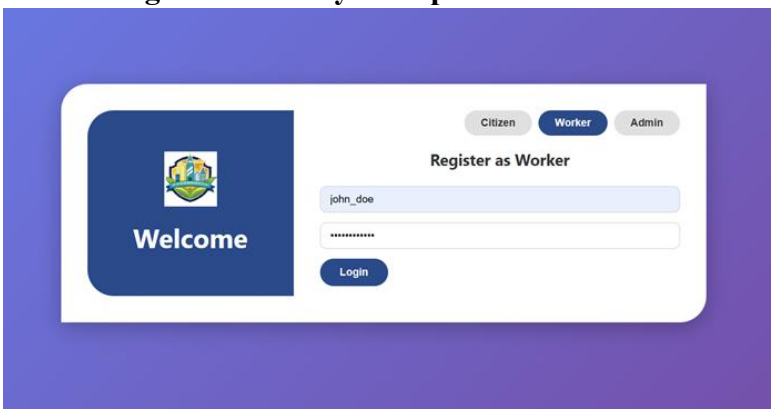
## 3.1 USER INTERFACE SNAPSHOTS

**Login page: It is page for logging in to the web application.**

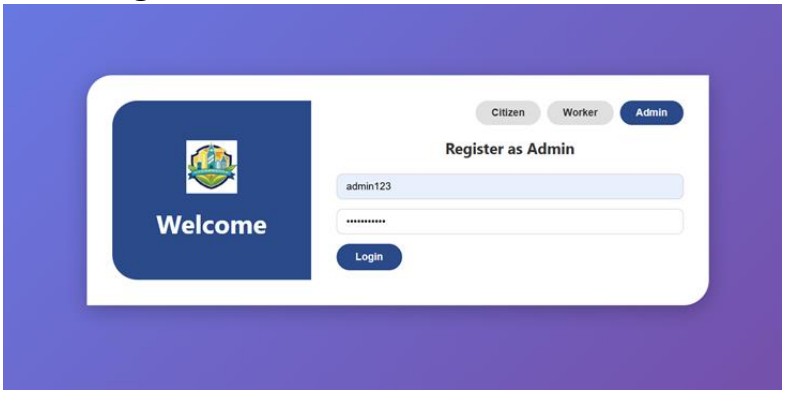
- **Citizen login form: The citizen must register first, and then can log in.**



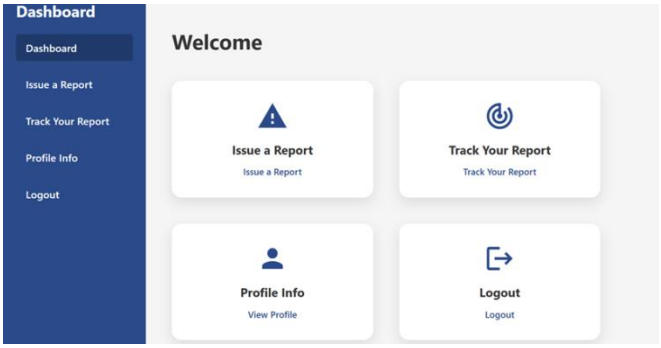

- **Worker login form: They have particular username and password.**



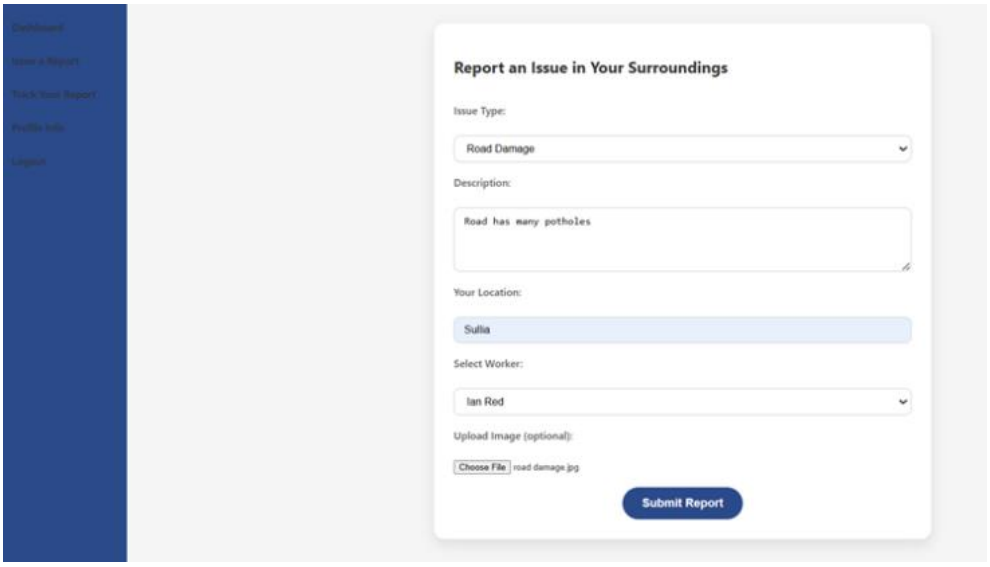

- **Admin login form:**



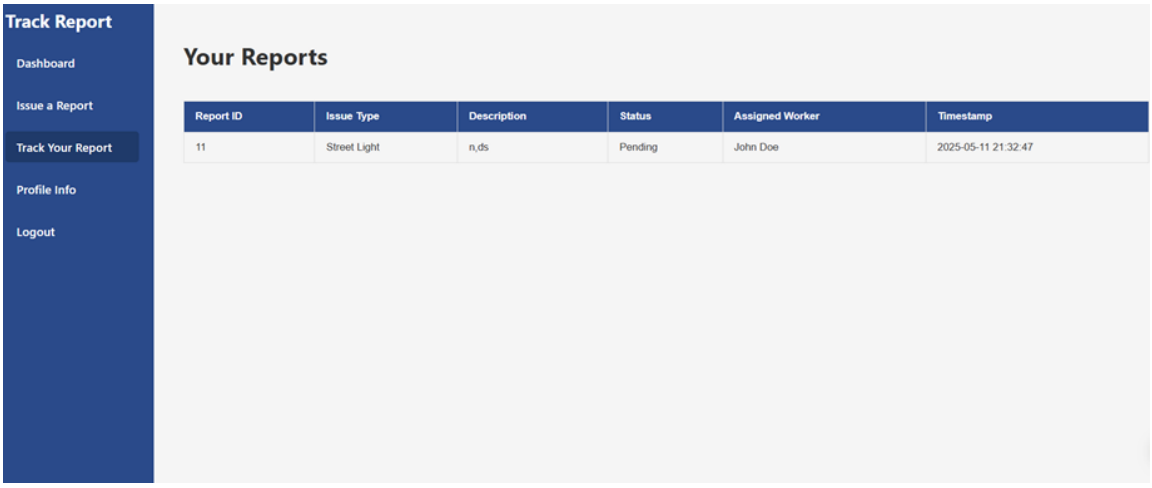
**Citizen Dashboard: It is like home page after the citizen logins where he can issue report, track his report, check his profile information.**



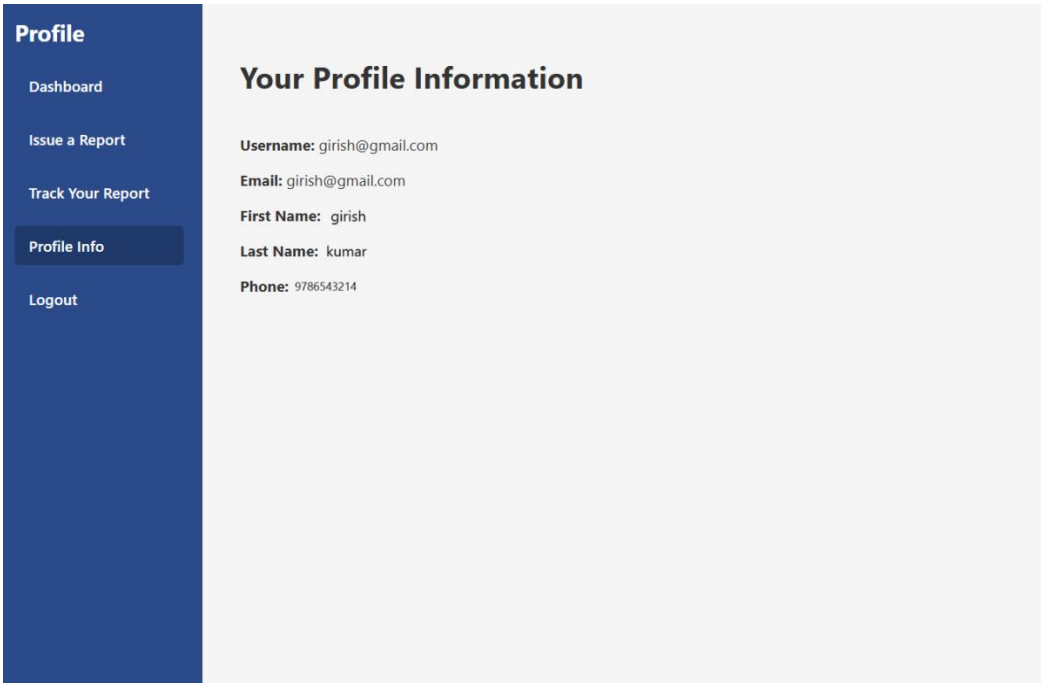

**Report Issue: This allows you to report the issue around your surroundings asking relevant information.**




**Track Report status: This will let you know the status of your report.**

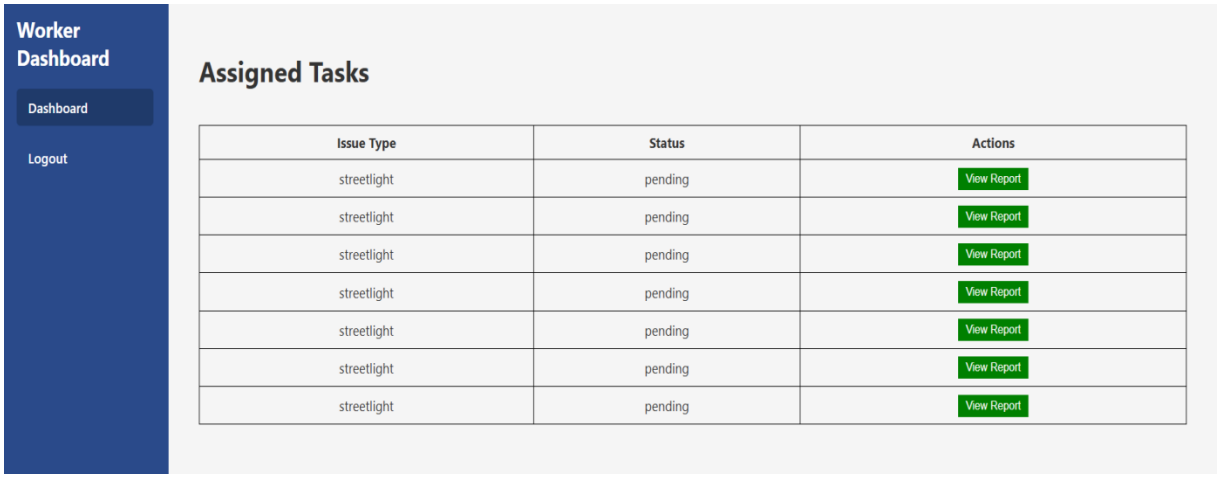


**Profile Information:**

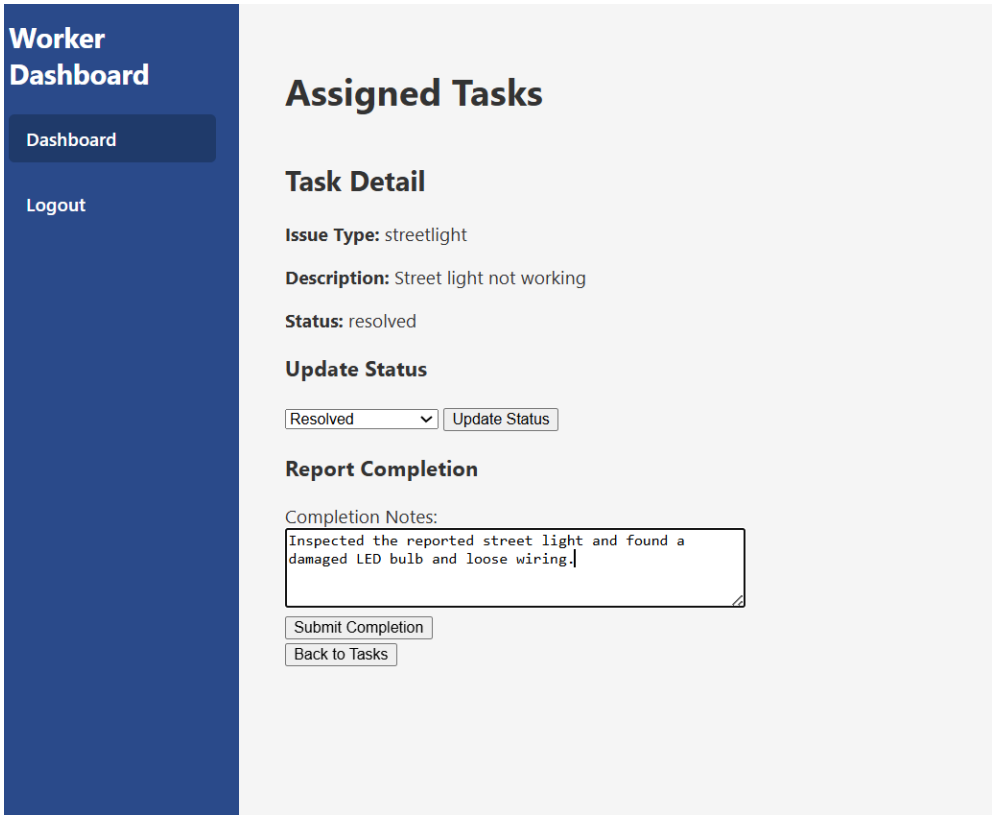



**Worker Dashboard: It displays the assigned task the worker logged in.**

- **Assigned task list:**

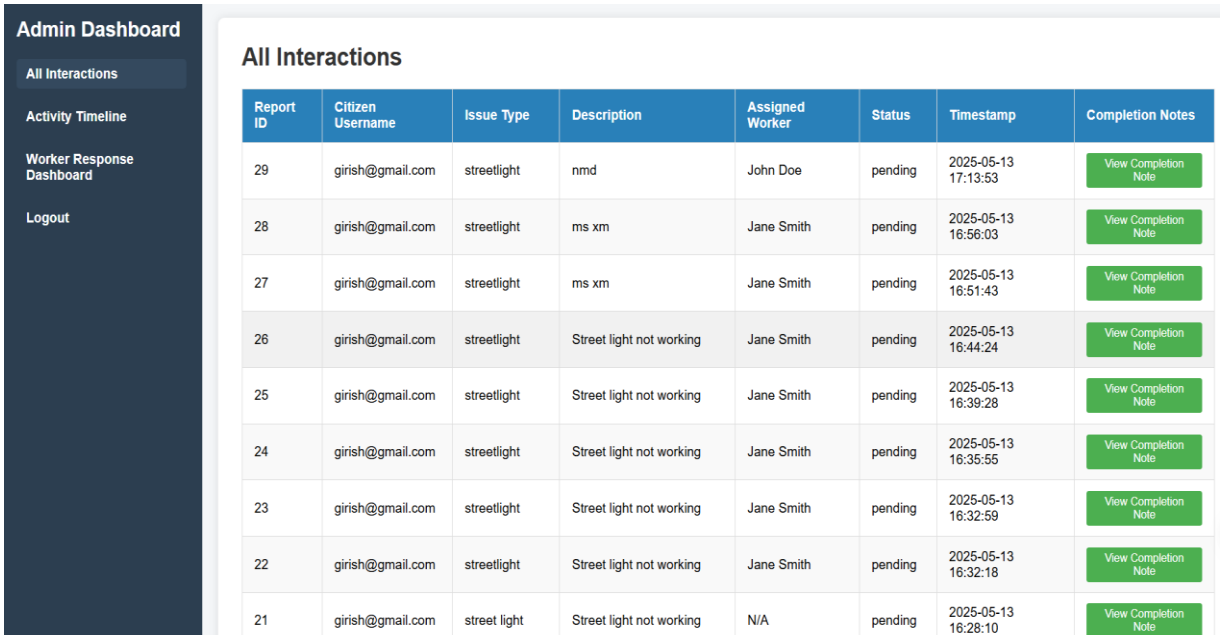


- **Task details**



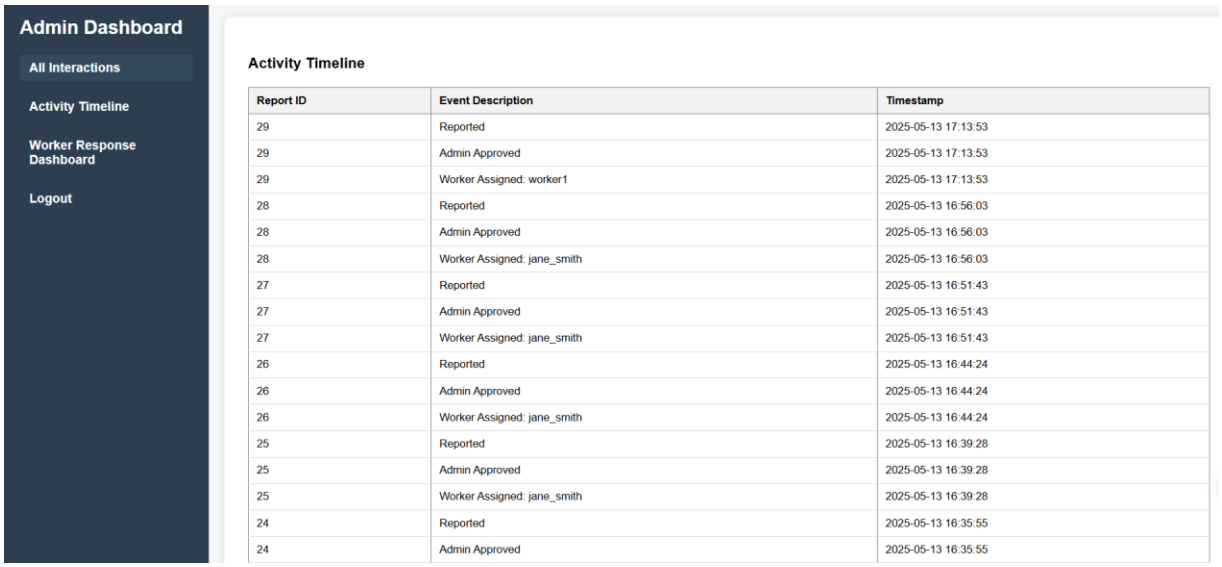

**Admin Dashboard: It shows all the interaction between worker and citizen.**

- **All interactions:**

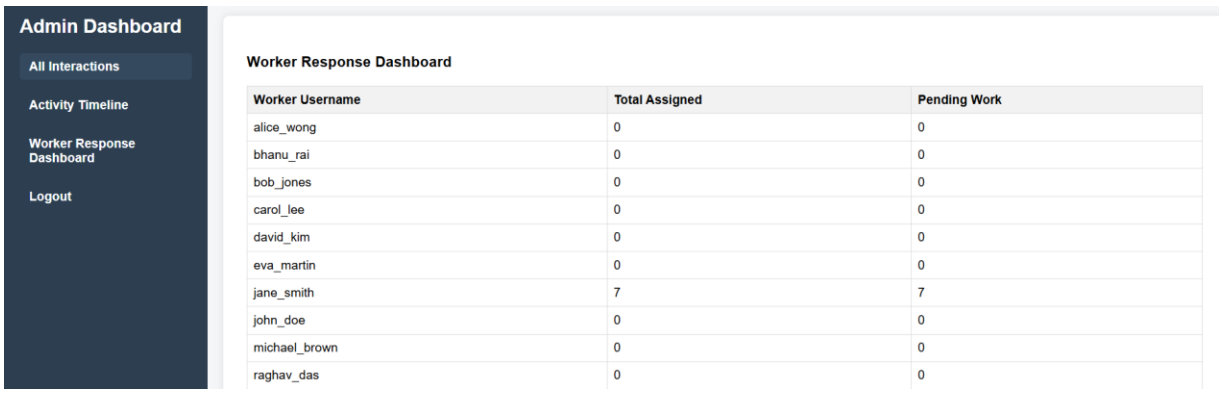



| Report ID | Citizen Username | Issue Type | Description | Assigned Worker | Status | Timestamp | Completion Notes |
|---|---|---|---|---|---|---|---|
| 29 | girish@gmail.com | streetlight | nmd | John Doe | pending | 2025-05-13 17:13:53 | View Completion Note |
| 28 | girish@gmail.com | streetlight | ms xm | Jane Smith | pending | 2025-05-13 16:56:03 | View Completion Note |
| 27 | girish@gmail.com | streetlight | ms xm | Jane Smith | pending | 2025-05-13 16:51:43 | View Completion Note |
| 26 | girish@gmail.com | streetlight | Street light not working | Jane Smith | pending | 2025-05-13 16:44:24 | View Completion Note |
| 25 | girish@gmail.com | streetlight | Street light not working | Jane Smith | pending | 2025-05-13 16:39:28 | View Completion Note |
| 24 | girish@gmail.com | streetlight | Street light not working | Jane Smith | pending | 2025-05-13 16:35:55 | View Completion Note |
| 23 | girish@gmail.com | streetlight | Street light not working | Jane Smith | pending | 2025-05-13 16:32:59 | View Completion Note |
| 22 | girish@gmail.com | streetlight | Street light not working | Jane Smith | pending | 2025-05-13 16:32:18 | View Completion Note |
| 21 | girish@gmail.com | street light | Street light not working | N/A | pending | 2025-05-13 16:28:10 | View Completion Note |

- **Activity timeline:**




- **Worker Response dashboard:**



## 3.2 CODING

**login_roles.html: This code snippet is used to design login pages for citizen, worker and admin.**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Register/Login</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='login_roles.css') }}" />
</head>
<body>
  <div class="container">
    <div class="welcome-section">
      <img src="{{ url_for('static', filename='images/logo.png') }}" alt="logo" class="icon" />
      <h1>Welcome</h1>
    </div>
    <div class="form-section">
      <div class="tab-buttons">
        <button class="tab-btn active" data-tab="citizen">Citizen</button>
        <button class="tab-btn" data-tab="worker">Worker</button>
        <button class="tab-btn" data-tab="admin">Admin</button>
      </div>
      <div class="tab-content active" id="citizen">
        <h2>Register as Citizen</h2>
        <form id="citizenForm">
          <div class="row">
            <input type="text" name="firstName" placeholder="First Name *" required />
            <input type="text" name="lastName" placeholder="Last Name *" required />
          </div>
          <div class="row">
            <input type="email" name="email" placeholder="Your Email *" required />
```

```
          <input type="tel" name="phone" placeholder="Your Phone *" required />
        </div>
        <div class="row">
          <input type="password" name="password" placeholder="Password *" required />
          <input type="password" name="confirmPassword" placeholder="Confirm Password
*" required />
        </div>
        <div class="row gender-row">
          <label><input type="radio" name="gender" value="male" required /> Male</label>
          <label><input type="radio" name="gender" value="female" required />
Female</label>
        </div>
        <div class="row">
          <a href="#" id="citizenLoginLink">Already have an account?</a>
          <button type="submit">Register</button>
        </div>
      </form>
      <form id="citizenLoginForm" style="display:none;">
        <div class="row">
          <input type="text" name="username" placeholder="Username *" required />
        </div>
        <div class="row">
          <input type="password" name="password" placeholder="Password *" required />
        </div>
        <div class="row">
          <a href="#" id="backToRegisterLink">Back to Register</a>
          <button type="submit">Login</button>
        </div>
      </form>
    </div>
    <div class="tab-content" id="worker">
      <h2>Register as Worker</h2>
      <form id="workerForm">
```

```
        <div class="row">
          <input type="text" name="username" placeholder="Username *" required />
        </div>
        <div class="row">
          <input type="password" name="password" placeholder="Password *" required />
        </div>
        <div class="row">
          <button type="submit">Login</button>
        </div>
      </form>
    </div>
    <div class="tab-content" id="admin">
      <h2>Register as Admin</h2>
      <form id="adminForm">
        <div class="row">
          <input type="text" name="username" placeholder="Username *" required />
        </div>
        <div class="row">
          <input type="password" name="password" placeholder="Password *" required />
        </div>
        <div class="row">
          <button type="submit">Login</button>
        </div>
      </form>
    </div>
  </div>
</div>
<script>
  const tabButtons = document.querySelectorAll('.tab-btn');
  const tabContents = document.querySelectorAll('.tab-content');
  tabButtons.forEach(button => {
    button.addEventListener('click', () => {
      tabButtons.forEach(btn => btn.classList.remove('active'));
```

```
        tabContents.forEach(content => content.classList.remove('active'));
        button.classList.add('active');
        document.getElementById(button.dataset.tab).classList.add('active');
      });
    });
    // Show login form and hide register form for citizen
    const citizenLoginLink = document.getElementById('citizenLoginLink');
    const backToRegisterLink = document.getElementById('backToRegisterLink');
    const citizenForm = document.getElementById('citizenForm');
    const citizenLoginForm = document.getElementById('citizenLoginForm');
    citizenLoginLink.addEventListener('click', (e) => {
      e.preventDefault();
      citizenForm.style.display = 'none';
      citizenLoginForm.style.display = 'block';
    });
    backToRegisterLink.addEventListener('click', (e) => {
      e.preventDefault();
      citizenLoginForm.style.display = 'none';
      citizenForm.style.display = 'block';
    });
    // Add form submission handlers here if needed
  </script>
  <script>
    // Handle citizen registration form submission
    document.getElementById('citizenForm').addEventListener('submit', async
function(event) {
      event.preventDefault();
      const form = event.target;
      const firstName = form.firstName.value;
      const lastName = form.lastName.value;
      const email = form.email.value;
      const phone = form.phone.value;
      const password = form.password.value;
```

```
const confirmPassword = form.confirmPassword.value;
const gender = form.gender.value;
if (!firstName || !lastName || !email || !phone || !password || !confirmPassword || !gender) {
  alert('Please fill in all required fields.');
  return;
}
if (password !== confirmPassword) {
  alert('Passwords do not match.');
  return;
}
try {
  const response = await fetch('/register', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      username: email,
      email: email,
      password: password,
      first_name: firstName,
      last_name: lastName,
      phone: phone,
      gender: gender
    })
  });
  if (response.redirected) {
    window.location.href = response.url;
    return;
  }
  const data = await response.json();
  if (response.ok) {
    alert(data.message || 'Registration successful');
    window.location.href = '/citizen_dashboard';
  } else {
```

```
        alert(data.error || 'Registration failed');
      }
    } catch (error) {
      alert('Error connecting to server');
      console.error('Citizen registration error:', error);
    }
  });
  // Handle citizen login form submission
  document.getElementById('citizenLoginForm').addEventListener('submit', async
function(event) {
    event.preventDefault();
    const form = event.target;
    const username = form.username.value;
    const password = form.password.value;
    if (!username || !password) {
      alert('Please enter both username and password.');
      return;
    }
    try {
      const response = await fetch('/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ username, password })
      });
      const data = await response.json();
      if (response.ok) {
        alert(data.message || 'Login successful');
        window.location.href = '/citizen_dashboard';
      } else {
        alert(data.error || 'Login failed');
      }
    } catch (error) {
      alert('Error connecting to server');
```

```
        console.error('Citizen login error:', error);
      }
    });
    // Handle worker login form submission
    document.getElementById('workerForm').addEventListener('submit', async
function(event) {
      event.preventDefault();
      const form = event.target;
      const username = form.username.value;
      const password = form.password.value;
      if (!username || !password) {
        alert('Please enter both username and password.');
        return;
      }
      try {
        const response = await fetch('/worker_login', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({ username, password })
        });
        const data = await response.json();
        if (response.ok) {
          alert(data.message || 'Login successful');
          window.location.href = data.redirect_url || '/worker_dashboard';
        } else {
          alert(data.error || 'Login failed');
        }
      } catch (error) {
        alert('Error connecting to server');
        console.error('Worker login error:', error);
      }
    });
  </script>
```

```
  <script>
    // Handle admin login form submission
    document.getElementById('adminForm').addEventListener('submit', async
function(event) {
      event.preventDefault();
      const form = event.target;
      const username = form.username.value;
      const password = form.password.value;
      if (!username || !password) {
        alert('Please enter both username and password.');
        return;
      }
      try {
        const response = await fetch('/login', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({ username, password })
        });
        const data = await response.json();
        if (response.ok) {
          alert(data.message || 'Login successful');
          window.location.href = data.redirect_url || '/login_roles';
        } else {
          alert(data.error || 'Login failed');
        }
      } catch (error) {
        alert('Error connecting to server');
        console.error('Admin login error:', error);
      }
    });
  </script>
</body>
</html>
```

**report.html :This code snippet is used to design the report form.**

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1" />
   <title>Report an Issue</title>
   <style>
    body {
      margin: 0;
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
      background: #f5f5f5;
      color: #333;
    }
    .container {
      display: flex;
      width: 100%;
      height: 100vh;
      background: #f5f5f5;
    }
    .sidebar {
    width: 220px;
    background-color: #2a4a8a;
    color: white;
    padding: 20px;
    box-sizing: border-box;
    }
    .sidebar nav ul {
      list-style: none;
      padding: 0;
      margin: 0;
    }
    .sidebar nav ul li {
```

```
  margin: 0;
}
.sidebar nav ul li a {
  display: block;
  padding: 15px 20px;
  color: #333;
  text-decoration: none;
  font-weight: 600;
  border-left: 4px solid transparent;
  transition: background 0.3s ease, border-color 0.3s ease;
}
.sidebar nav ul li a:hover {
  background: #e6e6e6;
}
.sidebar nav ul li a.active {
  background: #2a4a8a;
  color: white;
  border-left-color: #4a6edb;
}
.main-content {
  flex-grow: 1;
  padding: 40px 50px;
  overflow-y: auto;
  box-sizing: border-box;
}
form {
  max-width: 700px;
  margin: 0 auto;
  background: white;
  padding: 30px;
  border-radius: 15px;
  box-shadow: 0 8px 20px rgba(0,0,0,0.1);
  display: flex;
```

```
  flex-direction: column;
  gap: 20px;
}
label {
  font-weight: 600;
  margin-bottom: 8px;
  color: #555;
}
input[type="text"],
input[type="email"],
input[type="tel"],
select,
textarea {
  padding: 10px 15px;
  border-radius: 10px;
  border: 1px solid #ccc;
  font-size: 16px;
  box-sizing: border-box;
  width: 100%;
  transition: border-color 0.3s ease;
}
input[type="text"]:focus,
input[type="email"]:focus,
input[type="tel"]:focus,
select:focus,
textarea:focus {
  border-color: #2a4a8a;
  outline: none;
  box-shadow: 0 0 8px 2px #a18cd1;
}
button {
  background: #2a4a8a;
  color: white;
```

```
      border: none;
      padding: 14px 30px;
      border-radius: 50px;
      font-weight: 700;
      font-size: 18px;
      cursor: pointer;
      align-self: center;
      transition: background 0.3s ease;
    }
    button:hover {
      background: #1f3a6a;
    .report-image-preview {
      max-width: 100%;
      max-height: 300px;
      margin-top: 10px;
      border-radius: 10px;
      object-fit: contain;
    }
  </style>
</head>
<body>
  <div class="container">
    <aside class="sidebar">
      <nav>
        <ul>
          <li class="active"><a href="#">Dashboard</a></li>
          <li><a href="/report">Issue a Report</a></li>
          <li><a href="/track_report">Track Your Report</a></li>
          <li><a href="/profile">Profile Info</a></li>
          <li><a href="/logout">Logout</a></li>
        </ul>
      </nav>
    </aside>
```

```
<main class="main-content">
  <form id="reportForm" enctype="multipart/form-data">
    <h2>Report an Issue in Your Surroundings</h2>
    <label for="issue_type">Issue Type:</label>
    <select id="issue_type" name="issue_type" required>
      <option value="">Select an issue</option>
      <option value="Street Light">Street Light</option>
      <option value="Road Damage">Road Damage</option>
      <option value="Garbage">Garbage</option>
      <option value="Water Leakage">Water Leakage</option>
      <option value="Other">Other</option>
    </select>
    <input type="text" id="custom_issue" name="custom_issue" placeholder="Please specify the issue" style="display:none; margin-top:10px; padding: 10px; width: 100%; border-radius: 10px; border: none; background: rgba(255, 255, 255, 0.15); color: #000;" />
    <label for="description">Description:</label>
    <textarea id="description" name="description" rows="4" cols="50" placeholder="Describe the issue..." required></textarea>
    <label for="location">Your Location:</label>
    <input type="text" id="location" name="location" placeholder="Enter your location" required />
    <label for="worker_list">Select Worker: </label>
    <select id="worker_list" name="worker_list" required>
      <option value="">Select a worker</option>
      <option value="road_damage">John Doe</option>
      <option value="streetlight">Jane Smith</option>
      <option value="garbage">Raghav Das</option>
      <option value="water_leakage">Michael Brown</option>
      <option value="other">Bhanu Rai</option>
    </select>
    <label for="image">Upload Image (optional):</label>
    <input type="file" id="image" name="image" accept="image/*" />
    <button type="submit">Submit Report</button>
  </form>
```

```
    </main>
  </div>
    <script src="{{ url_for('static', filename='report.js') }}"></script>
</body>
</html>
```

**admin_dasboard.html: This code snipped is used for designing the admin dashboard so that admin can manage and see the interaction between worker and citizen.**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Admin Dashboard</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='admin.css') }}" />
</head>
<body>
  <div class="dashboard-container">
    <aside class="sidebar">
      <h2>Admin Dashboard</h2>
      <nav>
        <ul>
          <li class="active"><a href="/admin_dashboard">Admin Dashboard</a></li>
          <li><a href="/activity_timeline">Activity Timeline</a></li>
          <li><a href="/worker_response_dashboard">Worker Response Dashboard</a></li>
          <li><a href="/profile">Profile Info</a></li>
          <li><a href="/logout">Logout</a></li>
        </ul>
      </nav>
    </aside>
    <main class="main-content">
      <h1>All Interactions</h1>
      <table class="interaction-table">
```

```
        <thead>
          <tr>
            <th>Report ID</th>
            <th>Citizen Username</th>
            <th>Issue Type</th>
            <th>Description</th>
            <th>Assigned Worker</th>
            <th>Status</th>
            <th>Timestamp</th>
            <th>Completion Notes</th>
          </tr>
        </thead>
        <tbody>
          {% for interaction in interactions %}
          <tr>
            <td>{{ interaction.report_id }}</td>
            <td>{{ interaction.citizen_username }}</td>
            <td>{{ interaction.issue_type }}</td>
            <td>{{ interaction.description }}</td>
            <td>{{ interaction.worker_username if interaction.worker_username else 'N/A'
}}</td>
            <td>{{ interaction.status }}</td>
            <td>{{ interaction.timestamp }}</td>
            <td>{{ interaction.completion_notes if interaction.completion_notes else '' }}</td>
          </tr>
          {% else %}
          <tr>
            <td colspan="8">No interactions found.</td>
          </tr>
          {% endfor %}
        </tbody>
      </table>
    </main>
```

```
  </div>
</body>
</html>
```

**app.py: It helps in controlling the flask server**

```
from flask import Flask, request, jsonify, render_template, abort
from flask_mysqldb import MySQL
from werkzeug.security import generate_password_hash, check_password_hash
from flask_cors import CORS
from werkzeug.utils import secure_filename
import os
from datetime import datetime
import logging
app = Flask(__name__)
app.debug = True
@app.route('/register', methods=['POST'])
def register():
    data = request.get_json()
    if not data:
        return jsonify({'error': 'Invalid JSON data'}), 400
    username = data.get('username')
    email = data.get('email')
    password = data.get('password')
    first_name = data.get('first_name')
    last_name = data.get('last_name')
    phone = data.get('phone')
    gender = data.get('gender')
    if not all([username, email, password, first_name, last_name, phone, gender]):
        return jsonify({'error': 'Missing required fields'}), 400
    try:
        cur = mysql.connection.cursor()
        # Check if user already exists
        cur.execute("SELECT id FROM users WHERE email = %s", (email,))
```

```
        existing_user = cur.fetchone()
        if existing_user:
            cur.close()
            return jsonify({'error': 'User already exists'}), 409
        hashed_password = generate_password_hash(password)
        cur.execute("""
            INSERT INTO users (username, email, password, first_name, last_name, phone,
gender)
            VALUES (%s, %s, %s, %s, %s, %s, %s)
        """, (username, email, hashed_password, first_name, last_name, phone, gender))
        mysql.connection.commit()
        cur.close()
        return jsonify({'message': 'Registration successful'}), 201
    except Exception as e:
        app.logger.error(f"Error in register: {e}", exc_info=True)
        return jsonify({'error': 'Internal server error', 'details': str(e)}), 500
from flask import session
@app.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    if not data:
        return jsonify({'error': 'Invalid JSON data'}), 400
    username = data.get('username')
    password = data.get('password')

    if not username or not password:
        return jsonify({'error': 'Missing username or password'}), 400
    try:
        cur = mysql.connection.cursor()
        cur.execute("SELECT id, password, role FROM users WHERE username = %s",
(username,))
        user = cur.fetchone()
        cur.close()
```

```
        if user and check_password_hash(user[1], password):
            session['user_id'] = user[0]
            user_role = user[2] if len(user) > 2 else 'user'
            if user_role == 'admin':
                return jsonify({'message': 'Login successful', 'redirect_url': '/admin_dashboard'}), 200
            else:
                return jsonify({'message': 'Login successful', 'redirect_url': '/citizen_dashboard'}), 200
        else:
            return jsonify({'error': 'Invalid username or password'}), 401
    except Exception as e:
        app.logger.error(f"Error in login: {e}", exc_info=True)
        return jsonify({'error': 'Internal server error', 'details': str(e)}), 500
@app.route('/citizen_dashboard')
def citizen_dashboard():
    return render_template('citizen_dashboard.html')
@app.route('/report', methods=['POST', 'GET'])
def submit_report():
    if request.method == 'GET':
        return render_template('report.html')
    try:
        if 'image' not in request.files:
            return jsonify({'error': 'No image part'}), 400
        image = request.files['image']
        issue_type = request.form.get('issue_type')
        if issue_type:
            issue_type = issue_type.strip().lower().replace(" ", "")
        description = request.form.get('description')
        selected_worker_id = request.form.get('worker_list')
        if not issue_type or not description:
            return jsonify({'error': 'Please provide issue type and description'}), 400
        if image and allowed_file(image.filename):
            filename = secure_filename(image.filename)
            timestamp_str = datetime.now().strftime("%Y%m%d%H%M%S")
```

```
            filename = f"{timestamp_str}_{filename}"
            image.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
            image_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        else:
            image_path = None
        cur = mysql.connection.cursor()
        assigned_worker_id = None
        if selected_worker_id:
            # Validate selected worker exists
            cur.execute("SELECT id FROM workers WHERE id = %s", (selected_worker_id,))
            worker = cur.fetchone()
            if worker:
                assigned_worker_id = worker[0]
            else:
                # fallback to automatic assignment
                pass
        if not assigned_worker_id:
            cur.execute("SELECT id FROM workers WHERE
REPLACE(LOWER(TRIM(issue_type)), ' ', '') = %s LIMIT 1", (issue_type,))
            worker = cur.fetchone()
            assigned_worker_id = worker[0] if worker else None
            if not assigned_worker_id:
                # Assign default worker if no match found
                cur.execute("SELECT id FROM workers LIMIT 1")
                default_worker = cur.fetchone()
                assigned_worker_id = default_worker[0] if default_worker else None
        user_id = session.get('user_id')
        cur.execute("""
            INSERT INTO reports (issue_type, description, image_path, timestamp,
assigned_worker_id, user_id)
            VALUES (%s, %s, %s, %s, %s, %s)
        """, (issue_type, description, image_path, datetime.now(), assigned_worker_id,
user_id))
```

```
        mysql.connection.commit()
        report_id = cur.lastrowid
        cur.close()
        return jsonify({'message': 'Report submitted successfully', 'id': report_id}), 201
    except Exception as e:
        app.logger.error(f"Error in submit_report: {e}", exc_info=True)
        return jsonify({'error': 'Internal server error', 'details': str(e)}), 500
@app.route('/workers', methods=['GET'])
def get_workers():
    location = request.args.get('location')
    # issue_type = request.args.get('issue_type')  # Removed because column does not exist
    try:
        cur = mysql.connection.cursor()
        query = "SELECT id, name, location FROM workers WHERE 1=1"
        params = []
        if location:
            query += " AND location = %s"
            params.append(location)
        # Removed issue_type filtering
        cur.execute(query, tuple(params))
        workers = cur.fetchall()
        cur.close()
        workers_list = []
        for worker in workers:
            workers_list.append({
                'id': worker[0],
                'name': worker[1],
                'location': worker[2]
            })
        return jsonify(workers_list), 200
    except Exception as e:
        app.logger.error(f"Error fetching workers: {e}", exc_info=True)
        return jsonify({'error': 'Internal server error', 'details': str(e)}), 500
```

```
@app.route('/track_report')
def track_report():
    user_id = session.get('user_id')
    if not user_id:
        return redirect(url_for('login_roles'))
    reports = []
    try:
        cur = mysql.connection.cursor()
        # Fetch all reports submitted by the logged-in user
        cur.execute("""SELECT r.id, r.issue_type, r.description, r.image_path, r.timestamp,
r.status, r.assigned_worker_id, COALESCE(w.name, w.username) as assigned_worker
            FROM reports r LEFT JOIN workers w ON r.assigned_worker_id = w.id
            WHERE r.user_id = %s ORDER BY r.timestamp DESC""", (user_id,))
        rows = cur.fetchall()
        for row in rows:
            formatted_timestamp = row[4].strftime("%Y-%m-%d %H:%M:%S") if row[4] else
"N/A"
            app.logger.info(f"track_report: report_id={row[0]}, assigned_worker_id={row[6]},
assigned_worker_username={row[7]}")
            reports.append({
                'id': row[0],
                'issue_type': row[1],
                'description': row[2],
                'image_path': row[3],
                'timestamp': formatted_timestamp,
                'status': row[5] if row[5] else 'Pending',
                'assigned_worker': row[7] if row[7] else 'N/A'
            })
        cur.close()
    except Exception as e:
        app.logger.error(f"Error fetching reports for user {user_id}: {e}", exc_info=True)
    app.logger.info(f"track_report: user_id from session: {user_id}, reports count:
{len(reports)}")
    return render_template('track_report.html', reports=reports)
```

```
from flask import session, redirect, url_for
@app.route('/worker_dashboard')
def worker_dashboard():
    user_id = session.get('user_id')
    if not user_id:
        return redirect(url_for('login_roles'))
    return render_template('worker_dashboard.html')
@app.route('/login_roles')
def login_roles():
    return render_template('login_roles.html')
from flask import session, jsonify, request
@app.route('/api/worker/tasks', methods=['GET'])
def get_worker_tasks():
    user_id = session.get('user_id')
    app.logger.info(f"get_worker_tasks called with session user_id: {user_id}")
    if not user_id:
        return jsonify({'error': 'Unauthorized'}), 401
    try:
        cur = mysql.connection.cursor()
        # Fetch worker by id directly using session user_id
        cur.execute("SELECT id FROM workers WHERE id = %s", (user_id,))
        worker = cur.fetchone()
        if not worker:
            cur.close()
            return jsonify({'error': 'Worker not found'}), 404
        worker_id = worker[0]
        cur.execute("""SELECT r.id, r.issue_type, r.description, r.image_path, r.timestamp,
r.status, r.completion_notes, w.name FROM reports r
            LEFT JOIN workers w ON r.assigned_worker_id = w.id
            WHERE r.assigned_worker_id = %sORDER BY r.timestamp DESC
        """, (worker_id,))
        reports = cur.fetchall()
        app.logger.info(f"Number of tasks fetched: {len(reports)}")
```

```
        cur.close()
        tasks = []
        for report in reports:
            tasks.append({
                'id': report[0],
                'issue_type': report[1],
                'description': report[2],
                'image_path': report[3],
                'timestamp': report[4].strftime("%Y-%m-%d %H:%M:%S") if report[4] else "N/A",
                'status': report[5] if report[5] else 'Pending',
                'completion_notes': report[6] if report[6] else '',
                'assigned_worker_name': report[7] if report[7] else 'N/A'
            })
        return jsonify(tasks), 200
    except Exception as e:
        app.logger.error(f"Error fetching worker tasks: {e}", exc_info=True)
        return jsonify({'error': 'Internal server error'}), 500
@app.route('/api/worker/task/<int:task_id>/update_status', methods=['POST'])
def update_task_status(task_id):
    user_id = session.get('user_id')
    if not user_id:
        return jsonify({'error': 'Unauthorized'}), 401
    data = request.get_json()
    new_status = data.get('status')
    if not new_status:
        return jsonify({'error': 'Missing status'}), 400
    try:
        cur = mysql.connection.cursor()
        cur.execute("SELECT id FROM workers WHERE user_id = %s", (user_id,))
        worker = cur.fetchone()
        if not worker:
            cur.close()
            return jsonify({'error': 'Worker not found'}), 404
```

```
        worker_id = worker[0]
        cur.execute("SELECT assigned_worker_id FROM reports WHERE id = %s",
(task_id,))
        report = cur.fetchone()
        if not report or report[0] != worker_id:
            cur.close()
            return jsonify({'error': 'Task not found or unauthorized'}), 404
        cur.execute("UPDATE reports SET status = %s WHERE id = %s", (new_status,
task_id))
        mysql.connection.commit()
        cur.close()
        return jsonify({'message': 'Status updated successfully'}), 200
    except Exception as e:
        app.logger.error(f"Error updating task status: {e}", exc_info=True)
        return jsonify({'error': 'Internal server error'}), 500
@app.route('/api/worker/task/<int:task_id>/complete', methods=['POST'])
def complete_task(task_id):
    user_id = session.get('user_id')
    if not user_id:
        return jsonify({'error': 'Unauthorized'}), 401
    data = request.get_json()
    completion_notes = data.get('completion_notes')
    try:
        cur = mysql.connection.cursor()
        # Verify worker owns the task
        cur.execute("SELECT id FROM workers WHERE user_id = %s", (user_id,))
        worker = cur.fetchone()
        if not worker:
            cur.close()
            return jsonify({'error': 'Worker not found'}), 404
        worker_id = worker[0]
        cur.execute("SELECT assigned_worker_id FROM reports WHERE id = %s",
(task_id,))
```

```
        report = cur.fetchone()
        if not report or report[0] != worker_id:
            cur.close()
            return jsonify({'error': 'Task not found or unauthorized'}), 404
        cur.execute("""UPDATE reports
            SET completion_notes = %s, status = 'resolved'
            WHERE id = %s
        """, (completion_notes, task_id))
        mysql.connection.commit()
        cur.close()
        return jsonify({'message': 'Report completion submitted successfully'}), 200
    except Exception as e:
        app.logger.error(f"Error submitting report completion: {e}", exc_info=True)
        return jsonify({'error': 'Internal server error'}), 500
from flask import session, redirect, url_for, request, jsonify
@app.route('/worker_login', methods=['POST'])
def worker_login():
    data = request.get_json()
    if not data:
        return jsonify({'error': 'Invalid JSON data'}), 400
    username = data.get('username')
    password = data.get('password')
    if not username or not password:
        return jsonify({'error': 'Missing username or password'}), 400
    try:
        cur = mysql.connection.cursor()
        cur.execute("SELECT id, password FROM workers WHERE username = %s",
(username,))
        worker = cur.fetchone()
        cur.close()
        if worker and check_password_hash(worker[1], password):
            session['user_id'] = worker[0]
            app.logger.info(f"Worker {username} logged in with user_id {worker[0]}")
```

```
            return jsonify({'message': 'Worker login successful', 'redirect_url':
'/worker_dashboard'}), 200
        else:
            return jsonify({'error': 'Invalid username or password'}), 401
    except Exception as e:
        app.logger.error(f"Error in worker_login: {e}", exc_info=True)
        return jsonify({'error': 'Internal server error', 'details': str(e)}), 500
@app.route('/profile')
def profile():
    user_id = session.get('user_id')
    app.logger.info(f"Accessing profile page for user_id: {user_id}")
    if not user_id:
        return redirect(url_for('login_roles'))  # Redirect to login if not logged in
    try:
        cur = mysql.connection.cursor()
        cur.execute("SELECT username, email, first_name, last_name, phone FROM users
WHERE id = %s", (user_id,))
        user = cur.fetchone()
        app.logger.info(f"Database query result for user_id {user_id}: {user}")
        cur.close()
        if not user:
            return "User not found", 404
        profile_data = {
            'username': user[0],
            'email': user[1],
            'first_name': user[2],
            'last_name': user[3],
            'phone': user[4]
        }
        return render_template('profile.html', user=profile_data)
    except Exception as e:
        app.logger.error(f"Error loading profile: {e}", exc_info=True)
        return "Internal server error", 500
```

```
from flask import jsonify
@app.route('/debug/users_columns')
def debug_users_columns():
    try:
        cur = mysql.connection.cursor()
        cur.execute("""
            SELECT COLUMN_NAME
            FROM information_schema.columns
            WHERE table_schema = %s AND table_name = 'users'
        """, (app.config['MYSQL_DB'],))
        columns = [row[0] for row in cur.fetchall()]
        cur.close()
        return jsonify({'users_table_columns': columns})
    except Exception as e:
        app.logger.error(f"Error fetching users table columns: {e}", exc_info=True)
        return jsonify({'error': 'Internal server error'}), 500
@app.route('/debug/workers_columns')
def debug_workers_columns():
    try:
        cur = mysql.connection.cursor()
        cur.execute("""
            SELECT COLUMN_NAME
            FROM information_schema.columns
            WHERE table_schema = %s AND table_name = 'workers'
        """, (app.config['MYSQL_DB'],))
        columns = [row[0] for row in cur.fetchall()]
        cur.close()
        return jsonify({'workers_table_columns': columns})
    except Exception as e:
        app.logger.error(f"Error fetching workers table columns: {e}", exc_info=True)
        return jsonify({'error': 'Internal server error'}), 500
@app.route('/debug/worker_reports/<int:worker_id>')
def debug_worker_reports(worker_id):
```

```
    try:
        cur = mysql.connection.cursor()
        cur.execute("""
            SELECT id, issue_type, description, image_path, timestamp, status
            FROM reports
            WHERE assigned_worker_id = %s
        """, (worker_id,))
        reports = cur.fetchall()
        cur.close()
        tasks = []
        for report in reports:
            tasks.append({
                'id': report[0],
                'issue_type': report[1],
                'description': report[2],
                'image_path': report[3],
                'timestamp': report[4].strftime("%Y-%m-%d %H:%M:%S") if report[4] else "N/A",
                'status': report[5] if report[5] else 'Pending'
            })
        return jsonify(tasks), 200
    except Exception as e:
        app.logger.error(f"Error fetching worker reports: {e}", exc_info=True)
        return jsonify({'error': 'Internal server error'}), 500
@app.route('/debug/all_reports')
def debug_all_reports():
    try:
        cur = mysql.connection.cursor()
        cur.execute("""
            SELECT id, issue_type, assigned_worker_id
            FROM reports""")
        reports = cur.fetchall()
        cur.close()
        reports_list = []
```

```
        for report in reports:
            reports_list.append({ 'id': report[0], 'issue_type': report[1],'assigned_worker_id':
report[2] })
        return jsonify(reports_list), 200
    except Exception as e:
        app.logger.error(f"Error fetching all reports: {e}", exc_info=True)
        return jsonify({'error': 'Internal server error'}), 500
@app.route('/debug/all_workers')
def debug_all_workers():
    try:
        cur = mysql.connection.cursor()
        cur.execute("""
            SELECT id, username, issue_type
            FROM workers
        """)
        workers = cur.fetchall()
        cur.close()
        workers_list = []
        for worker in workers:
            workers_list.append({
                'id': worker[0],
                'username': worker[1],
                'issue_type': worker[2]
            })
        return jsonify(workers_list), 200
    except Exception as e:
        app.logger.error(f"Error fetching all workers: {e}", exc_info=True)
        return jsonify({'error': 'Internal server error'}), 500

@app.route('/debug/worker/<int:worker_id>')
def debug_worker(worker_id):
    try:
        cur = mysql.connection.cursor()
```

```
        cur.execute("SELECT id, username, issue_type FROM workers WHERE id = %s",
(worker_id,))
        worker = cur.fetchone()
        cur.close()
        if worker:
            return jsonify({'id': worker[0], 'username': worker[1], 'issue_type': worker[2]}), 200
        else:
            return jsonify({'error': 'Worker not found'}), 404
    except Exception as e:
        app.logger.error(f"Error fetching worker {worker_id}: {e}", exc_info=True)
        return jsonify({'error': 'Internal server error'}), 500
@app.route('/debug/reports_assigned_to_worker/<int:worker_id>')
def debug_reports_assigned_to_worker(worker_id):
    try:
        cur = mysql.connection.cursor()
        cur.execute("""
            SELECT id, issue_type, description, assigned_worker_id
            FROM reports
            WHERE assigned_worker_id = %s
        """, (worker_id,))
        reports = cur.fetchall()
        cur.close()
        reports_list = []
        for report in reports:
            reports_list.append({
                'id': report[0],
                'issue_type': report[1],
                'description': report[2],
                'assigned_worker_id': report[3]
            })
        return jsonify(reports_list), 200
    except Exception as e:
        app.logger.error(f"Error fetching reports for worker {worker_id}: {e}", exc_info=True)
```

```
        return jsonify({'error': 'Internal server error'}), 500
from flask import session, redirect, url_for
@app.route('/admin_dashboard')
def admin_dashboard():
    interactions = []
    try:
        cur = mysql.connection.cursor()
        # Fetch all reports with citizen username and assigned worker name
        cur.execute("""SELECT r.id, u.username, r.issue_type, r.description, w.name, r.status,
r.timestamp, r.completion_notes FROM reports r LEFT JOIN users u ON r.user_id = u.id
            LEFT JOIN workers w ON r.assigned_worker_id = w.id
            ORDER BY r.timestamp DESC""")
        rows = cur.fetchall()
        for row in rows:
            formatted_timestamp = row[6].strftime("%Y-%m-%d %H:%M:%S") if row[6] else "N/A"
            interactions.append({
                'report_id': row[0],
                'citizen_username': row[1] if row[1] else 'N/A',
                'issue_type': row[2],
                'description': row[3],
                'worker_name': row[4],
                'status': row[5] if row[5] else 'Pending',
                'timestamp': formatted_timestamp,
                'completion_notes': row[7] if row[7] else ''
            })
        cur.close()
    except Exception as e:
        app.logger.error(f"Error fetching data for admin dashboard: {e}", exc_info=True)
    return render_template('admin_dashboard.html', interactions=interactions)
@app.route('/activity_timeline')
def activity_timeline():
    activity_timeline = []
    try:
```

```
        cur = mysql.connection.cursor()
        cur.execute("""SELECT r.id, r.status, r.timestamp, w.username FROM reports r
            LEFT JOIN workers w ON r.assigned_worker_id = w.id
            ORDER BY r.timestamp DESC""")
        rows = cur.fetchall()
        for row in rows:
            report_id = row[0]
            reported_time = row[2]
            status = row[1] if row[1] else 'Pending'
            if reported_time:
                activity_timeline.append({
                    'report_id': report_id,
                    'event': 'Reported',
                    'timestamp': reported_time.strftime("%Y-%m-%d %H:%M:%S")
                })
            activity_timeline.append({'report_id': report_id, 'event': 'Admin Approved',
                'timestamp': reported_time.strftime("%Y-%m-%d %H:%M:%S") if reported_time
else 'N/A'
            })
            if row[3]:
                activity_timeline.append({report_id': report_id,'event': f'Worker Assigned:
{row[3]}','timestamp': reported_time.strftime("%Y-%m-%d %H:%M:%S") if reported_time
else 'N/A'})
            if status.lower() == 'resolved':
                activity_timeline.append({'report_id': report_id,'event': 'Worker Resolved',
                    'timestamp': reported_time.strftime("%Y-%m-%d %H:%M:%S") if
reported_time else 'N/A' })
        cur.close()
    except Exception as e:
        app.logger.error(f"Error fetching activity timeline data: {e}", exc_info=True)
    return render_template('activity_timeline.html', activity_timeline=activity_timeline)
@app.route('/worker_response_dashboard')
def worker_response_dashboard():
    worker_responses = []
```

```
    try:
        cur = mysql.connection.cursor()
        cur.execute(""" SELECT w.id, w.username, COUNT(r.id) AS total_assigned,
            SUM(CASE WHEN r.status != 'resolved' THEN 1 ELSE 0 END) AS pending_work
            FROM workers w
            LEFT JOIN reports r ON w.id = r.assigned_worker_id
            GROUP BY w.id, w.username """)
        workers = cur.fetchall()
        for worker in workers:
            worker_responses.append({
                'worker_id': worker[0],
                'worker_username': worker[1],
                'total_assigned': worker[2],
                'pending_work': worker[3]
            })
        cur.close()
    except Exception as e:
    app.logger.error(f"Error fetching worker response dashboard data: {e}", exc_info=True)
    return render_template('worker_response_dashboard.html', worker_responses=worker_responses)
if __name__ == '__main__':
    app.run(debug=True)
```

# Chapter 4

# TESTING

## 4.1 INTRODUCTION:

Software testing is the process used to help identify the correctness, completeness, security and quality of developed computer software. This includes the process of executing the program or application with the intent of finding errors. Quality is not an absolute; it is value to some person. With that in mind testing can never completely establish the correctness of arbitrary computer software; testing furnishes a criticism or comparison that compares the state and behavior of the product against a specification.

The testing phase consists of evaluating the software that has been developed in order to conform that it produces the output required in a safe and efficient manner. In this phase inherent errors that occur, have to be handled and the user should be informed so that he/she can follow the guidelines and instructions and get around the error and obtain the output.

During testing, the program to be tested is executed with a set of test cases and the output of the program for the test cases is evaluated to determine if the program is performing as expected. Due to its approach, dynamic testing can only ascertain the presence of errors in the program the exact nature of the errors is not usually decided by testing.

Testing forms the first step in determining the errors in a program. Clearly the success of testing in revealing errors in programs depends critically on the test cases. Because code is the only product that can be executed and whose actual behavior can be observed, testing is the phase where the errors remaining from all the previous phases must be detected.

The program to be tested is executed with a set of test cases and the output of the program for the test cases are evaluated to determine if the programming is performing as expected. Testing forms the first step in determining errors in a program. The success of testing in reveling errors in programs depends critically on the test cases.

## 4.2 OBJECTIVES OF TESTING

The objectives of testing are:

- Testing is a process of executing a program with the intent of finding errors.

A successful test case is one that discovers an as of yet and discovered error. System testing is a stage of implementation which is aimed at ensuring that the system works accurately

and efficiently as per the user need, before the live operation commences as stated before, testing is vital to the success of a system system testing makes a logical assumption that if all parts of system are correct the goal will successfully be achieved. A series of tests are performed before the system is ready for the user acceptance test.

## THERE ARE TWO TYPES OF SOFTWARE TESTING:

- **Black Box Testing**: Internal system design is not considered in this type of testing. Tests are based on requirements and functionality.
- **White box testing**: This testing is based on knowledge of the internal logic of an application's code. Also known as glass box testing. Internal software and code working should be known for this type of testing. Tests are based on coverage of code statements, branches, paths and conditions.

A test case is a software testing document, which consists of event, action, input, output, expected result and actual result. Clinically defined a test case is an input and an expected result. This can be pragmatic as 'for condition x your derived result is y'; whereas other test cases described in more detail the input scenario and what results might be expected. It can occasionally be a series of steps but one with expected results or expected outcome. A test case should also contain a place for the actual result. White box testing is applicable at the unit, integration and system levels of the software testing.

## 4.3 TESTING METHODOLOGY

The different types of testing are as follows:

### 4.3.1 UNIT TESTING:

Unit testing focuses efforts on the smallest unit of software design this is known as module testing or white box testing, the modules are tested separately. The test is carried out during the programming stage itself. In this step, each module is found to be working satisfactorily as regards to the expected output from the module.

### 4.3.2 INTEGRATION TESTING:

In integration testing the different units of the system are integrated together to form the complete system. This type of testing checks the system as a whole to ensure that it is doing what it's supposed to do the testing of an integrated system can be carried out top-down, bottom-up or Big-Bang. In this type of testing some parts are tested with white box testing and some with black box testing techniques. This type of testing plays a very important role in increasing systems productivity. We have checked the system by using integration testing techniques.

### 4.3.3 SYSTEM TESTING:

Apart from testing the system to validate the functionality of software against the requirements. It is also necessary to test the non-functional aspect of the system, some examples of non-functional tools include tests to check the performance data security usability volume load and stress that we have used in a project to test the various module. System testing consists of the following steps:

- Program(s) testing
- String testing
- System Testing
- System Documentation
- User Acceptance test

### 4.3.4 FIELD TESTING:

The special type of testing may be very important in some projects. Here the system is tested in actual operational surroundings the interfaces with other systems and the real worlds are checked. This type of testing is very rarely used so far our project is concerned we haven't tested a project using field testing.

### 4.3.5 ACCEPTANCE TESTING:

After the developer has completed all rounds of testing and he is satisfied with the system, then the user takes over and retest system from his point of view to judge whether it is acceptable according to some previously identified criteria. This is almost always a tricky situation in the project because of the inherent conflict between the developer and the user in this project. It is the job of the developer of the planner to check the system that whether the system fulfills the goals or not.

## 4.4 TESTING CRITERIA:

### 4.4.1 Testing for valid user name:

| Test case | Input | Test description | Output |
|---|---|---|---|
| 1 | User name starts with number | User name cannot start with number | Must Enter Characters |
| 2 | User name is left blank | User name cannot be left blank | Must Enter username |

### 4.4.2 Testing for valid password:

| Test case | Input | Test description | Output |
|---|---|---|---|
| 1 | Password is left blank | Password cannot be blank | Must Enter password |
| 2 | Invalid password entered | Valid password must be entered | Password mismatch |

### 4.4.3 Testing for valid Email address:

| Test case | Input | Test description | Output |
|---|---|---|---|
| 1 | Email address is not in Correct format | Email address Should have Correct format | Invalid Expression |
| 2 | Email address with space | Email address cannot have space | Invalid Expression |
| 3 | Email is left blank | Email cannot be blank | Must Enter Email ID |

**4.4.4 Testing for data insertion:**

| Test case | Input | Test description | Output |
|---|---|---|---|
| 1 | Mandatory fields left empty | Mandatory fields cannot be left empty | Must enter data |

**4.4.5 Testing for phone number:**

| Test case | Input | Test description | Output |
|---|---|---|---|
| 1 | Phone number entered with alphabets | Phone number cannot have alphabets | Enter only numbers |
| 2 | Phone number entered is more than 10 digits | Phone number with more than 10 digits cannot be entered | Invalid phone number |

**4.4.6 Other cases:**

| Test case | Input | Test description | Output |
|---|---|---|---|
| 1 | Click on submit | Submitting reports | Report is submitted successfully |
| 2 | Click on Drop down list | Selecting data | Data is displayed |
| 3 | Click on logout | Logging out! | Redirected back to login page |
| 4 | Click on login | User has logged in successfully | Redirects to respective page. |

# Chapter 5

## CONCLUSION

The development of ReportMate has provided a meaningful solution to a real-world problem faced by many communities—inefficient and delayed responses to civic issues. This web-based application serves as a bridge between citizens and authorities by offering a simple, accessible platform for reporting problems such as road damage, garbage accumulation, broken street lights, and water leakage.

By using powerful technologies like Python with Flask for backend processing, MySQL for secure data storage, and standard frontend technologies like HTML, CSS, and JavaScript, the project achieves seamless functionality and responsiveness. The system allows users to submit detailed reports with location and optional images, while the admin panel provides features for reviewing, approving/declining, and assigning workers based on skill and area. The admin login module adds a security layer, ensuring only authorized personnel can manage the system.

The report tracking mechanism improves transparency and boosts user confidence by allowing citizens to check the status of their complaints. The inclusion of dynamic worker suggestions based on the location and issue type enhances the resolution process by directing problems to the most suitable workers.

Overall, ReportMate is a scalable and adaptable platform that can be extended with mobile applications, real-time notifications, multilingual support, and advanced analytics. It has the potential to be integrated with municipal systems or smart city projects to improve urban infrastructure management. The project has not only fulfilled its academic goals but also opened up possibilities for real-world implementation and future development in the civic-tech domain.

# Chapter 6

# BIBILOGRAPHY

## BOOKS:


- **Flask Web Development**-Miguel Grinberg
- **JavaScript Programing-** Elisabeth Robson
- **HTML,CSS & JavaScript**-Laura Lemay, Rafe Colburn, Jenifer Kyrlin


## WEBSITES:


- **MySQL Documentation** –
https://dev.mysql.com/doc
*Reference for SQL syntax, database operations, and MySQL Workbench usage.*

- **W3Schools – HTML, CSS, JavaScript Tutorials** –
https://www.w3schools.com
*Used for designing the user interface and learning frontend basics.*

- **Visual Studio Code Documentation** –
https://code.visualstudio.com/docs
*Guide for using VS Code editor with Python and web development extensions.*

- **Bootstrap & CSS Styling References** *(if used)* –
https://getbootstrap.com
*Used for layout styling and responsive design support.*

- **Official Python Docs** –
https://docs.python.org/3/
*Used for learning Python syntax, file handling, and working with libraries.*