# WARUNA

(Water Assessment and Reporting in Unified Network Application)

**A PROJECT REPORT**

**Submitted by**

**SUJAY V KULKARNI**
**NIDHISHREE C**

*in partial fulfillment for the award of the degree*

*of*

**BUILDING APPS WITH PYTHON**

*in*

**COMPUTER SCIENCE ENGINEERING**


**School of Computer Science and Engineering**
**RV University**
**RV Vidyaniketan,8th Mile, Mysuru Road, Bengaluru, Karnataka, India - 562112**

**JUNE 2024**

# DECLARATION

I, **SUJAY V KULKARNI (1RVU23CSE482),** student of second semester B. Tech in **Computer Science & Engineering,** at School of Computer Science and Engineering, **RV University,** hereby declare that the project work titled "**WARUNA**" has been carried out by us and submitted in partial fulfillment for the award of degree in **Bachelor of Technology in Computer Science & Engineering** during the academic year **2023-2024**. Further, the matter presented in the project has not been submitted previously by anybody for the award of any degree or any diploma to any other University, to the best of our knowledge and faith.

Name: SUJAY V KULKARNI Signature

USN: 1RVU23CSE482

Place: BANGALORE

Date: 4th JUNE 2024

**JUNE 2024**

# DECLARATION

I, **NIDHISHREE C (1RVU23CSE307),** student of second semester B. Tech in **Computer Science & Engineering,** at School of Computer Science and Engineering, **RV University,** hereby declare that the project work titled "**WARUNA**" has been carried out by us and submitted in partial fulfillment for the award of degree in **Bachelor of Technology in Computer Science & Engineering** during the academic year **2023-2024**. Further, the matter presented in the project has not been submitted previously by anybody for the award of any degree or any diploma to any other University, to the best of our knowledge and faith.

Name: NIDHISHREE C Signature

USN: 1RVU23CSE307

Place: BANGALORE

Date: 4th JUNE 2024




## School of Computer Science and Engineering

RV University
RV Vidyaniketan,8th Mile, Mysuru Road, Bengaluru, Karnataka, India - 562112

# CERTIFICATE

This is to certify that the project work titled **"WARUNA"** is performed by SUJAY V KULKARNI **(1RVU23CSE307)** and NIDHISHREE C **(1RVU23CSE307),** debonair students of Bachelor of Technology at the School of Computer Science and Engineering, RV university, Bengaluru in partial fulfillment for the award of degree Bachelor of Technology in Computer Science & Engineering , during the Academic year **2023-2024**.

| **Prof. Sritama Banerjee** | **Dr.Mydhili Nair** | **Dr. G Shobha** |
|---|---|---|
| Assistant Professor | Head of the Department | Dean |
| SOCSE | SOCSE | SOCSE |
| RV University | RV University | RV University |
| Date: | Date: | Date: |

Name of the Examiner Signature of Examiner

1.

2.

# ACKNOWLEDGEMENT

It is a great pleasure for us to acknowledge the assistance and support of a large number of individuals who have been responsible for the successful completion of this project work.

First, we take this opportunity to express our sincere gratitude to the School of Computer Science and Engineering, RV University, for providing us with a great opportunity to pursue our Bachelor's Degree in this institution.

In particular we would like to thank Dr. Sanjay R. Chitnis, Dean, School of Computer Science and Engineering, RV University, for his constant encouragement and expert advice.

It is a matter of immense pleasure to express our sincere thanks to Dr.Mydhili Nair, Head of the department, Computer Science & Engineering University, for providing right academic guidance that made our task possible.

We would like to thank our guide Prof. , Designation, Dept. of Computer Science & Engineering, RV University, for sparing his/her valuable time to extend help in every step of our project work, which paved the way for smooth progress and fruitful culmination of the project.

We are also grateful to our family and friends who provided us with every requirement throughout the course.

We would like to thank one and all who directly or indirectly helped us in completing the Project work successfully.

Date: 4th June 2024 Sujay V Kulkarni (1RVU23CSE482)

# TABLE OF CONTENTS

# ABSTRACT

WARUNA is an application designed to enhance water management by enabling public issue reporting, efficient problem resolution, and real-time IoT data integration. This improves transparency and operational efficiency for users and officials overseeing water resources.

WARUNA empowers public users to report water-related issues without logging in, including details such as location, address, landmarks, issue description, images, and notice date. Upon submission, users receive confirmation emails. The application features an interactive map interface, powered by the Leaflet library, which displays water supply structures and reported issues with custom markers. Users can zoom, pan, filter, and toggle map layers to view different aspects of the water supply network.

Officials benefit from comprehensive dashboards tailored to their roles. Managers can assign tasks to inspectors, manage and view reports, perform CRUD operations on water supply structures, and monitor IoT sensor data. Inspectors can view and update task statuses, submit detailed reports, and respond to real-time IoT sensor alerts. The integration of IoT technology allows for the simulation of real-time updates, monitoring key parameters like flow rates and pressure levels, and ensuring accurate sensor placement through geospatial data integration. The grievance redressal system allows users to track the status of their reported issues, while admins efficiently manage and resolve grievances. The geospatial database management system supports CRUD operations on water supply structures using SQLite. Geospatial data visualization tools generate interactive maps and heatmaps, helping to identify areas of concern within the water supply network.

Additionally, WARUNA provides downloadable graphs created with Plotly and Seaborn, offering insights into issue statuses and analytics over time. These features make WARUNA a robust solution for water issue reporting and management, integrating advanced geospatial and IoT technologies to enhance the efficiency and responsiveness of water supply systems, ultimately empowering both public users and officials to ensure sustainable water resource management.

# LIST OF FIGURES

## LIST OF ABBREVIATIONS

| Abbreviation | Explanation |
|---|---|
| WARUNA | Water Assessment and Reporting in Unified Network Application |
| HTML | Hyper Text Markup Language |
| CSS | Cascading Style Sheet |
| CSV | Comma Separated Values |
| IoT | Internet of Things |
| CRUD | Create Read Update Delete |

# 1. INTRODUCTION

## 1.1 - Problem Statement and Solution Overview

Water is a fundamental resource crucial for sustaining life, yet its management and conservation pose ongoing challenges. Effective communication and data-driven decision-making are paramount in addressing these challenges. On our website, we recognize the significance of these issues and are committed to providing solutions. This is where WARUNA comes in.
WARUNA, short for Water Assessment and Reporting in Unified Network Application, is our innovative solution designed to streamline water management processes. It enables efficient communication, data analysis, and informed decision-making, ultimately contributing to more sustainable water management practices.

This problem statement is a part of the Smart India Hackathon and is proposed by the Ministry of Jal Shakti. It aligns perfectly with the objectives of our mobile app, WARUNA, which aims to crowdsource water-related problems and map them on a geographical platform. The app leverages geo-referenced images citizens share to provide situational awareness to emergency responders and assist with financial loss assessment. It also utilizes computer vision and deep learning algorithms to categorize water-related issues at different administrative levels.

### 1.2 - Features of WARUNA:

1. **Crowdsourced Data Collection:** Users can report water-related problems by uploading images and providing location details.
2. **Geo-Referenced Mapping:** The app maps reported issues on an interactive map, visually representing water-related problems within communities.
3. **Advanced Data Analysis:** Utilizes computer vision and deep learning algorithms to analyze images and categorize water-related issues.
4. **Real-Time Updates:** Provides real-time updates on reported issues, allowing for timely response and action.
5. **Administrative Planning Tool:** Serves as a valuable tool for administrators to plan and manage water-related problems at different administrative levels.

By integrating crowd-sourced data, advanced technologies, and community engagement, WARUNA aims to revolutionize water management practices, providing a comprehensive and effective solution for addressing water-related challenges within communities.

# 2. TOOLS AND TECHNOLOGY USED

HARDWARE & SOFTWARE REQUIREMENTS

## 2.1 - Hardware Requirements

- A computer with internet access
- Web camera or smartphone camera for image processing (if applicable)

## 2.2 - Software Requirements

- Flask: Used for creating web applications.
- HTML (Jinja): Used for rendering webpages and making components interactive.
- CSS: Used for styling the application.
- Bootstrap: Used for making the application responsive.
- SQL Alchemy: Used for database interactions and data safety.
- JavaScript: Used for client-side scripting to enhance interactivity.
- jQuery: Used for simplifying JavaScript programming.
- Leaflet: Used for displaying interactive maps
- SQLite: Used as the database management system.
- Chrome Developer Tools: Used for debugging and testing.
- Python Anywhere: Used for hosting the application on the cloud.

# 3. IMPLEMENTATION

## 3.1 - Essential functions' code

To begin, we created a folder named "Waruna" containing the app.py file, a templates folder, and a static folder. The template folder contained the index.html file and other templates. The static folder contained the images required for our webpage. As it is visible in the Figure, the website has been subdivided into 10 categories. The app.py file is for the app initialization and database creation. The graphs_generator.py file is for graph generation. The models.py file handles the creation and initialization of the different tables in our database. The backup folder is just a folder that contains another app.py in case this doesn't work. The data folder contains the necessary data for craft generation and the reports folder contains the reports submitted by the inspectors.


*Figure1.1 : Folders*

The next step was setting up the app.py. The first thing we did was import the necessary flask modules like sqlite3, Flask, etc. In Flask, modules are imported at the beginning of the script to enable their functionalities throughout the application.

```
import sqlite3
from flask import Flask, render_template, request, flash, redirect, url_for, session, send_from_directory, current_app, jsonify
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
import os
import time
from fpdf import FPDF
from PIL import Image
```



*Figure1.2 : Setting up app.py*

This Flask route function, index, renders the index.html template. It retrieves the top 5 reported problems from the 'reported_problems' table in the SQLite database and passes them to the template for display. The function establishes a connection to the database, fetches the data using a SQL query, closes the connection, and then renders the template with the fetched data.

```python
@app.route('/')
def index():
    """Render the index page."""
    # Fetch top 5 reported problems from the SQLite database
    conn = sqlite3.connect('waruna.db')
    c = conn.cursor()
    c.execute("SELECT * FROM reported_problems ORDER BY id DESC LIMIT 5")
    top_problems = c.fetchall()
    conn.close()
    return render_template('index.html', top_problems=top_problems)
```

*Figure1.3 : Flask route function*

The next step was creating tables. We wrote a function to create tables. The `create_tables` function is responsible for creating three tables in the SQLite database: `reported_problems`, `employees`, and `assigned_tasks`. Each table is designed to store specific information related to reported problems, employees (including inspectors and managers), and assigned tasks. The function ensures that if these tables already exist, they are not recreated.

```python
# Function to create SQLite database and tables
def create_tables():
    conn = sqlite3.connect('waruna.db')
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS reported_problems
                 (id INTEGER PRIMARY KEY AUTOINCREMENT, location TEXT, landmark TEXT,
                 location_description TEXT, issue_image TEXT, issue_description TEXT,
                 email TEXT, date TEXT, lat REAL, lng REAL, assigned_to TEXT, status TEXT)''')  # Add lat and lng columns
    c.execute('''CREATE TABLE IF NOT EXISTS employees
                 (employee_id TEXT PRIMARY KEY, name TEXT, password TEXT, role TEXT,
                 email TEXT, mobile_no TEXT, tasks_reviewed INTEGER DEFAULT 0,
                 tasks_completed INTEGER DEFAULT 0)''')  # Add name column

    # Create a table for assigned tasks
    c.execute('''CREATE TABLE IF NOT EXISTS assigned_tasks
                 (id INTEGER PRIMARY KEY AUTOINCREMENT, inspector_id TEXT,
                 issue_id INTEGER,
                 FOREIGN KEY(inspector_id) REFERENCES employees(employee_id),
                 FOREIGN KEY(issue_id) REFERENCES reported_problems(id))''')
    # c.execute("INSERT INTO employees VALUES ('manager1','Manager 1', 'password', 'Manager', 'manager1@example.com', '9380142763', 0, 0)")
    # c.execute("INSERT INTO employees VALUES ('inspector1','Inspector 1', 'password', 'Water Inspector', 'inspector1@example.com', '911351840

    conn.commit()
    conn.close()

# Create SQLite database and tables
create_tables()
```

*Figure1.4 : Create tables function*

The `dashboard` route function checks if the user is logged in by verifying the presence of 'employee_id' in the session. If the user is logged in, it retrieves the user's name from the session and renders the appropriate dashboard based on their role (Manager or Water Inspector). If the user is not logged in, it flashes an error message and redirects them to the login page.

```python
@app.route('/dashboard')
def dashboard():
    if 'employee_id' in session:
        name = session.get('name')
        if session['role'] == 'Manager':
            return render_template('manager_dashboard.html', name=name)
        elif session['role'] == 'Water Inspector':
            return redirect(url_for('inspector_dashboard'))
    flash('Please log in to access this page', 'error')
    return redirect(url_for('login'))
```

*Figure1.5 : Dashboard function*

The `create_inspector_task_table` function is designed to create a new table for storing tasks assigned to a specific water inspector if such a table does not already exist. The function takes the inspector's table name as an argument, connects to the SQLite database, and executes a SQL query to create the table with columns for task IDs and issue IDs. After committing the changes, the database connection is closed.

```python
def create_inspector_task_table(table_name):
    """Create a table for the inspector's tasks if not exists."""
    conn = sqlite3.connect('waruna.db')
    c = conn.cursor()
    c.execute(f"CREATE TABLE IF NOT EXISTS tasks_{table_name} (id INTEGER PRIMARY KEY AUTOINCREMENT, issue_id INTEGER)")
    conn.commit()
    conn.close()
```

*Figure1.6 : create_inspector_task_table function*

The `read_iot_data` function reads data from a CSV file containing IoT sensor data. It iterates over each row in the CSV file, extracting relevant information such as flow rate, pressure level, latitude, and longitude. Based on specified conditions (e.g., abnormal flow rate or low-pressure level), the function calls the `add_to_reported_problems` function to add a new entry to the 'reported_problems' database table, indicating the detected issue and its details, along with the location coordinates. This function demonstrates how IoT data can be processed and used to detect and report water-related problems in the system.

```python
# Function to read IoT data from a CSV file and detect problems based on specified conditions
def read_iot_data(filename):
    with open(filename, mode='r') as file:
        reader = csv.DictReader(file)
        for row in reader:
            #print(row)  # Here you can process each row as needed
            # Example: Detect problems based on specified conditions and add to reported problems database
            flow_rate = float(row['Flow Rate'])
            pressure_level = float(row['Pressure Level'])
            latitude = row['Latitude']
            longitude = row['Longitude']
            if flow_rate > 8.0:
                add_to_reported_problems('Abnormal Flow Rate', f'Flow rate is too high: {flow_rate}', latitude, longitude)
            elif pressure_level < 20.0:
                add_to_reported_problems('Low Pressure', f'Pressure level is too low: {pressure_level}', latitude, longitude)
```

*Figure1.7 : read_iot_data function*

The `assign_tasks` function allows managers to assign reported issues to water inspectors. When a POST request is made, it retrieves the selected issue and inspector from the form, updates the database to mark the issue as assigned, and inserts the task into the `assigned_tasks` table. The function then commits the changes and redirects back to the `assign_tasks` page, ensuring that the task assignment is successfully processed and recorded.pressure level), the function calls the `add_to_reported_problems` function to add a new entry to the 'reported_problems' database table, indicating the detected issue and its details, along with the location coordinates.

```python
# Function to assign tasks to water inspectors
@app.route('/assign_tasks', methods=['GET', 'POST'])
def assign_tasks():
    issues_data = []
    water_inspectors=[]
    if request.method == 'POST':
        # Get the selected reported issue and water inspector from the form
        issue_id = request.form.get('issue_id')
        inspector_id = request.form.get('inspector_id')

        # Update the database to assign the task to the selected water inspector
        conn = sqlite3.connect('waruna.db')
        c = conn.cursor()
        c.execute("UPDATE reported_problems SET assigned_to = ?, status = 'assigned' WHERE id = ?", (inspector_id, issue_id))

        # Insert the assigned task into the assigned_tasks table
        c.execute("INSERT INTO assigned_tasks (inspector_id, issue_id) VALUES (?, ?)", (inspector_id, issue_id))

        conn.commit()
        conn.close()

        # Create a table for the inspector's tasks if not exists
        table_name=f"{inspector_id}_tasks"
        # create_inspector_task_table(table_name)

        # Insert the assigned task into the inspector's task table
        # insert_inspector_task(table_name, issue_id)
        #print(reported_issues)
        session['task_assigned'] = True
        # Redirect to the assign tasks page after updating the database
        return redirect(url_for('assign_tasks'))
```

*Figure1.8 : assign_tasks function*

The `delete_employee` function handles the deletion of an employee from the database. It first checks if the specified employee exists, and if so, deletes the employee record and their associated tasks table. Upon successful deletion, it flashes a success message and redirects to the 'manage_employees' page; if the employee does not exist, it flashes an error message instead.

```python
@app.route('/delete_employee/<employee_id>', methods=['POST'])
def delete_employee(employee_id):
    conn = get_db_connection()
    c = conn.cursor()

    # Check if the employee exists
    c.execute("SELECT * FROM employees WHERE employee_id = ?", (employee_id,))
    employee = c.fetchone()

    if employee:
        c.execute("DELETE FROM employees WHERE employee_id = ?", (employee_id,))

        # Check if a table with the employee's name exists and delete it
        employee_table = f"tasks_{employee_id}_tasks"
        c.execute(f"DROP TABLE IF EXISTS {employee_table}")

        conn.commit()
        flash('Employee deleted successfully', 'success')
    else:
        flash('Employee does not exist', 'error')

    conn.close()
    return redirect(url_for('manage_employees'))
```

*Figure1.9 : delete_employee function*

The `botwarunacharya` route redirects users to the Warunacharya chatbot, which is hosted externally. This chatbot provides users with automated assistance and information related to water management issues.

```python
@app.route('/botwarunacharya')
def botwarunacharya():
    return redirect('https://mediafiles.botpress.cloud/c006a2d3-274b-4e5b-8567-37eb08511cce/webchat/bot.html')
```

*Figure1.10 : botwarunacharya function*

The block if __name__ == "__main__": ensures that the Flask application runs only if the script is executed directly, rather than being imported as a module. The app.run(debug=True) line starts the Flask development server and enables debug mode, which provides detailed error messages and

automatically reloads the server when code changes are detected.

```
if __name__ == "__main__":
    app.run(debug=True)
```

*Figure1.11 : app.run*

Given that our code is close to 1000 lines, it is not feasible to explain it in its entirety within this report. Therefore, I have focused on explaining the CRUD operations in SQL and other significant code segments.

However, this is a brief of all the functions present in our code.

## 3.2- Brief of all the functions

**Imports and Setup:**

- Standard imports for os, sqlite3, datetime, etc.
- Flask and Flask components like request, session, flash, etc.
- Custom functions like send_confirmation_email.

**App Configuration:**

- Flask app configuration, including UPLOAD_FOLDER, secret key, etc.
- User Authentication and Session Handling:
- Login (/login): Validates user credentials, sets session variables, and redirects based on the role.
- Logout (/logout): Clears the session and redirects to the login page.
- Dashboard (/dashboard): Renders the appropriate dashboard for managers or redirects for water inspectors.

**Issue Reporting and Management:**

- Submit Issue (/submit_issue): Handles the submission of reported issues, including saving images and data to the database.
- View Image (/view_image/<int:issue_id>): Retrieves and displays the uploaded issue image.
- Assign Tasks (/assign_tasks): Assigns reported issues to water inspectors and updates the database accordingly.
- Delete Task (/delete_task): Deletes a task from the reported issues.

**Water Inspector Tasks:**

- Inspector Dashboard (/inspector_dashboard): Displays tasks assigned to the logged-in water inspector.
- File Report (/file_report/<int:issue_id>): Allows inspectors to file a report on the assigned tasks, including uploading images and data.

**Employee Management:**

- Register Manager (/register_manager) and Register Inspector (/register_inspector): Registers new managers and inspectors.
- Manage Employees (/manage_employees): Displays a list of all employees.
- Edit Employee (/edit_employee_page/<employee_id>): Edits the details of an employee.
- Delete Employee (/delete_employee/<employee_id>): Deletes an employee from the database.

**IoT Data Management:**

- Generate Dummy IoT Data: Creates dummy IoT data and saves it to a CSV file.
- Read IoT Data: Reads IoT data from the CSV and detects problems, adding them to the reported problems database.
- Manage IoT Data (/manage_iot_data): Displays graphs and provides options to download IoT data and reports.
- Download IoT Data (/downloadiotdata): Allows users to download the dummy IoT data.

**Helper Functions:**

- Database Connection (get_db_connection): Returns a SQLite database connection.
- Issue Description (get_issue_description): Fetches the issue description from the database.
- Get Reported Issues (get_reported_issues): Retrieves reported issues from the database.
- Get Water Inspectors (get_water_inspectors): Retrieves water inspectors from the database along with the number of tasks assigned to each.
- Get Assigned Issues (get_assigned_issues): Retrieves the list of assigned issue IDs.
- Visualization and Reporting:
- Map Graph (/map_graph) and Geospatial Data Map Graph (/geospatialdatamapgraph): Renders HTML templates for map visualizations.
- Download PDF (/download_pdf): Combines images into a PDF and provides a download link.
- Create PDF: Generates a PDF from a list of image paths.

# 4. RESULT AND DISCUSSION

## 4.1 - Website Overview

This is the main page of our application. Here, users can find a "Report an Issue" option, enabling them to file complaints. At the top right corner, there is a login page specifically for employees, such as inspectors and managers. Additionally, a chat icon is located at the bottom right corner of the page, which users can click to access our Warunacharya chatbot. For mapping functionalities, we use Leaflet, an open-source JavaScript library for mobile-friendly interactive maps, enabling users to pinpoint and visualize the exact location of the problem.

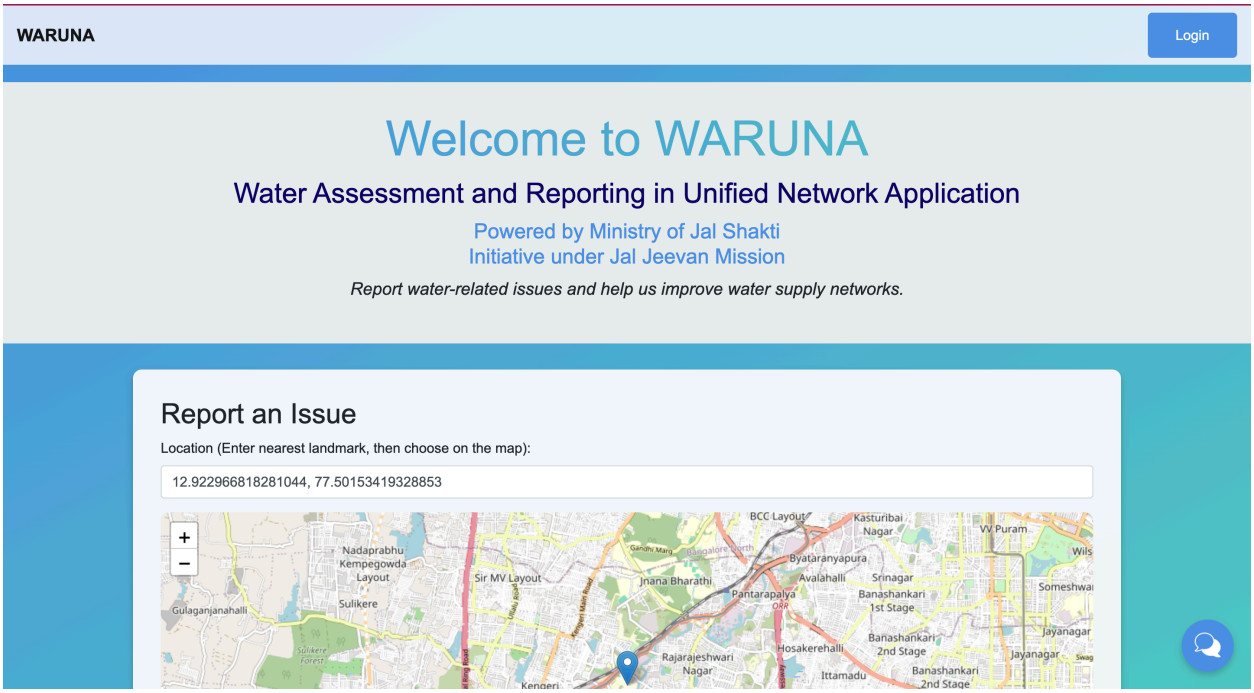


*Figure 2.1 : main page of the application*

The "Report a Problem" option allows users to provide detailed information about issues. They can select a specific location on the map, input a landmark, and describe the location. Users can also upload an issue image, provide a description of the issue, enter their email, and specify the date when the issue was noticed. On scrolling further down, we can see Top 5 reported problems.

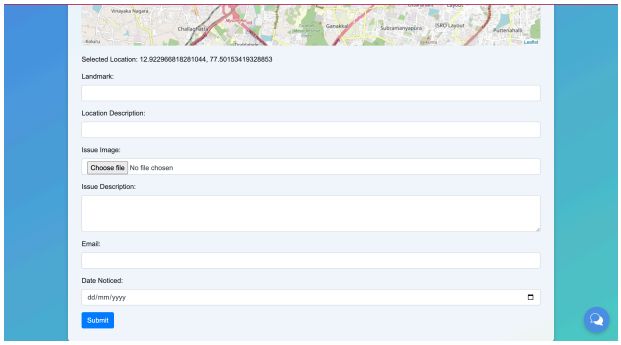


*Figure 2.2 : Report an issue*

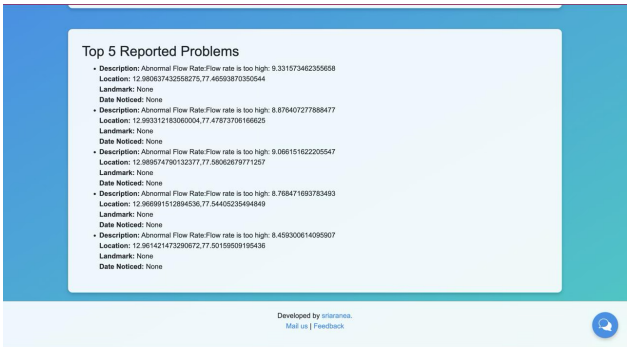


*Figure 2.3 : Top 5 reported problems*

On logging in with manager credentials, we can see the manager dashboard. The Manager Dashboard is the central hub for managing operations within the application. It offers options to assign tasks, view detailed reports, and analyze geospatial and IoT data. Managers can also register new managers and inspectors, view and update their profiles, and manage employee

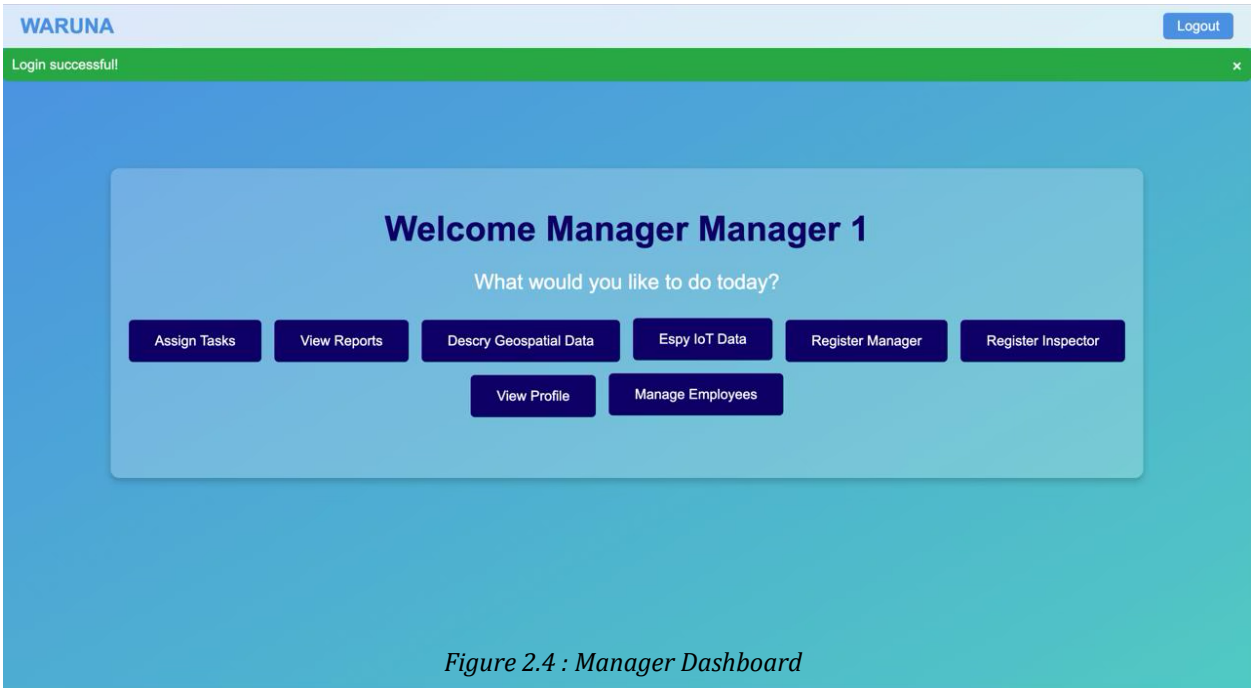


*Figure 2.4 : Manager Dashboard*

information. This streamlined interface ensures efficient coordination and oversight of all activities.

On clicking assign tasks, the manager can see a list of all the problems filed by the citizens and can assign particular tasks to particular water inspectors. While assigning the tasks, he can see the number of tasks already assigned to the inspector.

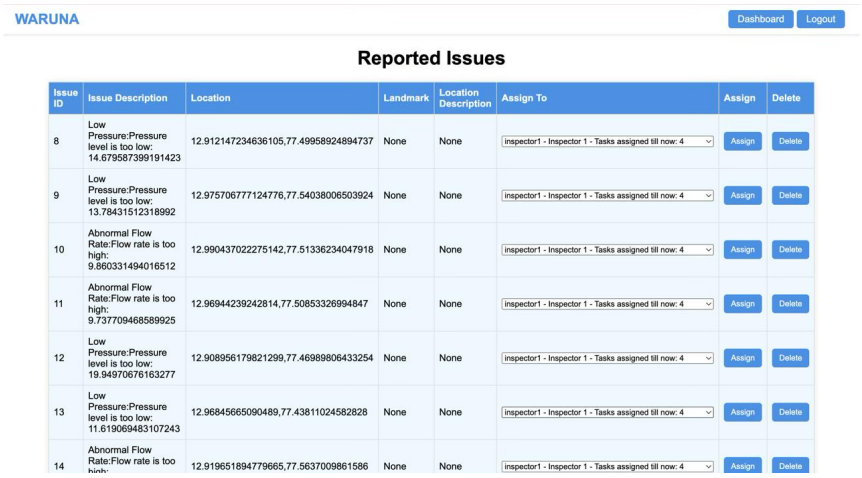


*Figure 2.5 : Assign tasks*

The "View Reports" button directs the manager to a page displaying reports filed by water inspectors post-inspection. Each issue has a "View Report" link, leading to a page that contains the text file which has the details the observations, actions taken, and recommendations by the inspector. Additionally, the "View Image" and "View Data" buttons allow users to see the submitted image and relevant data. The "Approve" button marks the task as approved, removes it

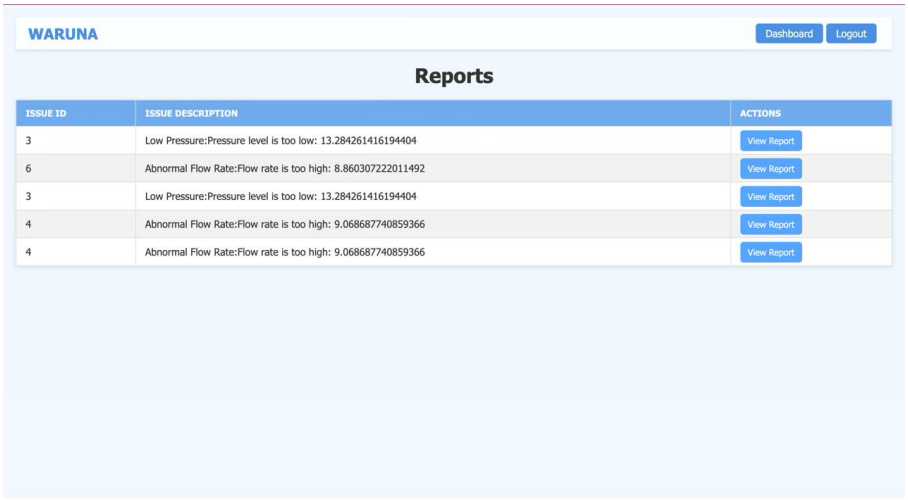


*Figure 2.6 : View  Reports*

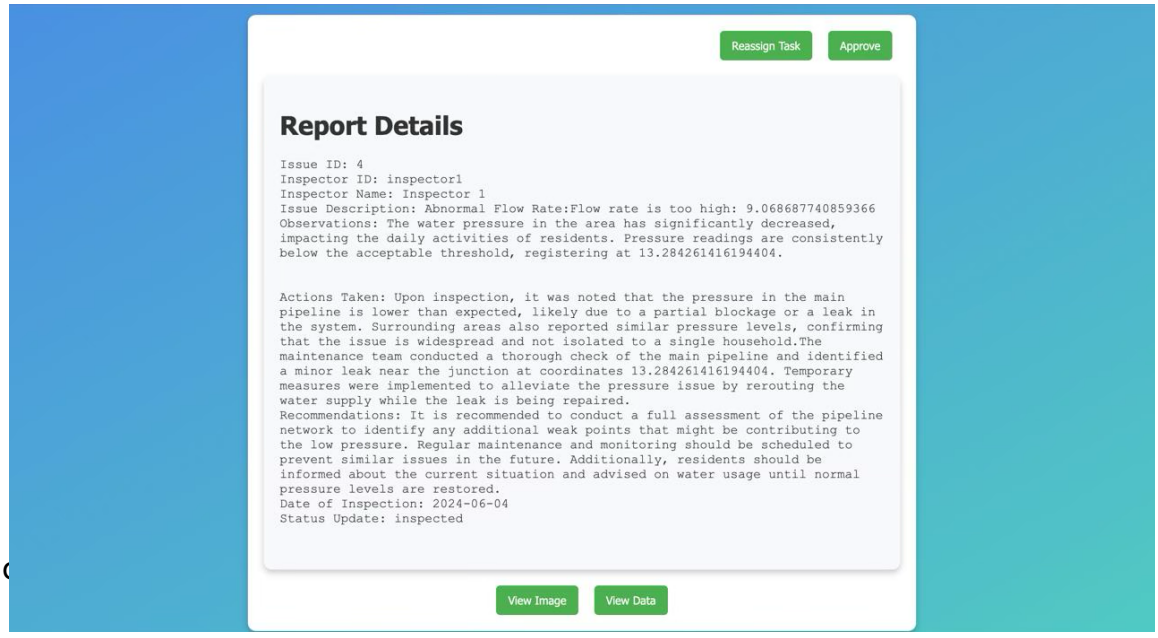


*Figure 2.7 : Report Details*

from the reported problems and assigned tasks list, and marks it as completed. The "Reassign Task" button reassigns the task while keeping the issue status as assigned.

The "Descry Geospatial Data" option on the manager dashboard provides a comprehensive view of geospatial information related to reported water issues. Utilizing interactive maps powered by Leaflet, managers can visualize the exact locations of all reported issues, enhancing their ability to analyze patterns and trends geographically.


*Figure 2.8 : Descry Geopolitical Data*

The "Espy IoT Data" feature on the manager dashboard offers real-time monitoring and insights from IoT devices deployed across the water management system. This functionality provides managers with critical data such as pressure levels, flow rates, and other sensor readings. By analyzing this data, managers can quickly identify and respond to anomalies or potential issues before they escalate.

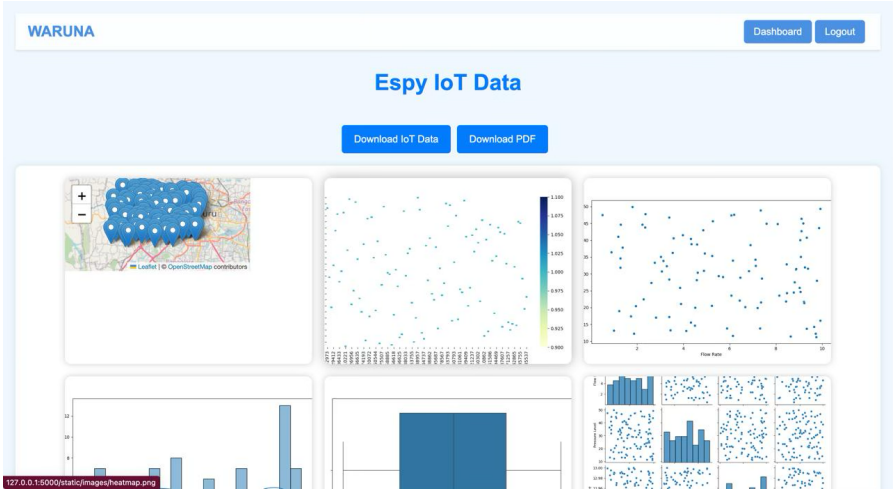




*Figure 2.9 : Espy IoT Data*

The Manage Employees section allows managers to oversee and update employee information such as contact details, roles, and responsibilities. It provides a comprehensive view of the workforce, enabling efficient management and allocation of tasks.

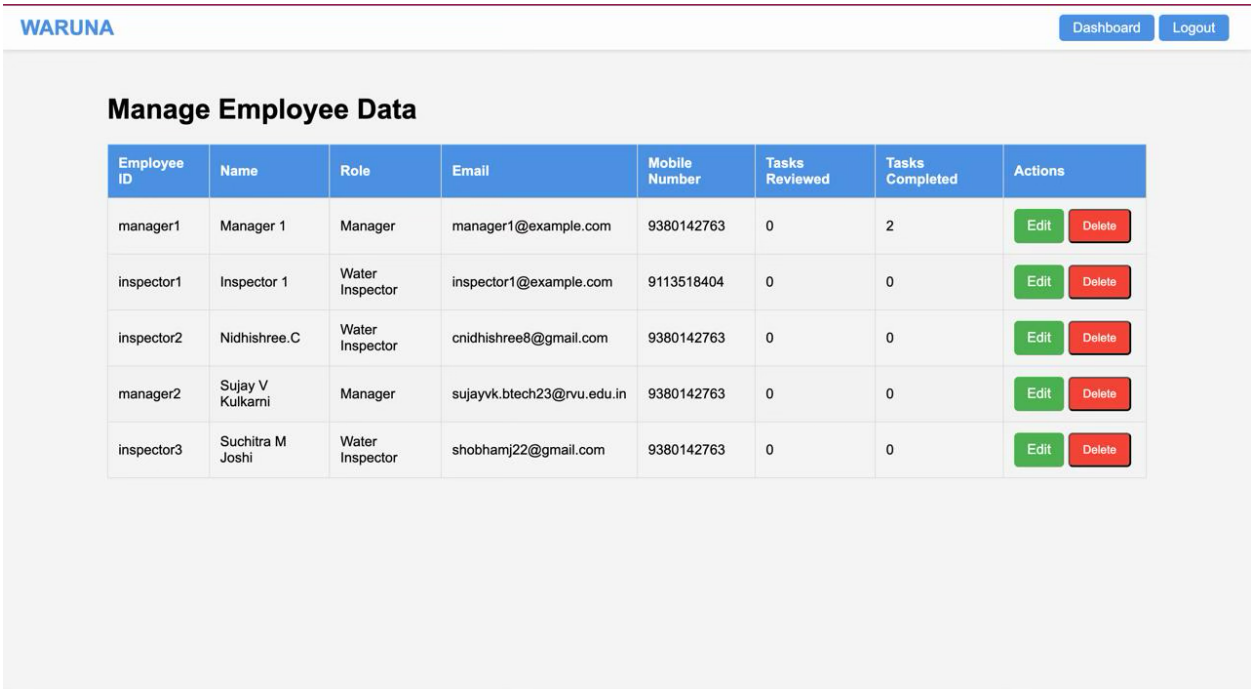


*Figure 2.10 : Manage Employee Data*

The "View Profile" option allows managers to access and update their personal information, including contact details and account settings. This feature ensures that all profile data is current and accurate for effective communication and system management.

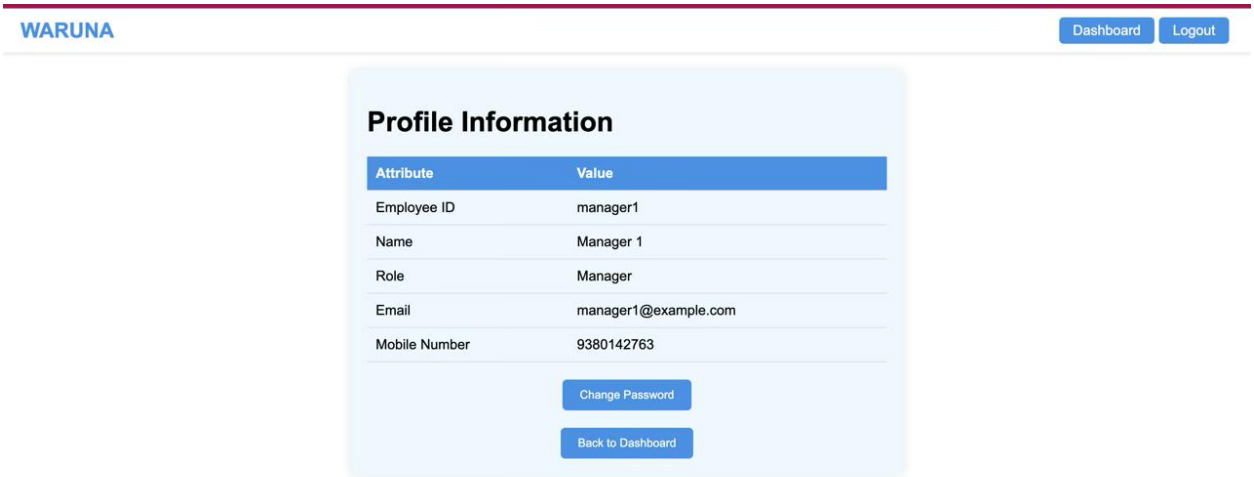


*Figure 2.11: Profile*

The Inspector Dashboard provides Water Inspectors with a clear overview of their assigned tasks, including issue IDs, problem descriptions, and locations. Inspectors can efficiently manage their workload by accessing options to view images, inspect locations, and file reports directly from the dashboard. Clicking on view image will lead them to the image uploaded by the complainant. Inspect location just leads them to a page

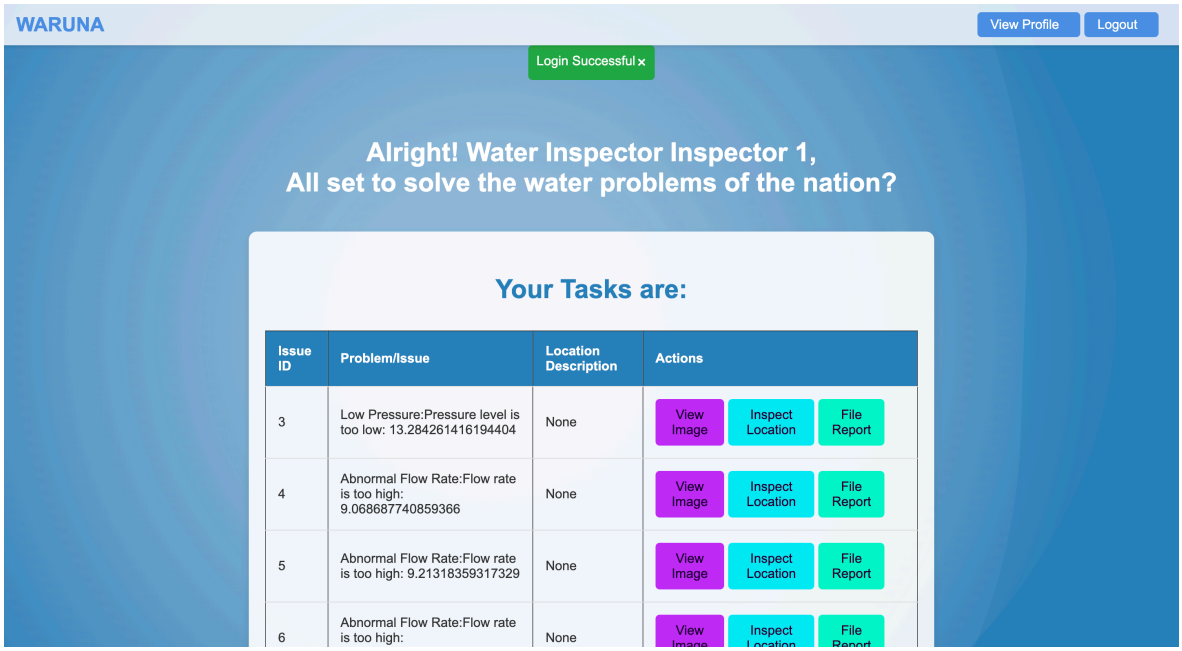


*Figure 2.12 : Inspector dashboard*

The "File Report" feature in the inspector dashboard enables inspectors to document their observations, actions taken, and recommendations for each reported issue. It also allows them to upload images and data related to their inspection, ensuring a thorough and well-documented resolution process.

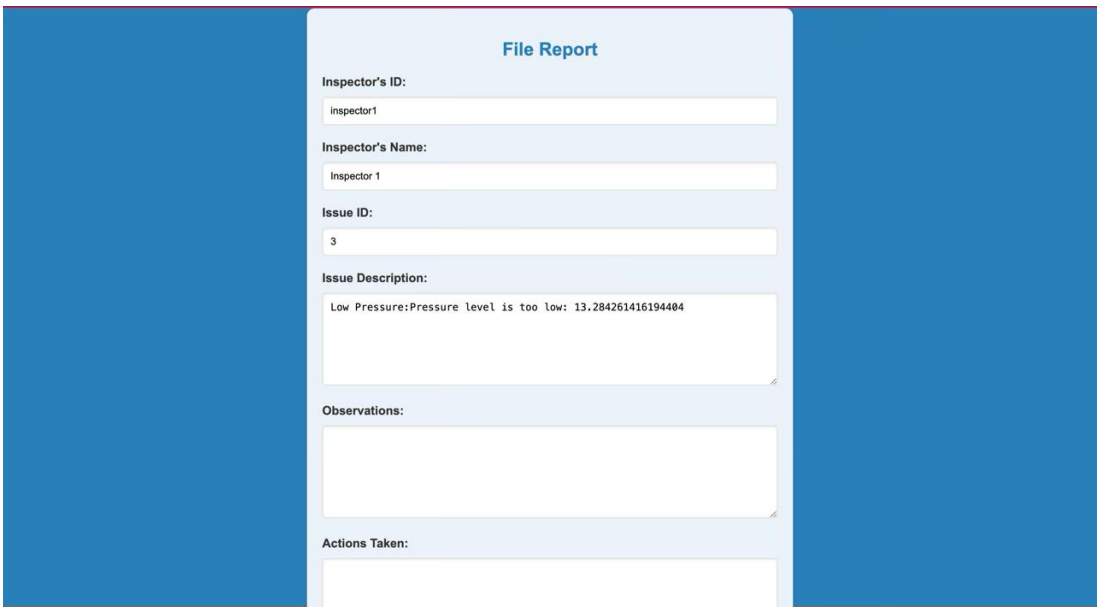


*Figure 2.13 : File  Report*

Warunacharya is a chatbot integrated into the water management system, designed to provide interactive assistance and information retrieval for users. It enhances user experience by offering quick responses to queries and guiding users through various functionalities of the application.

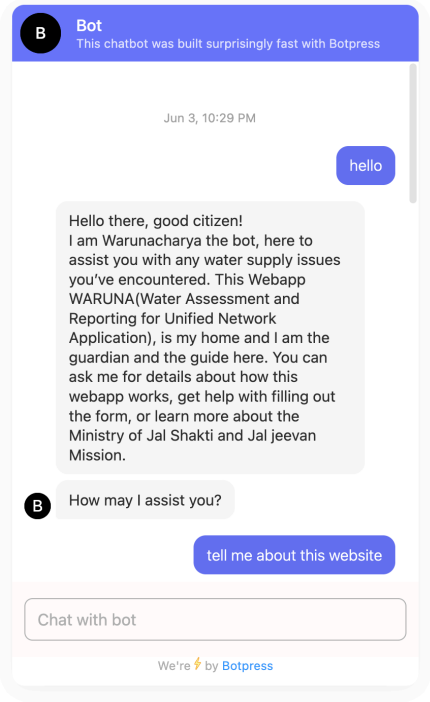


*Figure 2.14 : Chatbot Warunacharya*

## 4.2 - Incorporating Python Concepts

This project addresses the critical challenge of efficient water resource management through the development of a comprehensive web application. By integrating manual reporting and IoT-based monitoring, the system provides a robust platform for detecting, reporting, and managing water-related issues effectively.

The project applies various concepts of Python programming to achieve its objectives. It leverages Python's features such as input/output operations, variables, data types, operators, and expressions for efficient data processing. Additionally, functions and data structures like user-defined functions, lists, strings, tuples, dictionaries, and NumPy are extensively used for data manipulation and problem-solving. The application also utilizes file-handling techniques for data extraction, transformation, and storage, enabling seamless data management.

Furthermore, the project demonstrates the interface between Python applications and an SQLite

database, showcasing CRUD operations for efficient data handling. Object-oriented programming concepts such as classes, objects, inheritance, and modules are applied to ensure a structured and maintainable codebase. The use of Flask for web application development highlights the application of Python in creating user-friendly interfaces and web applications.

## 5. CONCLUSION

In conclusion, this report has detailed the development and implementation of a comprehensive water management system. It began with an overview of the project's objectives and scope, emphasizing the need for efficient water resource management. The report then delved into the various chapters, including the use of Python for problem-solving, data manipulation, and application development.

The significance of each chapter was highlighted, illustrating how concepts like object-oriented programming, Flask for web development, was used to address real-world water management challenges. The integration of technologies like SQLite for database management and tools like Git for version control ensured the project's robustness and scalability.

Furthermore, the report discussed the deployment of the application on PythonAnywhere.com, showcasing how various software requirements were tailored to suit the project's needs. Additionally, the integration of Botpress for the chatbot feature added a layer of interactivity and user engagement to the system.

Overall, the project's achievements are significant, as it provides a streamlined platform for reporting water-related issues, managing tasks efficiently, and analyzing geospatial and IoT data for better decision-making. The system's user-friendly interface, coupled with its robust backend architecture, makes it a valuable tool for water management authorities and stakeholders.

# REFERENCES

https://www.w3schools.com/css/css_table.asp

https://classroom.google.com/u/1/c/NjU4MDM2NTA2NDc2/m/Njc1Nzk3OTA3MDE3/details

https://classroom.google.com/u/1/c/NjU4MDM2NTA2NDc2/m/NjgyNjE2MTI3MjMx/details

## APPENDIX

GitHub Link: https://github.com/SujayKulkarni-2211/WARUNA.git