Project Report

CHAPTER 1
INTRODUCTION

1.1.Project Overview

The project aims to create a web application called "The Issue Tracker" that serves as a reliable complaint management system for improved customer service. The main objective is to provide a smart and simple method for individuals to register complaints, track their progress, and resolve them online, eliminating the need for frequent visits to administrative offices.

The target audience for this web application is the general population, particularly residents of housing schemes and societies in India. By utilizing this application, users can save time and money by reporting and monitoring their complaints remotely.

Additionally, the project seeks to tackle corruption in government offices by providing a transparent and efficient complaint management system.

The application allows customers to raise tickets for their complaints. They can provide a detailed description of the issue, such as problems with streetlights, water pipe leakages, or road renovations. Users can also attach image files to better explain their complaints and specify the issue's location. Once a ticket is raised, an agent or service person will be assigned to handle the problem.

The administrator plays a crucial role in overseeing the entire process. They have specific responsibilities, including logging in as an admin, creating and managing agents, and assigning issues to customer complaints. The administrator can also track the work assigned to agents, and all the updates will be recorded in the database.

Users can register for an account on the web application and log in to access their complaint management features. After logging in, they can create complaints, providing necessary details and attachments. Each user is assigned an agent who will be responsible for addressing their complaint. Users can track the status of their complaint in their personal page on the web application.

Overall, the project aims to automate and streamline the manual complaint management system, enhancing efficiency and improving customer service. By leveraging technology, individuals can easily report and track their complaints, leading to faster resolutions and better overall satisfaction.

1.2.PURPOSE

1. The purpose of "The IssueTracker" project, which is a complaint management system, can be summarized as follows: Improved Customer Service: The project aims to enhance customer service by providing a reliable and efficient platform for individuals to report their complaints. By offering a user-friendly web application, users can easily raise tickets, track the progress of their complaints, and receive timely updates on the

resolution process. This ultimately leads to better customer satisfaction.

2. Time and Cost Savings: The project aims to save time and money for both customers and administrative staff. By allowing individuals to submit complaints online, it eliminates the need for frequent visits to administrative offices, reducing travel costs and time spent in queues. Additionally, automating the complaint management process streamlines the workflow and improves response times, further saving time for both customers and administrators.
3. Transparency and Accountability: The project seeks to promote transparency and accountability in complaint management. By providing a centralized system where complaints are recorded, assigned to agents, and tracked, it ensures that all actions taken on each complaint are documented. This helps to eliminate potential corruption or negligence in addressing customer complaints.
4. Conversion to an Automated System: The project aims to convert the existing manual complaint management system into an automated one. By leveraging technology and a web application, it streamlines the entire process, from complaint registration to resolution. Automation reduces the likelihood of human errors and improves overall efficiency.
5. Corruption Elimination: By providing a transparent and accountable complaint management system, the project aims to contribute to the reduction of corruption in government offices. With clear documentation and tracking of complaints, it becomes more challenging for any unethical practices to occur, promoting a fair and corruption-free environment.

CHAPTER 2
IDEATION & PROPOSED SOLUTION

2.1 Problem Statement Definition

Customer Problem Statement Template:

Usually ,The people have met problems to complain about corruption in goverment Offices and problems with streetlights,water pipes leakages and road renovation. So people have tovisit the admin to complain about these problems .It takes long time and money .

Template 1:

Template 2:

Problem Statements

Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	Customer	Visit adminto complaint for problem	It takes long time and more money	Distance Between admin and customer isso long	Tired and frustrated
PS-2	People	Request tosolve the social problems	It might be threatening and harrasement	person who are reasonfor that problems will harrasement	scare and no saffety

2.2 Empathy Map Canvas

1. User:

- Thoughts: "I want a convenient way to report complaints without visiting administrative offices."
- Feelings: Frustrated with the current manual complaint management system, hopeful for a better solution.
- Behaviors: Searching for an online platform to register complaints, seeking efficient and transparent processes.
- Needs: A user-friendly web application that simplifies the complaint registration process and provides timely updates on complaint status.

2. Administrator:

- Thoughts: "I need an efficient system to manage and track complaints from users."
- Feelings: Overwhelmed with the manual process, seeking automation for improved productivity.
- Behaviors: Researching complaint management software, looking for a solution to assign and track agent activities.
- Needs: A comprehensive web application that allows for seamless complaint management, agent assignment, and tracking of complaint resolution progress.

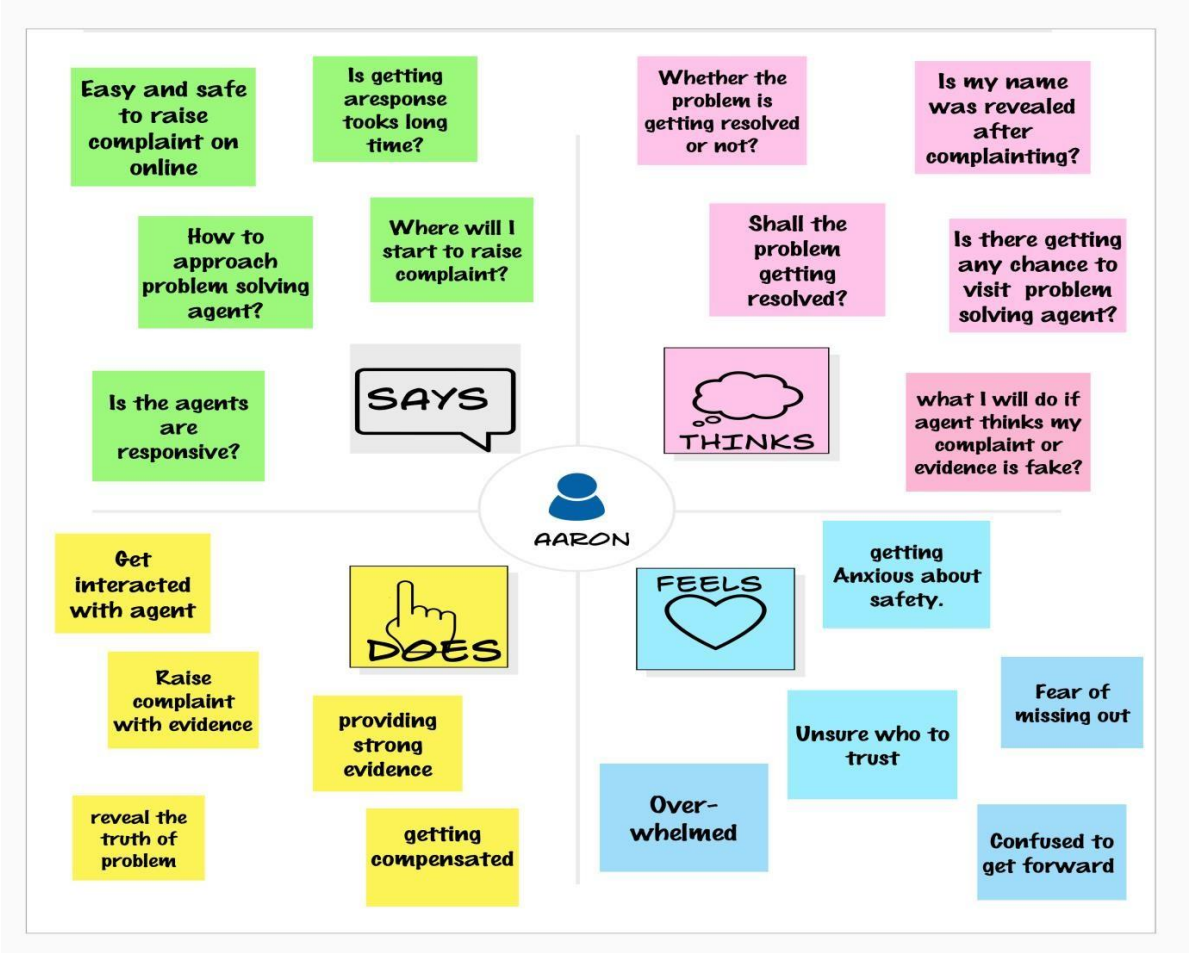
3. Agent/Service Personnel:

- Thoughts: "I want a clear understanding of the complaints assigned to me and the necessary actions to resolve them."
- Feelings: Responsible for addressing user complaints, wanting to provide efficient and satisfactory solutions.
- Behaviors: Checking for newly assigned complaints, prioritizing tasks, and ensuring timely responses.
- Needs: Access to a centralized platform that provides detailed information about assigned complaints, clear instructions on next steps, and effective communication channels with users and administrators.

4. General Public:

- Thoughts: "I want my complaints to be addressed promptly and transparently."
- Feelings: Frustrated with unresolved issues, seeking a reliable system for complaint resolution.
- Behaviors: Actively reporting complaints, hoping for a responsive and accountable process.
- Needs: A user-friendly web application that allows easy complaint registration, transparent tracking of complaint progress, and regular updates on issue resolution.

By understanding the perspectives and needs of different stakeholders, an empathy map can help in designing a complaint management system that addresses their pain points and provides a satisfying user experience.



2.3 Ideation & Brainstorming

1. Mobile Application: Develop a mobile application in addition to the web application to provide users with the convenience of reporting complaints on the go. (Priority: High, Impact: High, Feasibility: High)
2. Real-time Notifications: Implement a notification system that keeps users updated on the progress of their complaints, providing real-time status updates and estimated resolution times. (Priority: High, Impact: High, Feasibility: Medium)
3. Integrated Mapping: Integrate a mapping feature into the application, allowing users to pinpoint the exact location of their complaints and enabling administrators and agents to visualize and prioritize based on geographic data. (Priority: Medium, Impact: Medium, Feasibility: Medium)
4. Customer Feedback System: Incorporate a feedback mechanism that allows users to rate and provide feedback on the service provided by agents, enabling

continuous improvement in complaint resolution. (Priority: Medium, Impact: Medium, Feasibility: High)

5. Knowledge Base: Create a knowledge base within the application that provides users with self-help guides, FAQs, and troubleshooting tips to address common issues before submitting a complaint. (Priority: Low, Impact: Medium, Feasibility: High)
6. Data Analytics Dashboard: Develop a dashboard for administrators to visualize and analyze complaint data, including trends, response times, and agent performance metrics, facilitating data-driven decision-making. (Priority: Medium, Impact: High, Feasibility: Low)
7. Integration with Social Media Platforms: Allow users to report complaints and track their status through popular social media platforms, leveraging their widespread usage and accessibility. (Priority: Low, Impact: Low, Feasibility: Medium)
8. Multilingual Support: Provide multilingual support within the application to cater to a diverse user base, ensuring that language barriers do not hinder users from reporting complaints. (Priority: Low, Impact: Medium, Feasibility: Medium)

Prioritization is subjective and can vary based on specific project goals and constraints. It's essential to consider factors such as resources, timeline, and user needs when deciding which ideas to prioritize for implementation.
Regenerate response

Step-1: Team Gathering, Collaboration and Select the Problem Statement

Template

Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

- **10 minutes** to prepare
- **1 hour** to collaborate
- **2-8 people** recommended

Share template feedback

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

A Team gathering
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B Set the goal
Think about the problem you'll be focusing on solving in the brainstorming session.

C Learn how to use the facilitation tools
Use the Facilitation Superpowers to run a happy and productive session.

Open article →

Need some inspiration?

See a finished version of this template to kickstart your work.

Open example →

Step-2: Brainstorm, Idea Listing and Grouping

2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

TIP
You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

Arun R

- what if a person raise a fake complaint?
- How do approach fake problem solving officer agent?
- what to do if the agent was not responsive for problem?
- How to complaint again if the same problem get partially solved?
- what if the confidence level of the evidence is not enough?

Penyamine

- what if the resolving time is too high?
- how do we know whether the complaint had received the officer or not?
- what to do if the assigned agent was not responsive to solve the problem?
- what if the problem dont get resolved for a long time?
- what to do if the solution get solved only on online but not on offline?
- how to report and eliminate the user's fake complaint?

Tamizhselvan

- what if the complaint was not clearly mentioned?
- What if user does not get reply from officer?
- How do agent report fake user?
- what if evidence is not enough?

Nagarajan

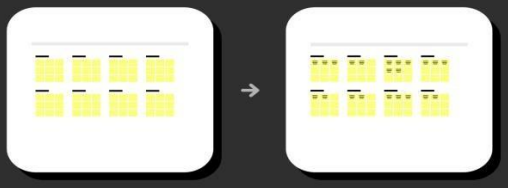
- what to do if complaint details is not enough?
- How to report a unsuppotive officer agent?
- what if the problem is not solved?
- what if complaint does not get acknowledgement from officer agent?
- How to find fake complaint by agent?
- what is maximum duration of solving a complaint type exceeded?

Person 5

Person 6

Person 7

Person 8



3

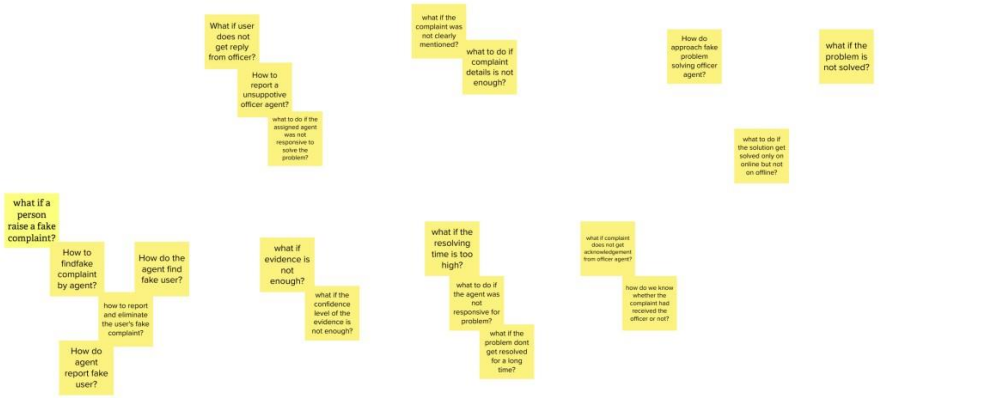
Group ideas

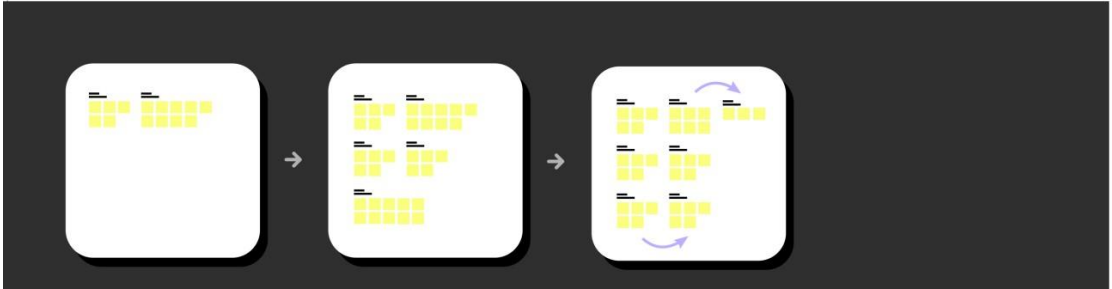
Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes

TIP

Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.





4

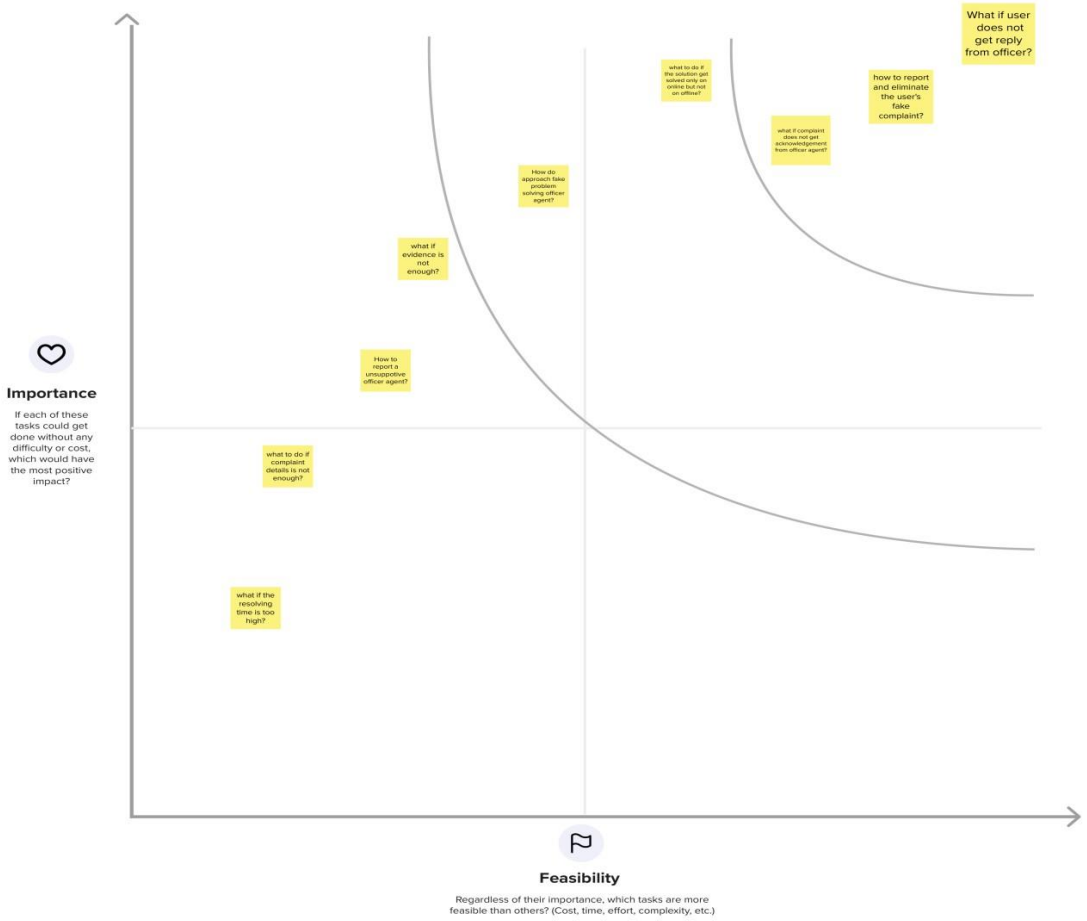
Prioritize

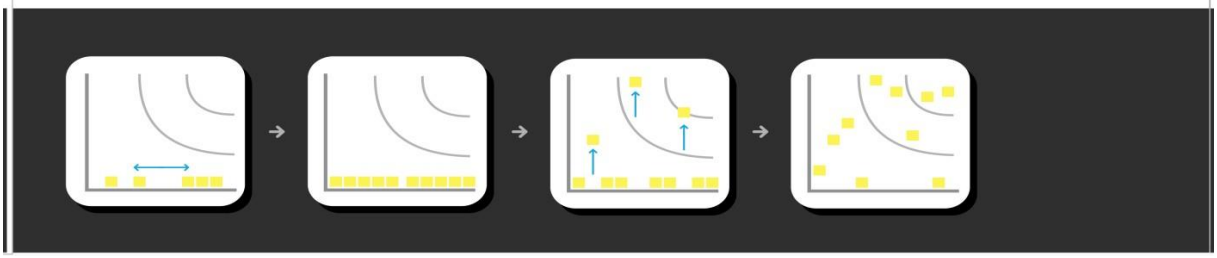
Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

TIP

Participants can use their cursors to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the **H key** on the keyboard.





CHAPTER 3

REQUIREMENT ANALYSIS

3.1 Functional requirement

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration Users should be able to register for an account by providing their necessary details.	User registration form should include fields like fullname, email,password .
FR-2	Agent registration Agent should be able to register for an account by providing their necessary details.	Agent registration form should include fields like username,password,secret key,role
FR-3	Admin registration admin should be able to register for an account by providing their necessary details.	Admin registration form should include fields like username,password,secret key,role
FR-4	User,Agent,Admin Login: Registered users should be able to log in to the web application using their credentials.	User,Admin,Agent authentication should be implemented to ensure secure access.
FR-5	Complaint Creation: Users should be able to create a complaint by providing detailed information about the issue.	Complaint form should include fields for issue description, type of issue (streetlights, water pipes leakages, road renovation), image upload option, and issue location.
FR-6	Complaint Tracking: Users should be able to track the status of their complaints.	Users should have a dedicated page where they can view the list of their complaints along with their current status.
FR-7	Agent Assignment: Admin should be able to assign agents to handle the user complaints	Admin interface should include a functionality to assign agents to specific complaints
FR-8	Admin Dashboard: Admin should have access to a dashboard that provides an overview of all complaints and their statuses.	Admin should be able to view, filter, and sort the complaints based on various parameters such as date, type, status, etc.
FR-9	Database Management: The system should have a robust database to store and retrieve complaint and user information.	Database should be properly designed to efficiently store and manage the required data.

3.2 Non-Functional requirements

Following are the non-functional requirements of the proposed solution.

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	This system aims to provide a user-friendly interface for easy complaint registration and tracking, enhancing usability.
NFR-2	**Security**	The system should ensure the security of user data and protect it from unauthorized access or manipulation.User authentication and secure data transmission should be implemented.
NFR-3	**Reliability**	The system should be reliable and available for users to access and report complaints at any time. The application should have appropriate backup mechanisms and error handling to prevent data loss and ensure system stability.
NFR-4	**Performance**	The application should be able to handle a large number of concurrent users and provide a responsive experience.Proper optimization techniques should be employed to ensure fast response times and minimize delays.
NFR-5	**Availability**	It ensures 24/7 availability, allowing users to access and monitor their complaints at any time, increasing convenience and efficiency.
NFR-6	**Scalability**	The system should be scalable to accommodate an increasing number of users and complaints over time. The system architecture and infrastructure should be designed to handle growing demands without performance degradation.

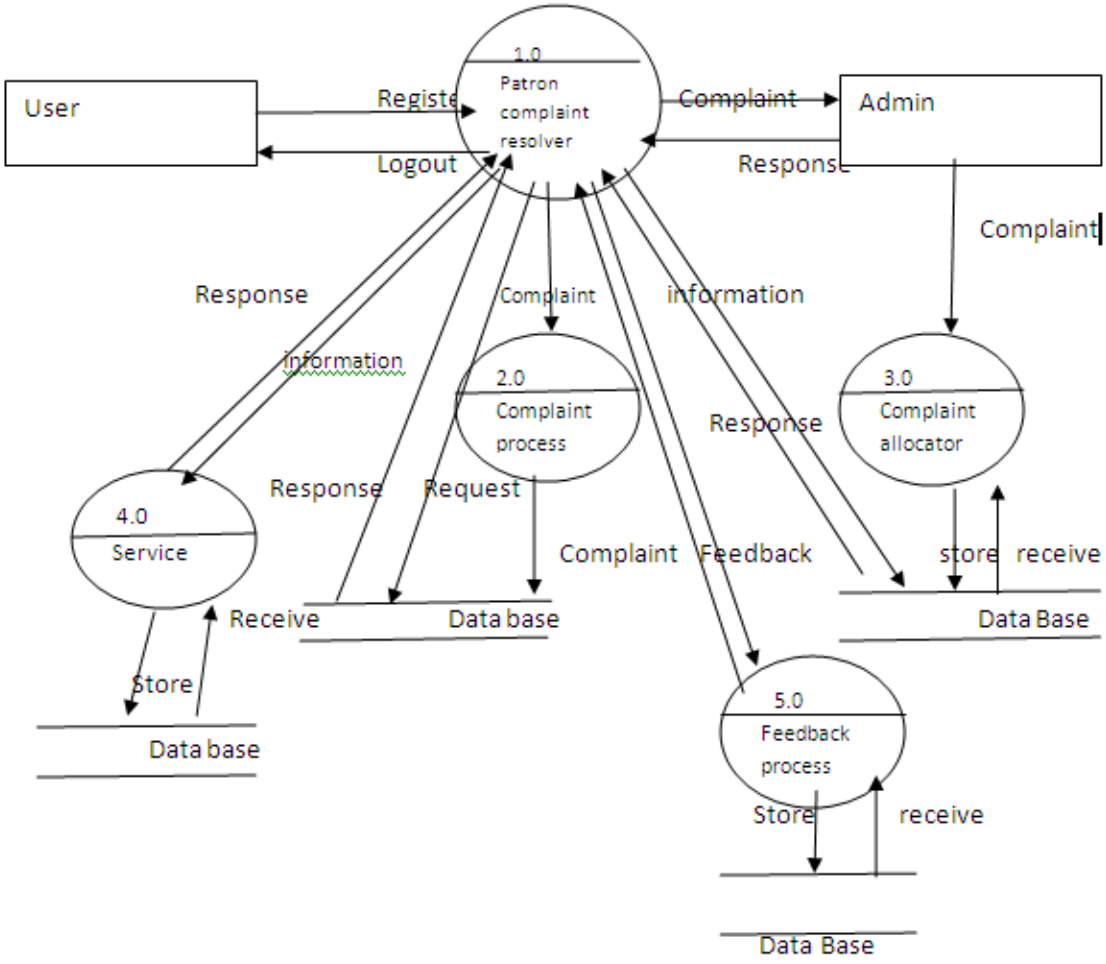
CHAPTER 4
PROJECT DESIGN

4.1 Data Flow Diagrams

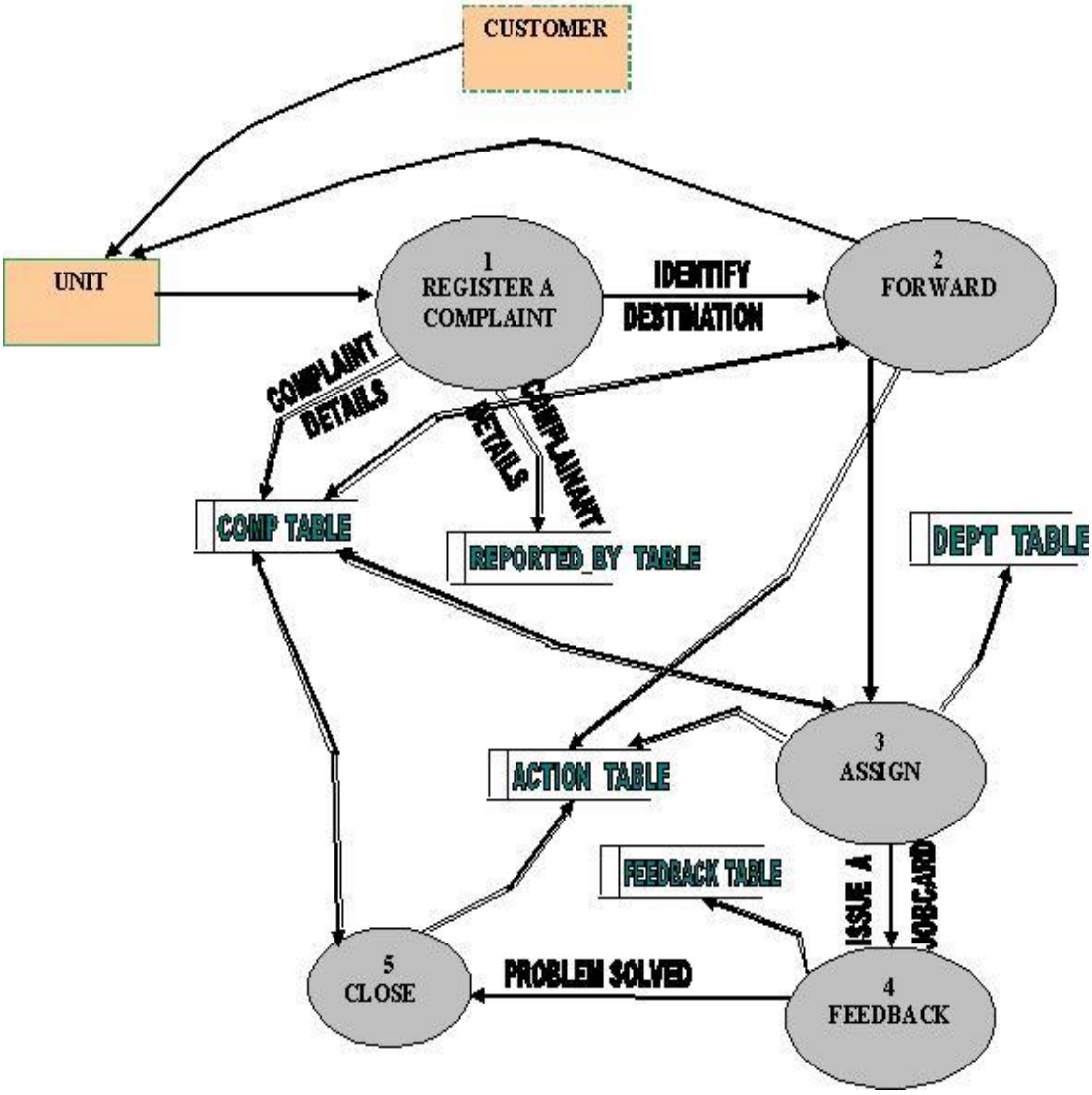
A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.

Data flow diagram:

Dataflow Diagram



DATA FLOW DIAGRAM



User Stories:

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Team Member
User of this system	Complaint creation	USN-1	As a user, I want to be able to create a complaint with detailed information about the problem I am facing.	• The user should be able to access the complaint creation form. • The complaint creation form should include fields for issue description, type of issue (streetlights, water pipes leakages, road renovation), image upload option, and issue location. • The user should be able to submit the complaint after filling in all the required information. • Upon successful submission, the complaint should be stored in the database with a unique complaint ID.	High	Penyamine

User of this System	Complaint tracking	USN-2	As a user, I want to be able to track the status of my complaints.	• The user should have a dedicated page where they can view the list of their complaints. • The list of complaints should display the current status of each complaint. • The user should be able to click on a specific complaint to view more details, including the assigned agent and any updates on the progress. • The complaint status should be updated in real-time as the complaint progresses.	High	Arun

Admin of this system	Agent assignment	USN-3	As an admin, I want to be able to assign agents to handle user complaints.	• The admin should have access to an agent management interface. • The admin should be able to create new agent profiles, including their name, contact information, and expertise. • The admin should be able to assign agents to specific complaints. • Once an agent is assigned, the complaint details and assignment should be updated in the database. • The admin should be able to view the list of assigned complaints for each agent.	Medium	Tamizh selvan

4.2 Solution & Technical Architecture

Proposed Solution for Issue Tracker:

S. No.	Parameter	Description
1	Problem Statement (Problemto be solved)	People visiting the Admin for their problems until the situation is resolved. The public couldnot save time and money by using this approach, and corruption in

		government offices can be eliminated.
2	Idea / Solution description	We want to create a web application for a complaint management system where people can report problems with streetlights, water pipesleakages, and road renovation. To convert a manual compliance management system into an automated system.
3	Novelty / Uniqueness	The customers can raise the ticket with a detailed description, with his issue. A agent or service man will be assigned to the Customerraised issue to solve the problem. Whenever the Admin is assigned to a agent and the datawill be updated in the database, Customers can view the status of the ticket till the service is provided.
4	Social Impact / Customer Satisfacti on	Its primary goal is to give a smart and simple method for registering complaints, tracking theircomplaints, and eliminating them via a web application,
5	Business Model (Revenue Model)	Revenue model for a customer complaint system could be asubscription-based model. This model could offer different levels of subscription to businesses, depending on the size of their customer base and the number of complaints they receive. Businesses could pay a monthly or annual fee for access to the complaint system, and the level of subscription could determine the number of complaints they can submit or the level of support they receive.

Solution Architecture:

In this diagram, we can see that the system is deployed on a Kubernetes cluster, which provides scalability, reliability, and ease of management. The Flask framework is used as the frontend to provide a user-friendly interface for both customers and agents.

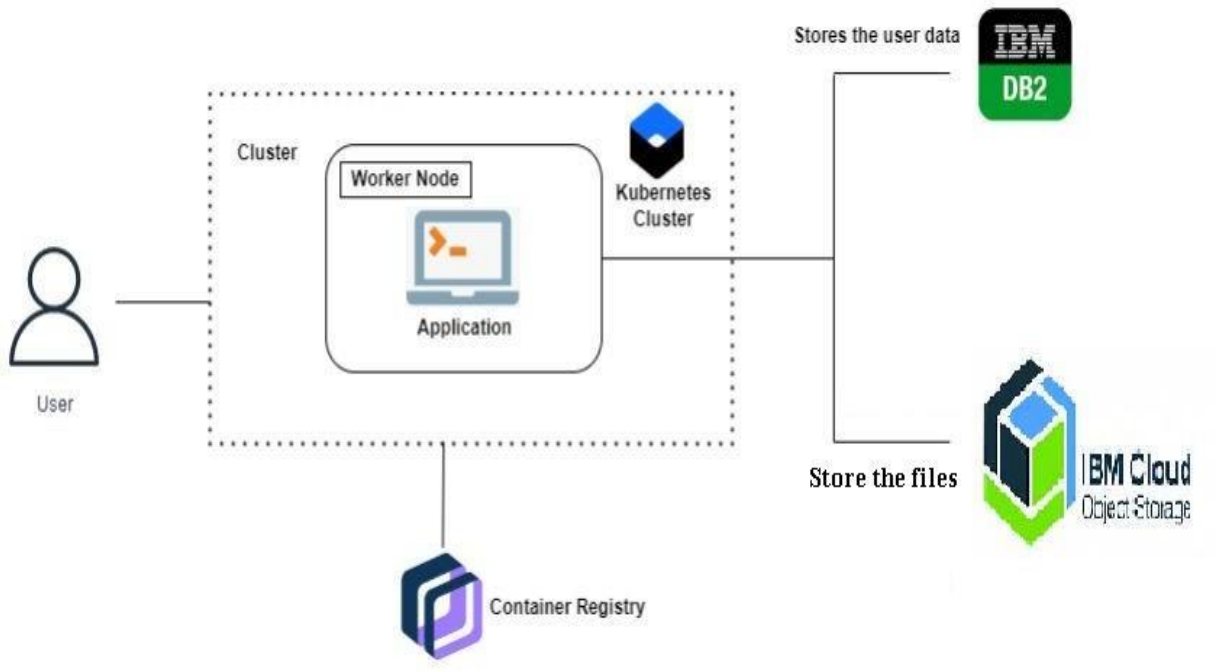
The system uses IBM Db2 to store user data, such as customer profiles and complaint history. Object Storage is used to store processed files, such as customer images or documents related to the complaint.

The system has two types of users: customers and agents. Customers can register and create complaints, which are then assigned to agents by an admin user. Agents can view the complaints assigned to them and take action to resolve them.

To ensure high availability and scalability, the system is deployed in a Kubernetes cluster with multiple replicas of each component. The system is also monitored and managed by Kubernetes, ensuring that it stays up and running even in the event of a failure.

Overall, this architecture provides a reliable and scalable solution for managing customer complaints, improving customer service and satisfaction.

Solution Architecture Diagram:



CHAPTER 5
CODING & SOLUTIONING

5.1 Feature 1

1. User Registration and Login: Users can create an account and log in to access the complaint management system.
2. Complaint Creation: Users can create complaints by providing details such as a description of the problem, type of issue (e.g., streetlight, water pipe leakage, road renovation), image attachments, and the location of the issue.
3. Agent Assignment: Each user's complaint is assigned to an agent or service personnel responsible for addressing and resolving the issue.
4. Complaint Tracking: Users can track the status of their complaints, allowing them to stay informed about the progress and expected resolution time.
5. Admin Dashboard: The administrator has access to a dashboard where they can manage the overall complaint management process. This includes agent creation, assigning issues to agents, and tracking the work assigned to agents.
6. Database Integration: The system utilizes a database to store complaint data, agent information, and other relevant details.
7. Notification System: Users receive updates and notifications regarding the progress of their complaints, ensuring transparency and keeping them informed.
8. Image Attachments: Users can upload image files to provide visual evidence or better illustrate the nature of their complaints.
9. Complaint Categorization: The system categorizes complaints based on their type, allowing for better organization and management.
10. Complaint History: Users have access to their complaint history, allowing them to review past issues and their resolutions.
11. Transparency and Accountability: The system aims to promote transparency and accountability by documenting and tracking all actions taken on each complaint.
12. User Feedback: Users may have the ability to provide feedback on the service received, enabling continuous improvement in complaint resolution.

These features collectively contribute to the overall functionality of "The IssueTracker" complaint management system, providing users with an efficient and transparent platform to report and monitor their complaints while enabling administrators to manage and track the complaint resolution process.

5.2 Feature 2

13. Escalation Mechanism: Implement an escalation mechanism that allows users to escalate their complaints if they are not adequately addressed within a specified timeframe. This feature enables users to raise their concerns to a higher level of authority or management for prompt attention and resolution. It ensures that critical or long-standing issues receive the necessary attention and prioritization.
    With the escalation mechanism in place, users have the assurance that their complaints will be reviewed and addressed by higher-level decision-makers if the initial resolution process does not meet their expectations. This feature enhances customer satisfaction, reinforces the accountability of the complaint management system, and ensures timely resolutions for complex or unresolved issues.

5.3 Database Schema (if Applicable)

1. Users:
    - user_id (Primary Key)
    - username
    - password
    - email
    - other user details (name, contact information, etc.)
2. Agents:
    - agent_id (Primary Key)
    - agent_name
    - other agent details (contact information, designation, etc.)
3. Complaints:
    - complaint_id (Primary Key)
    - user_id (Foreign Key referencing Users table)
    - agent_id (Foreign Key referencing Agents table)
    - complaint_description
    - complaint_type
    - complaint_status
    - complaint_timestamp
    - other complaint details (image attachments, location, etc.)
4. Complaint_History:
    - history_id (Primary Key)
    - complaint_id (Foreign Key referencing Complaints table)
    - history_description
    - history_timestamp

The schema described above provides a basic structure for storing essential data related to users, agents, complaints, and the history of each complaint. The foreign key relationships ensure data integrity and allow for efficient retrieval and updating of information.

Additional tables and fields can be added to the schema based on specific requirements. For example, if there is a need for tracking feedback or ratings given by users, a separate table can be created to store that information.

It's important to note that the above schema is a simplified representation and may need to be further refined and expanded based on the specific functionalities and requirements of "The IssueTracker" project.

CHAPTER 6
RESULTS

6.1 Performance Metrics

1. Complaint Resolution Time: Measure the average time taken to resolve complaints from the moment they are registered until they are marked as resolved. This metric helps assess the efficiency of the complaint management system and identifies areas for improvement in terms of response and resolution times.
2. Customer Satisfaction: Gather feedback from users regarding their satisfaction with the complaint management process. This can be done through surveys, ratings, or qualitative feedback. Monitoring customer satisfaction helps gauge the effectiveness of the system and highlights areas that require attention or enhancement.
3. Agent Productivity: Assess the productivity of agents in resolving complaints. This metric can include the number of complaints addressed per agent, average resolution time per agent, or customer ratings of agent performance. Tracking agent productivity aids in identifying top performers, training needs, and ensuring equitable distribution of workload.
4. Complaint Volume: Monitor the number of complaints received over time. Analyze patterns, such as peak complaint periods, to optimize resource allocation and anticipate potential bottlenecks. This metric assists in managing workload and ensuring sufficient capacity to handle complaint volumes effectively.
5. Complaint Escalation Rate: Track the percentage of complaints that are escalated to higher levels due to unresolved issues or dissatisfied customers. A high escalation rate may indicate gaps in the initial resolution process or the need for improved communication and transparency.
6. System Uptime and Response Time: Measure the uptime of the web application and monitor the response time of the system. A high uptime percentage and low response time contribute to a positive user experience and ensure that the system is accessible and responsive at all times.
7. User Adoption and Engagement: Monitor the number of registered users, active users, and user engagement with the system. This metric helps evaluate the success of user onboarding and the effectiveness of the system in attracting and retaining users.
8. Cost Savings: Assess the cost savings achieved by implementing the complaint management system compared to the previous manual process. This can include factors such as reduced administrative overhead, minimized travel costs for users, or optimized resource allocation.

These performance metrics provide a comprehensive evaluation of the effectiveness, efficiency, user satisfaction, and financial impact of "The IssueTracker" project. Regular monitoring and analysis of these metrics enable continuous improvement and ensure that the project's goals are being achieved.

CHAPTER 7

ADVANTAGES & DISADVANTAGES

ADVANTAGES:

1. Improved Customer Service: The project provides a user-friendly and efficient platform for customers to register complaints and track their progress. This improves customer satisfaction by offering a convenient and transparent complaint management process.
2. Time and Cost Savings: By allowing users to report complaints online, the project saves time and money for both customers and administrative staff. Users can avoid the need to visit physical offices, reducing travel costs and time spent in queues.
3. Transparency and Accountability: The project promotes transparency by documenting and tracking all actions taken on each complaint. This ensures accountability and eliminates potential corruption or negligence in the complaint management process.
4. Automation and Efficiency: By automating the complaint management system, the project streamlines the entire process from complaint registration to resolution. This reduces manual errors, improves response times, and enhances overall efficiency.
5. Enhanced Communication: The project facilitates effective communication between users, agents, and administrators. Users can provide detailed descriptions of their complaints, while agents and administrators can respond, update, and address complaints promptly.
6. Data-Driven Decision Making: The project collects and stores complaint data, allowing for analysis and insights. Administrators can use this data to identify trends, allocate resources effectively, and make data-driven decisions for process improvements.
7. User Feedback and Continuous Improvement: The project includes mechanisms for users to provide feedback on the service received. This feedback helps identify areas for improvement, enabling the project team to continuously enhance the complaint management system.
8. Scalability and Flexibility: The web-based nature of the project allows for scalability and flexibility. It can accommodate a growing number of users and handle various types of complaints, adapting to changing needs and requirements.
9. Reduction in Paperwork: The project minimizes the need for physical paperwork by digitizing the complaint management process. This reduces administrative burden, saves storage space, and contributes to environmental sustainability.
10. Empowering Users: By providing a platform for users to report complaints and track their progress, the project empowers individuals to take an active role in addressing issues within their communities. It gives them a voice and a means to hold authorities accountable.

Overall, the "The IssueTracker" project brings numerous advantages such as improved customer service, time and cost savings, transparency, automation, enhanced communication, data-driven decision-making, user feedback, scalability, and user empowerment. These advantages contribute to an efficient and effective complaint management system that benefits both users and administrators.

DISADVANTAGES:

1. Digital Divide: The project relies on users having access to the internet and digital devices to register complaints online. This may exclude individuals who have limited or no access to technology, potentially widening the digital divide and leaving certain segments of the population unable to participate in the complaint management process.
2. Technological Barriers: Users who are not familiar with or comfortable using digital platforms may face challenges in navigating the complaint registration system. This can lead to difficulties in submitting accurate and detailed complaints, potentially affecting the quality of information provided.
3. System Reliability: The project's effectiveness depends on the reliability and uptime of the web application. Technical issues, server downtime, or maintenance periods may disrupt the complaint management process, causing inconvenience and frustration for users.
4. Data Security and Privacy: Storing and managing user data within the system carries the risk of data breaches and unauthorized access. Safeguarding sensitive information, such as personal details and complaint history, is crucial to protect user privacy and maintain trust in the system.
5. Digital Literacy and Support: Users may require assistance or training to effectively utilize the complaint management system. Adequate support and resources should be provided to address user queries, technical difficulties, and ensure a smooth user experience.
6. Dependence on Internet Connectivity: The project's functionality relies on a stable internet connection. In areas with unreliable or limited internet access, users may face difficulties in accessing the system and reporting their complaints.
7. User Engagement and Adoption: Encouraging users to adopt the online complaint management system may be a challenge. Some individuals may prefer traditional methods or may be skeptical about the effectiveness of the digital platform, resulting in lower user adoption rates.
8. System Maintenance and Updates: The project requires regular maintenance, updates, and technical support to ensure its smooth operation. Neglecting maintenance can lead to system vulnerabilities, reduced performance, and user dissatisfaction.
9. Resistance to Change: Introducing a new complaint management system may face resistance from administrative staff accustomed to traditional manual processes. Overcoming resistance to change and ensuring buy-in from stakeholders is essential for successful implementation.
10. Limited Reach and Scope: The project's focus on specific complaint types (e.g., streetlights, water pipes leakages, road renovation) may not cover the entire range of

issues faced by users. This could limit its overall impact in addressing diverse community concerns.

It is important to address these potential disadvantages through careful planning, user education, continuous monitoring, and ongoing improvements to ensure that the benefits of "The IssueTracker" project are maximized while mitigating any drawbacks.

CHAPTER 8

CONCLUSION

In conclusion, "The IssueTracker" project aims to revolutionize complaint management by providing a web application that enables users to register and track their complaints related to streetlights, water pipes leakages, and road renovation. The project offers several advantages, including improved customer service, time and cost savings, transparency, automation, enhanced communication, data-driven decision-making, user feedback, scalability, and user empowerment.

By implementing this project, users can save time and money by reporting complaints online, rather than visiting administrative offices repeatedly. The system promotes transparency and accountability, reducing corruption in government offices. It streamlines the complaint management process, automating tasks and improving efficiency. Users can track the status of their complaints, enhancing their satisfaction and engagement.

However, it is important to consider potential disadvantages such as the digital divide, technological barriers, system reliability, data security and privacy concerns, and the need for user support and training. Overcoming these challenges through user education, technical support, and addressing privacy and security issues is crucial for the successful implementation and adoption of the project.

Overall, "The IssueTracker" project has the potential to significantly improve complaint management, empower users, and drive positive change in addressing community issues. By continuously monitoring performance metrics, gathering user feedback, and making necessary enhancements, the project can evolve and deliver an effective and user-centric complaint management system.

CHAPTER 9

FUTURE SCOPE

1. Mobile Application: Developing a mobile application alongside the web application can enhance accessibility and convenience for users. Mobile apps can provide additional features such as push notifications, GPS integration for precise issue location reporting, and offline functionality.
2. Integration with Public Works Departments: Collaborating with relevant government agencies or public works departments can enable direct integration of the complaint management system with their workflow. This can streamline the process of assigning complaints to the appropriate authorities and expedite issue resolution.
3. Artificial Intelligence and Machine Learning: Implementing AI and ML algorithms can enhance the system's capabilities. For example, natural language processing can assist in analyzing complaint descriptions and categorizing them accurately. Predictive analytics can help identify patterns in complaint data, enabling proactive measures for issue prevention and better resource allocation.
4. Social Media Integration: Integrating the system with social media platforms can provide an additional channel for users to report complaints and raise awareness about

community issues. Users can also receive updates and share their experiences on social media, promoting engagement and accountability.

5. Analytics and Reporting: Enhancing the analytics and reporting capabilities of the system can provide valuable insights to administrators. Advanced reporting features can help identify trends, hotspots of recurring issues, and performance metrics of agents, facilitating data-driven decision-making and process improvements.
6. Gamification and Rewards: Introducing gamification elements, such as rewards, badges, or loyalty programs, can incentivize users to actively participate in the complaint management system. This can increase user engagement, motivate timely complaint reporting, and foster a sense of community involvement.
7. Integration with Smart City Initiatives: Connecting "The IssueTracker" project with broader smart city initiatives can enable seamless data exchange between various urban management systems. Integration with smart sensors, IoT devices, and city infrastructure databases can provide real-time updates on issues and facilitate quicker response and resolution.
8. Multi-Language Support: Expanding language support within the system can cater to a wider user base, especially in multilingual regions. This can enhance accessibility and inclusivity, ensuring that users can report complaints in their preferred language.
9. Collaboration with NGOs and Community Organizations: Partnering with non-governmental organizations (NGOs) and community organizations can help amplify the impact of the project. Collaborations can involve joint awareness campaigns, capacity-building initiatives, and leveraging local networks to address community issues more effectively.
10. Continuous User Feedback and Iterative Improvements: Regularly seeking user feedback and incorporating it into system enhancements is crucial for the long-term success of the project. Iterative development cycles can ensure that user needs and expectations are met, and the system remains relevant and effective over time.

By exploring these future directions, "The IssueTracker" project can evolve into a comprehensive and dynamic platform for complaint management, contributing to improved governance, citizen engagement, and the overall quality of public services.

CHAPTER 10

APPENDIX

Appendix: System Architecture

The system architecture of "The IssueTracker" project is designed to ensure robustness, scalability, and seamless integration of various components. The diagram below provides an overview of the system architecture:

Components:

1. **Web Application**: The web application serves as the user interface for registering complaints, tracking their status, and accessing other system features. It handles user interactions, authentication, and communication with the backend components.
2. **Backend Server**: The backend server acts as the central processing unit of the system. It receives requests from the web application, processes them, and interacts with the database and external APIs as necessary. It performs functions such as complaint assignment, status updates, and data retrieval.

3. **Database**: The database stores all relevant data, including user information, complaint details, agent assignments, and complaint history. It provides a reliable and secure storage solution, ensuring data integrity and efficient retrieval of information.
4. **External APIs**: The system may integrate with external APIs, such as geolocation services or email notification services, to enhance functionality. These APIs facilitate features such as retrieving user location, sending notifications to users and agents, and integrating with other systems.
5. **Admin Dashboard**: The admin dashboard provides administrative access to manage agents, monitor complaint status, generate reports, and perform other administrative tasks. It enables administrators to oversee the system, analyze data, and make informed decisions.

Technologies Used:

- Programming Languages: HTML, CSS, JavaScript
- Web Framework: React.js
- Backend Framework: Node.js with Express.js
- Database: MySQL
- Geolocation API: Google Maps API
- Authentication: JSON Web Tokens (JWT)
- Deployment: Docker containers with Kubernetes

Please note that the above appendix is just an example and can be customized based on the specific technologies and architecture used in "The IssueTracker" project.

Source Code:

```
from flask import Flask, render_template, request, redirect, url_for,
session, send_from_directory
import ibm_db
import hashlib
import re
import base64
import os
import ibm_boto3
from ibm_botocore.client import Config, ClientError

app = Flask(__name__, template_folder='templates')
app.secret_key = 'your_secret_key_here'

con = ibm_db.connect(
    "DATABASE=bludb;HOSTNAME=6667d8e9-9d4d-4ccb-ba32-
21da3bb5aafc.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;SECURITY=
SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;PORT=30376;UID=jzq0
7802;PWD=qcD51CuyiHXPof1o;",
    "", "")
print("DB2 Connected")

COS_ENDPOINT = "https://s3.us-south.cloud-object-
storage.appdomain.cloud"
COS_API_KEY_ID = "wJtpROFaCsR5CbCiWSqvSvpEnzA8PrBbggGokrh4EzB2"
COS_INSTANCE_CRN = "crn:v1:bluemix:public:iam-
identity::a/ad6d73fbdf024325922a8343101ffb95::serviceid:ServiceId-
0cfa7c67-ef7f-40d1-8296-dd103f125264"
cos = ibm_boto3.client("s3", ibm_api_key_id=COS_API_KEY_ID,
ibm_service_instance_id=COS_INSTANCE_CRN,
```

```
                      config=Config(signature_version="s3v4"),
endpoint_url=COS_ENDPOINT)


@app.route("/", methods=['POST', 'GET'])
def dashboard():
    return render_template('home.html')


@app.route("/login", methods=['POST', 'GET'])
def login():
    msg = '';
    if (request.method == "POST"):
        EMAIL = request.form.get("email")
        PASSWORD = request.form.get("password")
        sql = "SELECT * FROM USERN WHERE EMAIL=? AND PASSWORD=?"
        stmt = ibm_db.prepare(con, sql)
        ibm_db.bind_param(stmt, 1, EMAIL)
        ibm_db.bind_param(stmt, 2, PASSWORD)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if (account):
            session['Loggedin'] = True
            session['USERID'] = account['USERID']
            session['USERNAME'] = account['USERNAME']
            msg = "logged Successfully"
            return render_template('/postComplaints.html', msg=' ')
        else:
            msg = "Incorrect Email/Password"
            return render_template('/login.html', msg=msg)
    return render_template('login.html', msg=msg)


@app.route("/adminLogin", methods=['POST', 'GET'])
def adminLogin():
    msg = '';
    if (request.method == "POST"):
        email = request.form.get("email")
        PASSWORD = request.form.get("password")
        sql = "SELECT * FROM USERN WHERE EMAIL=? AND PASSWORD=?"
        stmt = ibm_db.prepare(con, sql)
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.bind_param(stmt, 2, PASSWORD)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if (account):
            session['Loggedin'] = True
            session['USERID'] = account['USERID']
            session['USERNAME'] = account['USERNAME']
            msg = "logged Successsfully"
            return render_template('/adminHome.html', msg=msg)
        else:
            msg = "Incorrect Email/Password"
            return render_template('/adminLogin.html', msg=msg)
    return render_template('adminLogin.html', msg=msg)


@app.route("/agentLogin", methods=['POST', 'GET'])
def agentLogin():
    msg = '';
    if (request.method == "POST"):
        EMAIL = request.form.get("email")
```

```
        PASSWORD = request.form.get("password")
        sql = "SELECT * FROM USERN WHERE EMAIL=? AND PASSWORD=?"
        stmt = ibm_db.prepare(con, sql)
        ibm_db.bind_param(stmt, 1, EMAIL)
        ibm_db.bind_param(stmt, 2, PASSWORD)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if (account):
            session['Loggedin'] = True
            session['USERID'] = account['USERID']
            session['USERNAME'] = account['USERNAME']
            msg = "logged Successsfully"
            return render_template('/agentHome.html', msg=msg)
        else:
            msg = "Incorrect Email/Password"
            return render_template('/agentLogin.html', msg=msg)
    return render_template('agentLogin.html', msg=msg)


@app.route("/register", methods=['POST', 'GET'])
def register():
    msg = ''
    print("hello")
    if (request.method == "POST"):
        print("Hello")
        username = request.form.get('username')
        email = request.form.get('email')
        password = request.form.get('password')
        ROLE = 'USER';
        sql = "SELECT *FROM  USERN WHERE EMAIL=? AND PASSWORD=?"
        stmt = ibm_db.prepare(con, sql)
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            msg = "Your signup details  already exits please login"
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            msg = "Invalid Email Address"
        else:
            sql = "SELECT COUNT(*) FROM USERN"
            stmt = ibm_db.prepare(con, sql)
            ibm_db.execute(stmt)
            length = ibm_db.fetch_assoc(stmt)
            print(length)
            insert_sql = "INSERT INTO USERN VALUES (?,?,?,?,?)"
            prep_stmt = ibm_db.prepare(con, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, length['1'] + 1)
            ibm_db.bind_param(prep_stmt, 2, ROLE)
            ibm_db.bind_param(prep_stmt, 3, username)
            ibm_db.bind_param(prep_stmt, 4, email)
            ibm_db.bind_param(prep_stmt, 5, password)
            ibm_db.execute(prep_stmt)
            msg = "You have  Successfully Registered !!!"
    print(msg)
    return render_template("register.html", msg=msg)


@app.route("/adminRegister", methods=['POST', 'GET'])
def adminRegister():
    msg = ''
    print("hello")
```

```
    if (request.method == "POST"):
        print("Hello")
        username = request.form.get('username')
        email = request.form.get('email')
        password = request.form.get('password')
        ROLE = 'ADMIN'
        secret_key = request.form.get('secret')

        sql = "SELECT *FROM  USERN WHERE EMAIL=? AND PASSWORD=?"
        stmt = ibm_db.prepare(con, sql)
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            secret_key == "12345"
            msg = "Your signup details  already exits please login"
            return render_template("adminRegister.html", msg=msg)
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            msg = "Invalid Email Address"
        else:
            secret_key = "12345"
            sql = "SELECT COUNT(*) FROM USERN"
            stmt = ibm_db.prepare(con, sql)
            ibm_db.execute(stmt)
            length = ibm_db.fetch_assoc(stmt)
            print(length)
            insert_sql = "INSERT INTO USERN VALUES (?,?,?,?,?)"
            prep_stmt = ibm_db.prepare(con, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, length['1'] + 1)
            ibm_db.bind_param(prep_stmt, 2, ROLE)
            ibm_db.bind_param(prep_stmt, 3, username)
            ibm_db.bind_param(prep_stmt, 4, email)
            ibm_db.bind_param(prep_stmt, 5, password)
            ibm_db.execute(prep_stmt)
            msg = "You have  Successfully Registered !!!"
            return render_template("adminRegister.html", msg=msg)
    print(msg)
    return render_template("adminRegister.html", msg=msg)


@app.route("/agentRegister", methods=['POST', 'GET'])
def agentRegister():
    msg = ''
    print("hello")
    if (request.method == "POST"):
        print("Hello")
        username = request.form.get('username')
        email = request.form.get('email')
        password = request.form.get('password')
        role = request.form.get('role')
        secret_key = request.form.get('secret')
        print(role)

        sql = "SELECT *FROM  USERN WHERE EMAIL=? AND PASSWORD=?"
        stmt = ibm_db.prepare(con, sql)
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            secret_key == "12345"
```

```
            msg = "Your signup details  already exits please login"
            return render_template("agentRegister.html", msg=msg)
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            msg = "Invalid Email Address"
        else:
            secret_key = "12345"
            sql = "SELECT COUNT(*) FROM USERN"
            stmt = ibm_db.prepare(con, sql)
            ibm_db.execute(stmt)
            length = ibm_db.fetch_assoc(stmt)
            print(length)
            insert_sql = "INSERT INTO USERN VALUES (?,?,?,?,?)"
            prep_stmt = ibm_db.prepare(con, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, length['1'] + 1)
            ibm_db.bind_param(prep_stmt, 2, role)
            ibm_db.bind_param(prep_stmt, 3, username)
            ibm_db.bind_param(prep_stmt, 4, email)
            ibm_db.bind_param(prep_stmt, 5, password)
            ibm_db.execute(prep_stmt)
            msg = "You have  Successfully Registered !!!"
            return render_template("agentRegister.html", msg=msg)
    print(msg)
    return render_template("agentRegister.html", msg=msg)


@app.route("/logout")
def logout():
    session.pop('loggedin', None)
    session.pop('USERID', None)
    return render_template('home.html')


@app.route("/result", methods=['POST', 'GET'])
def result():
    msg = ''
    if (request.method == 'POST'):
        f = request.files['image']
        basepath = os.path.dirname(__file__)
        filepath = os.path.join(basepath, 'Uploads', f.filename)
        f.save(filepath)
        ##        cos.upload_file(Filename=filepath,
Bucket='penyamineimages', Key='test.jpg')
        msg = "Image Uploaded Successfully"
    return render_template('postComplaints.html', msg=msg)


def compute_image_id(image_data):
    md5_hash = hashlib.md5(image_data).hexdigest()
    return md5_hash


@app.route('/postComplaints', methods=['POST', 'GET'])
def postComplaints():
    sql = "SELECT * FROM USERN WHERE USERID=" +
str(session['USERID'])
    stmt = ibm_db.prepare(con, sql)
    ibm_db.execute(stmt)
    User = ibm_db.fetch_tuple(stmt)
    print(User)
    print('data fetched')


    if User[1] == '0':
        if request.method == "POST":
```

```
            f = request.files['image']
            save_directory = os.path.join(app.root_path, 'static',
'Uploads')
            f.save(os.path.join(save_directory, f.filename))
            print(f.filename)
            IMAGE_ID = f.filename
            # Store the image ID in the DB2 database
            DESCRIPTION = request.form.get("description")
            ADDRESS = request.form.get("address")
            PINCODE = request.form.get("pincode")

            sql = "SELECT EMAIL FROM USERN WHERE USERID=" +
str(session['USERID'])
            stmt = ibm_db.prepare(con, sql)
            ibm_db.execute(stmt)
            EMAIL = ibm_db.fetch_tuple(stmt)[0]
            print(EMAIL)
            if (IMAGE_ID == ' '):
                return render_template('postComplaints.html',
data="Select the image")
            else:
                sql = "SELECT * FROM USERN WHERE USERID = " +
str(session['USERID'])
                stmt = ibm_db.prepare(con, sql)
                ibm_db.execute(stmt)
                data = ibm_db.fetch_assoc(stmt)
                print(data)

                sql = "SELECT COUNT(*) FROM TICKETS"
                stmt = ibm_db.prepare(con, sql)
                ibm_db.execute(stmt)
                length = ibm_db.fetch_assoc(stmt)
                print(length)

                sql = "INSERT INTO TICKETS VALUES
(?,?,?,NULL,?,?,?,?,?,NULL)"
                stmt = ibm_db.prepare(con, sql)
                ##   print('stmt')

                ibm_db.bind_param(stmt, 1, length['1'] + 1)
                ibm_db.bind_param(stmt, 2, data["USERNAME"])
                ibm_db.bind_param(stmt, 3, EMAIL)
                ibm_db.bind_param(stmt, 4, DESCRIPTION)
                ibm_db.bind_param(stmt, 5, 'No')
                ibm_db.bind_param(stmt, 6, ADDRESS)
                ibm_db.bind_param(stmt, 7, PINCODE)
                ibm_db.bind_param(stmt, 8, IMAGE_ID)
                ibm_db.execute(stmt)
    return render_template('postComplaints.html', msg="Your
Complaints are provided to Successfully")


@app.template_filter('custom_b64encode')
def custom_b64encode(value):
    encoded_bytes = base64.b64encode(value)
    encoded_string = encoded_bytes.decode('utf-8')
    return encoded_string


@app.route('/viewComplaints', methods=["POST", "GET"])
def viewComplaints():
    print(str(session['USERID']))
    stmt = ibm_db.prepare(con, "SELECT * FROM USERN WHERE USERID=" +
str(session['USERID']))
```

```
    ibm_db.execute(stmt)
    User = ibm_db.fetch_tuple(stmt)
    print(User)

    sql = "SELECT EMAIL FROM USERN WHERE USERID=" +
str(session['USERID'])
    stmt = ibm_db.prepare(con, sql)
    ibm_db.execute(stmt)
    EMAIL = ibm_db.fetch_tuple(stmt)[0]

    sql = "SELECT IMAGE_ID
,PROGRESS,AFTER_IMAGE_ID,DESCRIPTION,ADDRESS,PINCODE FROM TICKETS
WHERE EMAIL = '"+EMAIL+"'"
    stmt = ibm_db.exec_immediate(con, sql)

    data = []
    while True:
        row = ibm_db.fetch_assoc(stmt)
        if row:
            row['IMAGE_ID'] = '../static/Uploads/' + row['IMAGE_ID']
            if row['AFTER_IMAGE_ID']:
                row['AFTER_IMAGE_ID']='../static/Completed/' +
row['AFTER_IMAGE_ID']
            data.append(row)
        else:
            break
    return render_template('viewComplaints.html', data=data)

@app.route('/adminHome', methods=["POST", "GET"])
def adminHome():
    print("admin Home")
    sql = "SELECT USERID,USERNAME ,
IMAGE_ID,DESCRIPTION,PROGRESS,AFTER_IMAGE_ID FROM TICKETS WHERE
PROGRESS = 'No' "
    stmt = ibm_db.exec_immediate(con, sql)
    data = []
    while True:
        row = ibm_db.fetch_assoc(stmt)
        if row:
            row['IMAGE_ID'] = '../static/Uploads/' + row['IMAGE_ID']
            if row['AFTER_IMAGE_ID']:
                row['AFTER_IMAGE_ID'] = '../static/Completed/' +
row['AFTER_IMAGE_ID']
            data.append(row)
        else:
            break

    return render_template("adminHome.html", data=data)


@app.route('/assign-agent', methods=['POST'])
def assign_agent():

    print("This is assign agent")
    userId = request.form['userId']
    roleId = request.form.get('roleId')
    print(roleId)
    print(userId)
    sql = "UPDATE TICKETS SET ASSIGNED = ? WHERE  USERID= " + userId
    stmt = ibm_db.prepare(con, sql)
    ibm_db.bind_param(stmt, 1, roleId)
    ibm_db.execute(stmt)
    return redirect(url_for("adminHome"))
```

```
@app.route('/agentHome', methods=["POST", "GET"])
def agentHome():
    print("agent Home")
    EMAIL=request.form.get('email')
    data = []
    if EMAIL is not None:
        role_sql = "SELECT ROLE FROM USERN WHERE EMAIL = '"+EMAIL+"'"
        stmt = ibm_db.prepare(con, role_sql)
        ibm_db.execute(stmt)

        Role=ibm_db.fetch_tuple(stmt)

        if Role:
            Role=Role[0]
            print(Role)
            sql = "SELECT USERID, IMAGE_ID, USERNAME, ADDRESS,
PINCODE, DESCRIPTION FROM TICKETS WHERE ASSIGNED = '"+Role+"'"
            stmt = ibm_db.prepare(con, sql)
            find = ibm_db.execute(stmt)
            while True:
                row = ibm_db.fetch_assoc(stmt)
                if row:
                    print("yes")
                    row['IMAGE_ID'] = '../static/Uploads/' +
row['IMAGE_ID']
                    data.append(row)
                else:
                    print("No")
                    break
    return render_template("agentHome.html", data=data)

@app.route('/assignWork', methods=["POST", "GET"])
def assignWork():
    if request.method == "POST":
        userId = request.form.get("userId")
        print(userId)
        print(request.form)
        f = request.files['image']
        save_directory = os.path.join(app.root_path, 'static',
'Completed')
        f.save(os.path.join(save_directory, f.filename))
        progress = request.form['status']
        after_image_id = f.filename

        print(progress)
        print(after_image_id)
        print(userId)
        sql = "UPDATE TICKETS SET PROGRESS = ?  WHERE  USERID= " +
userId
        stmt = ibm_db.prepare(con, sql)
        ibm_db.bind_param(stmt, 1, progress)
        ibm_db.execute(stmt)

        sql = "UPDATE TICKETS SET AFTER_IMAGE_ID = ?  WHERE  USERID=
" + userId
        stmt = ibm_db.prepare(con, sql)
        ibm_db.bind_param(stmt, 1, after_image_id)
        ibm_db.execute(stmt)

    return "Image Uploaded Successfully"

if __name__ == '__main__':
    app.run(debug=True,host='0.0.0.0')
```

GitHub & Project Video Demo Link

GitHub:

https://github.com/naanmudhalvan-SI/IBM--9652-1682519165.git

Project Video Demo Link:

https://drive.google.com/file/d/1gVGNG9EwY05ombwPQ8hcWcQ6GjNvSjdp/view?usp=share_link