DATA SCIENECE

ONLINE COMPLAINT REGISTRATION AND MANAGEMENT SYSTEM

Submitted by

In partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE AND

DATA SCIENCE

THANGAVELU ENGINEERING COLLEGE

KARAPAKKAM

CHENNAI 600097

ANNA UNIVERSITY

CHENNAI 600025

November 2024

THANGAVELU ENGINEERING COLLEGE

ANNA UNIVERSITY CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this internship of Data Science by Hamari Pahchan's is the bonafide work of Dilli babu .D who carried out the internship under my supervision.

SIGNATURE

MRS.D.BEULAH PRETTY M.E.,

HEAD OF THE DEPARTMENT

Department of artificial intelligence

And Data Science

Thangavelu Engineering College

Thoraipakkam,Chennai-600097

SIGNATURE

MS.C.KANAGALAKSHMI M.E.,

SUPERVISOR

ASSISTANT PROFESSOR

Department of artificial intelligence

And Data Science

Thangavelu Engineering College

Thoraipakkam,Chennai-600097

ACKNOWLEDGEMENT

The success and final outcome of learning the MERN stack required immense guidance and assistance from many people. I feel extremely privileged to have received such support throughout the completion of my course and various projects. All that I have accomplished is due to the valuable supervision and assistance provided to me, and I would like to express my heartfelt gratitude to everyone involved. I am deeply thankful and fortunate to have received constant encouragement, support, and guidance from all the teaching staff. Their invaluable help has been instrumental in successfully completing my internship and project work.

_____

(Signature of Student)

Name: Dilli babu.D

Reg no:312621243011

Date: _____

TEAM MEMBERS:

1. Dilli babu.D (TL)
2. Harish.S
3.Emi hedrif
4.Gayathri

ONLINE COMPLAINT REGISTRATION AND MANAGEMENT SYSTEM

- ER DIAGRAM
- PRE REGUISTES
- PROJECT STRUCTURE
- PROJECT FLOW

ER DIAGRAM

An **Online Complaint Registration and Management System** is a digital platform designed to facilitate the efficient handling of complaints from users, customers, or stakeholders. Below is a **brief explanation** of its key components and functionalities:

Purpose

To streamline the process of lodging, tracking, managing, and resolving complaints in an organized and transparent manner.

- ER DIAGRAM
- ere's a simple **ER Diagram (Entity-Relationship Diagram)** outline for an **Online Complaint Registration and Management System**:
- ---
- **Entities and Relationships**

- **1. User**
- Attributes: User_ID (Primary Key), Name, Email, Password, Contact_Number, Address
- Relationships:
- Submits → Complaint
- **2. Complaint**
- Attributes: Complaint_ID (Primary Key), User_ID (Foreign Key), Complaint_Type, Description, Submission_Date, Status (e.g., Pending, Resolved), Resolution_Date
- Relationships:
- Assigned to → Admin/Staff
- Related to → Category
- **3. Admin/Staff**
- Attributes: Admin_ID (Primary Key), Name, Role, Contact_Number, Email
- Relationships:
- Manages → Complaint
- **4. Category**
- Attributes: Category_ID (Primary Key), Name, Description
- Relationships:
- Has → Complaint
- **5. Feedback**
- Attributes: Feedback_ID (Primary Key), User_ID (Foreign Key), Complaint_ID (Foreign Key), Rating, Comments, Feedback_Date
- Relationships:
- Linked to → Complaint
- ---
- **ER Diagram (Description)**
- **User** entity is connected to **Complaint** via a "Submits" relationship (one-to-many).
- **Complaint** is linked to **Admin/Staff** via an "Assigned to" relationship (many-to-one).

- **Complaint** is linked to **Category** via a "Related to" relationship (many-to-one).
- **Complaint** is connected to **Feedback** via a "Linked to" relationship (one-to-one).
- **Feedback** is also connected to **User** as it originates from the user.
- ---
- Would you like a **visual representation** of this ER diagram? If so, I can create an image for you.
- 4o

PRE REGUISTES

Technical Prerequisites

a. Hardware Requirements

- **Servers**:
    - Web Server (e.g., Apache, Nginx).
    - Database Server (e.g., MySQL, PostgreSQL).
- **Client Devices**:
    - Desktop, mobile, or tablet for end-users and administrators.
    - Storage space depending on expected complaint volume.

b. Software Requirements

- **Programming Languages**:
    - Backend: Python, Java, PHP, Node.js.
    - Frontend: HTML, CSS, JavaScript, frameworks like React, Angular, or Vue.js.
- **Database**:

    - Relational databases like MySQL, PostgreSQL, or NoSQL like MongoDB.
- **Development Tools**:
    - IDE (e.g., VS Code, IntelliJ IDEA).
    - Version Control (e.g., Git).
- **Operating System**:
    - Linux/Windows for servers.
- **Middleware and Frameworks**:
    - Django, Flask, Laravel, or Spring for backend logic.
    - APIs for integration (REST/GraphQL).
- **Hosting Environment**:
    - Cloud providers like AWS, Azure, Google Cloud, or on-premise servers.

c. Network Requirements

- Reliable internet connection for 24/7 accessibility.
- SSL/TLS for secure data transfer.

---

2. Functional Prerequisites

- **User Authentication and Authorization**:
    - Secure login for users and admins with role-based access control.
- **Complaint Categorization**:

    - Define categories for complaints (e.g., technical, administrative, service-related).
- **Tracking Mechanism**:
    - Assign unique complaint IDs.
    - Enable real-time status updates.
- **Notification System**:
    - Email/SMS APIs for automated alerts.
- **Feedback Mechanism**:
    - System to collect and analyze user feedback.

---

3. Organizational Prerequisites

- **Requirement Analysis**:
    - Define objectives and scope.
    - Identify stakeholders (users, admin teams, IT staff).
- **Team Setup**:
    - Developers, testers, UI/UX designers, and project managers.
- **Training and Documentation**:
    - Manuals or tutorials for users and administrators.
- **Support Infrastructure**:
    - IT helpdesk for addressing technical issues.

---

4. Legal and Compliance Requirements

- **Data Privacy Laws**:
    - Ensure compliance with GDPR, CCPA, or other local data protection regulations.
- **Accessibility Standards**:
    - Follow WCAG guidelines for an inclusive user experience.
- **Terms and Conditions**:
    - Clear policies for complaint submission and resolution.

---

5. Test Environment

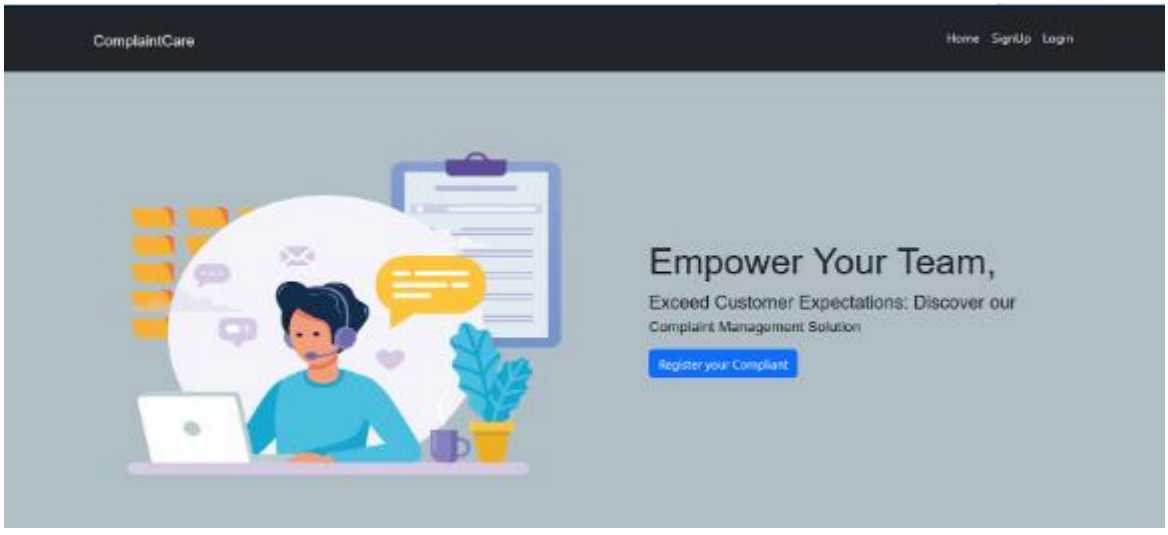
- A separate staging environment for testing and bug fixes before going live.

Would you like further details on any specific aspect, like technical tools or user interface design

PROJECT FLOW

Project Implementation

On completing the development part, we then run the application one last time to verify all the functionalities and look for any bugs in it. The user interface of the application looks a bit like the one's provided below.



Project implementation

Planning Phase

- **Requirement Gathering**:
    - Identify all functional and non-functional requirements from stakeholders.
    - Define user roles (e.g., User, Admin, Staff).
    - Outline use cases (e.g., complaint registration, tracking, resolution).
- **Feasibility Study**:
    - Assess technical, operational, and financial viability.
- **Project Plan**:
    - Create a roadmap with timelines, milestones, and deliverables.
    - Allocate resources (team, budget, tools).

---

2. Design Phase

- **System Design**:
    - Create architecture diagrams, including ER diagrams and flowcharts.
    - Design database schemas for entities like User, Complaint, and Feedback.
- **UI/UX Design**:
    - Develop user-friendly interfaces for web and mobile platforms.

- Use tools like Figma, Adobe XD, or Sketch for mockups and prototypes.

- **Technology Stack Selection**:
  - Decide on the programming languages, frameworks, and databases.

---

3. Development Phase

- **Frontend Development**:
  - Implement the user interface using HTML, CSS, and JavaScript frameworks (e.g., React, Angular).
- **Backend Development**:
  - Develop APIs and server-side logic using frameworks like Django, Flask, or Spring Boot.
- **Database Development**:
  - Set up the database schema and configure the database server.
- **Integration**:
  - Integrate the frontend with the backend via REST or GraphQL APIs.

---

4. Testing Phase

- **Unit Testing**:
  - Test individual components for functionality and reliability.
- **System Testing**:
  - Test the complete system for compatibility and performance.
- **User Acceptance Testing (UAT)**:
  - Allow end-users to test the system for usability and effectiveness.
- **Bug Fixing**:
  - Address issues identified during testing.

---

5. Deployment Phase

- **Hosting Setup**:
  - Deploy the system on a cloud platform (e.g., AWS, Azure) or on-premise servers.

- **Domain Configuration**:
    - Link the system to a domain name for accessibility.
- **SSL Certification**:
    - Secure the system with SSL/TLS.
- **Go-Live**:
    - Launch the system for public or restricted access as planned.

---

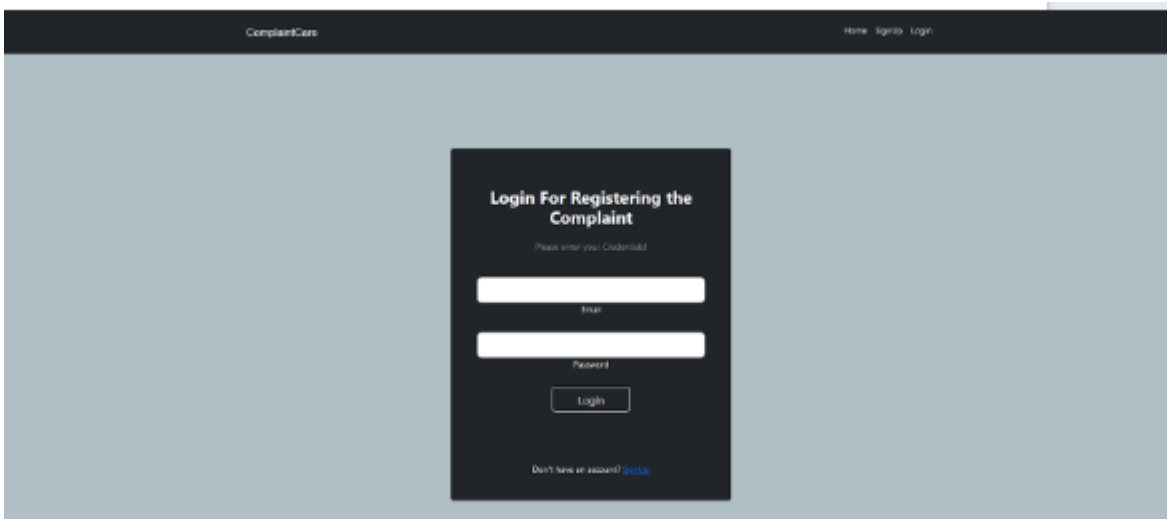
6. Post-Implementation Phase

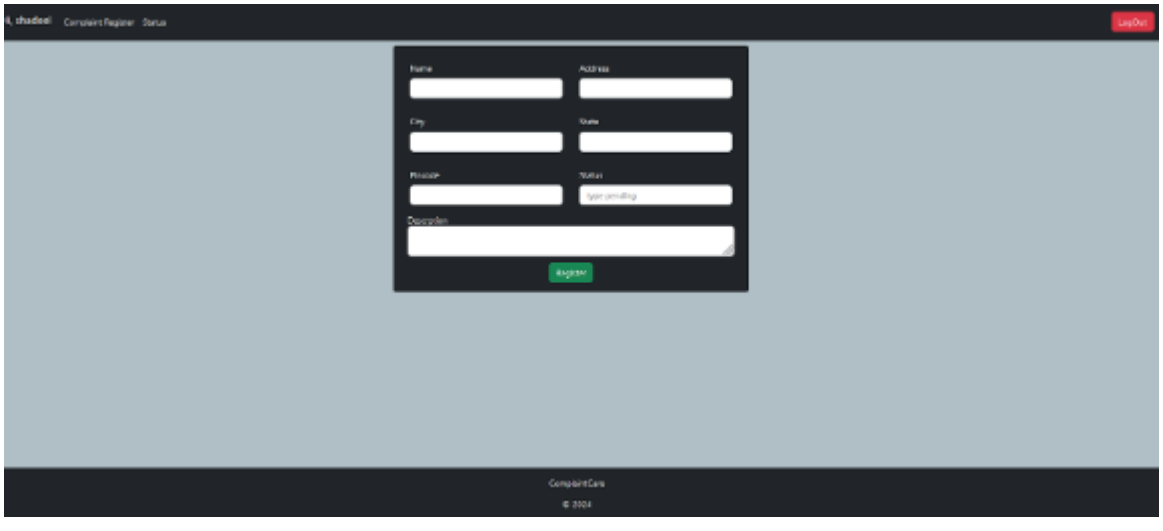
- **Monitoring**:
    - Monitor the system for performance, user activity, and issues.
- **User Training**:
    - Provide training materials or sessions for users and admins.
- **Maintenance**:
    - Regular updates for bug fixes, security patches, and feature enhancements.
- **Feedback Collection**:
    - Gather feedback from users to improve the system.

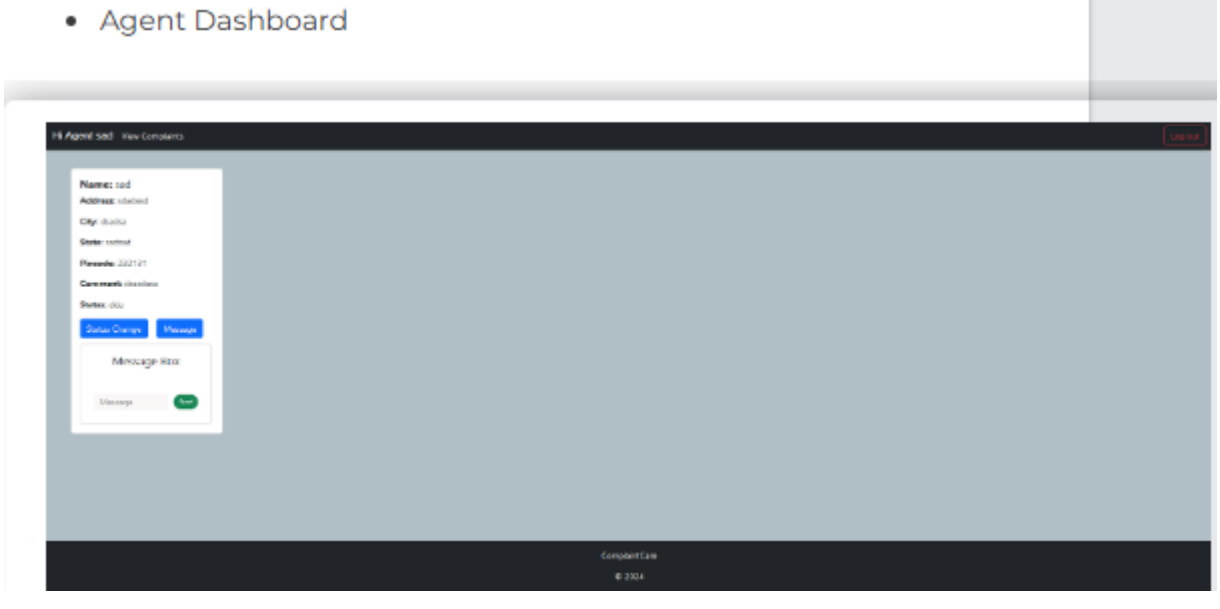
Tools for Implementation

- **Project Management**: Jira, Trello, or Asana.
- **Code Management**: GitHub, GitLab, or Bitbucket.
- **Testing Tools**: Selenium, Postman, or JMeter.

Would you like detailed examples or templates for any specific phase







Frontend Development

1. **Setup React Application:**

Bringing Customer Care Registry to life involves a three-step development process. First, a solid foundation is built using React.js. This includes creating the initial application structure, installing necessary

libraries, and organizing the project files for efficient development. Next, the user interface (UI) comes to life. To start the development process for the frontend, follow the below steps.

- Install required libraries.
- Create the structure directories.

2. **Design UI components:**

Reusable components will be created for all the interactive elements you'll see on screen, from stock listings and charts to buttons and user profiles. Next, we'll implement a layout and styling scheme to define the overall look and feel of the application. This ensures a visually-appealing and intuitive interface. Finally, a navigation system will be integrated, allowing you to effortlessly explore different sections of Customer Care Registry, like making specific complaints or managing your Product complaints.

3. **Implement frontend logic:**

In the final leg of the frontend development, we'll bridge the gap between the visual interface and the underlying data. It involves the below stages.

- Integration with API endpoints.
- Implement data binding.

Database Development

1. **User Schema:**
    - The user schema defines the structure of user data stored in the database. It includes fields such as name, email, password, phone, and userType.
    - Each user must provide a name, email, password, phone number, and userType (e.g., customer, agent, admin).
    - User data is stored in the "user_Schema" collection in the MongoDB database.
2. **Complaint Schema:**
    - The complaint schema specifies the format of complaint data registered by users.
    - It contains fields like userId, name, address, city, state, pincode, comment, and status.
    - Complaints are associated with users through the userId field, and each complaint must have a name, address, city, state, pincode, comment, and status.
    - Complaint data is stored in the "complaint_schema" collection in the MongoDB database.
3. **Assigned Complaint Schema:**
    - The assigned complaint schema defines how complaints are assigned to agents for resolution.
    - It includes fields such as agentId, complaintId, status, and agentName.

- Each assigned complaint is linked to a specific agent (identified by agentId) and complaint (identified by complaintId).
- The status field indicates the current status of the assigned complaint.
- Assigned complaint data is stored in the "assigned_complaint" collection in the MongoDB database.

4. **Chat Window Schema:**
    - The chat window schema governs the structure of messages exchanged between users and agents regarding specific complaints.
    - It comprises fields like name, message, and complaintId.
    - Messages are associated with a complaint through the complaintId field, allowing for easy tracking and retrieval of chat history for each complaint.
    - Message data is stored in the "message" collection in the MongoDB database

Backend Development

- **Set Up Project Structure:**
- Create a new directory for your project and set up a package.json file using npm init command.
- Install necessary dependencies such as Express.js, Mongoose, and other required packages.
- **Set Up Project Structure:**
    - Create a new directory for your project and set up a package.json file using npm init command.
    - Install necessary dependencies such as Express.js, Mongoose, and other required packages.
- **Create Express.js Server:**
    - Set up an Express.js server to handle HTTP requests and serve API endpoints.
    - Configure middleware such as body-parser for parsing request bodies and cors for handling cross-origin requests.

- **Define API Routes:**
    - Create separate route files for different API functionalities such as authentication, stock actions, and transactions.
    - Implement route handlers using Express.js to handle requests and interact with the database.

- **Implement Data Models:**
    - Define Mongoose schemas for the different data entities like Bank, users, transactions, deposits and loans.
    - Create corresponding Mongoose models to interact with the MongoDB database.
    - Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.
- **User Authentication:**
    - Implement user authentication using strategies like JSON Web Tokens (JWT) or session-based authentication.
    - Create routes and middleware for user registration, login, and logout.
    - Set up authentication middleware to protect routes that require user authentication.
- **Handle new transactions:**
    - Allow users to make transactions to other users using the user's account id.
    - Update the transactions and account balance dynamically in real-time.
- **Admin Functionality:**
    - Implement routes and controllers specific to admin functionalities such as fetching all the data regarding users, transactions, stocks and orders.
- **Error Handling:**
    - Implement error handling middleware to catch and handle any errors that occur during the API requests.
    - Return appropriate error responses with relevant error messages and HTTP status codes.