**Project Design Phase-II**

**Technology Stack (Architecture & Stack)**

|  |  |
| --- | --- |
| **DATE:** | **26-06-2025** |
| **Team ID :** | **LTVIP2025TMID52692** |
| **Project Name :** | **ResolveFlow: Online Complaint Registration and Management System** |

**1. Architectural Style**

The system adopts a **Client-Server Architecture**. This style ensures a clear separation between the user-facing interface and the underlying data processing and storage. All communication between these layers is standardized via **RESTful APIs**, promoting modularity, scalability, and ease of maintenance.

**2. Core System Components & Their Roles**

The system's functionality is delivered through distinct yet interconnected components:

* **Frontend (Client-Side):**
  + **Purpose:** Manages user interaction and presents the UI/UX.
  + **Functions:** Handles user registration/login, complaint submission, displays user/admin/agent dashboards for tracking, and supports in-app messaging and notifications.
* **Backend (Server-Side):**
  + **Purpose:** The central hub for business logic, data validation, and core processing.
  + **Functions:** Manages user accounts, processes complaints (including routing/assignment), handles real-time chat, manages status updates, provides secure APIs, and enforces security policies.
* **Database:**
  + **Purpose:** Ensures persistent storage of all system data.
  + **Functions:** Reliably stores and retrieves user profiles, complaint details, messages, and attachment metadata, while enabling efficient querying.
* **APIs (Application Programming Interfaces):**
  + **Purpose:** Defines the communication contract between the frontend and backend.
  + **Functions:** Standardizes data exchange using JSON format, ensuring secure and efficient communication.
* **External Services/Integrations:**
  + **Purpose:** Provides specialized functionalities outside the core application.
  + **Functions:** Primarily responsible for delivering real-time user notifications via email or SMS.

**3. Technology Stack**

The following technologies have been chosen for their robustness, scalability, and developer-friendliness:

* **Frontend Development:**
  + **UI/Responsiveness:** **Bootstrap** and **Material UI** are used for building a responsive, visually appealing, and consistent user interface across devices.
  + **API Communication:** **Axios** is used to make efficient HTTP requests from the frontend to the backend APIs.
* **Backend Development:**
  + **Framework:** **Express.js** serves as the robust and minimalist web application framework for building the RESTful APIs.
  + **Real-time Communication:** **Socket.io** is specifically implemented for the in-app messaging/chat feature, enabling low-latency, bidirectional communication between users and agents.
* **Database:**
  + **System:** **MongoDB** (NoSQL) is selected for its flexibility in handling various data structures, scalability, and performance, particularly suitable for dynamic complaint and chat data.

**4. Scalability and Performance Considerations**

The chosen architecture and technology stack inherently support scalability and performance:

* **Stateless Backend:** The Express.js backend will be designed to be largely stateless, facilitating easy horizontal scaling by adding more server instances as user load increases.
* **MongoDB Sharding:** MongoDB's native sharding capabilities offer a clear path for horizontal scaling of the database when data volume grows significantly.
* **Optimized APIs:** API endpoints will be optimized, and database queries will be crafted for efficiency to ensure quick response times.
* **Socket.io Efficiency:** Socket.io is a lightweight and efficient protocol for real-time communication, minimizing overhead for chat and live updates.

**5. Deployment Strategy (Render)**

* **Platform:** **Render** is chosen as the unified cloud platform for deployment, streamlining the hosting process.
* **Strategy:**
  + The Frontend (whether a static site or a web service) and the Backend (Web Service) will be deployed as separate, independently scalable services on Render.
  + Render's integrated Git capabilities will enable automatic deployments upon code pushes.
  + Managed features like automatic SSL certificates, custom domains, and environment variable management provided by Render will be leveraged.
  + The MongoDB database will be sourced from a dedicated managed service (e.g., MongoDB Atlas) and securely connected to the Render-deployed backend using connection strings.

This comprehensive technology stack and architecture ensure that the Online Complaint Registration and Management System will be robust, scalable, secure, and deliver a high-quality user experience.