**Project Design Phase**

**Proposed Solution Template**

|  |  |
| --- | --- |
| **DATE:** | **26-06-2025** |
| **Team ID :** | **LTVIP2025TMID55625** |
| **Project Name :** | **ResolveFlow: Online Complaint Registration and Management System** |

**1. Architectural Style**

* **Chosen Style:** Client-Server Architecture
* **Rationale:** Clear separation of concerns, enabling modularity, scalability, and easier maintenance. All communication via RESTful APIs.

**2. Core System Components**

* **Frontend (Client-Side):**
  + **Purpose:** User interaction, UI/UX presentation.
  + **Key Functions:** Registration/Login, Complaint Submission, User Dashboard (tracking), In-app Messaging, Admin/Agent Dashboards, Notifications.
* **Backend (Server-Side):**
  + **Purpose:** Business logic, data validation, database interaction, authentication/authorization, external service communication.
  + **Key Functions:** User account management, complaint processing/routing, real-time chat handling, status updates, API provision, security enforcement.
* **Database:**
  + **Purpose:** Persistent data storage.
  + **Key Responsibilities:** Reliable storage/retrieval, data integrity, efficient querying.
* **APIs (Application Programming Interfaces):**
  + **Purpose:** Define client-backend communication contract.
  + **Key Responsibilities:** Standardized data exchange (JSON), secure communication.
* **External Services/Integrations:**
  + **Purpose:** Specific functionalities outside core backend.
  + **Key Functions:** Real-time user notifications (e.g., email, SMS).

**3. Technology Stack**

* **Frontend:**
  + **UI/Responsiveness:** Bootstrap, Material UI
  + **API Communication:** Axios
* **Backend:**
  + **Framework:** Express.js
  + **Real-time Communication:** Socket.io (for chat)
* **Database:**
  + **System:** MongoDB (NoSQL)

**4. High-Level Data Flow**

1. **User Interaction:** User (any role) uses Frontend.
2. **Request to Backend:** Frontend (via Axios) sends API request to Backend (Express.js).
3. **Backend Processing:** Backend executes business logic, interacts with MongoDB.
4. **Database Interaction:** Data stored/retrieved from MongoDB.
5. **Real-time Updates:** Socket.io handles bidirectional communication for chat/status.
6. **External Services:** Backend triggers notifications (email/SMS) via external services.
7. **Response to Frontend:** Backend sends data back to Frontend.
8. **UI Update:** Frontend updates user interface.

**5. Key Non-Functional Considerations**

* **Security:**
  + User Authentication (hashing, JWTs, potential 2FA).
  + Authorization (RBAC).
  + Data Encryption (in transit/at rest).
  + Input Validation.
  + API Security (rate limiting).
  + Confidentiality measures.
* **Scalability & Performance:**
  + Stateless Backend (horizontal scaling).
  + MongoDB Sharding.
  + Optimized API/Database queries.
  + Socket.io for efficient real-time communication.
* **Deployment Strategy:**
  + **Platform:** Render (as the unified cloud platform).
  + **Strategy:**
    - Frontend (Static Site/Web Service) and Backend (Web Service) deployed as separate services on Render.
    - Leverage Render's automatic deployments via Git integration.
    - Utilize Render's managed features (automatic SSL, custom domains, environment variables).
    - MongoDB database will be a managed service (e.g., MongoDB Atlas) connected securely to the Render-deployed backend.