**Project Design Phase**

**Solution Architecture**

|  |  |
| --- | --- |
| **DATE:** | **26-06-2025** |
| **Team ID :** | **LTVIP2025TMID52693** |
| **Project Name :** | **ResolveFlow: Online Complaint Registration and Management System** |

**1. Introduction to the Project Design Phase**

The Project Design Phase translates the conceptual ideas and user requirements into a concrete technical plan. Solution Architecture is a critical part of this phase, providing a high-level design that defines the system's structure, behavior, and various views of the system to its stakeholders. It ensures that the system is scalable, secure, maintainable, and aligned with business objectives.

**2. Solution Architecture Overview**

The Online Complaint Registration and Management System will employ a **Client-Server Architecture**. This model clearly separates the user interface (client) from the data storage and processing logic (server), enabling modularity, scalability, and easier maintenance. All communication between the client and server will be handled via **RESTful APIs**, ensuring a standardized and efficient data exchange.

**3. Key Architectural Components**

The system is composed of the following primary components:

**3.1. Frontend (Client-Side)**

* **Purpose:** Provides the user interface (UI) and user experience (UX) for all user roles (Customer, Agent, Administrator). It is responsible for presenting data, handling user input, and making requests to the backend.
* **Key Responsibilities:**
  + User Registration and Login Interfaces.
  + Complaint Submission Forms (with attachments).
  + User Dashboard for tracking complaint status.
  + In-app Messaging/Chat interface.
  + Admin and Agent dashboards for complaint management.
  + Displaying notifications.

**3.2. Backend (Server-Side)**

* **Purpose:** Acts as the central processing unit of the system. It handles business logic, data validation, database interactions, user authentication/authorization, and communication with external services.
* **Key Responsibilities:**
  + Managing user accounts (registration, login, profiles).
  + Processing complaint submissions and attachments.
  + Implementing complaint routing and assignment logic.
  + Handling real-time chat messages between users and agents.
  + Managing status updates and sending notifications.
  + Providing secure APIs for frontend interaction.
  + Enforcing security and access control policies.

**3.3. Database**

* **Purpose:** Stores all persistent data required by the system, including user profiles, complaint details, messages, attachments metadata, and system configuration.
* **Key Responsibilities:**
  + Reliable storage and retrieval of structured and unstructured data.
  + Ensuring data integrity and consistency.
  + Providing efficient querying capabilities.

**3.4. APIs (Application Programming Interfaces)**

* **Purpose:** Define the contract for how the frontend and backend communicate. All interactions between components will be exposed as RESTful endpoints.
* **Key Responsibilities:**
  + Standardizing data exchange formats (JSON).
  + Enabling secure and efficient communication.

**3.5. External Services / Integrations**

* **Purpose:** Components responsible for specific functionalities not handled by the core backend, such as sending emails or SMS messages.
* **Key Responsibilities:**
  + Delivering real-time user notifications.

**4. Technology Stack**

* **Frontend Development:**
  + **Libraries/Frameworks:** Bootstrap and Material UI (for UI components and responsiveness).
  + **API Communication:** Axios (for making HTTP requests to the backend).
* **Backend Development:**
  + **Framework:** Express.js (for building robust RESTful APIs).
  + **Real-time Communication:** Socket.io (for the in-app messaging/chat feature, enabling real-time updates).
* **Database:**
  + **Database System:** MongoDB (NoSQL database, for flexible and scalable data storage).

**5. Data Flow (High-Level)**

1. **User Interaction:** A user (Customer, Agent, Admin) interacts with the **Frontend Application**.
2. **Request to Backend:** The Frontend uses **Axios** to send API requests (e.g., submit complaint, fetch status, send message) to the **Backend Server** (Express.js).
3. **Backend Processing:** The Backend processes the request, performs necessary business logic, and interacts with the **MongoDB Database** for data storage or retrieval.
4. **Database Interaction:** Data is stored in or retrieved from MongoDB.
5. **Real-time Updates:** For chat messages or real-time status updates, **Socket.io** facilitates bidirectional communication between the Backend and connected Frontends.
6. **External Services:** The Backend may trigger **External Services** (e.g., email/SMS notification service) based on certain events (e.g., complaint status change).
7. **Response to Frontend:** The Backend sends a response back to the Frontend.
8. **UI Update:** The Frontend updates the UI based on the response received.

**6. Security Considerations**

Security is paramount and will be integrated at every layer:

* **User Authentication:** Secure user registration, password hashing, and authentication mechanisms (e.g., JWTs for session management). Consideration for 2FA as a Major Project.
* **Authorization & Access Control:** Role-based access control (RBAC) to ensure users can only access authorized features and data.
* **Data Encryption:** Data encrypted in transit (HTTPS/SSL/TLS) and at rest (database encryption where applicable).
* **Input Validation:** Strict input validation on the backend to prevent injection attacks (SQL Injection, XSS, etc.).
* **API Security:** Rate limiting, API keys (if applicable for external services), and proper error handling to prevent information disclosure.
* **Confidentiality:** Measures to ensure user and complaint data confidentiality.

**7. Scalability & Performance**

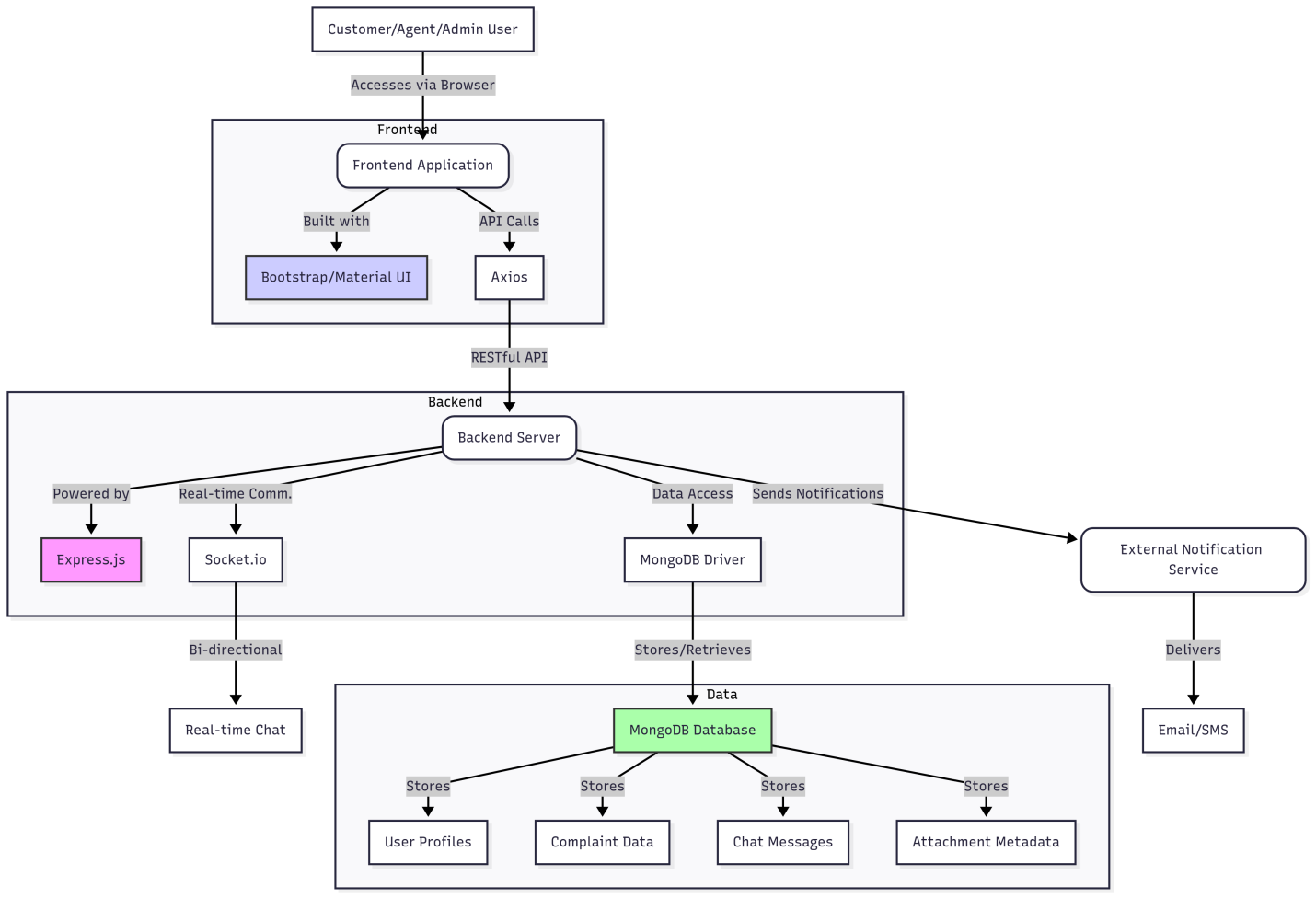
* **Stateless Backend:** Express.js backend will be designed to be largely stateless for easier scaling horizontally (adding more server instances).
* **MongoDB Scalability:** MongoDB's sharding capabilities provide a path for horizontal scaling of the database.
* **API Efficiency:** Optimized API endpoints and efficient database queries will be prioritized.
* **Real-time Efficiency:** Socket.io is designed for efficient, low-latency real-time communication.

**8. Deployment Strategy (High-Level)**

The system will likely be deployed in a cloud environment (e.g., Google Cloud Platform, AWS, Azure) to leverage managed services for database, hosting, and scalability. Frontend and Backend components will be deployed separately.

**9. Architectural Diagram**

This diagram provides a high-level visual representation of the proposed Solution Architecture:



**Explanation of Diagram Elements:**

* **Users:** Represented by the initial A node accessing the Frontend.
* **Frontend Application:** The client-side part, built with UI libraries and using Axios for API calls.
* **Backend Server:** The core logic powered by Express.js, handling database interactions and real-time communication via Socket.io.
* **MongoDB Database:** The central data store.
* **External Notification Service:** For sending email/SMS updates.
* **Real-time Chat:** The communication channel enabled by Socket.io.
* **Data Sub-elements:** Details what the MongoDB database stores.

This Solution Architecture document provides a clear roadmap for the development team, outlining the technical framework for the Online Complaint Registration and Management System.