Government Grievance Portal: Project Documentation

Module 4: Database & Security

Mankaran Singh Tandon (UID: 23BCS10204)

Section: KRG2B
Course: Full Stack Development (PBLJ / Web Technologies)

7. Database Schema

The application utilizes a MongoDB NoSQL database, which is composed of collections of JSON-like documents. The schema is flexible but is primarily organized around two main collections: `Users` and `Complaints`.

Collection: `Users`

Stores information for all registered users (Citizens, Admins, and Super Admins).

Field	Data Type	Description
`_id`	`ObjectId`	Unique identifier for the user (auto-generated by MongoDB).
`name`	`String`	The full name of the user.
`email`	`String`	The user's unique email address (used for login).
`password`	`String`	The user's hashed password (using bcrypt).
`role`	`String`	Defines user permissions (e.g., "citizen", "admin", "superadmin").

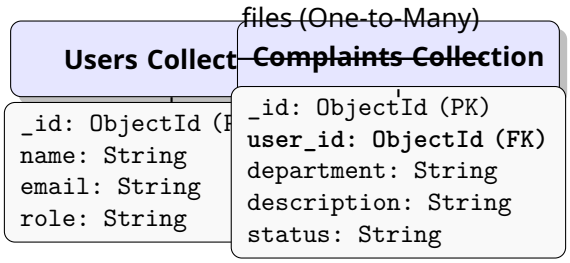
Collection: `Complaints`

Stores all details related to the grievances filed by citizens.

Field	Data Type	Description
`_id`	`ObjectId`	Unique identifier for the complaint.
`user_id`	`ObjectId`	A reference to the `_id` of the user who filed the complaint.
`department`	`String`	The government department the complaint is assigned to.
`description`	`String`	The full text of the citizen's grievance.
`status`	`String`	The current status (e.g., "Pending", "In Progress", "Resolved").
`created_at`	`Timestamp`	The date and time the complaint was filed.
`resolved_at`	`Timestamp`	(Optional) The date and time the complaint was resolved.
`feedback`	`String`	(Optional) Feedback provided by the citizen after resolution.
`attachments`	`Array`	(Optional) An array of URLs or file paths for attached documents.

Schema Relationship Diagram

This diagram illustrates the one-to-many relationship between the Users and Complaints collections.



9. Security Measures

A multi-layered security approach is implemented to protect user data and ensure system integrity.

- **Password Hashing:** The **bcrypt** library is used to hash all user passwords before they are stored in the database. This one-way encryption ensures that even if the database is compromised, plain-text passwords are not exposed.
- **JWT Authentication:** JSON Web Tokens (JWT) are used to manage user sessions. After a successful login, the server issues a signed JWT to the client. This token must be included in the header of all subsequent API requests to access protected routes, ensuring the user is authenticated.
- **Role-Based Access Control (RBAC):** The system enforces strict access control based on the `role` field in the User model ("citizen", "admin", "superadmin"). Backend API routes are protected by middleware that checks the user's role, preventing unauthorized actions (e.g., a "citizen" cannot access the admin dashboard).
- **Input Validation:** All data received from the client (e.g., in forms or API requests) is validated on the server side to prevent common attacks such as Cross-Site Scripting (XSS) and NoSQL Injection.

- **Secure Communication:** The application is intended to be deployed using **HTTPS**, which encrypts all data in transit between the client's browser and the server, protecting against man-in-the-middle attacks.