I had pushed the Code to git below is the URL link

https://github.com/R-NandaKumar//

Algorithm:

1. **User Registration:**
    - Prompt the user to provide necessary details for registration (name, address, contact information, etc.).
    - Validate the entered information and assign a unique user ID.
2. **User Login:**
    - Ask the user to log in using their credentials (username and password).
    - Validate the entered credentials against the stored user data.
3. **Apply for a new Aadhar Card:**
    - Collect personal information such as name, date of birth, address, etc.
    - Verify the details and generate a unique Aadhar number.
    - Store the Aadhar details in the database for future reference.
4. **Request Aadhar Details Update:**
    - Allow users to request updates to their Aadhar details (address change, name change, etc.).
    - Collect the updated information and validate it.
    - Update the user's Aadhar details in the database.
5. **Apply for a Duplicate Aadhar Card:**
    - Provide an option for users to apply for a duplicate Aadhar card.
    - Verify the user's identity and generate a new Aadhar card with a unique number.
    - Store the new Aadhar card details in the database.
6. **Admin Approval for Aadhar Application:**
    - Design an admin interface to view pending Aadhar applications.
    - Allow the admin to review and approve/reject applications.
    - If approved, issue a new Aadhar number and update the database.

7. **Apply to Close Aadhaar Card (Due to Death):**
    - Implement a feature allowing users to apply for the closure of an Aadhar card due to the account holder's death.
    - Verify the request and update the status of the Aadhar card in the database.

Frontend (Angular):

1. **Create Angular Service:**
    - Develop an Angular service using the **HttpClient** module.
    - Define methods in the service for each API endpoint, encapsulating HTTP requests.
2. **Inject Service into Components:**
    - Inject the created service into Angular components that need to interact with the backend.
    - Utilize the service methods to make HTTP requests to the backend.
3. **Handle Responses and Errors:**
    - Implement logic in Angular components to handle responses from the backend.
    - Handle errors gracefully, providing appropriate feedback to users.

Backend (Spring Boot):

1. **Create Spring Boot Controller:**
    - Develop a Spring Boot controller class to handle incoming requests.
    - Define RESTful endpoints in the controller for different functionalities.
2. **Enable Cross-Origin Resource Sharing (CORS):**
    - Configure CORS settings in the Spring Boot application to allow requests from the Angular frontend.

3. **Implement Business Logic:**
    - Implement the necessary business logic within the Spring Boot controller methods.
    - Process incoming requests, perform operations, and return appropriate responses.
4. **Run the Spring Boot Application:**
    - Start the Spring Boot application, making the defined endpoints accessible to the Angular frontend.

Notes:

- **Communication Protocol:**
    - Ensure that the communication between the Angular frontend and Spring Boot backend follows a consistent protocol (e.g., RESTful API using JSON).
- **Error Handling:**
    - Implement error handling mechanisms on both the frontend and backend to gracefully manage unexpected scenarios.
- **Security Considerations:**
    - Consider securing communication channels using HTTPS.
    - Implement authentication and authorization mechanisms based on your application requirements.
- **Testing:**
    - Test the integration between frontend and backend thoroughly to identify and resolve any issues.