COLLEGE CODE:9628

COLLEGE NAME:UNIVERSITY COLLEGE OF ENGINEERING, NAGERCOIL.

DEPARTMENT:COMPUTER SCIENCE AND ENGINEERING.

STUDENT NM-ID:

443E0C77651A61FB2E34B443B6BDB61B

ROLLNO:962823104103

DATE:22/9/2025

Completed the project named as

Phase: 2 – FE

NAME:LOGIN AUTHENTICATION SYSTEM

SUBMITTED BY,

NAME:ABINOV I

MOBILE NO:6383764595

Login Authentication System

Phase 2 – Solution Design & Architecture

1. Tech Stack Selection

The selection of a tech stack is crucial in defining how the authentication system will be implemented, maintained, and scaled in the future. For Phase 1, we are using a mock backend approach, so the focus is on frontend implementation, session simulation, and routing.

1.1 Frontend

- **AngularJS (v1.x)**: Chosen for its simplicity, two-way data binding, and built-in form validation. AngularJS allows rapid development of a single-page application (SPA) with modular architecture.
- **Reasoning**: AngularJS controllers and services simplify data handling, validation, and mock API interactions. Its routing system (ngRoute or ui-router) enables smooth page transitions without page reloads.

1.2 Backend

- **Mock Backend**: No real server exists at this stage. Instead, all user credentials are hardcoded within an AngularJS service (AuthService).
- **Reasoning**: Using a mock backend allows developers to test the authentication flow and session management without needing database setup or server infrastructure. This approach speeds up development and reduces complexity for Phase 1.

1.3 Storage

- **$rootScope**: Provides a global variable accessible throughout the AngularJS application. Can temporarily store session data for SPA runtime.
- **localStorage**: Enables persistent session storage across browser reloads.
- **Reasoning**: Combining $rootScope for runtime session and localStorage for persistence simulates a real-world session handling mechanism.

1.4 Navigation

- **AngularJS Routing (ngRoute or ui-router)**: Used to manage page transitions and

enforce access control.
- **Reasoning**: Proper routing ensures that unauthorized users cannot access the dashboard without logging in. It also simplifies navigation between login, registration, and dashboard pages.

1.5 Styling (Optional)

- **Bootstrap / TailwindCSS**: Provides prebuilt UI components and responsive design support.
- **Reasoning**: Using a CSS framework improves usability and ensures a consistent layout across devices.

2. UI Structure

The User Interface (UI) is designed to be simple, intuitive, and functional. Each page has a distinct purpose, guiding users through the authentication process.

2.1 Login Page

- **Email Input**: Two-way data binding with ng-model for validation.
- **Password Input**: Input masked to protect user privacy.
- **Submit Button**: Calls login function on click.
- **Error Messages**: Displayed below fields using AngularJS form validation (ng-required, ng-pattern).
- **Flow**: User enters credentials → form validates → login service called → user redirected on success.

2.2 Dashboard Page

- **Welcome Message**: Displays logged-in user's email.
- **Logout Button**: Calls AuthService.logout() to clear session and redirect to login.
- **Flow**: If user navigates without an active session → redirect to login page.

2.3 Error Page (Optional)

- **Unauthorized Access**: Displays a message when users attempt to access dashboard without authentication.
- **Navigation**: Redirects user to login page after a timeout or with a button click.

3. API Schema Design (Mock Service)

Since the system currently does not connect to a real backend, all API interactions are simulated via an AngularJS service named **AuthService**.

3.1 AuthService Methods

```
AuthService = {
  login(email, password) {
    // Hardcoded check
    if(email === "admin@example.com" && password === "admin123") {
      return { success: true, user: { email: email } };
    }
    return { success: false, message: "Invalid credentials" };
  },
  logout() {
    // Remove session from $rootScope/localStorage
  },
  isAuthenticated() {
    // Check if session exists
  }
}
```

3.2 API Behavior

- **login(email, password)**: Validates input against hardcoded credentials. Returns success or error message.
- **logout()**: Clears session data from $rootScope and localStorage.
- **isAuthenticated()**: Returns a Boolean indicating if a user session exists.

This design abstracts authentication logic into a reusable service, making future integration with a real backend straightforward.

4. Data Handling Approach

Data handling ensures secure and consistent management of user input, session information, and routing control.

4.1 Form Validation

- **AngularJS Form Features**:
  - ng-model: Binds input fields to controller variables.
  - ng-required: Ensures required fields are filled.
  - ng-pattern: Validates email format using regex.
- **Behavior**: Prevents invalid input submission and provides immediate feedback to the user.

4.2 Authentication Flow

- User submits login form → AuthService validates credentials →
  - If valid → session saved in $rootScope or localStorage → dashboard accessed.
  - If invalid → error message displayed, user stays on login page.

4.3 Routing Control

- **Access Enforcement**: Dashboard route checks isAuthenticated().
  - Not authenticated → redirect to login page.
  - Authenticated → access dashboard.

4.4 Logout Handling

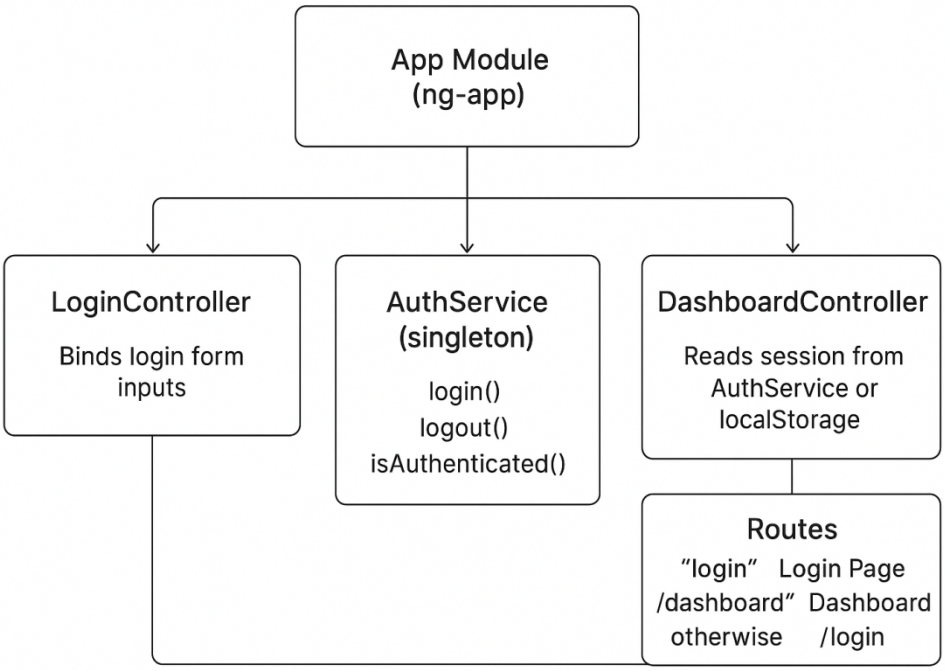
- User clicks "Logout" → AuthService.logout() called → session cleared → redirect to login page.

5. Component / Module Diagram

The system is modular and follows AngularJS best practices.

Explanation

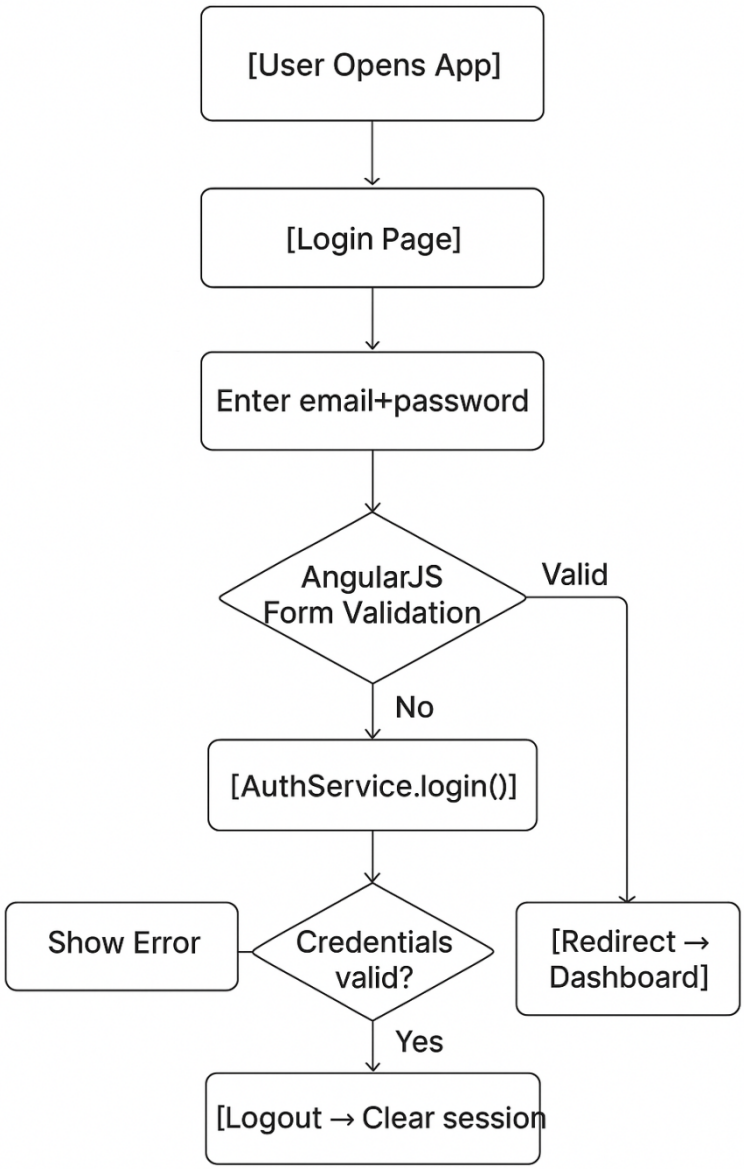
- **Controllers**: Handle UI interactions and bind form data.



- **Service**: Centralizes authentication logic and session management.
- **Routes**: Ensure smooth navigation and access control.

6. Basic Flow Diagram

The flow diagram illustrates user interaction with the application from login to dashboard and logout.



Description

- **Validation Step**: Ensures form inputs are correct before authentication.
- **Authentication Step**: Checks credentials against hardcoded values.
- **Session Management**: Simulates a persistent login using `$rootScope` and `localStorage`.
- **Routing Enforcement**: Ensures users cannot bypass login to access the dashboard.

7. Conclusion

The **Solution Design & Architecture** for the Login Authentication System provides a structured approach to implementing secure user authentication with AngularJS. Key takeaways:

- **Tech Stack**: AngularJS for frontend, `$rootScope`/`localStorage` for session, mock AuthService simulating backend.
- **UI Structure**: Simple, clear, and functional with login, dashboard, and optional error pages.
- **Data Handling**: Ensures secure form validation, session simulation, and routing control.
- **Modular Architecture**: Controllers, services, and routes provide maintainability and scalability.
- **Flow Diagram**: Visualizes the complete login-to-logout process.

This Phase 2 document forms the blueprint for implementing the MVP in Phase 3 and ensures that the system design aligns with functional and usability requirements.