



**COLLEGE CODE: 9123**

**COLLEGE NAME: SACS MAVMM Engineering College**

**STUDENT NM-ID: 4822D33F698C3B88D634C38C8237F3F6**

**STUDENT NM-ID: 82DBE839B5BEB7697C643AE558BB639A**

**STUDENT NM-ID: 18C5FEC84FBE1F2ADD54DD0B31E5C2E1**

**STUDENT NM-ID: 26F7A4E3FB2E532BFC44304B2A1EBED9**

**ROLL NO:23CS030**

**ROLL NO:23CS021**

**ROLL NO:23CS052**

**ROLL NO:23CS022**

**DATE: 06.10.2025**

**COMPLETED THE PROJECT NAMED AS PHASE 5 TECHNOLOGY**

**PROJECT NAME: LOGIN AUTHENTIFICATION**

| NAME | REG NO | PHONE NO | EMAIL ID |
|---|---|---|---|
| J. MEENAKSHI | 912323104027 | 8838321932 | meenu.mailforyou@gmail.com |
| A.V. HARINI | 912323104021 | 9787862865 | harinisaravindababu@gmail.com |
| R. THIBITHRA | 912323104049 | 8124810977 | thibithra06@gmail.com |
| T.S. JANANI | 912323104022 | 9043006559 | janasri255@gmail.com |

**SUBMITTED BY,**

**J. Meenakshi**

**A.V. Harini**

**R. Thibithra**

**T.S. Janani**

# ENHANCEMENTS & DEPLOYMENT(Deadline – Week 9)

## 1. Login Authentication – Overview

Login authentication is a process that verifies a user's identity before granting access to a system, application, or website. It ensures that only authorized users can access protected resources. In your project, this typically involves:

- **Frontend:** Login form (username/email + password)
- **Backend:** API endpoint to validate credentials
- **Database:** Stores encrypted passwords & user details
- **Security:** Hashing (e.g., bcrypt) & session/token management (e.g., JWT)

## 2. Final Demo Walkthrough

When presenting during your final demo:

1. **Show Login Page** – Clean UI with fields for username/email & password.
2. **User Flow:**
    - New user → Registration → Stored in DB.
    - Existing user → Login → Verification → Redirect to dashboard.
3. **Error Handling:** Show messages for invalid login, wrong password, or unregistered user.
4. **Security Features:** Mention password hashing, session expiry, CAPTCHA, or multi-factor authentication (if implemented).
5. **Demo Flow:**
    - Step 1: Register a new account
    - Step 2: Logout & re-login with correct credential
    - Step 3: Attempt login with wrong password → show error
    - Step 4: Access dashboard only after authentication

## 3. Project Report Section

In your report, explain:

- **Objective:** Why authentication is necessary.
- **Architecture:**
    - Frontend → sends credentials to backend API
    - Backend → validates credentials with DB
    - JWT/Session → returned to client
    - Middleware → checks authentication for protected routes
- **Technologies Used:** (HTML/CSS/JS, React/Angular/Vue, Node.js/Flask/Django, MongoDB/MySQL, JWT, bcrypt)
- **Implementation Details:** How you encrypted passwords, handled sessions, and secured APIs.
- **Testing & Validation:** How you tested valid/invalid login cases.

## 4. Screenshots / API Documentation

- **Screenshots to include:**
    - Login Page (empty form)
    - Login Page (with error)
    - Successful login (redirect to dashboard)
    - Registration form (if included)
    - API testing in Postman (showing /login and /register endpoints)
- **API Documentation Example:**
    - **POST /register**
        - Request: { "username": "test", "email": "test@mail.com", "password": "1234" }
        - Response: {"message": "User registered successfully"}

- **POST /login**
    - Request: { "email": "test@mail.com", "password": "1234" }
    - Response: { "token": "JWT_TOKEN", "message": "Login successful" }
- **GET /profile** (protected)
    - Request Header: Authorization: Bearer <JWT_TOKEN>
    - Response: { "username": "test", "email": "test@mail.com" }

## 5. Challenges & Solutions

- **Challenge 1:** Storing passwords securely
    - *Solution:* Used hashing with bcrypt/argon2 instead of plain text.
- **Challenge 2:** Protecting routes
    - *Solution:* Added middleware to validate JWT before accessing protected APIs.
- **Challenge 3:** Session expiration / token invalidation
    - *Solution:* Set JWT expiry (e.g., 1 hour) and implemented refresh tokens.
- **Challenge 4:** Deployment issues (CORS, environment variables)
    - *Solution:* Configured CORS headers properly & used .env for secrets.

## 6. GitHub README & Setup Guide

Your README should include:

- **Project Title & Description**
- **Tech Stack**
- **Features (Login, Register, Authentication, Protected Routes)**
- **Setup Guide:**
- # Clone repo
- git clone <repo-link>
- cd project-folder

- # Install dependencies
- npm install  # or pip install -r requirements.txt
- # Setup environment variables
- # Example: create .env file with DB_URL and JWT_SECRET
- # Run project
- npm start  # or python app.py
- **API Endpoints Documentation** (login/register)
- **Screenshots/GIFs** showing working demo
- **Deployed Link** (Netlify/Vercel + Render/Heroku/Railway)

## 7. Final Submission (Repo + Deployed Link)

- **GitHub Repo:** Should contain
    - Source code (frontend + backend)
    - README with setup guide
    - Screenshots folder
    - API docs (markdown or Postman collection)

- **Deployed Link:**
    - Frontend (Netlify/Vercel)
    - Backend (Render/Railway/Heroku)
    - Provide demo credentials (test user) for reviewers