COLLEGE CODE : 9604

COLLEGE NAME : CSI INSTITUTE OF TECHNOLOGY

DEPARTMENT : COMPUTER SCIENCE AND ENGINEERING

STUDENT NM- ID : 74381B7E0686DCFB608474BC09F05350

ROLL NO : 960423104053

DATE : 30/09/2025

SUBMITTED BY,

NAME : MOHANPRASATH.P

MOBILE NO: 9345758070

PHASE4 – ENHANCEMENT DEPLOYMENT:

LOGIN AUTHENTICATION SYSTEM

ADDITIONAL FEATURES:-

- **Multi-Factor Authentication (MFA)**:

  Combines two or more factors—something you know (password), something you have (a phone or token), or something you are (a biometric)—to create a stronger defense against unauthorized access.

- **Passwordless Authentication**:

  Eliminates the need for passwords entirely, often using email magic links, one-time passcodes (OTPs), or Passkeys, which are more secure and user-friendly.

- **Biometric Authentication**:

  Verifies users based on unique biological characteristics like fingerprints, facial recognition, or retina scans.

- **Adaptive and Risk-Based Authentication (RBA)**:

  An intelligent system that assesses the risk level of each login attempt, prompting additional verification steps for unusual activities from unknown locations or devices.

- **Token-Based Authentication:**

  Uses time-sensitive security tokens or codes generated by an app or sent via SMS to verify identity.

- **Certificate-Based Authentication**:
  Involves using digital certificates to authenticate both users and devices, especially in enterprise settings.
  Deployment & Usability Enhancements

- **Single Sign-On (SSO)**:

  Allows users to log in once and access multiple applications without re-authenticating, improving both security and user experience.

- **Passwordless Authentication (continued)**:

  Improves user experience by simplifying the login process and reducing the risk of password theft or forgetting passwords.

- **Session Management**:

  Efficiently handles user sessions to maintain security and prevent session hijacking after authentication.

- **Audit Trails and Logging**:

Essential for deployment, these logs record all authentication events, providing detailed records for security monitoring, auditing, and compliance.

- **User Registration and Password Recovery**:
  Secure and efficient processes for users to register and recover their credentials are fundamental for any authentication system.

PROGRAM:-

```
// C++ program to demonstrate Segmentation Fault

#include <iostream>

using namespace std;


// Driver Code

int main()

{

   // An array of size 100

   int arr[100] = { 0 };


   // When we try to access the array out

   // of bound, it will give Segmentation Fault

   cout << arr[100001];

   return 0;

}
```

Output:

Below is the output of the above program:

Runtime Errors:

```
Segmentation Fault (SIGSEGV)
```

Output: Copy

```
No Output
```

Importance of UI Authentication improvements:-

UI Improvements for Login Authentication

1. Layout & Design

Use a minimal, centered card layout for the login box.

- Provide **ample spacing** between fields for readability.
- Add a **subtle brand/logo** for trust and familiarity.

---

2. Form Fields

- Clearly label inputs: **Email / Username** and **Password**.
- Include **placeholder text** (e.g., "Enter your email").
- Add **"Show/Hide password" toggle** (with an eye icon).
- Support **autofill** & browser password managers.

---

3. Buttons & Actions

- **Primary login button** should be **large, high-contrast, and full width**.
- Provide **secondary actions**:
  - "Forgot password?" (small text link).
  - "Register / Create account" (clear CTA).

---

4. Error & Validation Feedback

- Show **inline validation**:
  - Invalid email format → highlight with red border + tooltip.
  - Password too short → show rules dynamically.
- Error messages should be **clear and friendly**.

---

5. Additional Login Options

- Offer **social login buttons** (Google, Apple, Microsoft, GitHub).
- Use official branding styles for recognition.
- Place them **below the main form**, separated by a divider ("OR").

---

6. Security Indicators

- Show a **lock icon** or **"Secure login"** text.
- If MFA is enabled, display a **step indicator** ("Step 1 of 2").
- Display **caps lock warning** if enabled during password typing.

---

7. Accessibility

- Ensure **keyboard navigability** (Tab/Enter works smoothly).
- Use **ARIA labels** for screen readers.
- Provide high **contrast ratios** for text and buttons.

---

8. Mobile Responsiveness

- Fields should be **touch-friendly** (44px height minimum).
- Use **responsive layout** (stacked on mobile, side-by-side on desktop).
- Autofocus the first field on load.

Example Modern Login UI Layout:-

```
 -------------------------------------
|    Company Logo                     |
|                                     |
|  Welcome back                      |
|  Please sign in to continue         |
|                                     |
|  [ Email address _____________ ]    |
|  [ Password _______________ 👁 ]     |
|                                     |
|  [ Login Button ▶ ]                 |
|                                     |
|  Forgot password?   |   Register    |
| ----------------------------------  |
|   Or continue with:                 |
|   [ Google ] [ Apple ] [ GitHub ]   |
 -------------------------------------
```

Importance of UX Authentication improvements:-

1.Clear & Simple Login Form

- Minimal fields: **Email/Username + Password**.
- Use **"Show/Hide Password" toggle** (helpful on mobile).
- Provide **password requirements upfront** (e.g., "Min 8 chars, 1 special symbol").

---

2. Multiple Login Options

- **Social logins** (Google, Apple, Microsoft, GitHub, etc.).
- **Passwordless login** (magic links, OTP to email/SMS).
- **Biometric login** (Face ID, fingerprint on supported devices).

---

3. Helpful Error Handling

- Use **clear, friendly messages**:
  - Bad: "401 Unauthorized"
  - Good: "Invalid email or password. Please try again."
- Keep errors **generic** to avoid leaking info (don't say "User not found").
- Highlight the **exact field in error** (e.g., invalid email format).

---

4. Frictionless Security

- **Adaptive authentication**:
  - If login is from a trusted device/location → normal login.
  - If login is from a new device/IP → require MFA.
- "Remember me" option with proper token/session management.

---

5. Passwordless & MFA UX

- Allow login without remembering passwords:
  - **One-click login links** via email.
  - **Short OTPs** via SMS/email (but only as fallback, not primary).
- Provide smooth **MFA UX**:
  - Autofill OTP if sent via SMS (on mobile).
  - Allow users to save MFA devices.

---

6. Accessibility & Inclusivity

- Support **keyboard navigation** (Tab to move between fields).
- Add **screen reader labels** (ARIA attributes).
- Use sufficient **color contrast** for error/success states.

---

7. Security Without Annoyance

- Avoid **CAPTCHA on first attempt** (only show after multiple failures).
- Offer **"Forgot Password?"** link near the login field.
- Enable **progressive disclosure** → show MFA prompt only when needed.

---

8. Transparency & Control

- Show **active sessions & devices** → let users log out remotely.
- Provide **login activity history** (last login time, IP, location).
- Send alerts for **new device logins**.

Example UX Flow:-

1. User enters email & password.
2. If correct:
    - If **trusted device** → login success immediately.
    - If **new device** → show MFA prompt.
3. On failure:
    - Show friendly message + "Forgot Password?" link.
    - After 3 failed attempts, show CAPTCHA.

API Enhancements for Login Authentication:-

1. **Input Validation & Rate Limiting**

- Validate email format and password length (e.g., minimum 8 chars).
- Add **rate limiting** (e.g., max 5 failed attempts per 15 mins) to prevent brute-force attacks.
- Return **generic error messages** (avoid "user not found" vs. "wrong password" distinctions).

---

2. **Secure Password Handling**

- Store passwords using **bcrypt, Argon2, or PBKDF2** (never plain text).
- Enforce password rules: min length, special chars, etc.
- Optionally support **passwordless login** (magic links, OTPs).

---

3. **Token-Based Authentication**

- Use **JWT (JSON Web Tokens)** or **opaque tokens** for session handling.
- Include:
  - **Access Token** (short-lived, e.g., 15 min).
  - **Refresh Token** (longer-lived, e.g., 7 days, stored securely).
- Rotate tokens and allow revocation.

---

4. **Multi-Factor Authentication (MFA)**

- Support **TOTP apps** (Google Authenticator, Authy).
- Optionally allow **SMS/email OTP** for fallback.
- Provide “remember this device” option.

---

5. **OAuth2 / Social Login Integration**

- Allow login with **Google, Apple, Microsoft, GitHub, etc.**
- Useful for lowering friction while still secure.

---

6. **Account Security Features**

- Implement **account lockout** after repeated failures.
- Provide **password reset flow** via email link or OTP.
- Enable **session/device management** (users can see & revoke logins).

---

7. **API Best Practices**

- Always use **HTTPS** (TLS).
- Return **HTTP status codes** properly:
  - 200 OK for success
  - 401 Unauthorized for invalid credentials
  - 429 Too Many Requests for rate limits
- Use **CSRF protection** if sessions are cookie-based.

---

8. **Logging & Monitoring**

- Log failed/successful attempts.
- Monitor unusual login patterns (e.g., many failures, new geolocation).
- Enable alerts for suspicious activity.

Performance Checks for Login Authentication:-

1.**Response Time**

- Login API should typically respond within **200–500 ms** under normal load.
- Measure DB lookup + password hash verification times.

2. **Load & Stress Testing**
   - Simulate thousands of concurrent login attempts.
   - Check system behavior under **peak loads** and identify bottlenecks (e.g., DB, cache, network).
3. **Scalability**
   - Use **connection pooling** for DB queries.
   - Enable **caching** (but **never cache passwords**! Only user profile/session info).
   - Implement **horizontal scaling** (multiple auth servers with load balancer).
4. **Rate Limiting**
   - Prevent overload by **throttling excessive requests** (e.g., 5–10 per minute per IP).
5. **Session Management**
   - Tokens (JWT) should be **lightweight** and **quick to validate** (avoid DB lookup if possible).
   - Use Redis/memory cache for **refresh token management** if needed.

Security Checks for Login Authentication:-

1. Password Handling

- Use strong hashing (bcrypt, Argon2, PBKDF2).
- Enforce **min length, complexity rules, and rotation policies**.
- Never log or transmit passwords in plaintext.

2. Brute Force Protection

- Implement **rate limiting, IP blocking, and CAPTCHA** after multiple failures.
- Lock account temporarily after repeated failed logins.

3. Transport Layer Security

- Always enforce **HTTPS (TLS 1.2/1.3)**.
- Prevent **MITM attacks** by rejecting insecure requests.

4. Token Security

- Use **short-lived access tokens** + **long-lived refresh tokens**.

- Store refresh tokens securely (HttpOnly, Secure cookies).
- Implement **token revocation** on logout or suspicious activity.

5. MFA (Multi-Factor Authentication)

- Support **TOTP, SMS, or email OTP**.
- Optionally device fingerprinting.

Testing of Enhanced Login Authentication

1. Functional Testing

- **Valid Login** → user should log in successfully.
- **Invalid Credentials** → show correct error messages.
- **Case sensitivity** → check email/username (if case-insensitive) and password (case-sensitive).
- **Password reset flow** → request, receive, and use reset link/OTP.
- 🏷 **Session handling** → tokens/cookies should be issued, validated, and revoked properly.
- **MFA flow** → test with correct/incorrect OTP, backup codes, remember device option.

---

2. Security Testing

- **Password security**:
  - Minimum strength enforcement.
  - Hashing (bcrypt/Argon2).
- **Brute force prevention**:
  - Rate limiting works after multiple failed attempts.
  - CAPTCHA appears after suspicious behavior.
- **Token/session validation**:
  - Expired tokens rejected.
  - Refresh token reuse blocked.
- **Vulnerability checks (OWASP Top 10)**:
  - SQL injection (use parameterized queries).
  - CSRF (verify tokens).
  - XSS (sanitize error messages).
  - Session fixation/hijacking.

---

3. Performance Testing

- **Load testing**: simulate 1,000+ concurrent logins.
- **Stress testing**: push system to breaking point, observe recovery.

- **Response time**: check login API latency (should be <500 ms).
- **Database performance**: monitor user table queries and index usage.

---

4. UX/UI Testing

- **Clarity of form**: fields labeled correctly, placeholders helpful.
- **Button behavior**: disabled until inputs are valid, shows loading state.
- **Error messages**: friendly, consistent, and non-technical.
- **Responsive design**: works on desktop, tablet, mobile.
- **Accessibility**: keyboard navigation, ARIA labels, contrast checks.

5.Browsers

Devices: Android, iOS (including autofill/FaceID/fingerprint login).

- **OS**: Windows, macOS, Linux.

---

6. Monitoring & Logging Verification

- Ensure **failed login attempts are logged** (with IP & timestamp).
- Test **alerts** for suspicious activity (e.g., multiple attempts from same IP).
- Verify **audit trail** of login sessions in DB/logs.

Deployment of Login Authentication:-

1. Frontend (UI) Deployment on Netlify

Netlify is excellent for **static frontends (React, Vue, Angular, HTML/CSS)** that call an **API backend** for authentication.

Steps:

1. **Push your frontend code** (e.g., React app) to GitHub/GitLab/Bitbucket.
2. Go to **Netlify Dashboard** → *New Site from Git*.
3. Connect repository → choose branch (usually `main`).
4. Configure **build command** (for React: `npm run build`) and **publish directory** (`build/`).
5. Add **environment variables**:
   - `API_BASE_URL` → your backend login API (AWS, Azure, Firebase, etc.).
   - `JWT_SECRET`, `OAUTH_CLIENT_ID` (if needed).

6. Deploy → Netlify gives you a secure HTTPS URL.
7. Configure **redirects** in `netlify.toml` or `_redirects` for API calls if needed.

---

2. Backend Deployment (Authentication API)

Since Netlify doesn't run servers (except Netlify Functions), you need a **backend** for login/auth. Options:

a) Netlify Functions

- Use **serverless functions** (`/netlify/functions/`) to handle login requests.
- Ideal for **JWT-based login** or **OAuth flows**.
- Pros: Scales automatically, simple.
- Cons: Not good for heavy auth workloads.

b) Cloud Platforms

If you need a full backend service:

- **AWS** → API Gateway + Lambda + DynamoDB (serverless auth flow).
- **Firebase Authentication** → pre-built login/auth system (with Google, Apple, Email, etc.).
- **Azure App Service** → deploy Node.js/Java/Spring backend with DB.
- **Heroku** → simple PaaS for small auth APIs.
- **Render / Railway** → lightweight alternatives for backend APIs.