COLLAGE CODE: 9604

COLLAGE NAME: CSI INSTITUTE OF TECHNOLOGY

DEPARTMENT: COMPUTER SCIENCE AND ENGINEERING

STUDENT NM-ID: D3AAE798B05A3FA9FEA49AA542B7E1E0

ROLL NO: 960423104072

DATE: 23/09/2025

SUBMITTED BY,

NAME: SREE ASWIN.S

MOBILE NO: 9342338715

PHASE 4-Ehancements & Deployment:

USER REGISTRATION WITH VALIDATION

1.Additional feauters:

- Validation rules:

Username must be at least 3 characters.

Email must be valid format.

Password must be strong (min 8 chars, at least 1 uppercase, 1 lowercase, 1 number, 1 special symbol).

Confirm password should match.

Mobile number validation (10 digits).

- Extra features:

Show/Hide password toggle.

Password strength indicator.

Remember me checkbox.

Terms & conditions acceptance.

Store user data temporarily (localStorage) or send to server.

CODE:

HTML :

```
<input type="text" placeholder="Name" required />

<input type="email" placeholder="Email address" required />

<input type="text" placeholder="Country" required />

<input type="tel" placeholder="Phone" required />
```

```
    <div class="password-wrapper">
      <input type="password" id="password" placeholder="Password" required />
      <span class="toggle" onclick="togglePassword()">👁️</span>
    </div>
    <div class="terms">
      <input type="checkbox" required />
      <label>I agree to the terms and conditions.</label>
</div>
```

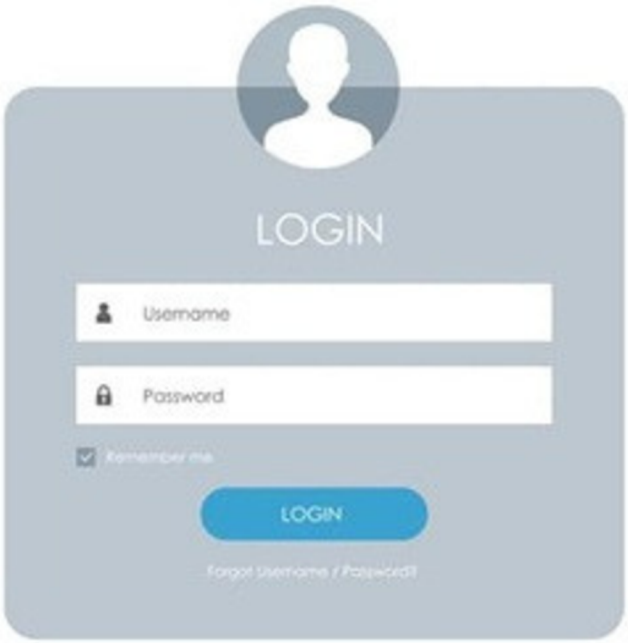
Java script:

```
<script>
function togglePassword() {
    const pwd = document.getElementById("password");
    pwd.type = pwd.type === "password" ? "text" : "password";
  }
</script>
```

Output :

2.UI/UX Improvenments :

- UI Improvements Use a clean single-column form (easy to scan). Add labels
- above inputs (not just placeholders).
- Provide a show/hide password toggle.
- Use consistent button styling (e.g., “Register” button in one color).
- Make the form mobile-friendly (responsive layout).
- UX Improvements
- Real-time validation (show error instantly, not after submission).
- Clear error messages (e.g., “Email is not valid” instead of “Error”).
- Password strength indicator (weak → strong).
- Use green checkmarks (✅) for correct inputs.
- Show a success message after registration (“Account created successfully!”).
-



3.API Enhancements:

1. Input Validation & Sanitization

- Client-side + Server-side validation (never rely only on client-side).
- Validate:
  - Email format (regex, DNS check if needed).
  - Password strength (length, uppercase, numbers, special chars).
  - Username constraints (unique, allowed characters).
- Sanitize inputs to prevent SQL Injection / XSS.

2. Security Enhancements

- Password hashing (e.g., bcrypt, Argon2).
- Rate limiting to prevent brute-force attacks.
- CSRF & CORS protection for API endpoints.
- Captcha or bot detection for high-risk endpoints.
- Use HTTPS (TLS encryption) for all API calls.

3. User Verification

- Email verification API (send OTP / link).
- Phone number verification API (OTP via SMS).
- Support resend verification with rate limits.

4.API Performance Enhancements

- Caching repeated validation checks (e.g., existing username lookup).
- Asynchronous email/SMS sending (via queue).

- Pagination / throttling for logs and requests.

5.Logging & Monitoring

- Store audit logs of registration attempts (success & failure).
- Use API monitoring tools (rate of registrations, unusual patterns).
- Alert on suspicious activities.

4.Performance & Security Checkup:

- Performance Checkup

1. Efficient Validation

- Perform lightweight validation at the API gateway (basic regex, required fields).
- Defer heavy checks (e.g., MX record lookup for email) to async services if possible.
- Use server-side validation libraries instead of custom regex everywhere.

2. Database Optimization

- Index email and username columns for faster uniqueness checks.
- Use connection pooling to avoid slow DB connections.
- Optimize queries (e.g., SELECT 1 FROM users WHERE email=? instead of SELECT *).

3. Scalability

- Support horizontal scaling with stateless APIs.
- Store session/OTP data in distributed cache (Redis/Memcached).
- Security Checkup

1. Password Security

- Enforce strong password policy (length, complexity).
- Hash passwords using bcrypt, scrypt, or Argon2 (never MD5/SHA1).
- Use per-user salt to protect against rainbow table attacks.

2. Input Validation & Sanitization

- Sanitize all user inputs to prevent SQL Injection & XSS.
- Use parameterized queries (e.g., PreparedStatement, ORM safe queries).

3. API Security

- Enable CORS restrictions (only trusted origins).
- Implement CSRF protection.
- Protect endpoints with rate limiting, WAF (Web Application Firewall).

4. Error Handling

- Never expose stack traces or SQL errors in API response.
- Return consistent safe error messages (e.g., “Invalid credentials” instead of “User not found”).

7. Monitoring & Alerts

- Log registration attempts with timestamps, IPs, device info.

- Monitor unusual activities (e.g., same IP registering multiple accounts).
- Set up alerts for suspicious spikes in registration requests.

5.Testing of Enhancements:

1. Functional Testing

- Input Validation Tests
  - ✅ Email: valid vs invalid formats (test@example.com, user@.com).
  - ✅ Password: test strong vs weak (1234, Pass@12345).
  - ✅ Username: allowed/disallowed characters, length boundaries.
  - ✅ Mandatory fields: leave blank → expect proper error messages.
- Duplicate Check
  - Try registering with an existing email/username → expect rejection.
- Verification Workflow
  - Email/SMS OTP sent → correct OTP activates account.
  - Expired/invalid OTP → expect error.
  - Resend OTP → works but respects rate limits.

2. Security Testing

- Password Security
  - Ensure hashed passwords in DB (not plain text).
  - Verify bcrypt/Argon2 hashing is used.
- SQL Injection
  - Input: "' OR '1'='1" in username/email → registration should fail, not bypass.
- XSS Injection
  - Input <script>alert(1)</script> in username → must be sanitized.
- Brute Force Protection
  - Multiple failed OTP/registration attempts → account lockout / captcha trigger.
- Transport Security
  - Ensure all API calls are over HTTPS.

3. Performance Testing

- Load Test
  - Simulate 1000+ concurrent registration requests.
  - Check API response time remains acceptable (< 500ms avg).
- Stress Test
  - Push beyond capacity to ensure system fails gracefully (proper error codes, no crash).
- Rate Limiting
  - From a single IP, send 100+ requests/minute → confirm rate limit is applied.

4. Usability Testing

- Clear error messages (e.g., “Email already registered” instead of vague errors).
- Mobile + Desktop form responsiveness.
- Accessibility: labels, error messages readable by screen readers.

5. Logging & Monitoring Tests

- Logs record both successful & failed attempts.
- Alerts trigger on suspicious patterns (e.g., same IP creating 20 accounts).

6.Deployment (Netlify, Vercel, or Cloud Platform):

- Netlify:
- Good for frontend + light backend (serverless functions)

Steps:

1. Push your project (HTML/JS/React app) to GitHub.
2. Go to Netlify → New Site from Git → connect your repo.
3. Add Netlify Functions for backend APIs (user registration, validation).
    - Example: /netlify/functions/register.js handles signup logic.
4. Connect to a cloud DB (e.g., Firebase, Supabase, MongoDB Atlas).
5. Add environment variables (DB_URL, API_KEY) in Netlify dashboard.
6. Click Deploy → app live with HTTPS.

- Vercel:
- Best for Next.js / full-stack JavaScript apps

Steps:

1. Push your Next.js app to GitHub.
2. Go to Vercel → Import Project → select repo.
3. Deploy → Vercel auto-detects Next.js & builds.
4. Use Vercel Serverless Functions (/api/register.js) for validation + signup API.
5. Connect external DB (PostgreSQL, MongoDB Atlas, or Supabase).
6. Add environment variables (DB_URL, SECRET_KEY) in Vercel dashboard.
7. App is deployed → auto-HTTPS + scaling included.

- Cloud Platform (AWS/GCP/Azure)
- Best for enterprise / scalable apps

Steps (AWS example):

1. Host frontend in S3 + CloudFront.
2. Deploy backend API in AWS Lambda + API Gateway (or EC2 if full server).
3. Use AWS RDS (MySQL/Postgres) or DynamoDB for user data.
4. Store secrets in AWS Secrets Manager.
5. Enable monitoring with CloudWatch.