0

COLLEGE CODE: 9504
COLLEGE NAME: DR.G.U.POPE COLLEGE OF ENGINEERING
DEPARTMENT: CSE
STUDENT NM-ID : 4E2319674F89610866446F7CAE6308F0
ROLL NUMBER: 950423104025
DATE: 06/10/2025

TECHNOLOGY PROJECT NAME
LOGIN AUTHENTICATION SYSTEM

SUMITTED BY:
NAME: S.Sri kamini
MOBILE NO: 6374681440

Problem Statement:

Modern web and mobile applications require secure and reliable user authentication to protect sensitive data, ensure privacy, and manage user access. Without a proper login authentication system, applications are vulnerable to unauthorized access, identity theft, and data breaches. A login authentication system provides a secure gateway for users to access services while maintaining system integrity and compliance with security standards.

Users & Stakeholders

Users

- End-users (individuals registering and logging into the system).
- Admin users (managing accounts, roles, and permissions).

Stakeholders

- Application owners/businesses (concerned with security, compliance, and usability).
- Developers (responsible for building and maintaining the system).
- Security teams (ensuring standards like encryption, password policies, and secure storage).
- Customer support (assisting users with login issues like password resets).

User Stories

- As a user, I want to register with my email and password so that I can create an account.
- As a user, I want to log in securely so that I can access my personal dashboard.
- As a user, I want to reset my password if I forget it so that I can regain access.
- As a user, I want my session to remain active until I log out, so that I don't have to re-enter credentials frequently.
- As an admin, I want to view and manage user accounts so that

I can ensure proper access control.

MVP Features

- User Registration (sign up with email/username & password).
- Secure Login (authentication via email/username & password).
- Password Management (reset and update password functionality).
- Session Handling (token-based authentication e.g., JWT).
- Basic Admin Panel (view users, deactivate accounts).
- Error Handling (clear error messages for invalid login or failed registration).

Wireframes / API Endpoint List

1.Wireframes (basic flow):

- Login Page → Input email & password.
- Registration Page → Input name, email, password, confirm password.
- Dashboard Page → Shows welcome message & logout button.
- Forgot Password Page → Email input to send reset link.

2.API Endpoints (example):

- POST /api/auth/register → Create new user.
- POST /API/auth/login → Authenticate user and return JWT token.
- POST /api/auth/logout → End user session.
- POST /api/auth/forgot-password → Send reset link.
- POST /api/auth/reset-password → Reset password using token.
- GET /API/users → (Admin) Fetch list of users.

Acceptance Criteria

- User should be able to register with a unique email.
- Login should succeed only with valid credentials.
- Passwords must be stored securely (hashed & salted).
- After login, a JWT/session token must be generated and validated for secure access.
- Forgot password feature should send a reset email/link to the registered address.
- System should prevent common vulnerabilities (SQL injection, brute force, session hijacking).

Tech Stack Selection:

- Frontend: React.js (for building dynamic, responsive UI)
- Backend: Node.js with Express.js (lightweight & scalable API development)
- Database: MongoDB (NoSQL, schema-less, flexible for dynamic data)
- Authentication: JWT (JSON Web Token) for secure user authentication
- Hosting / Deployment:
- Frontend → Vercel / Ntlify
- Backend → AWS EC2 / Render / Heroku
- Database → MongoDB Atlas
- Version Control: Git + GitHub
-

UI Structure / API Schema Design

UI Structure

- Login / Signup Page – Authentication screen
- Dashboard Page – Main user panel
- Feature Module Pages (based on your project use case)
- Settings / Profile Page – User management

API Schema Design

- User API
- POST /api/auth/signup → Register user
- POST /api/auth/login → Authenticate user
- GET /api/user/:id → Fetch user details
- Data API (example for tasks / products / posts, based on use case)
- POST /api/data/create
- GET /api/data/:id
- PUT /api/data/:id
- DELETE /api/data/:id

Data Handling Approach

- Frontend State Management: React Context API / Redux
- API Communication: Axios / Fetch (JSON format)

Database Handling:

- MongoDB collections for storing entities
- Use Mongoose schema for structured access

Security:

- Passwords → Hashed using bcrypt
- Tokens → JWT-based secure session
- Input validation → JOI / Validator.js

Component / Module Diagram

Frontend Modules:

- Authentication Module
- Dashboard Module
- Data Management Module
- User Profile Module

Backend Modules:

- Authentication Service
- Data CRUD Service
- Middleware (Auth, Error handling, Validation)
- Database Service

Basic Flow Diagram

User Authentication & Data Flow

User → UI (React) → API Request → Backend (Node/Express) → Auth Check (JWT) → Database (MongoDB) → Response → UI Update

This structure is generic and fits most web applications. You can tailor it to Login system, E-commerce, Task Manager, or any project you're working on manage users.

MVP Implementation :

Login Authentication System (React + Node.js)

1. Project Setup:

Initialize **React app** (create-react-app or Vite).

Set up **backend server** with Node.js + Express.

Install dependencies:

- ✓ Frontend: axios, react-router-dom
- ✓ Backend: bcrypt, jsonwebtoken, mongoose (for MongoDB) or mysql2 (for MySQL)

Create **GitHub repository** and push initial code.

2. Core Features Implementation:

Frontend (React):

- Signup form (username, email, password).
- Login form (email, password).
- Private route component to protect authenticated pages.
- State management for user session (React useState or Context API).

Backend (Node.js/Express):

API endpoints:

- POST /register → store new user with hashed password.
- POST /login → validate user credentials, return JWT.
- GET /profile → return user info if JWT is valid.
  - ✓ Use bcrypt for **password hashing**.
  - ✓ Use JWT for **session management**.

Data Storage (Local State / Database):

Frontend:

- Store JWT token in localStorage or sessionStorage.
- Maintain user state (logged in / logged out).

Backend Database (MongoDB example):

User schema:

- userId
- Username
- email
- hashedPassword
- created At

Testing Core Features:

Unit Testing:

- Check React components render correctly (forms, buttons).
- Validate backend login/register functions.

Integration Testing:

- Ensure user can sign up, log in, and access protected routes.
- Invalid credentials return error messages.

Manual Testing:

Run through user flows (register → login → logout).

Version Control (GitHub):

- Create feature branches (feature/login, feature/signup).
- Commit regularly with clear messages (feat: add JWT auth).
- Use Pull Requests for merging.
- Maintain README with setup instructions.

Enhancements & Deployment :

1. Additional Features

- **Password Reset**: Allow users to reset their password via email/OTP.
- **Remember Me Option**: Keep users logged in using secure cookies/local Storage.
- **Role-based Authentication**: Different access for Admin, User, Guest.
- **Social Logins**: Enable Google / GitHub sign-in using OAuth.
- **Two-Factor Authentication (2FA)**: Add an extra OTP/email verification step.

2. UI/UX Improvements

- **Clean and responsive design (works on desktop and mobile).**
- Show **password strength meter** while typing.
- Add **eye icon** to toggle password visibility.
- Better error messages (e.g., *"Invalid password format"* instead of *"Error 400"*).
- Loading indicators & success/failure toasts.

3. API Enhancements

- **Validation Layer**: Check inputs for SQL injection / XSS attempts.
- **JWT Refresh Tokens**: Extend session without re-login.
- **Rate Limiting**: Prevent brute-force login attempts.
- **Error Handling**: Send clear structured JSON error responses.
- **Logging & Monitoring**: Track failed logins, unusual IPs, and errors.

3. Performance & Security Checks

- Optimize database queries (indexes for faster lookups).
- Use HTTPS in production (SSL certificate).
- Hash passwords using **bcrypt** or **argon2**.
- Test with **OWASP Top 10** security guidelines.
- Reduce bundle size & enable lazy loading for frontend.

Testing of Enhancements:

- **Unit Testing**: Check individual functions (password hashing, token generation).
- **Integration Testing**: Ensure frontend and backend communicate properly.
- **End-to-End Testing (E2E)**: Simulate full login → logout → reset flow (e.g., Cypress).
- **Cross-Browser Testing**: Chrome, Firefox, Edge, Safari.
- **Mobile Responsiveness Testing**.

Deployment:

I use to deploy the project in github pages.

Project Demonstration & Documentation

Final Demo Walkthrough

Show the login page with fields: Username / Email and Password.

Demonstrate:

1. Signup / Registration (if included) → user creates an account.
2. Login → correct credentials allow access.
3. Invalid Login → wrong password shows error message.
4. Session / Token → user stays logged in until logout.
5. Logout → user session ends.

2. Project Report

Objective:

To implement a secure login authentication system where users can register and login using their credentials, with security features like password hashing and session management.

Technologies Used:

We only make frontend page using HTML,CSS, VITE +REACT

The Technologies of HTML,CSS and REACT are used for:

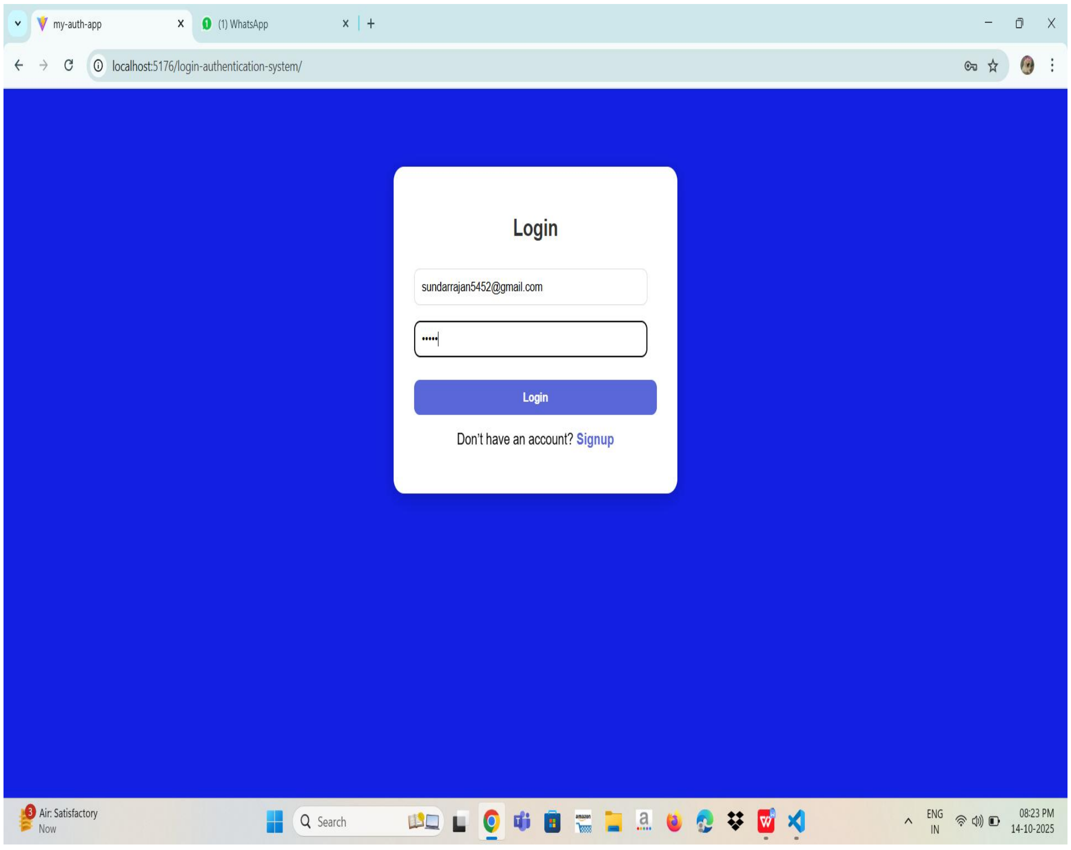
- HTML to **structure and organize content on the web** so that browsers and users can understand it.
- CSS is used on your login/signup page to make it **attractive, usable, and properly arranged**, instead of looking like plain, boring browser default elements.
- VITE+ React for this page because it lets us **dynamically render components**, **manage state easily**, **handle events efficiently**, and **build a scalable, interactive web app** instead of static HTML pages.

Features:

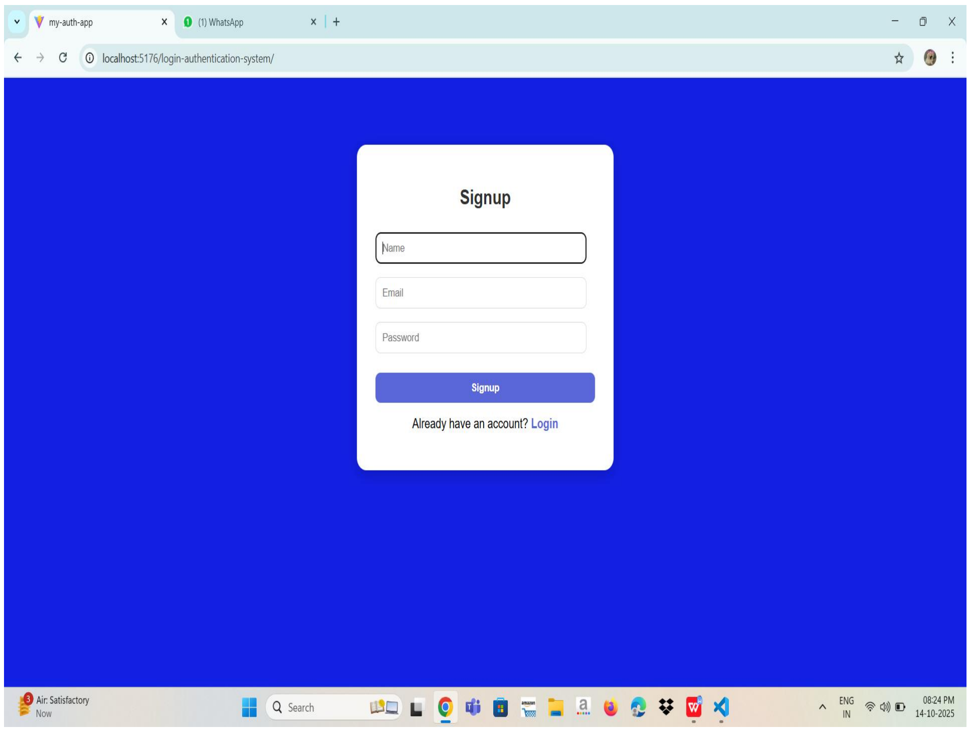
- User Registration & Login.
- Password hashing for security.
- Authentication middleware to protect routes.
- Error handling for invalid login attempts.
- Logout functionality.

3. Screenshots / API Documentation

Screenshot of login page:


my-auth-app
(1) WhatsApp
localhost:5176/login-authentication-system/
Login
sundarrajan5452@gmail.com
•••••
Login
Don't have an account? Signup
Air: Satisfactory
Now
Search
ENG
IN
08:23 PM
14-10-2025

Screenshot of signup page:



Signup

Name

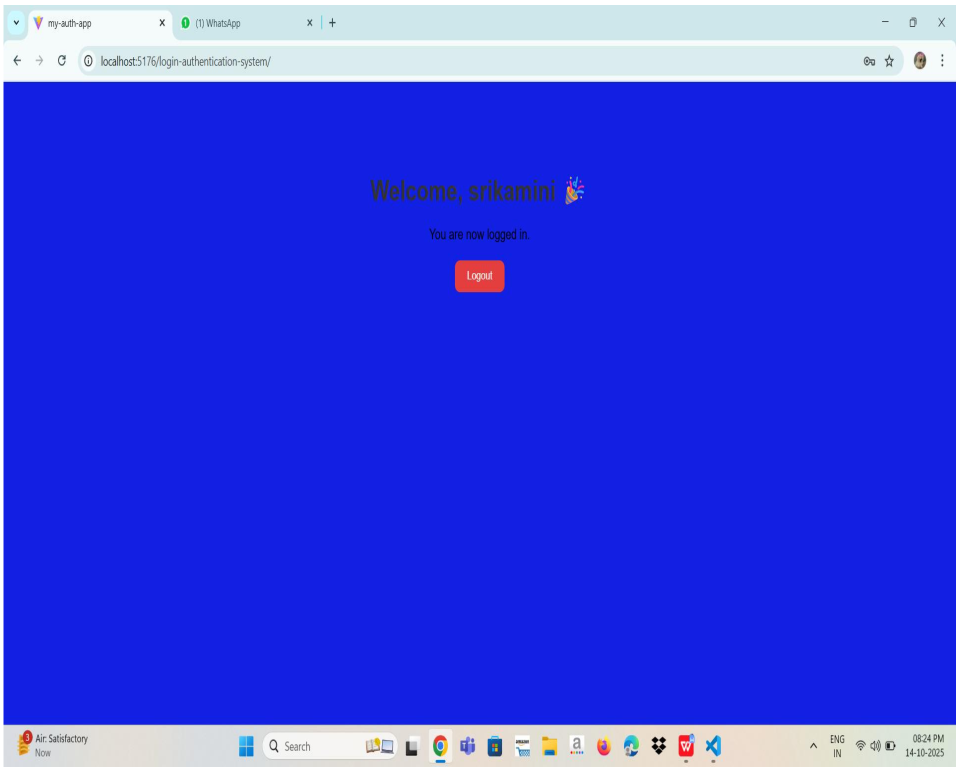
Email

Password

Signup

Already have an account? Login

Screenshot of Dashboard (after login):

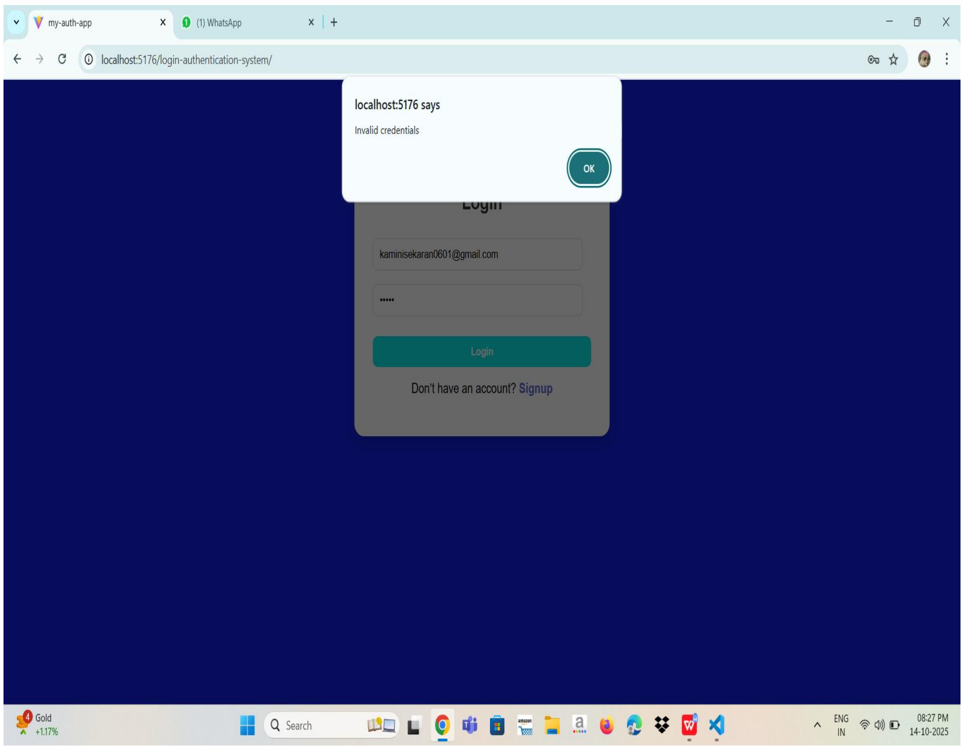


Welcome, srikamini 🎉

You are now logged in.

Logout

Screenshot of Error message for wrong credentials.


localhost:5176 says

Invalid credentials

my-auth-app
(1) WhatsApp
localhost:5176/login-authentication-system/
OK
Login
kaminisekaran0601@gmail.com
Login
Don't have an account? Signup
Gold
+1.17%
Search
ENG
IN
08:27 PM
14-10-2025

API Endpoints Example:

POST /signup → Registers new user.

POST /login → Validates user credentials.

GET /dashboard → Protected route (requires token/session).

POST /logout → Ends session.

4. Challenges & Solutions

Challenge	Solution
Passwords stored in plain text	Used bcrypt to hash passwords
Session hijacking risk	Implemented JWT / secure session cookies
Database connection errors	Added proper error handling & retry logic
Invalid login attempts	Displayed error message with validation
Deployment issues	Used GitHub + Render/Netlify/Vercel/Heroku for hosting

5. GitHub README & Setup Guide

# React + Vite

This template provides a minimal setup to get React working in Vite with HMR and some ESLint rules.

Currently, two official plugins are available:

- [@vitejs/plugin-react](https://github.com/vitejs/vite-plugin-react/blob/main/packages/plugin-react) uses [Babel](https://babeljs.io/) (or [oxc](https://oxc.rs) when used in [rolldown-vite](https://vite.dev/guide/rolldown)) for Fast Refresh

- [@vitejs/plugin-react-swc](https://github.com/vitejs/vite-plugin-react/blob/main/packages/plugin-react-swc) uses [SWC](https://swc.rs/) for Fast Refresh

## React Compiler

The React Compiler is not enabled on this template because of its impact on dev & build performances. To add it, see [this documentation](https://react.dev/learn/react-compiler/installation).

## Expanding the ESLint configuration

If you are developing a production application, we recommend using TypeScript with type-aware lint rules enabled. Check out the [TS template](https://github.com/vitejs/vite/tree/main/packages/create-vite/template-react-ts) for information on how to integrate TypeScript and [`typescript-eslint`](https://typescript-eslint.io) in your project.

Setup Guide

Follow these steps to run the project locally:

1. Clone the repository

git clone https://github.com/srikamini/login-authentication-system

open your web browser and access the application @http://localhost:5176/login-authentication-system/