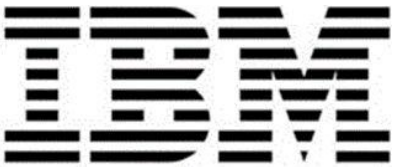


COLLEGE CODE : 9216

COLLEGE NAME : SBM COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT : Computer Science & Engineering

NM-ID : 42F81605FAAF64684E244A6566761A04

DATE : 25/10/2025

ROLL NO : 921623104006

COMPLETED THE PROJECT NAMED AS :
User Authentication System

TECHNOLOGY PROJECT NAME : NODE JS

SUBMITTED BY :

NAME : Balaji S
MOBILE NO : 8754095421

Phase 5 : Project Demonstration & Documentation
Title : User Authentication System

Final Demo Walkthrough :

The User Authentication System project demonstrates a complete and secure login–registration mechanism built using Node.js, Express.js, and MongoDB. It ensures safe user access, password encryption, and token-based authentication, featuring a clean, responsive web interface.

• Home Screen Overview:

When the application is launched, users are greeted with a Registration Page that collects name, email, and password. Once registered, they can proceed to the Login Page, where valid credentials generate a JWT token that allows access to the protected Profile Page.

• Secure Login Flow:

After login, a JWT (JSON Web Token) is generated and stored in the browser (via local storage). The token is automatically sent with every protected API call to validate the user session. Unauthorized users are denied access.

• User Interface (UI):

The frontend is built with HTML, CSS, and JavaScript — designed to be clean, responsive, and user-friendly. It includes color-coded alerts for success or failure messages, password visibility toggles, and responsive layouts for both desktop and mobile devices.

• Protected Profile Page:

Once authenticated, users are redirected to the Profile Dashboard, where personal details (name, email) are displayed using a secure API call (`/api/auth/profile`). The dashboard also includes a Logout button, which clears the JWT token and redirects the user back to login.

• **Data Security:**

All passwords are hashed using bcrypt.js, and sensitive information like database URLs and JWT secrets are stored in a .env file. This prevents any direct exposure of credentials.

• **Final Output:**

The completed demo provides a fully functional secure authentication system with registration, login, and profile access — built using modern web standards and suitable for both local and cloud deployment.

Project Report :

Project Title: Secure User Authentication System
Technologies Used: Node.js, Express.js, MongoDB, bcrypt.js, JWT, HTML, CSS, JavaScript

Objective:

To build a secure user management system allowing registration, login, and profile access with password encryption and token-based authentication.

System Design:

- **Backend:** Node.js + Express handles user requests, authentication, and token validation.
- **Database:** MongoDB stores user credentials (hashed passwords).
- **Frontend:** HTML, CSS, JS interface for registration, login, and profile views.
- **API Structure:** RESTful endpoints (`/register`, `/login`, `/profile`) manage data flow between frontend and backend.

Key Features:

- User Registration & Login using JWT authentication.
- Password encryption using bcrypt.
- Profile access only for verified users.
- Responsive and minimal UI design.
- Environment variables for secure deployment.

Testing & Validation:

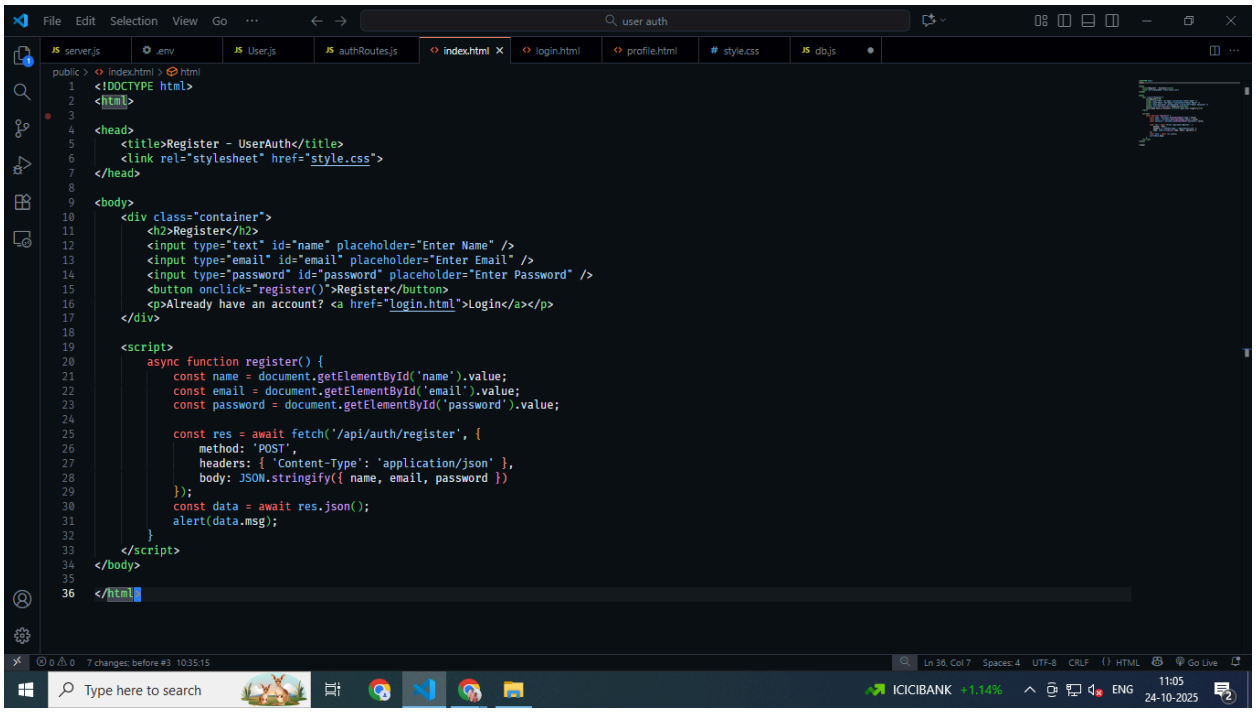
- Verified registration and login flows through Postman.
- Confirmed JWT token generation and validation.
- Tested invalid credentials, duplicate users, and missing fields.
- Checked token-based access for protected routes.

Outcome:

The project successfully demonstrates a secure user authentication mechanism that's scalable, lightweight, and ready for deployment on cloud platforms like Vercel or Render.

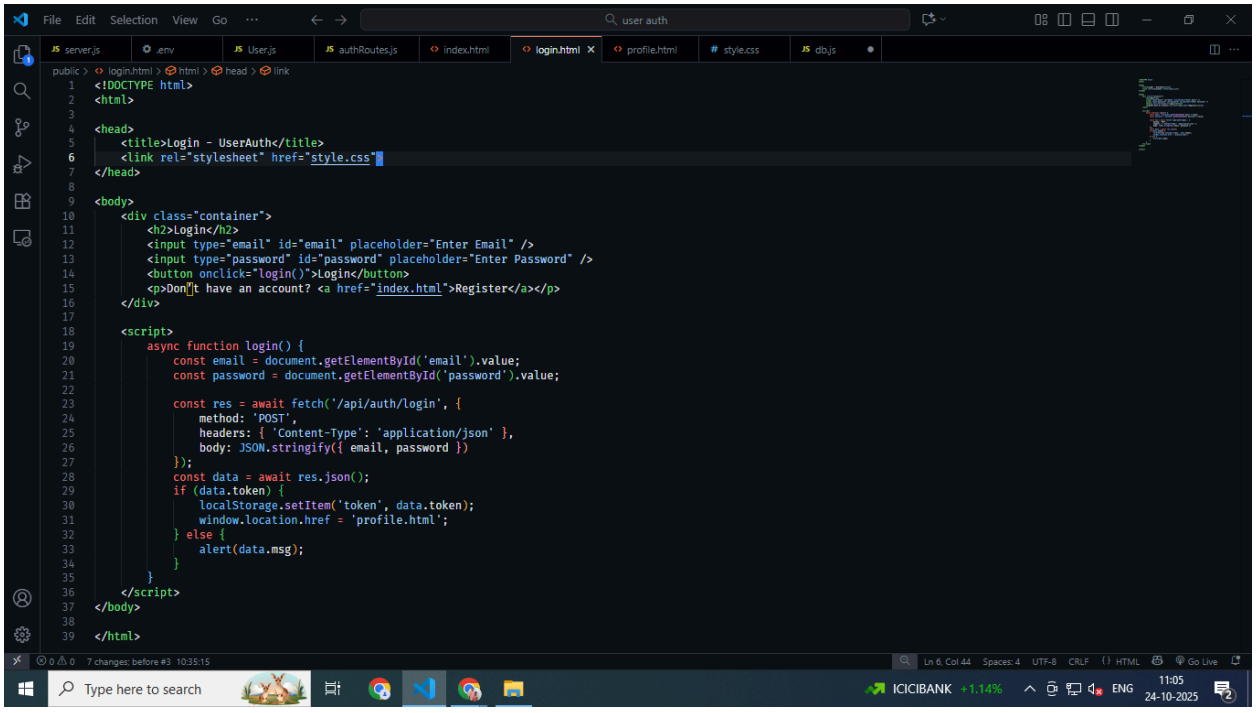
Screenshots : (CODE)

index.html



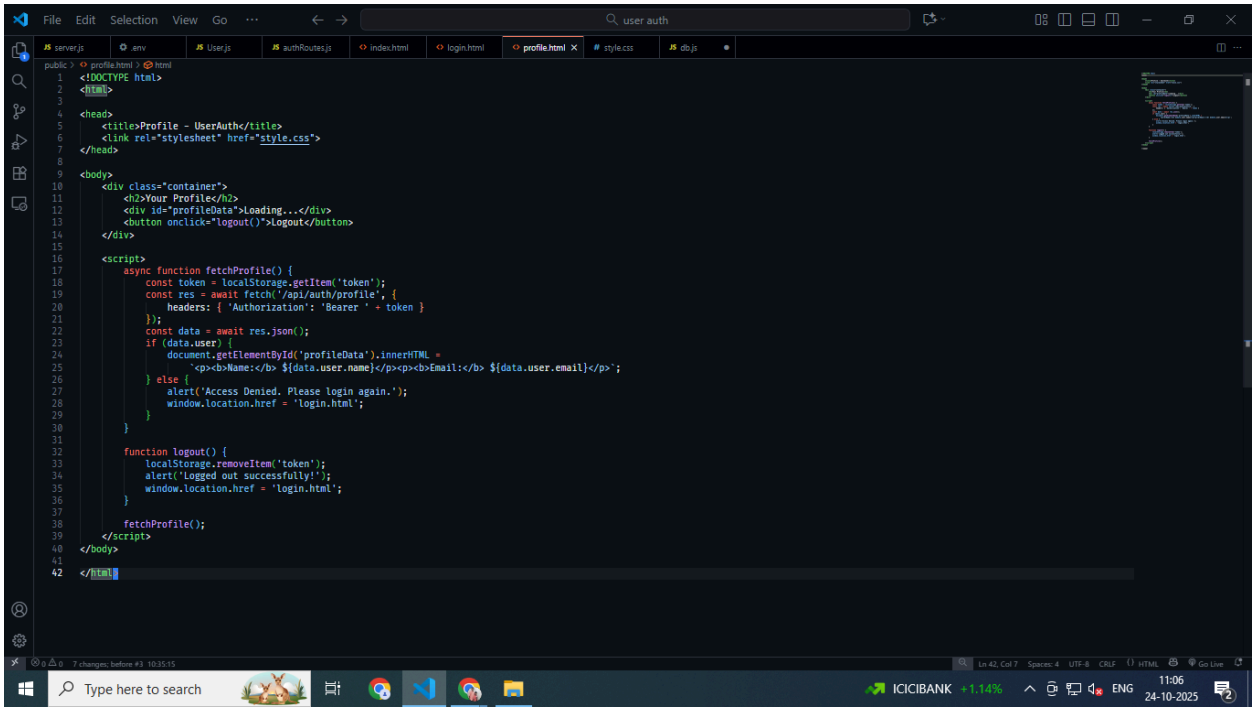
```html
<!DOCTYPE html>
<html>

<head>
    <title>Register - UserAuth</title>
    <link rel="stylesheet" href="style.css">
</head>

<body>
    <div class="container">
        <h2>Register</h2>
        <input type="text" id="name" placeholder="Enter Name" />
        <input type="email" id="email" placeholder="Enter Email" />
        <input type="password" id="password" placeholder="Enter Password" />
        <button onclick="register()">Register</button>
        <p>Already have an account? <a href="login.html">Login</a></p>
    </div>

    <script>
        async function register() {
            const name = document.getElementById('name').value;
            const email = document.getElementById('email').value;
            const password = document.getElementById('password').value;

            const res = await fetch('/api/auth/register', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ name, email, password })
            });
            const data = await res.json();
            alert(data.msg);
        }
    </script>
</body>

</html>
```

login.html



```html
<!DOCTYPE html>
<html>

<head>
    <title>Login - UserAuth</title>
    <link rel="stylesheet" href="style.css">
</head>

<body>
    <div class="container">
        <h2>Login</h2>
        <input type="email" id="email" placeholder="Enter Email" />
        <input type="password" id="password" placeholder="Enter Password" />
        <button onclick="login()">Login</button>
        <p>Don't have an account? <a href="index.html">Register</a></p>
    </div>

    <script>
        async function login() {
            const email = document.getElementById('email').value;
            const password = document.getElementById('password').value;

            const res = await fetch('/api/auth/login', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ email, password })
            });
            const data = await res.json();
            if (data.token) {
                localStorage.setItem('token', data.token);
                window.location.href = 'profile.html';
            } else {
                alert(data.msg);
            }
        }
    </script>
</body>

</html>
```

profile.html



```
<!DOCTYPE html>
<html>

<head>
    <title>Profile - UserAuth</title>
    <link rel="stylesheet" href="style.css">
</head>

<body>
    <div class="container">
        <h2>Your Profile</h2>
        <div id="profileData">Loading...</div>
        <button onclick="logout()">Logout</button>
    </div>

    <script>
        async function fetchProfile() {
            const token = localStorage.getItem('token');
            const res = await fetch('/api/auth/profile', {
                headers: { 'Authorization': 'Bearer ' + token }
            });
            const data = await res.json();
            if (data.user) {
                document.getElementById('profileData').innerHTML =
                    `<p><b>Name:</b> ${data.user.name}</p><p><b>Email:</b> ${data.user.email}</p>`;
            } else {
                alert('Access Denied. Please login again.');
                window.location.href = 'login.html';
            }
        }

        function logout() {
            localStorage.removeItem('token');
            alert('Logged out successfully!');
            window.location.href = 'login.html';
        }

        fetchProfile();
    </script>
</body>

</html>
```

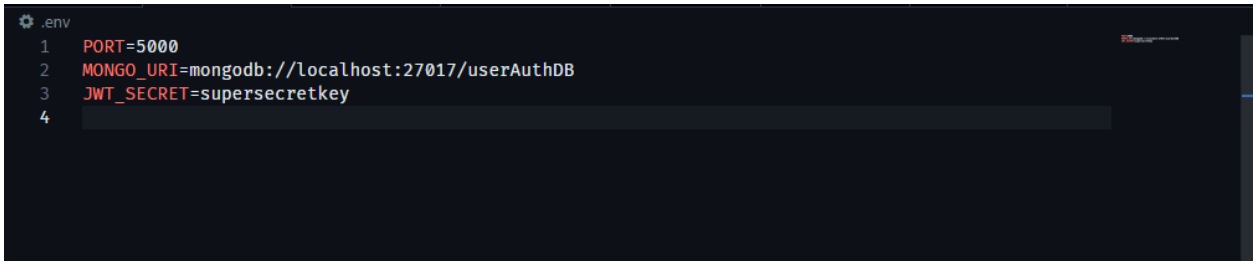
db.js

```
const mongoose = require("mongoose");

const connectDB = async () => {
    try {
        await mongoose.connect(process.env.MONGO_URI);
        console.log("✅ MongoDB Connected...");
    } catch (err) {
        console.error("❌ DB Connection Failed:", err.message);
        process.exit(1);
    }
};

module.exports = connectDB;
```

server.js

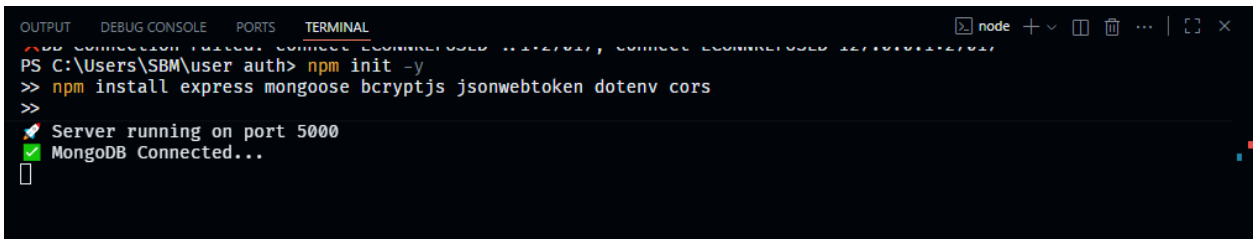
```
JS server.js > ...
1   const express = require("express");
2   const dotenv = require("dotenv");
3   const cors = require("cors");
4   const connectDB = require("./config/db");
5   const authRoutes = require("./routes/authRoutes");
6
7   dotenv.config();
8   const app = express();
9
10  app.use(cors());
11  app.use(express.json());
12  app.use(express.static("public"));
13
14  connectDB();
15
16  app.use("/api/auth", authRoutes);
17
18  const PORT = process.env.PORT || 5000;
19  app.listen(PORT, () => console.log(`🚀 Server running on port ${PORT}`));
20
```

.env



```
.env
1   PORT=5000
2   MONGO_URI=mongodb://localhost:27017/userAuthDB
3   JWT_SECRET=supersecretkey
4
```

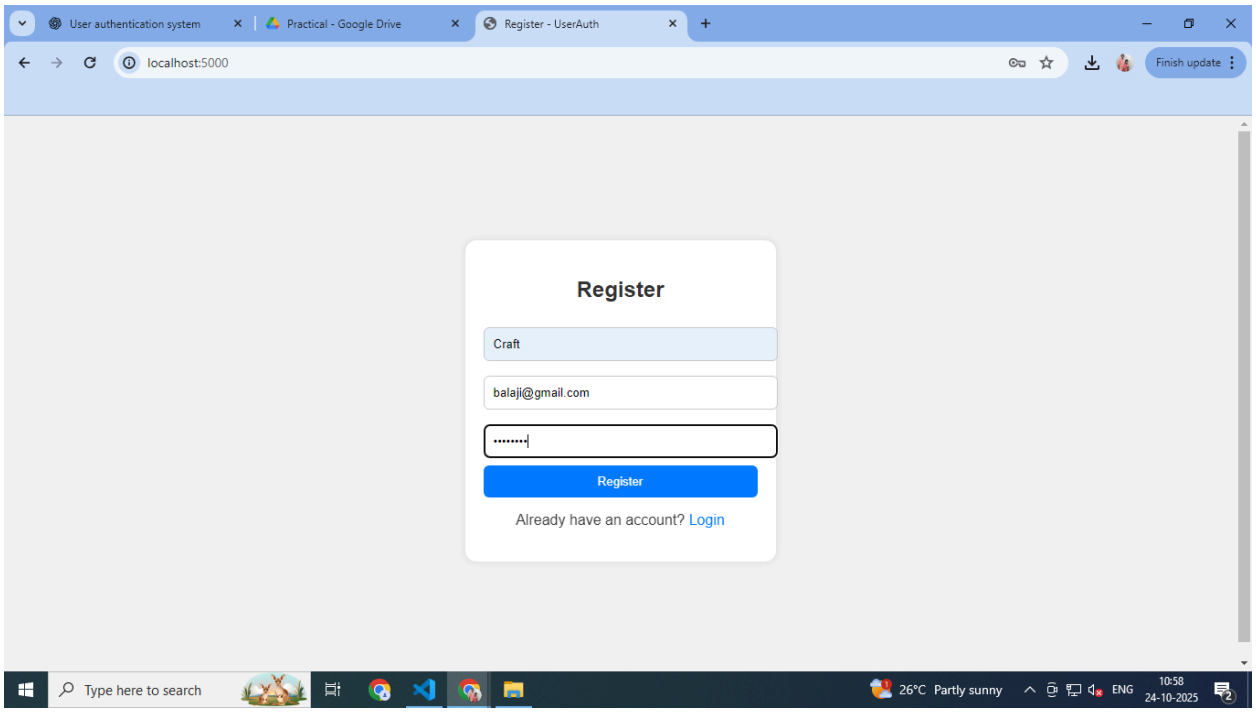
Terminal



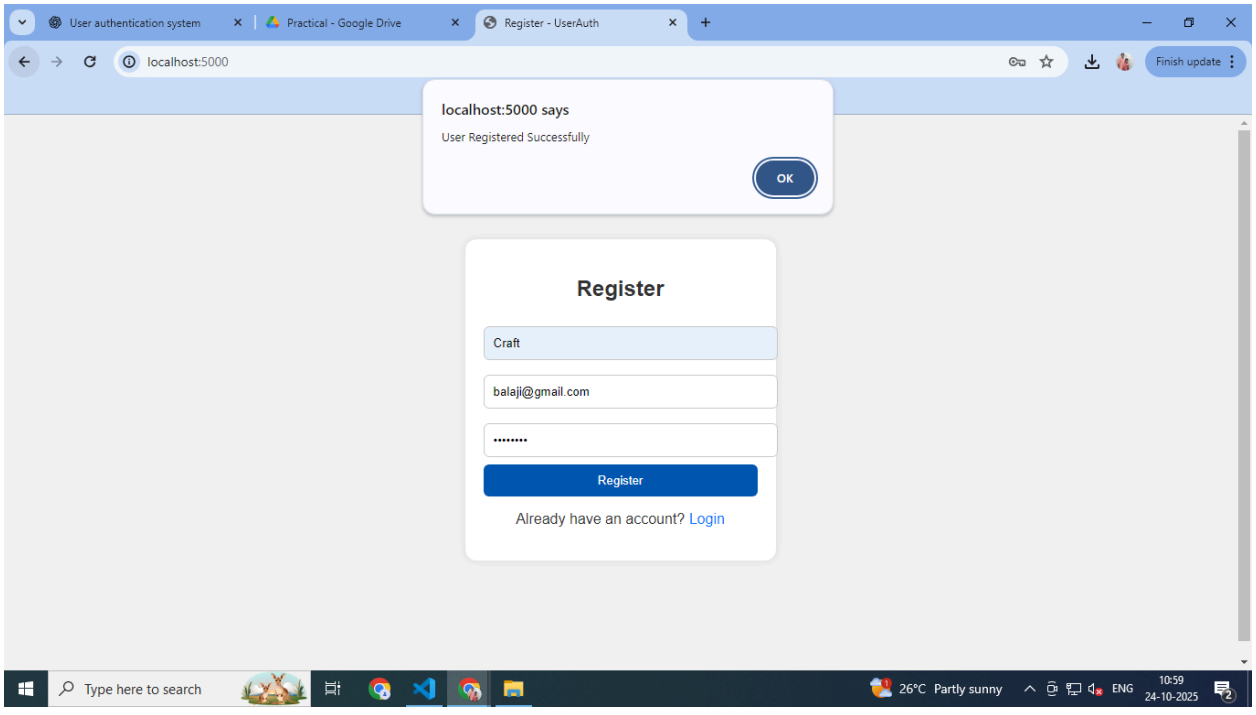
```
OUTPUT  DEBUG CONSOLE  PORTS  TERMINAL                                  node
PS C:\Users\SBM\user auth> npm init -y
>> npm install express mongoose bcryptjs jsonwebtoken dotenv cors
>>
🚀 Server running on port 5000
✅ MongoDB Connected...
```

UI AND UX OUTPUT :

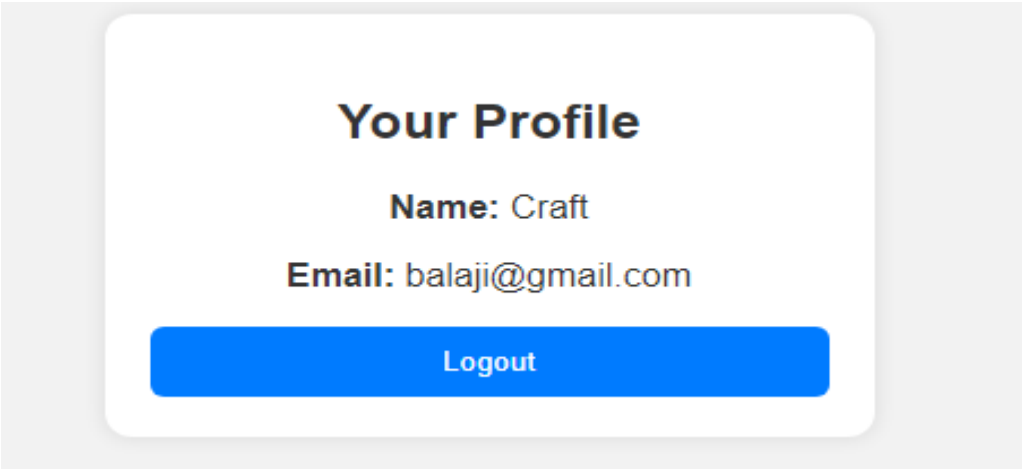
Register



Successfully Register



After Login



Challenges and Solutions :

1. **Challenge 1 – Password Security**
    - ***Problem:*** Storing passwords in plain text made the system vulnerable.
    - ***Solution:*** Implemented bcrypt.js hashing with salt rounds so passwords are securely encrypted before saving to MongoDB.
2. **Challenge 2 – Token Authentication**
    - ***Problem:*** Needed a secure way to keep users logged in without storing passwords.
    - ***Solution:*** Added JWT (JSON Web Tokens) for session management, verifying tokens on every protected route.

3. **Challenge 3 – Database Connection Errors**
    - ***Problem:*** Local MongoDB connection failed during deployment.
    - ***Solution:*** Migrated to MongoDB Atlas, configured a cloud URI, and stored it in the `.env` file for stable connectivity.
4. **Challenge 4 – Invalid or Duplicate Users**
    - ***Problem:*** The app allowed duplicate registrations and crashed on invalid inputs.
    - ***Solution:*** Added backend validation using Express Validator and handled duplicate emails gracefully.
5. **Challenge 5 – CORS and API Access Issues**
    - ***Problem:*** Frontend fetch requests were blocked due to CORS policy.
    - ***Solution:*** Enabled CORS middleware in Express to allow safe cross-origin communication.
6. **Challenge 6 – Session Expiry and Logout**
    - ***Problem:*** Tokens remained valid indefinitely, causing security risks.
    - ***Solution:*** Set JWT expiry (1 hour) and added a logout function that removes the token from storage.
7. **Challenge 7 – Error Handling**
    - ***Problem:*** API errors were inconsistent and confusing.
    - ***Solution:*** Centralized all responses into a unified format with clear status codes and messages.

8. **Challenge 8 – UI/UX Design**
    - ***Problem:*** Early UI lacked responsiveness and user feedback.
    - ***Solution:*** Redesigned pages with HTML + CSS Flexbox, added alerts for success/errors, and made it mobile-friendly.
9. **Challenge 9 – Environment Security**
    - ***Problem:*** Sensitive data (JWT secret, DB URI) appeared in source code.
    - ***Solution:*** Moved all secrets into a .env file and used `dotenv` to load them securely.
10. **Challenge 10 – Deployment Setup**
    - ***Problem:*** Backend failed to start after uploading to the cloud.
    - ***Solution:*** Added a start script in `package.json` (`"start": "node server.js"`) and correctly configured environment variables on Vercel and Netlify.

Version Control (GitHub)

My Project Code GitHub Link → User-Authentication-System

My Project Pdf Upload GitHub Link →Project-Demonstration&Documentation

Team Members

Balaji S
Alagu Pandi K
Vetri Saravanan V