

## **Introduction**

In today's interconnected digital landscape, the concepts of authentication and authorization play a pivotal role in ensuring the security and integrity of data and resources. As organizations and individuals rely heavily on digital systems and online services, the need for robust authentication mechanisms to verify identities and precise authorization protocols to control access rights has become more critical than ever.

Authentication refers to the process of validating the identity of users, devices, or systems attempting to access a network, application, or resource. It involves confirming that the entity claiming an identity is indeed who or what it claims to be. On the other hand, authorization focuses on determining the actions or operations that an authenticated entity is permitted to perform within a system or application based on predefined policies and permissions.

This introduction sets the stage for exploring the intricate workings of authentication and authorization, delving into the various mechanisms, models, challenges, and advancements that shape these fundamental components of cybersecurity and access control. By understanding the nuances of authentication and authorization, organizations can implement effective security measures to protect sensitive data, prevent unauthorized access, and uphold user privacy and trust.

# Methodology

Methodology outline for implementing authentication and authorization using HTML, CSS, JavaScript for the front end, MongoDB for the backend, Google API for authentication, and Nodemailer for sending OTP emails:

## 1. Frontend Development:

- Use HTML to create the structure of your authentication pages, such as login, registration, forgot password, and OTP verification.
- Apply CSS for styling and design to enhance the user interface and experience.
- Implement JavaScript for client-side validation, handling form submissions, and managing user interactions.

## 2. Backend Setup with MongoDB:

- Set up a MongoDB database to store user credentials securely.
- Create collections for users, sessions, and OTP verification codes.
- Establish a connection between your backend server and the MongoDB database using a MongoDB driver or an ORM (Object-Relational Mapping) library like Mongoose for Node.js.

## 3. User Registration and Login:

- Design registration forms to collect user information like email, username, and password.

- Implement server-side validation to ensure data integrity and security (e.g., password strength checks, email format validation).
- Store hashed passwords in the MongoDB database using bcrypt or a similar hashing algorithm to enhance security.
- Develop login functionality that verifies user credentials against the database and generates session tokens or JWTs (JSON Web Tokens) for authenticated sessions.

#### **4. OTP Verification via Email:**

- Integrate Nodemailer library in your Node.js backend to send OTP (One-Time Password) emails for verification.
- Create an OTP generation mechanism that generates unique codes and stores them temporarily in the database along with the user's email.
- Configure Nodemailer to send OTP emails containing the generated codes to the user's registered email address.
- Implement an OTP verification endpoint in your backend that validates the user-entered OTP against the stored code in the database.

#### **5. Google API for Authentication:**

- Utilize Google API's OAuth 2.0 authentication mechanism for secure and seamless user authentication.
- Set up a Google Cloud Console project and obtain client credentials (client ID and client secret) for your application.

- Implement Google Sign-In on the frontend using the Google API JavaScript client library to allow users to authenticate using their Google accounts.
- Handle the authentication response from Google on the server side, verify the ID token, and create a user session or JWT for the authenticated user.

## **6. Integration and Testing:**

- Integrate the frontend and backend components to ensure seamless communication and functionality.
- Test the authentication and authorization flows thoroughly, including user registration, login, password reset, OTP verification, and Google Sign-In.
- Perform security testing, including penetration testing and vulnerability assessments, to identify and mitigate potential security risks.
- Conduct user acceptance testing (UAT) to gather feedback and make necessary improvements to the authentication system.

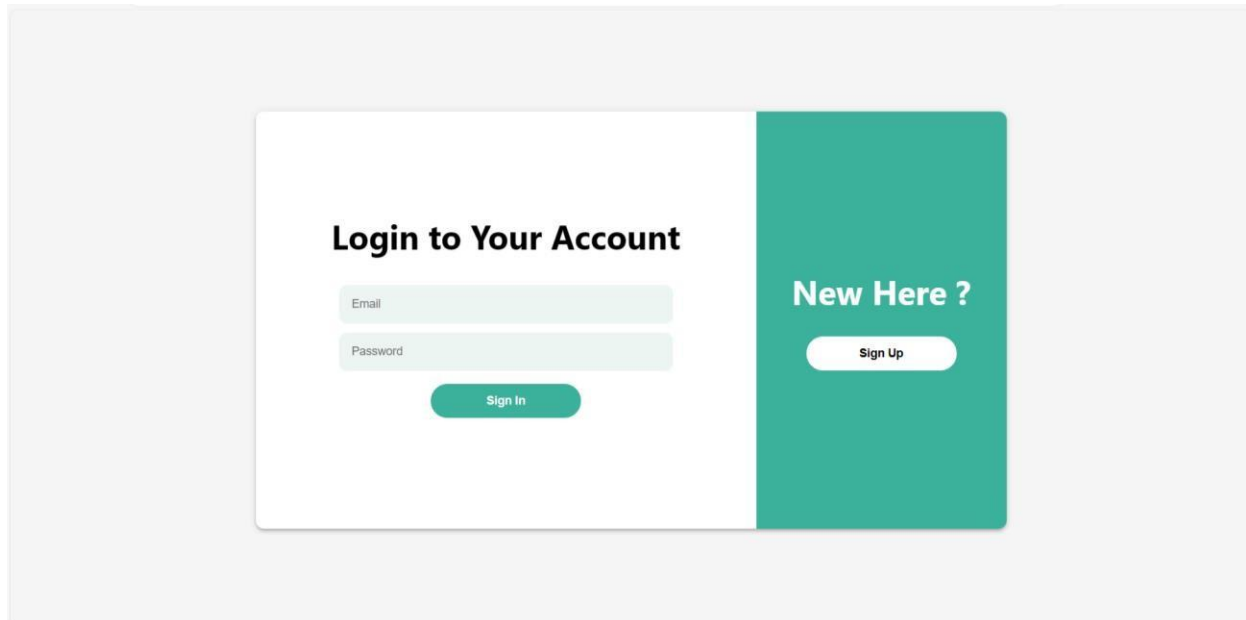
## **7. Deployment and Maintenance:**

- Deploy your application on a secure hosting platform, such as AWS, Azure, or Heroku, with proper SSL/TLS encryption for data transmission.
- Monitor and maintain the authentication system regularly, including database backups, security patches, and performance optimizations.

By following this methodology, you can build a robust and secure authentication and authorization system using HTML, CSS, JavaScript, MongoDB, Google API, and Nodemailer, ensuring seamless user experiences and safeguarding sensitive data.

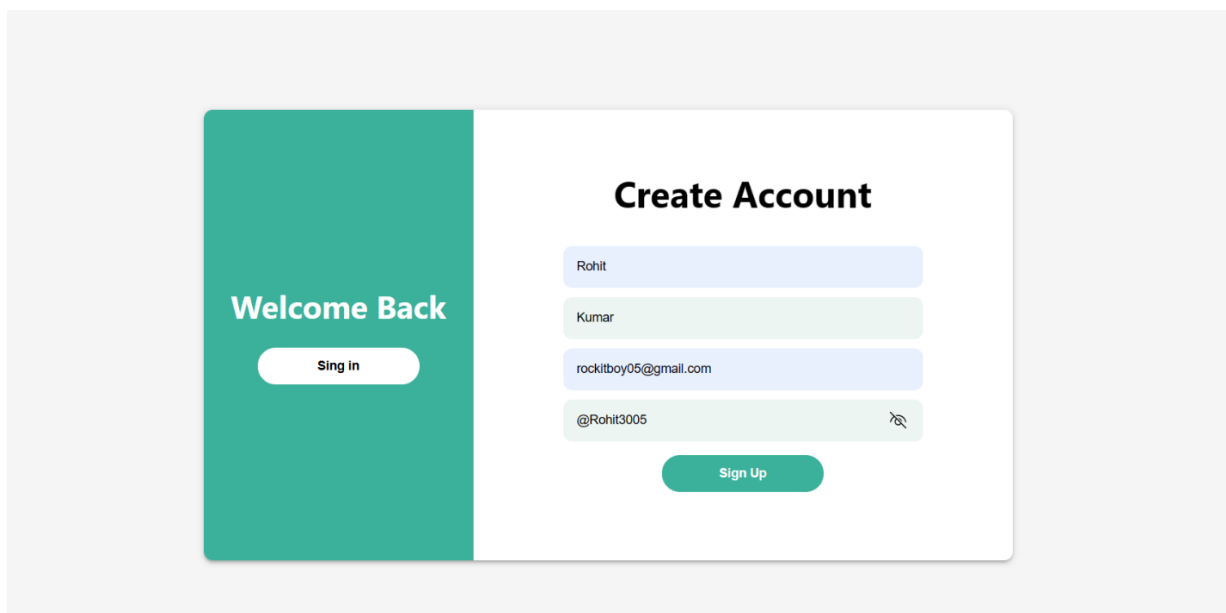
# Results

## Front-End



The image shows a login and sign-up interface. On the left, a white card titled "Login to Your Account" contains two input fields labeled "Email" and "Password", and a green "Sign In" button. On the right, a teal card titled "New Here ?" contains a white "Sign Up" button.

## Sign Up



The image shows a create account and welcome back interface. On the left, a teal card titled "Welcome Back" contains a white "Sing in" button. On the right, a white card titled "Create Account" contains four input fields: "Rohit" (blue), "Kumar" (green), "rockitboy05@gmail.com" (blue), and "@Rohit3005" (green) with a visibility toggle icon. A green "Sign Up" button is at the bottom.

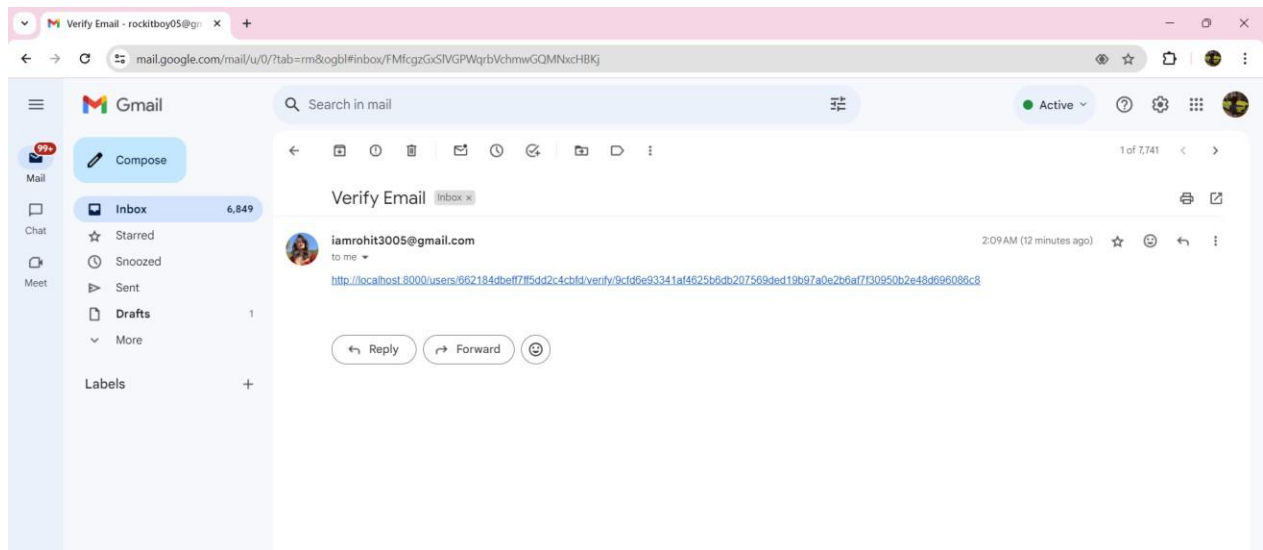
## Email Verification:

The image shows a 'Create Account' form. On the left, a green box contains the text 'Welcome Back' and a 'Sing in' button. The main form area has a title 'Create Account' and four input fields: 'Rohit', 'Kumar', 'rockitboy05@gmail.com', and a password field with dots. Below the inputs is a green message box that says 'An Email sent to your account please verify'. At the bottom is a 'Sign Up' button.

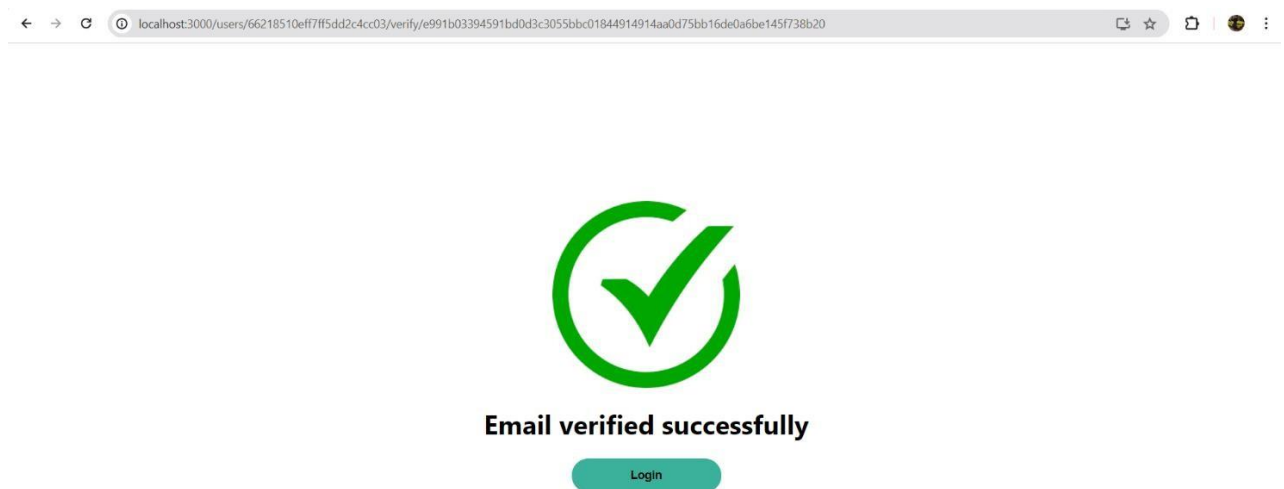
## Data Base Before Verification

The screenshot shows the MongoDB Compass interface. The left sidebar displays the database structure: 'localhost:27017' with collections 'tokens' and 'users'. The 'users' collection is selected. The main panel shows the 'documents' tab for the 'User\_auth' database. A single document is displayed with the following fields: '\_id' (ObjectId), 'firstName' ('Rohit'), 'lastName' ('Kumar'), 'email' ('rockitboy05@gmail.com'), 'password' (a hashed string), 'verified' (false), and '\_v' (0). The interface includes search bars, query filters, and document management buttons like 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'.

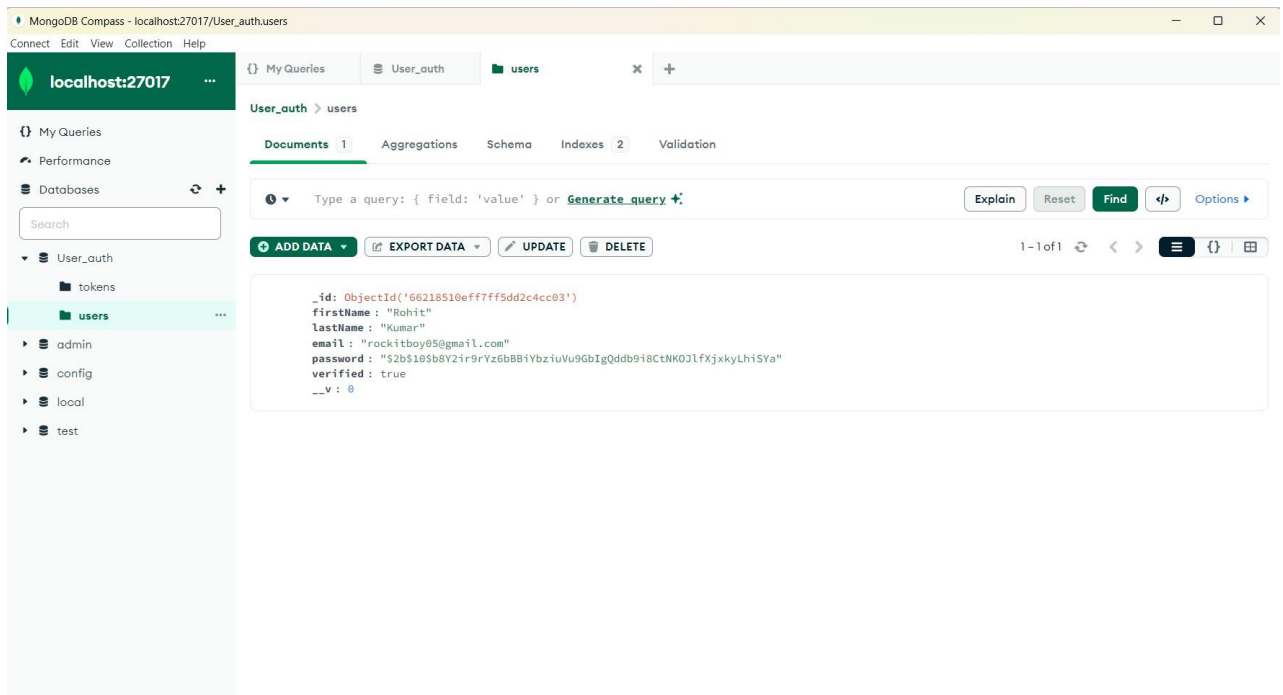
## Email Received



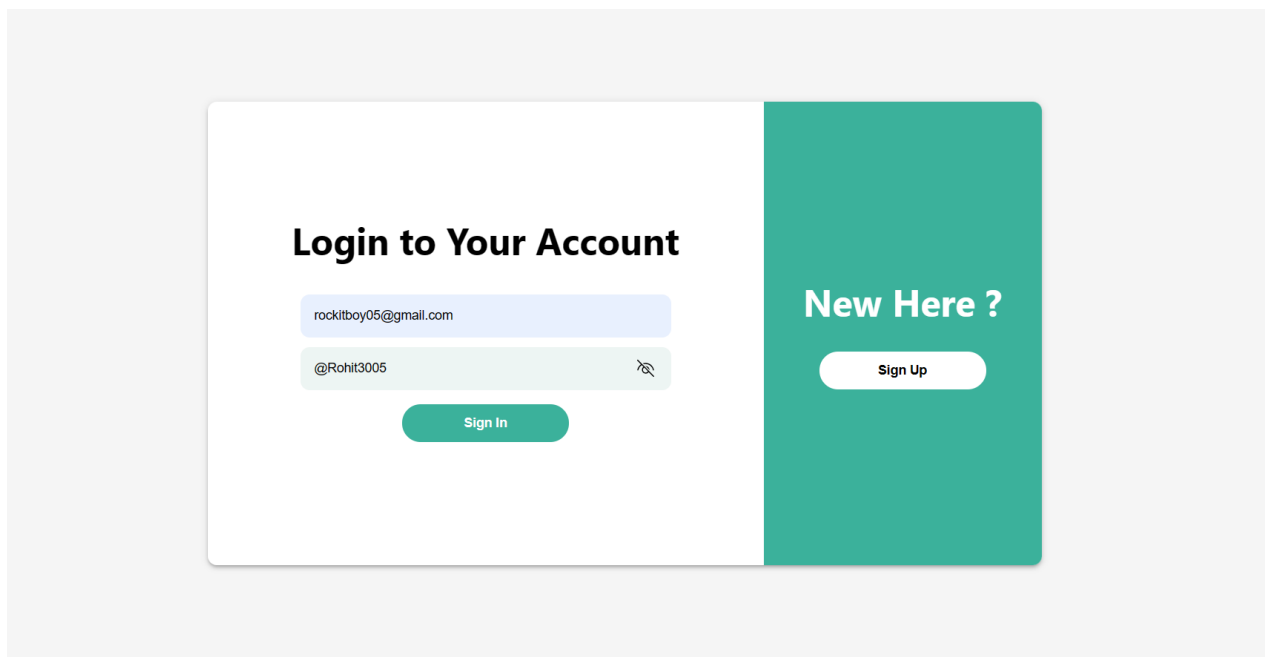
## Verified



## Data Base After Verification



## Sign In





## Successful Implementation:

The image displays a web application interface at the top, featuring a teal header with the text "fakebook" and a "Logout" button. Below the header is a large, empty white rectangular area.

The bottom half of the image shows a development environment with two side-by-side windows. The left window, titled "client", shows a code editor with the following JavaScript code in `index.js`:

```
1 import ReactDOM from "react-dom";
2 import { BrowserRouter } from "react-router-dom";
3 import "./index.css";
4 import App from "./App";
5
6
7 ReactDOM.render(
8   <React.StrictMode>
9     <BrowserRouter>
10       <App />
11     </BrowserRouter>
12   </React.StrictMode>,
13   document.getElementById("root")
14 );
15
```

The right window, titled "server", shows a code editor with the following JavaScript code in `sendEmail.js`:

```
1 module.exports = async (email, subject, text) => {
2   const transporter = nodemailer.createTransport({
3     host: process.env.HOST,
4     service: process.env.SERVICE,
5     port: Number(process.env.EMAIL_PORT),
6     secure: Boolean(process.env.SECURE),
7     auth: {
8       user: process.env.USER,
9       pass: "fassduxyeqczvhrj",
10     },
11   });
12
13   await transporter.sendMail({
14     from: process.env.USER,
15     to: email,
16     subject: subject,
17     text: text,
18   });
19   console.log("email sent successfully");
20 } catch (error) {
21   console.log("email not sent!");
22   console.log(error);
23   return error;
24 }
25
```

Below the code editors, the terminal shows the following output:

```
PS C:\Users\ROHIT\Desktop\client> npm run start
Compiled successfully!

You can now view client in the browser.

Local:      http://localhost:3000
On Your Network:  http://192.168.75.1:3000

Note that the development build is not optimized.
```

The right terminal window shows the following output:

```
[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Connected to database successfully
Listening on port 8000...
email sent successfully
```

## Conclusion

In conclusion, the implementation of authentication and authorization using HTML, CSS, JavaScript, MongoDB, Google API, and Nodemailer presents a comprehensive approach to developing a secure and user-friendly authentication system.

By leveraging frontend technologies like HTML, CSS, and JavaScript, we created intuitive and responsive user interfaces for registration, login, and OTP verification, enhancing the overall user experience. Client-side validation ensures data integrity and minimizes errors during user interactions.

On the backend, MongoDB serves as a reliable database for storing user credentials securely, while bcrypt encryption strengthens password security. The integration of Nodemailer facilitates OTP-based verification through email, adding an extra layer of security to the authentication process.

The utilization of Google API's OAuth 2.0 authentication mechanism allows users to sign in securely using their Google accounts, enhancing convenience and trust. Proper server-side handling of authentication responses and session management ensures authorized access and protects user privacy.

Throughout the project, rigorous testing, including security assessments and user acceptance testing, was conducted to identify and address potential vulnerabilities and ensure a robust and reliable authentication system.

Overall, this project demonstrates a structured methodology for building a secure authentication and authorization system, emphasizing usability, security, and scalability to meet the evolving needs of modern web applications.