

# Web-Based User Identification

CS23332 - Database Management Systems

Submitted by

Sugumar B (231501162/AIML-C)

Vijay R (231501183/AIML-C)

Vishal M (231001157/AIML-C)

Mini Project November 2024

Department of Information Technology Rajalakshmi Engineering College Thandalam, Chennai - 602105

**BONAFIDE CERTIFICATE**

**Certified that this project report “**Web-Based User Identification**” is the Bonafide work of “SUGUMAR B(231501162),VIJAY R(231501183), VISHAL M (231501184)” who carried out the project work under my supervision.**

**Submitted for the Practical Examination held on ----------------------------------------------**

**SIGNATURE SIGNATURE**

**INTERNAL EXAMINER EXTERNAL EXAMINER**

## ABSTRACT

The User Identification System is a robust, scalable, and secure solution designed to address the growing demand for efficient user authentication and management in modern digital environ- ments. In an era where user data is a critical asset, ensuring its security and accessibility while maintaining user convenience is paramount. This project employs advanced technologies such as Flask, MongoDB, and QR code authentication to create a comprehensive framework that inte- grates seamlessly with existing systems while offering a user-friendly experience.

The system's primary objective is to simplify user identification processes by providing secure login mechanisms, personalized profile management, and efficient data handling. It leverages state-of-the-art encryption techniques for password security and QR codes for rapid authentica- tion, reducing the risks associated with traditional login methods. Additionally, the system in- cludes features like real-time profile updates, password recovery via email, and a profile comple- tion tracker to enhance user engagement and data accuracy.

A significant focus of this project is on scalability and adaptability. The modular architecture al- lows the system to cater to diverse organizational needs, ranging from small businesses to large- scale enterprises. Its compatibility with a wide range of devices and platforms ensures seamless integration, making it a versatile solution for industries such as education, e-commerce, healthcare, and corporate environments.

Security remains a cornerstone of this system, with robust measures implemented to safeguard user data against unauthorized access and breaches. By using hashed passwords, secure session handling, and data validation, the system adheres to best practices in cybersecurity. Furthermore, the inclusion of QR code-based login enhances both security and user convenience, enabling a seamless and efficient authentication process.

This project highlights the potential for future enhancements, such as incorporating multi-factor authentication, artificial intelligence for user behavior analysis, and role-based access control to meet evolving user needs. The system's design ensures it can adapt to technological advancements and organizational growth, making it a long-term solution for user management.

# TABLE OF CONTENTS

CHAPTER NO. TITLE PAGE NO.

ABSTRACT 3

1. INTRODUCTION 5
2. SYSTEM DESIGN 6
3. IMPLEMENTATION 8
4. KEY FEATURES 13
5. CONCLUSION 15
6. REFERENCES 16

## CHAPTER 1 INTRODUCTION

### Problem Statement

Current user management systems often face challenges related to data security, user authentication, and profile management. There is a need for a robust User Identification System that allows users to securely register, authenticate, and manage their profiles while ensuring data integrity and privacy. This system should streamline user interactions, enhance security measures, and improve overall user experience.

### Objective of the Project

* + 1. The application allows users to create an account, log in securely, update their profiles, and store notes that can be retrieved and edited as needed. The system is designed with user experience in Secure User Authentication: Implement a robust authentication mechanism that ensures secure login and protects user data from unauthorized access.
    2. **User Profile Management**: Develop a user-friendly interface that allows users to create, update, and manage their profiles, including personal information and preferences.
    3. **Data Privacy and Integrity**: Ensure that all user data is stored securely and complies with relevant data protection regulations to maintain user privacy and data integrity.
    4. **Password Recovery**: Provide a secure password recovery process to assist users in regain- ing access to their accounts in case of forgotten passwords.
    5. **Scalability**: Design the system to be scalable, accommodating an increasing number of users and data without compromising performance.
    6. **User Experience Enhancement**: Focus on creating an intuitive user interface that en- hances the overall user experience, making it easy for users to navigate and manage their accounts.
    7. **Reporting and Analytics**: Integrate reporting features to analyze user behaviour and sys- tem performance, enabling continuous improvement of the user management process.

## CHAPTER 2 SYSTEM DESIGN

### System Architecture

The User Identification System follows a client-server architecture with a modular design. It con- sists of two main components: the client-side (front-end) and the server-side (back-end).

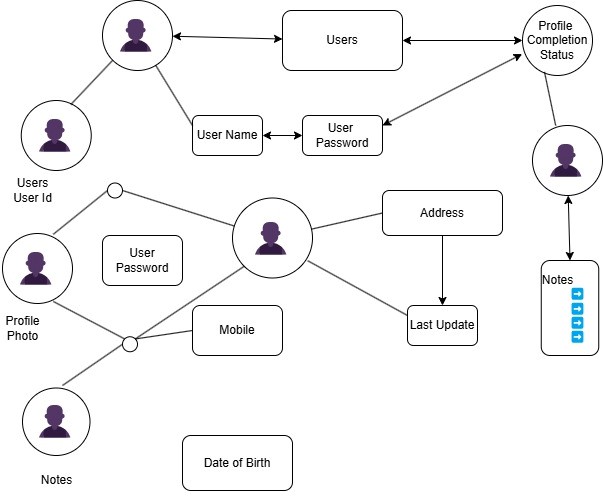
#### Client-Side:

Users interact with the system through a web interface built with HTML, CSS, and Ja- vaScript. It includes pages for login, registration, profile management, and viewing notes.

#### Server-Side:

The back-end is developed using the Flask web framework. It handles user authentication, profile management, and data storage. MongoDB is used for database management. The server also inte- grates features like QR code generation for login and email-based password recovery.

### ER Diagram



* 1. **System Specification Software Requirements:**

The User Identification System relies on several software tools to ensure efficient operation and smooth development. Flask, a lightweight Python web framework, serves as the

backbone for the backend, handling routing, user authentication, and request processing. Mon- goDB, a NoSQL database, is used to store and manage user data, notes, and system-related infor- mation, offering scalability and flexibility. Python, chosen for its simplicity and extensive librar- ies, facilitates backend development and ensures seamless interaction between different system components. For the front end, HTML, CSS, and JavaScript are employed to create a user- friendly and visually appealing interface. Additionally, the system uses Werkzeug for secure

password hashing and user authentication, ensuring the security of user data.

### Hardware Requirements:

The system requires a hardware configuration capable of handling local testing and efficient op- eration. A multi-core processor, such as an Intel i5, is recommended to manage multiple tasks simultaneously and ensure smooth data processing. The system should have a minimum of 4GB of RAM to support concurrent operations and prevent lags or interruptions during data handling. In terms of storage, at least 10GB of free disk space is necessary to store user data, application files, and temporary system data. This hardware setup ensures the system runs smoothly during development and deployment, delivering a responsive and reliable user experience.

### Tools/Platforms:

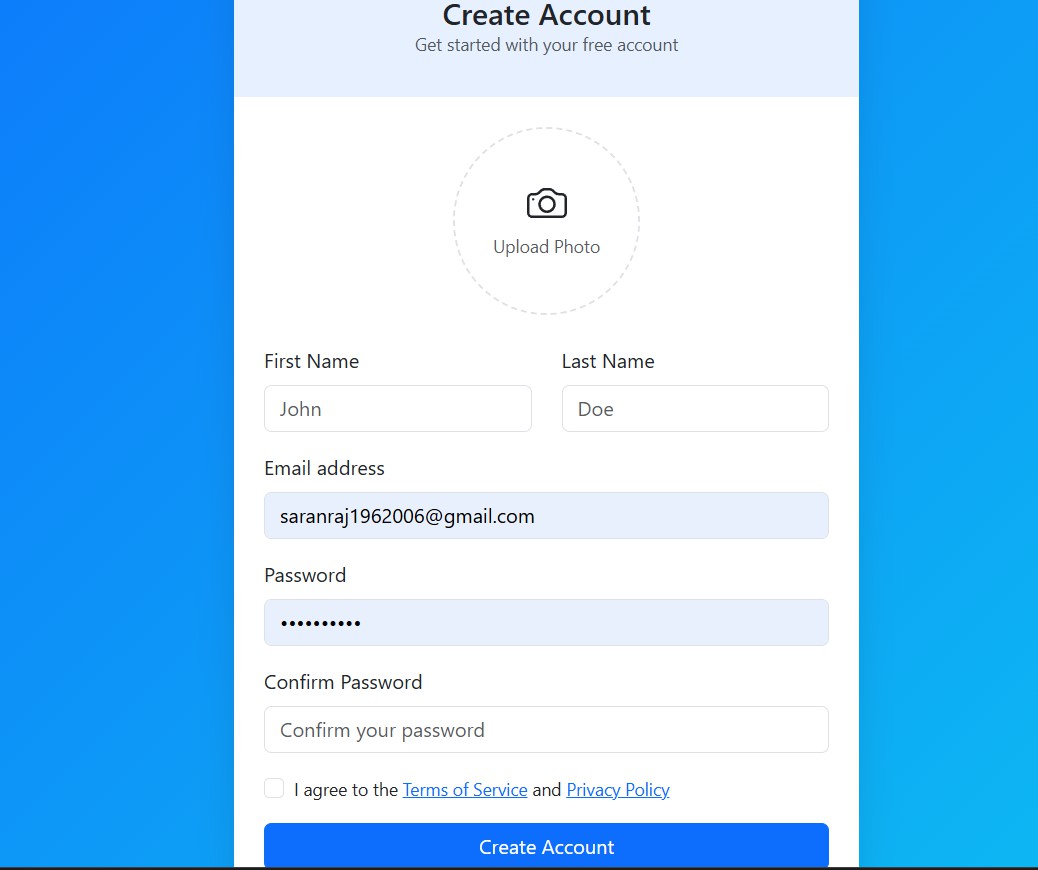
Database Management Tool (Mongo DB Compass (GUI)). Code Editor (VSCode, IDLE).

#### Technologies Used:

* Flask, MongoDB, Werkzeug, Qr-code, Flask-Mail, HTML/CSS

## CHAPTER 3 IMPLEMENTATION

### Sign-up Page



@app.route('/signup', methods=['GET', 'POST']) def signup():

if request.method == 'POST':

first\_name = request.form.get('firstName') last\_name = request.form.get('lastName') email = request.form.get('email') password = request.form.get('password') mobile = request.form.get('mobile')

dob = request.form.get('dob') address = request.form.get('address') # Calculate age if dob is provided age = None

if dob:

dob\_date = datetime.strptime(dob, '%Y-%m-%d') today = datetime.today()

age = today.year - dob\_date.year - ((today.month, today.day) < (dob\_date.month, dob\_date.day))

if users\_collection.find\_one({'email': email}):

flash('You already have an account. Please log in.', 'warning') return redirect(url\_for('signup'))

profile\_photo = request.files.get('profilePhoto') profile\_photo\_data = None

if profile\_photo and allowed\_file(profile\_photo.filename): profile\_photo\_data = base64.b64encode(profile\_photo.read()).de-

code('utf-8')

new\_user = {

'first\_name': first\_name, 'last\_name': last\_name, 'email': email,

'password': generate\_password\_hash(password), 'mobile': mobile,

'dob': dob,

'age': age, 'address': address,

'profile\_photo': profile\_photo\_data

}

result = users\_collection.insert\_one(new\_user) session['user\_id'] = str(result.inserted\_id)

flash('Account created successfully!') return redirect(url\_for('home'))

return render\_template('signup.html')

@app.route('/login', methods=['GET', 'POST']) def login():

if request.method == 'POST':

email = request.form.get('email') password = request.form.get('password')

user = users\_collection.find\_one({'email': email})

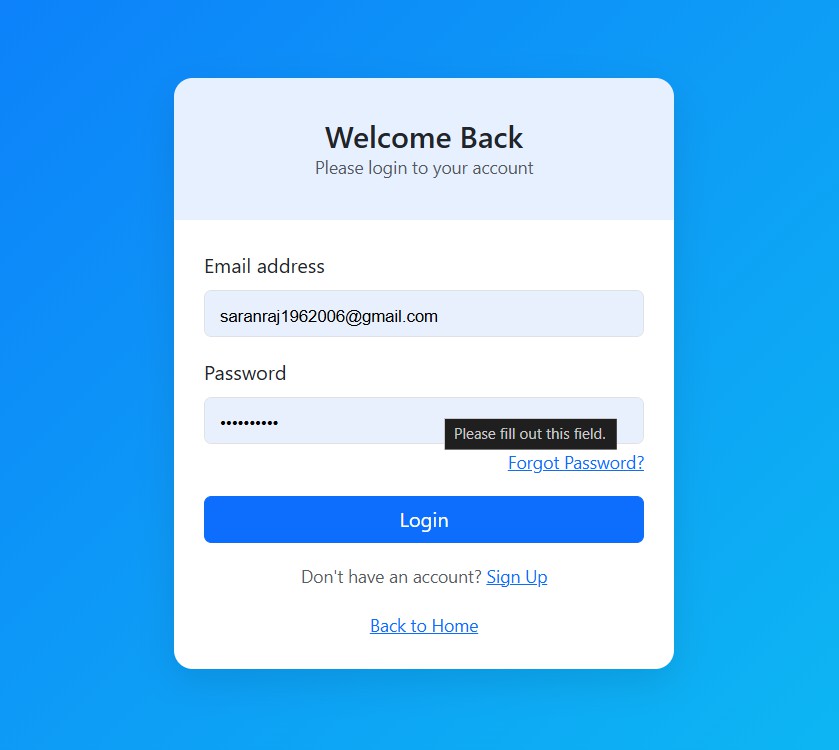
if user and check\_password\_hash(user['password'], password): session['user\_id'] = str(user['\_id'])

flash('Logged in successfully!') return redirect(url\_for('home'))

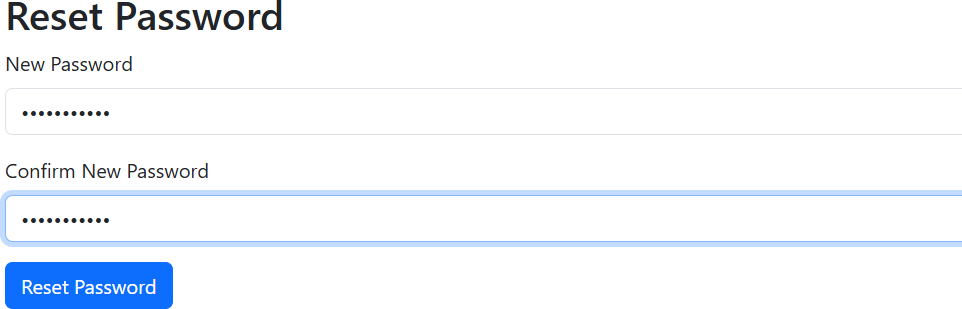
flash('Invalid email or password') return redirect(url\_for('login'))

return render\_template('login.html')

* 1. **Login:**



* 1. **Forgot Password:**



def reset\_password(token): try:

email = serializer.loads(token, salt='password-reset-salt', max\_age=3600) except:

flash('The password reset link is invalid or has expired.', 'error') return redirect(url\_for('login'))

if request.method == 'POST':

new\_password = request.form.get('password') confirm\_password = request.form.get('confirm\_password') if new\_password != confirm\_password:

flash('Passwords do not match.', 'error') return render\_template('reset\_password.html')

hashed\_password = generate\_password\_hash(new\_password)

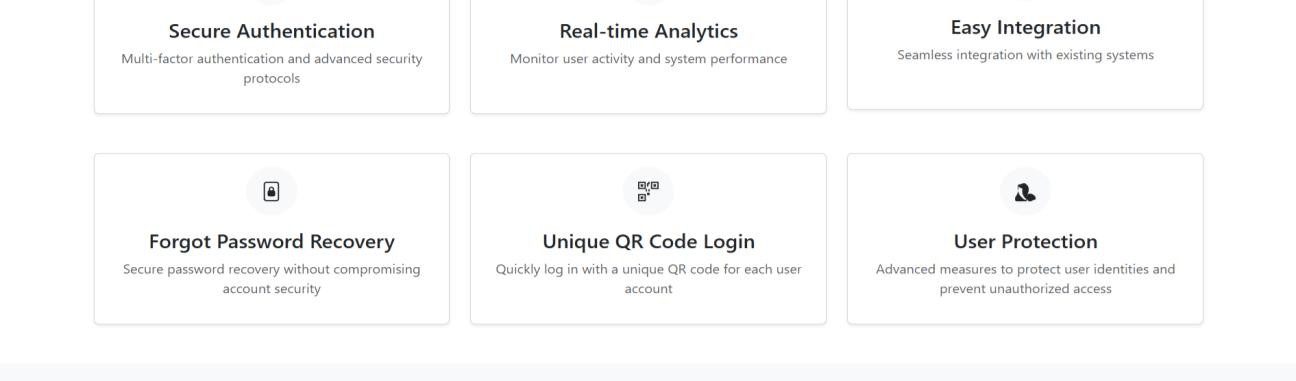
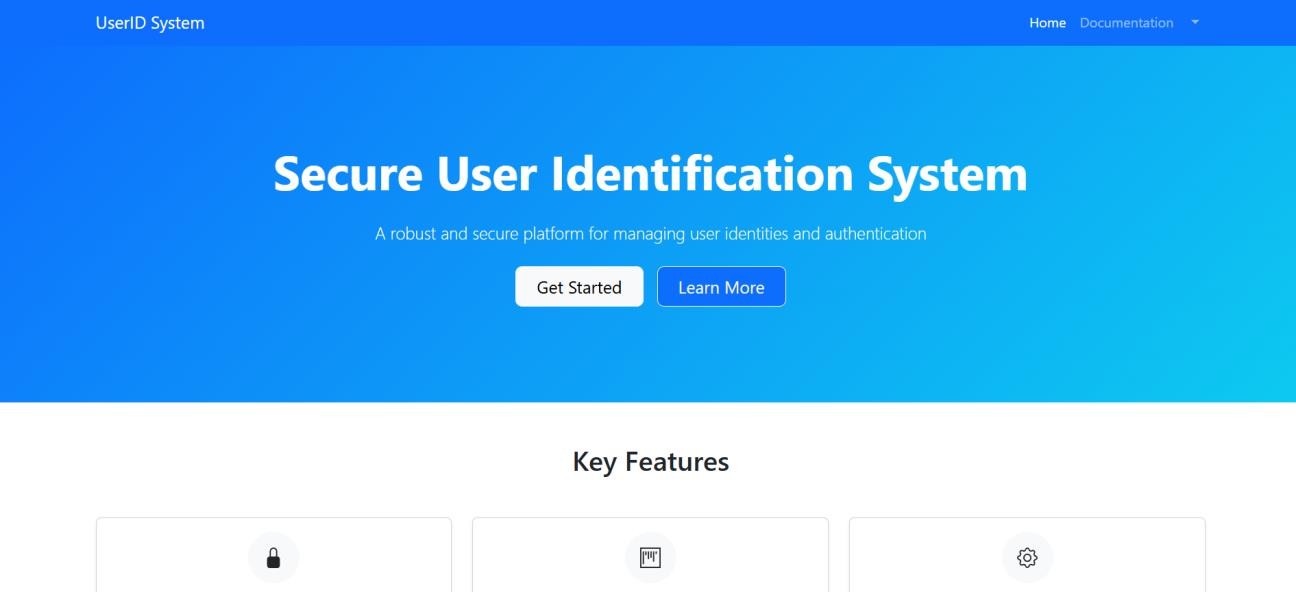
update\_result = users\_collection.update\_one({'email': email}, {'$set': {'password': hashed\_password}}) if update\_result.modified\_count > 0:

flash('Your password has been updated!', 'success') else:

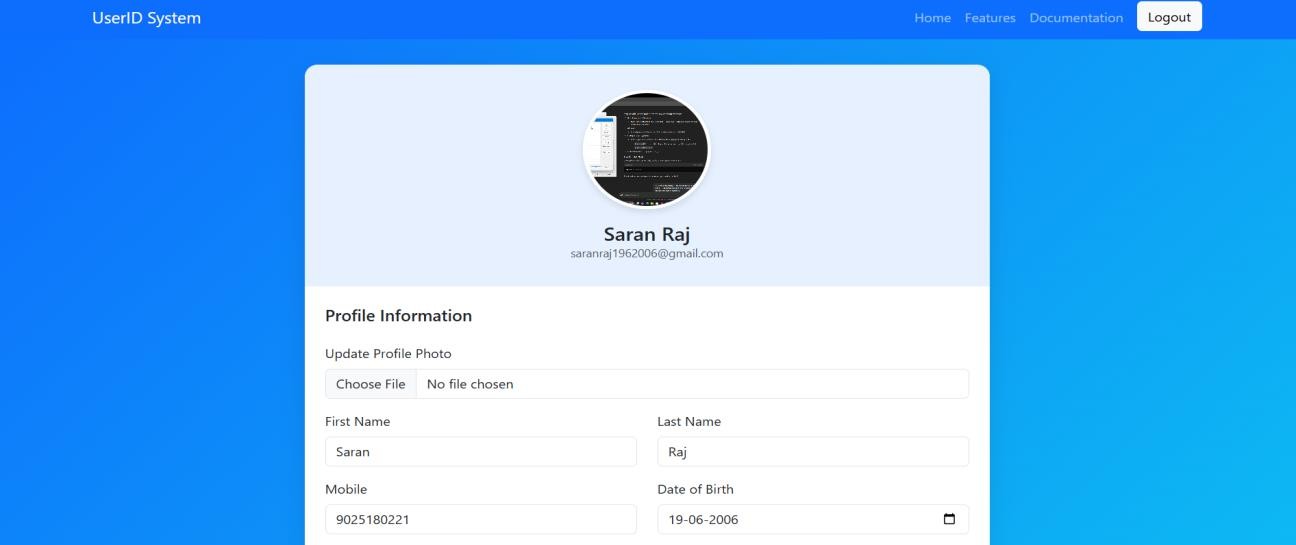
flash('Password update failed. Please try again.', 'error')

return redirect(url\_for('login'))

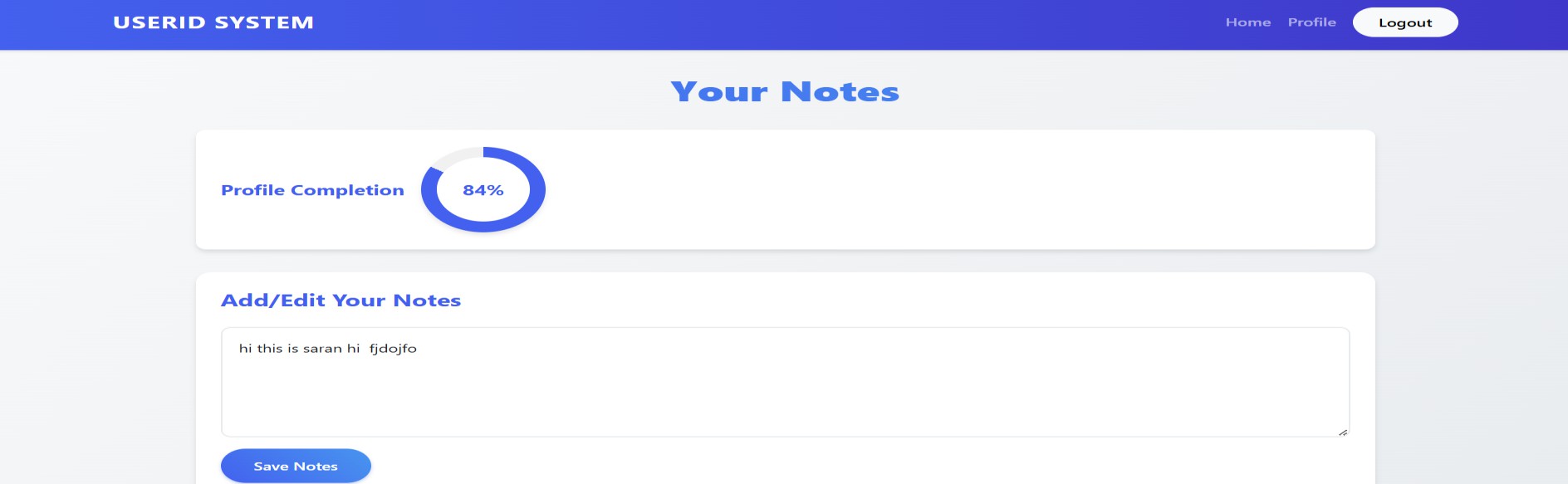
* 1. **Home Page:**



* 1. **Profile Section:**



* 1. **Notes and Profile Completion:**



* 1. **Generated Unique QR Code:**

@app.route('/generate\_qr\_code/<user\_id>') def generate\_qr\_code(user\_id):

# Generate a unique URL for QR login for each user

login\_url = f"http://127.0.0.1:5000/qr\_login?user\_id={user\_id}"

qr = qrcode.make(login\_url) img\_io = BytesIO() qr.save(img\_io, 'PNG') img\_io.seek(0)

return send\_file(img\_io, mimetype='image/png')

@app.route('/qr\_login') def qr\_login():

user\_id = request.args.get('user\_id') if not user\_id:

flash("Invalid QR code. Please try again.", "warning")

return redirect(url\_for('login'))

user = users\_collection.find\_one({"\_id": Ob- jectId(user\_id)})

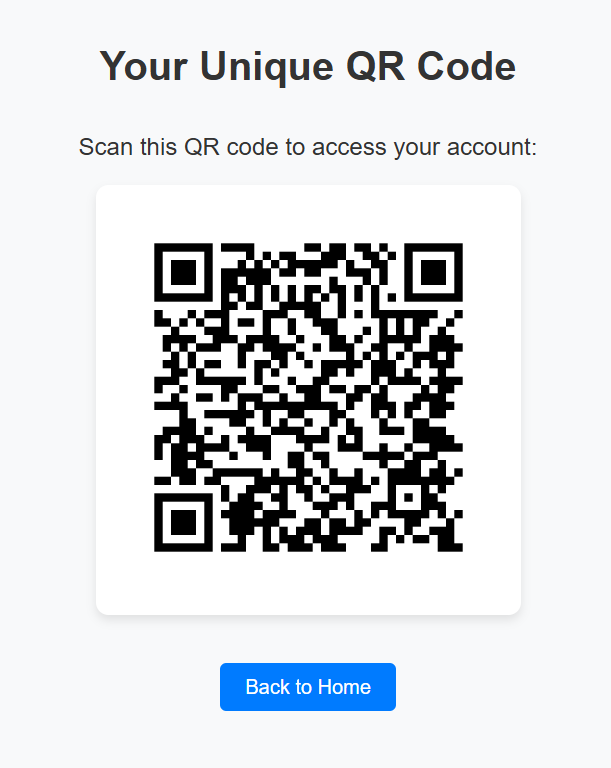
if user:

session['user\_id'] = str(user['\_id']) flash("Logged in via QR code successfully!") return redirect(url\_for('home'))

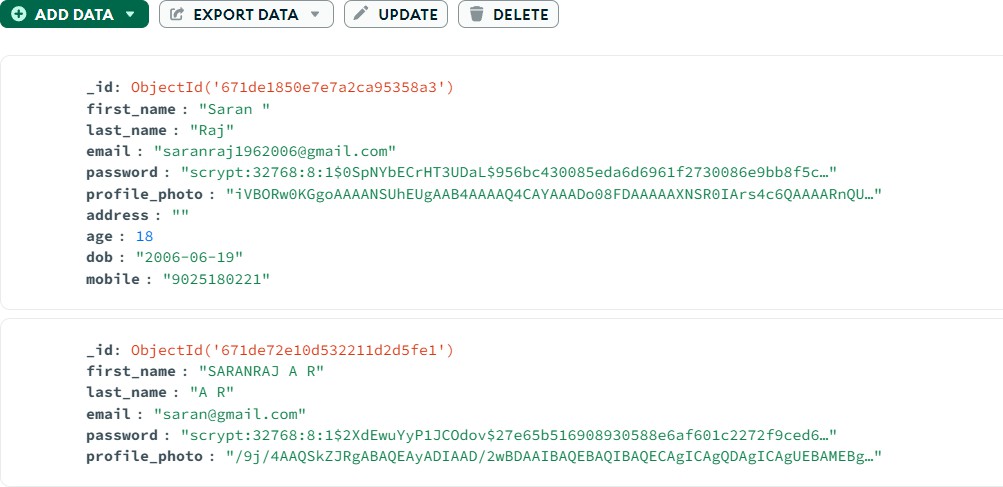
else:

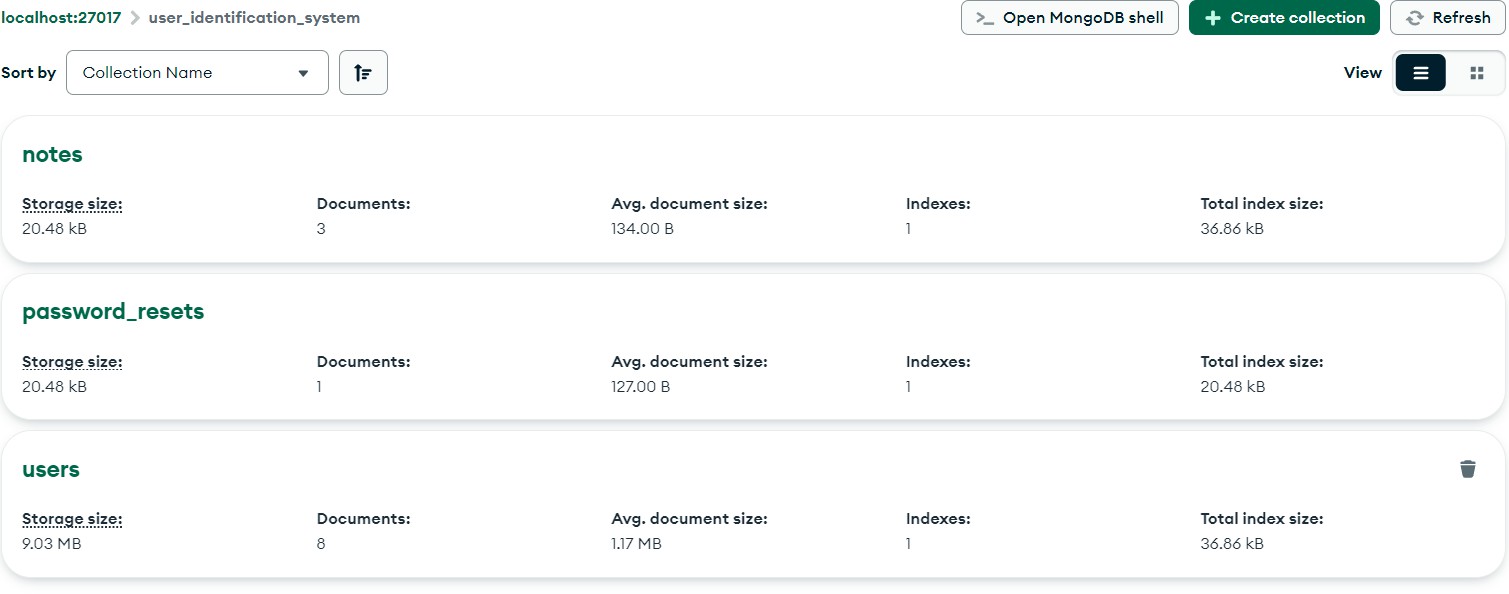
flash("User not found. Please sign up first.", "warning")

return redirect(url\_for('signup'))



* 1. **Sample Databases:**





# Update user profile in the database users\_collection.update\_one(

{"\_id": ObjectId(user\_id)},

{

"$set": {

"first\_name": first\_name, "last\_name": last\_name, "email": email,

"mobile": mobile,"dob": dob, "age": age, "address": address, "profile\_photo": prfile\_photo\_data

}

}

# MongoDB connection

client = MongoClient('mongodb://lo- calhost:27017/')

db = client['user\_identification\_sys- tem']

users\_collection = db['users'] notes\_collection = db['notes']

**CHAPTER-4 KEY-FEATURES**

1. **User Registration**
   * Endpoint: /signup
   * Users register with personal details such as name, email, mobile number, date of birth, address, and an optional profile picture.
   * Passwords are hashed using werkzeug.security.generate\_password\_hash for secure stor- age.
   * The profile picture, if provided, is encoded in Base64 and stored in the database.
2. **User Login**
   * Endpoint: /login
   * Users authenticate using their email and password. Successful login redirects them to the home page, and their user ID is stored in a session.
   * Login is protected with hashed password verification using check\_password\_hash**.**
3. **Profile Completion Calculation**
   * Calculates the percentage of profile fields completed and displays this value to the user on the notes page.
   * Fields tracked include name, email, mobile, date of birth, and address.
4. **Notes Feature**
   * Endpoint: /notes
   * Authenticated users can create and save notes. The notes are stored in the MongoDB col- lection, and users can retrieve and edit their notes.
   * The notes page also displays profile completion status.
5. **Profile Update**
   * Endpoint: /update\_profile
   * Users can update their profile details and upload a new profile picture if desired. The age is recalculated based on the date of birth provided.
6. **QR Code Login**
   * Endpoint: /generate\_qr\_code/<user\_id>
   * Generates a QR code with a unique URL for each user. When scanned, it allows users to log in without entering their credentials.
   * The QR code directs users to /qr\_login with a unique user ID parameter, validating the session and granting access.
7. **Password Reset**
   * Endpoints: /forgot\_password, /reset\_password/<token>
   * Users can request a password reset link, which is sent to their email. The link includes a token for verification and leads to a password reset page.
   * Users must enter and confirm the new password, which is then hashed and updated in the database.
8. **Documentation Page**
   * Endpoint: /documentation
   * Serves as a static page that provides an overview or instructions for using the application.

**CHAPTER-5 CONCLUSION**

The User Identification System represents a comprehensive solution for secure user management and authentication in a wide range of applications. By utilizing Flask as the web framework and MongoDB as the database, the system ensures high performance, scalability, and flexibility. The integration of advanced features such as QR code-based login, profile completion tracking, and password encryption enhances security while maintaining a seamless user experience.

This system demonstrates the potential of combining robust back-end technologies with intuitive front-end interfaces to address real-world challenges in identity management. The project fo-

cuses on creating a user-centric platform that prioritizes security, accessibility, and ease of use. By incorporating essential functionalities like email-based password recovery and profile cus-

tomization, the system provides users with complete control over their accounts while safeguard- ing their sensitive data.

Furthermore, the architecture of this project lays the groundwork for scalability, making it suita- ble for deployment across various domains, including educational institutions, corporate envi-

ronments, and e-commerce platforms. The use of lightweight and efficient tools ensures that the system remains responsive and adaptable to different user requirements and technological ad-

vancements.

In conclusion, the User Identification System not only addresses the critical need for secure au- thentication but also serves as a versatile and innovative framework for future enhancements. It exemplifies the effectiveness of modern technologies in solving complex challenges and reaf- firms the importance of user data protection in the digital age. This project provides a strong foundation for continuous development, ensuring its relevance and value in an ever-evolving

technological landscape

**CHAPTER- 6 REFRENCES**

1. Flask Documentation
   * URL: Flask Documentation
   * Official documentation for Flask, covering routing, sessions, request handling, and templating.
2. MongoDB Integration with Flask
   * URL: [MongoDB and Flask - A Complete Tutorial](https://www.mongodb.com/developer/languages/python/quickstart-python-flask/)
   * Guide on using MongoDB with Flask, including setup, CRUD operations, and us- ing pymongo.
3. User Authentication in Flask
   * URL: Flask Authentication Tutorial
   * Step-by-step on implementing secure user authentication with werkzeug pass- word hashing.
4. Profile Management with Flask-Login
   * URL: Profile Management in Flask
   * Introduction to Flask-Login for managing sessions, user logins, and profile access control.
5. Generating QR Codes in Python
   * URL: [QR Code Generator with Python](https://pypi.org/project/qrcode/)
   * Documentation for the qrcode library, helpful for implementing QR-based logins.
6. File Uploads and Security in Flask
   * URL: File Uploads in Flask
   * Best practices for handling secure file uploads, including profile photo functional- ity.
7. Flask-Mail for Password Reset
   * URL: Flask-Mail Documentation
   * Guide for using Flask-Mail to enable password reset emails with secure tokenized links.
8. Secure Password Hashing
   * URL: Werkzeug Security Documentation
   * Functions for secure password hashing (generate\_password\_hash and check\_password\_hash).
9. Organizing Flask Project Structure
   * URL: Flask Project Structure
   * Tips for scalable project organization, useful for maintaining large applications.
10. Jinja2 Templating

* URL: Jinja2 Documentation
* Reference for Jinja2, the templating engine used in Flask, enabling dynamic content ren- dering.