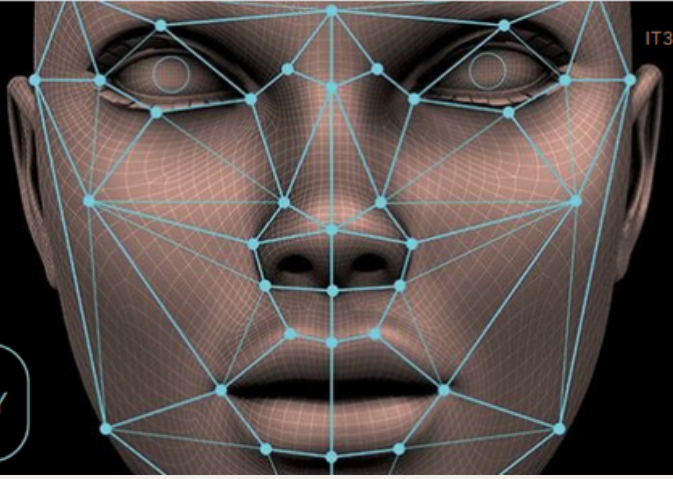
21/03/2024

IT360: INFORMATION INSURANCE AND SECURITY

Journey To The Future



FOR VERIFIED USERS ONLY

Introducing

3 E Y E S

Web-Based Facial Authentication System

Realized by:

Sarra Ibn El Haj - Ju.Fin/IT

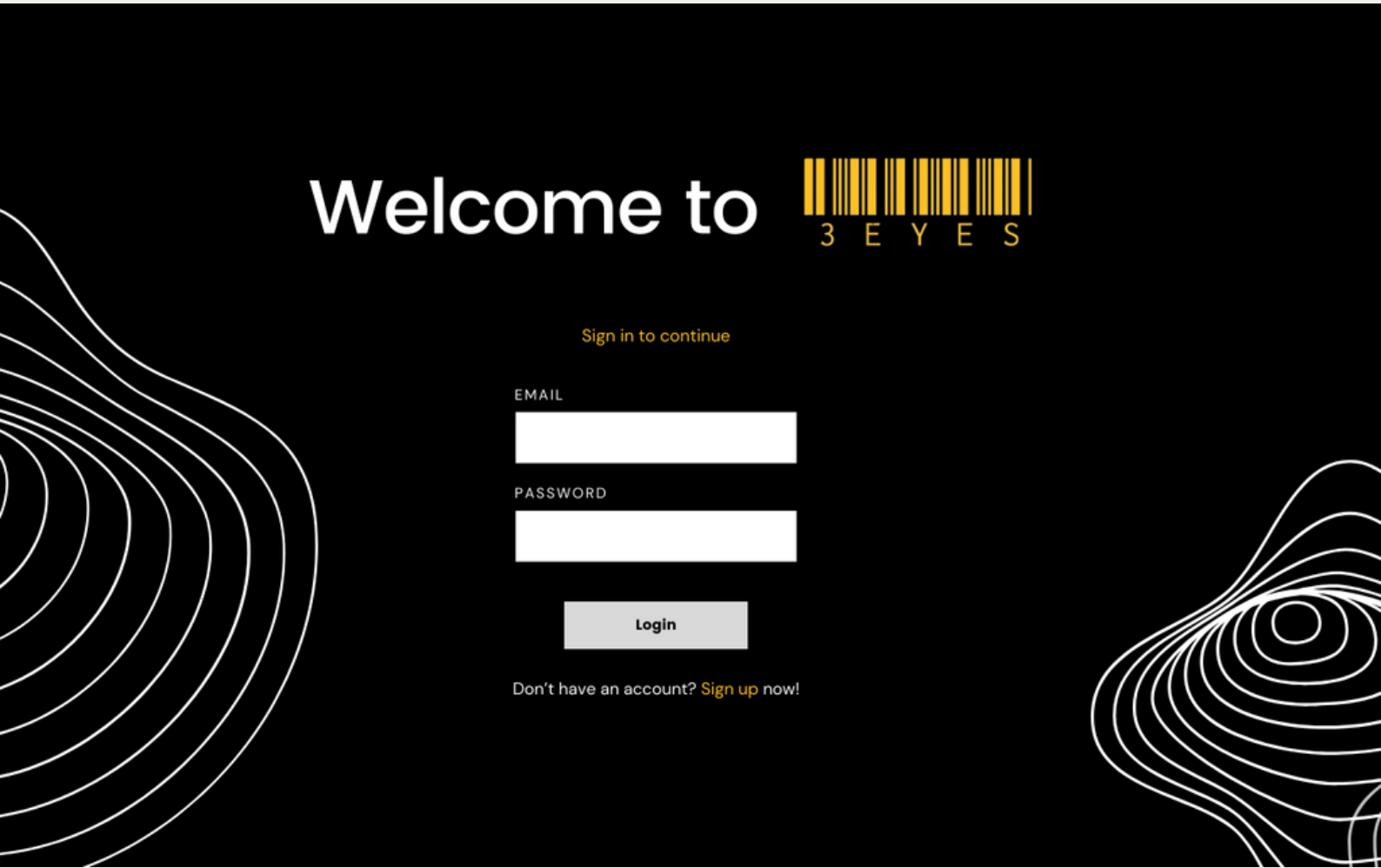
Becher Zribi- Ju.Fin/IT

Amine Maaloul - Ju.BA/IT

Raouf Lakhoues - Ju.BA/IT

High Level Design of our proposed solution

Our project aims to develop a secure web-based dashboard that utilizes facial recognition technology for user authentication. The primary objective is to create a user-friendly platform where authorized members can access and manage information.



Objectives:

- **Enhanced Security**: Facial recognition technology offers a stronger layer of authentication compared to traditional password-based systems. Passwords can be vulnerable to hacking, phishing attacks, or simply being forgotten. Facial recognition adds an extra layer of verification, making it more difficult for unauthorized individuals to gain access.
- **Improved User Experience**: Facial recognition provides a convenient and contactless login experience. Users don't need to remember complex passwords or manage multiple login credentials for different systems. Simply looking into a webcam can grant access, streamlining the login process.
- **Potential for Multi-Factor Authentication (MFA)**: Facial recognition can be integrated with other authentication methods like one-time passwords (OTP) or security tokens to create a multi-factor authentication (MFA) system. This adds an extra layer of security by requiring not only the correct face but also another verification factor.
- **Streamlined Access Control:** Facial recognition can be combined with role-based access control (RBAC) systems. This allows administrators to define specific access levels for different users based on their facial recognition and assigned roles within the system.

Functional Requirements

1- User Management

- **Registration:**

Users should have a secure registration process. This involves creating an account using his name, an email address and a password. During registration, the system should capture a facial image via the user's webcam. A secure facial recognition engine should then extract unique features from the captured image. These features are converted into a secure representation (hash) to protect user privacy before being stored in the system's database.

- **Login:**

The login process allows users to attempt login using their registered credentials, such as their email address and password. Once the credentials are verified, the system should activate the user's webcam and capture a live facial image. The facial recognition engine then extracts features from this live image and compares them with the user's stored hashed facial data.

Functional Requirements

2- Facial Recognition Engine

This is the core component responsible for verifying user identity through facial recognition. It consists of several functionalities:

- **Facial Detection:** The engine should accurately identify a user's face within the captured image or live webcam feed.
- **Feature Extraction:** Once a face is detected, the engine extracts unique features from the image, like the distance between the eyes or the shape of the jawline. These features form a mathematical representation of the user's face.
- **Matching:** The extracted features from the live image are then compared with the user's stored facial data (hashed representation). The engine determines a match if the features are sufficiently similar, indicating the user is likely the authorized person.
- **Security:** The facial recognition engine should be robust against spoofing attempts. This means it should be able to differentiate between a real user's face and attempts to bypass the system using photos or videos.

Functional Requirements

3- Access Control

- Upon **successful facial recognition**, the system should **grant access** to the user based on their assigned permissions. This ensures only authorized users can access the protected platform.
- Additionally, the system can integrate with **Role-Based Access Control (RBAC)**. RBAC allows administrators to define different user roles within the system (e.g., standard user, administrator). Users are assigned roles, and each role has specific permissions associated with it. This way, RBAC restricts functionalities within the platform based on a user's role. For instance, a standard user might only have access to view information, while an administrator might have additional functionalities for managing user accounts or system settings.
- **Error Handling:** The system should provide clear and informative error messages in case of login failures. This could include messages for incorrect credentials, failed facial recognition attempts, or other unexpected issues.
- **Session Management:** To maintain security, the system should enforce session timeouts. This means that if a user is inactive for a certain period, they will be automatically logged out of the system.

Functional Requirements

4- Security Considerations

- **Secure Storage:** User credentials and hashed facial data should be stored securely using strong hashing techniques. Hashing transforms data into a unique string that cannot be easily reversed back to the original data. This protects user privacy even if the system database were to be compromised.
- **Data Transmission:** Secure communication protocols like HTTPS should be used to encrypt all data transmission during registration, login, and facial recognition processing. HTTPS ensures that data exchanged between the user's device and the system remains confidential and cannot be intercepted by unauthorized parties.
- **Regular Security Audits:** Regularly conducting security audits helps identify and address any vulnerabilities within the system. This proactive approach helps maintain a strong security posture and minimize the risk of security breaches.

Non-Functional Requirements

- **Performance:**

The system should deliver fast response times for user registration, login attempts, and facial recognition processing. Delays can create frustration and hinder user experience. The system should be able to handle a reasonable number of concurrent users without significant performance degradation.

- **Availability:**

The system should be highly available, meaning users can access it and perform logins most of the time. Downtime due to technical issues should be minimized to ensure reliable access for authorized users.

- **Usability:**

The user interface (UI) for registration, login, and any error messages should be user-friendly and intuitive. Users wil be able to navigate the system easily and understand the steps involved in registration and login.

Non-Functional Requirements

- **Security:**

Security is a top priority. Beyond the secure storage and transmission of data mentioned in the functional requirements, our solution will consider additional security measures:

- Implement strong password policies if user credentials are part of the login process.
- Regularly update the facial recognition engine to address potential vulnerabilities and improve accuracy.
- Offer two-factor authentication (2FA) as an optional layer of security, requiring users to provide an additional verification code besides facial recognition.

- **Privacy:**

User privacy is a major concern with facial recognition technology. The system will:

- Obtain explicit user consent for collecting and storing facial data.
- Provide clear and transparent privacy policies outlining how facial data is collected, used, and stored.
- Allow users to request deletion of their facial data upon account termination.

User Flow

- **Registration:**

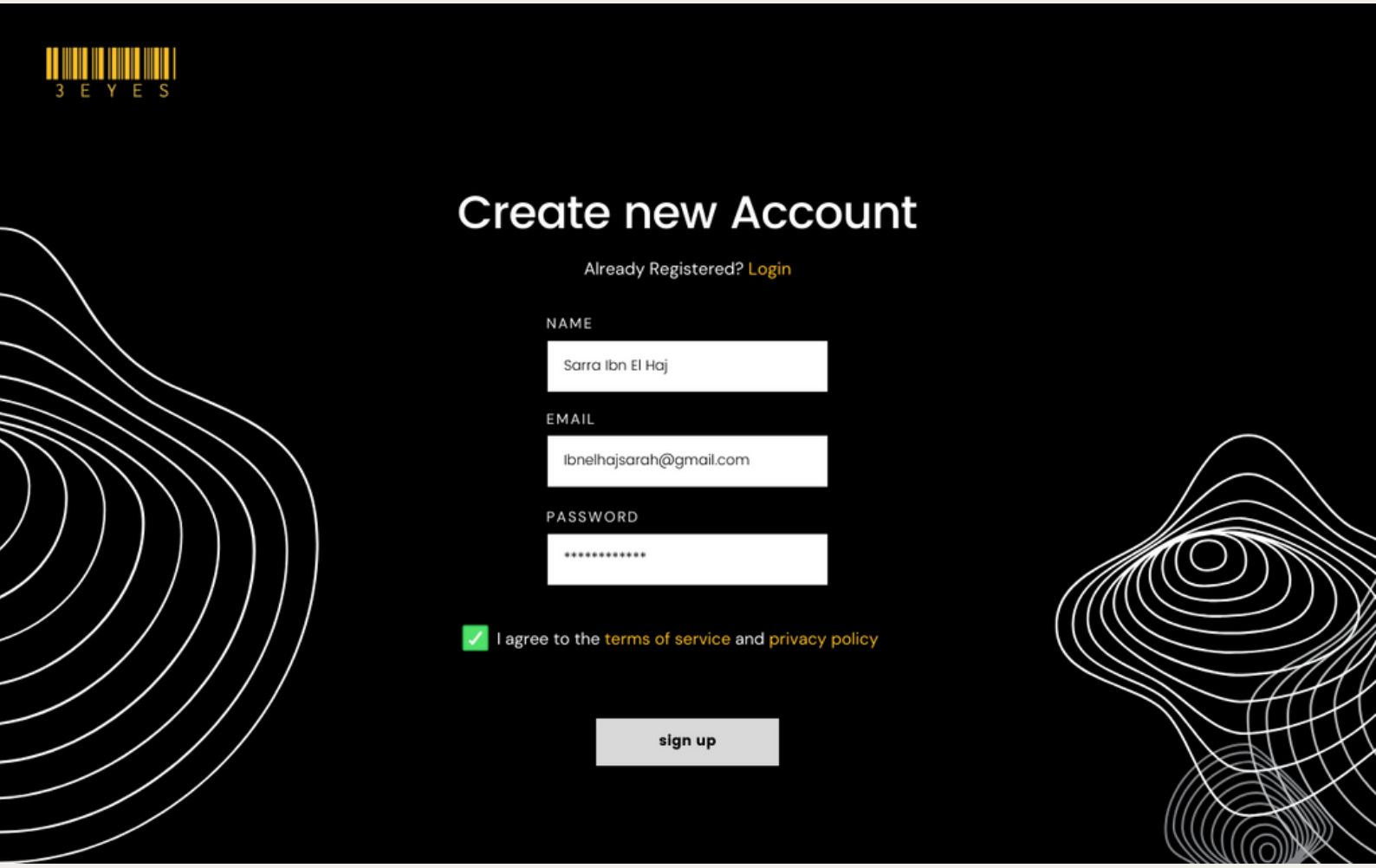
1. The user visits the **registration page**.
2. The user enters their **email address** and **name** in the provided field.
3. The user creates a strong **password** by following the system's password policy guidelines (e.g., minimum length, character complexity).
4. The user clicks the "**Sign-up**" button, activating their webcam.
5. The UI displays on-screen instructions guiding the user on proper facial positioning for capturing a clear image.
6. The user adjusts their position based on the instructions and clicks the "**Capture Image**" button.
7. The system captures the user's facial image through the webcam and sends it to the facial recognition engine.
8. The engine **extracts facial features** and creates a **secure hash** for storage.
9. The user's data (email/identifier, hashed password, and hashed facial data) is securely stored in the user management system.
10. Upon successful registration, the user receives a confirmation message (potentially sent to the registered email address) and might be redirected to a login page.

User Flow

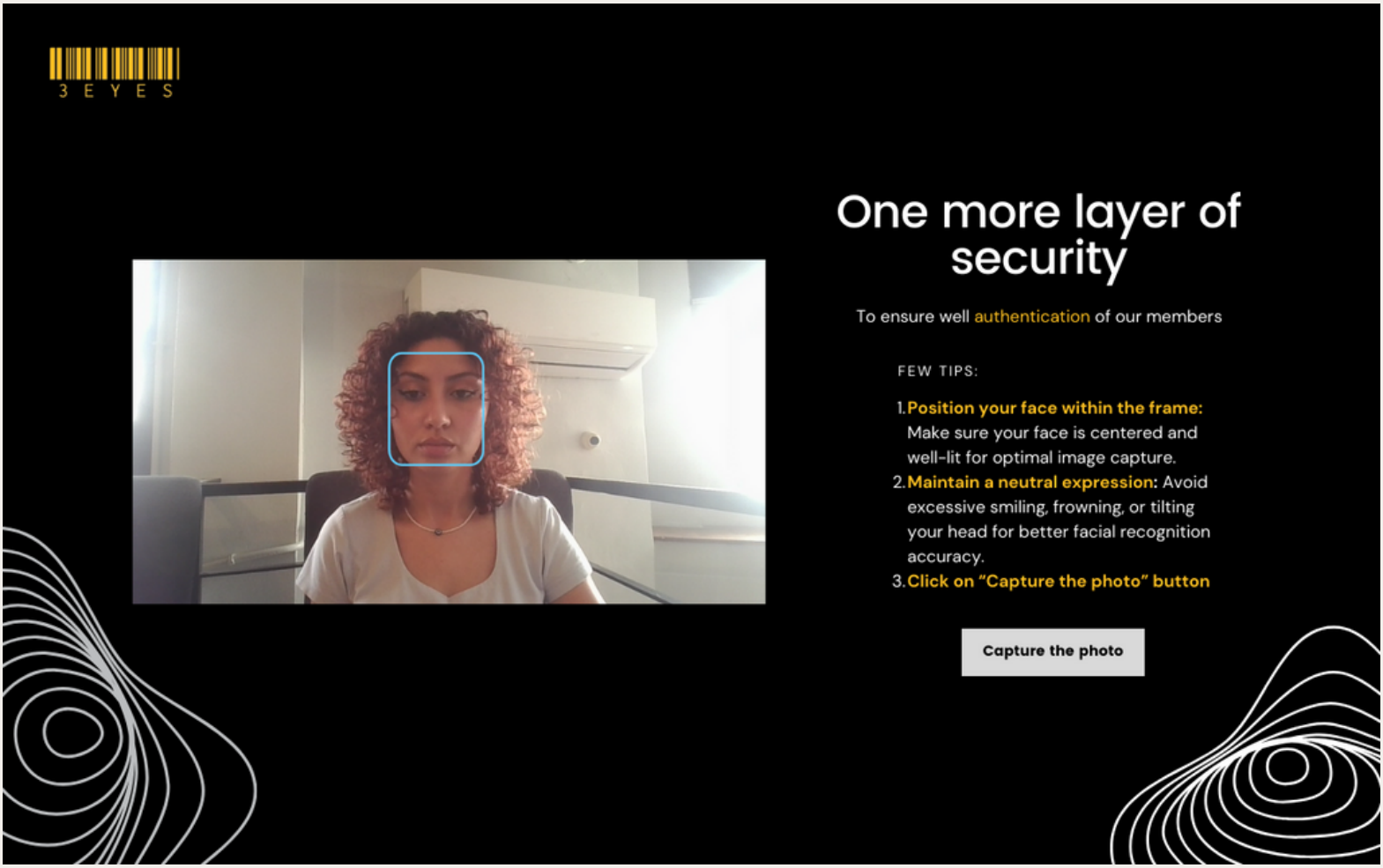
- **Login:**

1. The user visits the **login page**.
2. The user enters their **registered email address** in the designated field.
3. The user enters their **password** in the password field.
4. The user clicks the "**Login**" button, activating their webcam.
5. The system **captures a live image** of the user's face through the webcam.
6. The live image is sent to the facial recognition engine.
7. The engine extracts features from the live image and **compares** them with the user's stored hashed facial data.
8. If there's a **successful facial recognition match**, the system then verifies the entered password against the user's stored hashed password.
9. If both the facial recognition and password verification are successful, the access control system verifies the user's permissions.
10. Based on permissions (or RBAC roles), the user is granted access to the protected platform or receives an error message if unauthorized.

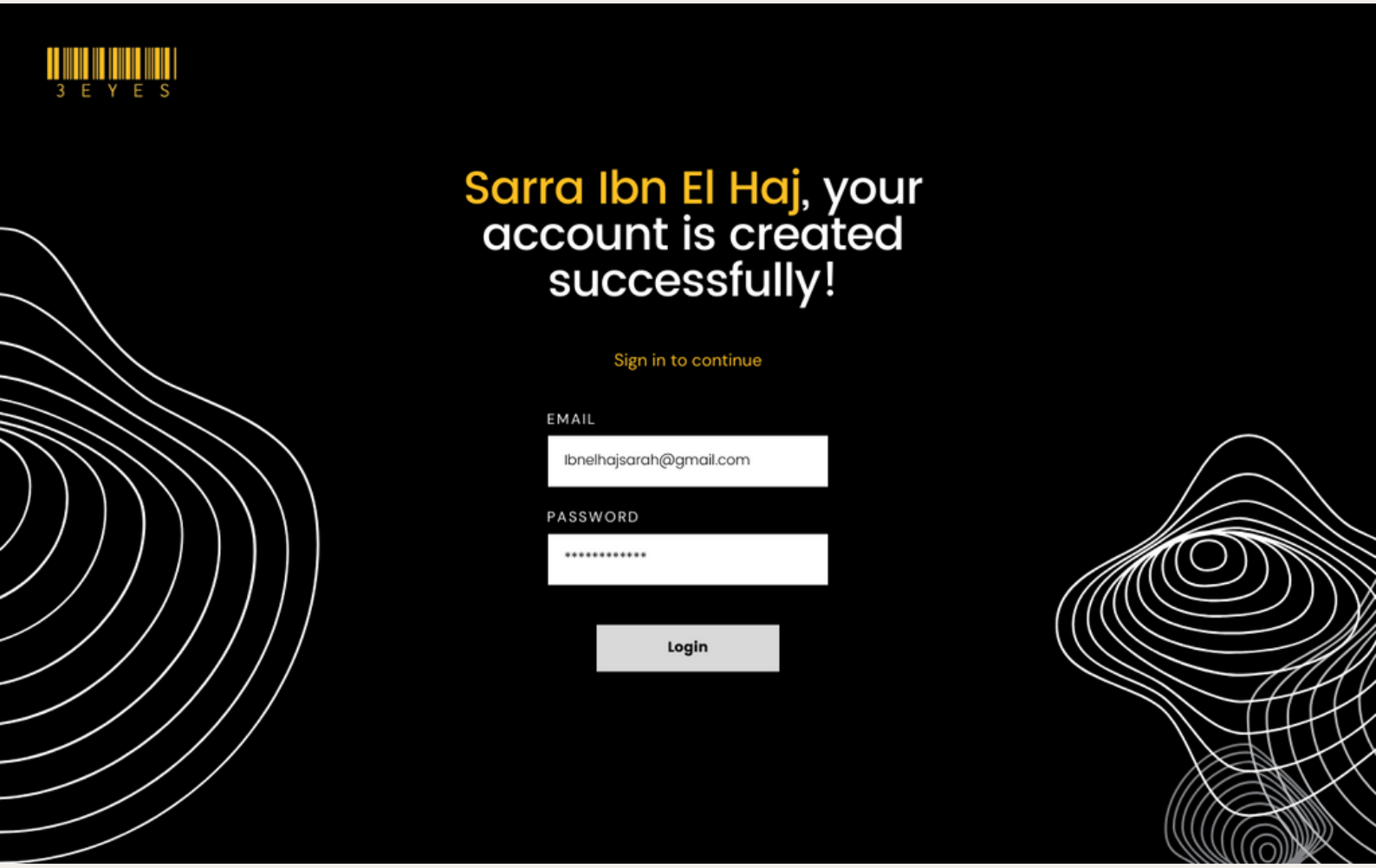
UI Prototype



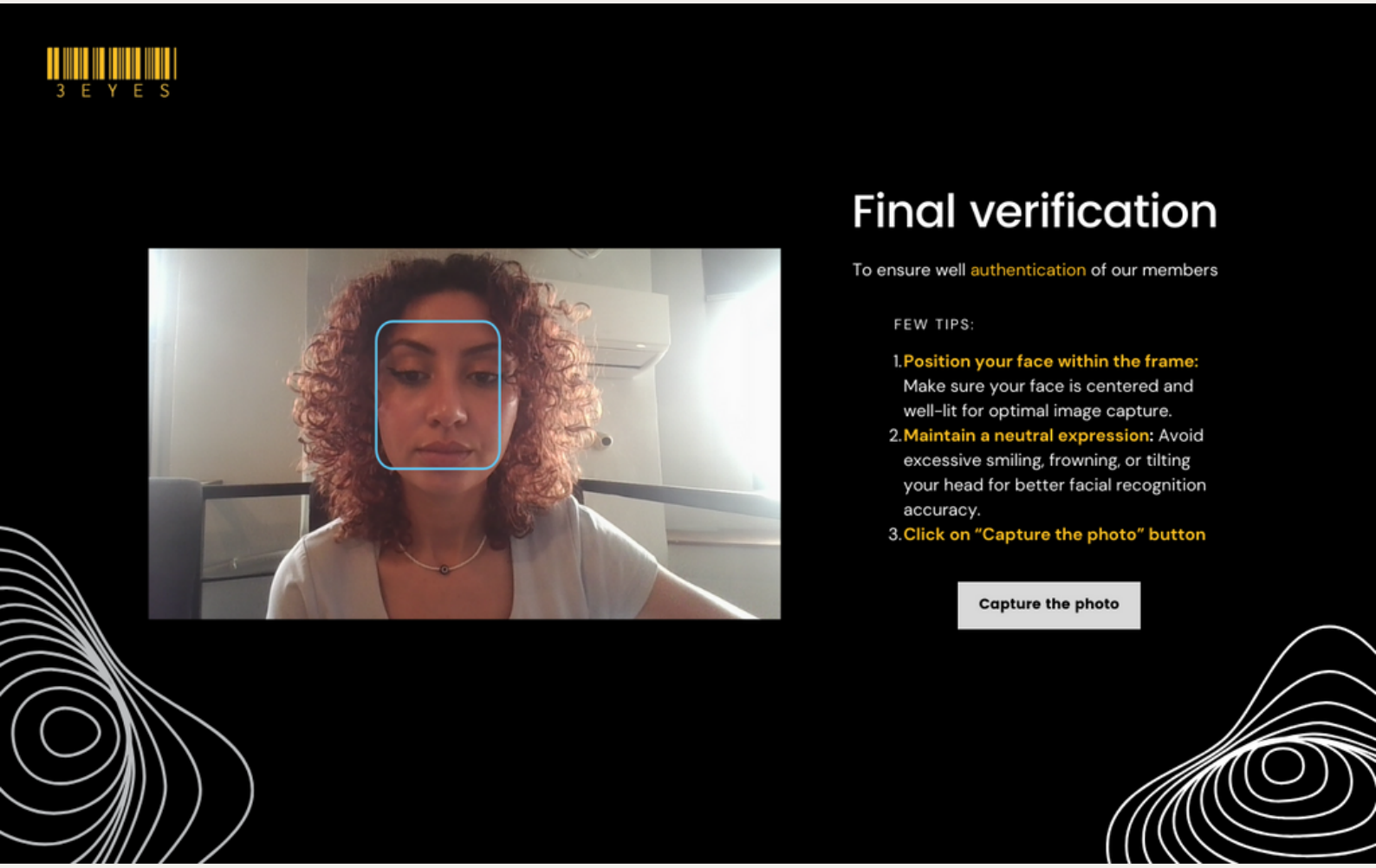
UI Prototype



UI Prototype



UI Prototype



UI Prototype

Methodology

1- Building the User Interface (UI):

- Leveraging a Python web framework like **Flask** to create a user-friendly interface for registration, login, and handling any error messages that might arise during the process.
- Design the UI elements to be clear and intuitive, guiding users through the registration and login procedures.
- During registration, the UI should allow users to enter their credentials and capture their facial image directly through their webcam. Libraries like **websockets** can facilitate real-time communication between the UI and the backend for a seamless user experience.
- Similarly, the login UI should have designated fields for credentials and a button to initiate the facial recognition process.

Methodology

2- Developing the Secure User Management System (Backend):

- Implementing the backend functionalities using Python. This involves user registration, login attempts, and secure storage of user data.
- For user credentials , employing robust hashing techniques like **bcrypt** or **argon2** to protect passwords from unauthorized access. These functions store passwords in a secure, irreversible format.
- Implementing secure storage methods for hashed facial data representations. This involves creating a separate database table where facial data is stored after being processed with a one-way hashing function like **SHA-256**.
- Enforcing strong password policies during registration to encourage users to create complex and difficult-to-guess passwords, further enhancing security.

Methodology

3- Integrating the Facial Recognition Engine:

- We'll utilize **OpenCV**, a powerful Python library for computer vision tasks, to handle the facial recognition aspects of the system.
- Within your backend code, we will leverage OpenCV's capabilities to:
  - Capture live video frames from the user's webcam during registration and login (using **cv2.VideoCapture)**.
  - Preprocess the captured images (e.g., convert to grayscale, reduce noise) for improved facial recognition accuracy.
  - Detects faces within the images using pre-trained face detection models provided by **OpenCV's Haar cascade classifiers**.
  - Extract facial features (e.g., eye positions, nose bridge) from the detected faces.
  - Compare the extracted features with stored hashed facial data representations of registered users using techniques like **Eigenfaces** or **Local Binary Patterns Histograms (LBPH)**.
  - Return a match/no-match result based on the comparison between the user's live image and the stored data.

Methodology

4- Implementing Access Control:

- Developing the access control system based on the pre-defined user permissions and access rules you've established.
- Upon successful facial recognition (and credential verification if applicable), the system needs to verify the user's permissions stored in the user management system.
- Grant access to the protected platform based on the user's verified permissions. If the user is unauthorized, display an appropriate error message.

Methodology

5- Security Measures and Testing:

- Implementing **HTTPS** throughout the system using libraries like **requests** to ensure secure communication between the UI and backend. This encrypts data transmission during registration, login, and facial recognition processing, protecting sensitive information.
- Following secure coding practices to minimize vulnerabilities in our Python code. Consider using static code analysis tools to identify potential security issues early on in the development process.
- Integrating security libraries like **SQLAlchemy** or **peewee** to protect against common database injection attacks when interacting with your user data storage.
- Conducting regular security audits to proactively identify and address potential vulnerabilities in the system. Utilizing security testing tools like **Bandit** or **Pylint** to scan our code for security weaknesses.
- Performing thorough testing of the system's functionalities, including facial recognition accuracy under various conditions, performance under load with multiple users, and security testing to identify and patch vulnerabilities.