Project Report

CSE 299 Junior Design

Submitted to

Nabeel Mohammed

Submitted By:

Chowdhury Nafis Faiyaz

Ayman ibne hakim

Tausif Jahangir

Section: 6

Date: 22-12-2022

Abstract:

With the emergence of time and the progression of usage of internet, data security has become a global phenomenon. Keeping our data safe in the online world is no longer a want but now it has become a need the people. New ways of penetrating into an individual's personal data are being discovered constantly. Hackers are also finding modern techniques of spoofing into others accounts hence the traditional 2-Factor authentications are also becoming vulnerable to the attacks. For that reason, keeping confidential information safely has become a threat to this tech-based civilization.

To overcome this problem the concept of encrypting data and facial recognition for logging in to an account can be a vital solution. Just imagine keeping a file somewhere in the cloud storage without worrying to be theft and logging into your account using your unique biometric features. All these features by implementing facial recognition algorithms to extract the biological landmarks of a user's face and use those to implement facial login. Along with that, "The Amazing App" uses AES 256 encryption algorithm to encrypt data or files and then decrypt the encrypted file and view them in the app

"The Amazing App", a web-based application which is used for encrypting and decrypting confidential files online is a implementation targeting to overcome the crisis. The app is designed in such a way so that a user is able to user their google account to sign up and use facial recognition system as a 2nd factor authentication to log into the app. After a successful login, the user can import files from their google drive, encrypt or decrypt them as when necessary.

Introduction

The initial goal of the “The Amazing App” web-app is to create a platform where the user can encrypt and decrypt a file using the app. The decrypted file can be viewed within the app using a file viewer, and the encrypted file can be stored to the users google drive. The user can also import files from their local disk use the app features.

The app provides a 2-Factor authentication where the first step is Google Oauth2.0 authentication and the second step is an image authentication. In google Oauth authentication, the user needs to sign up using their google account and after successful sign up, the user is redirected for image authentication. For image authentication and recognition, the app uses FACE IO API which is a robust service for face authentication service.

At first the user will sign up to the web app using Google OAuth services. Then as the user is enrolling to the ap for the first time, the Face-IO API will be called and a prompt will be given for image enrollment. The FACE-IO API automatically captures the users photograph and extracts the image matrix and saves it in their database, and then every time the user tries to log in to the app, it crosschecks the saved matrix with the current photograph. If it matches then the user is redirected to the account page form where they can use the encrypting and decrypting features. The image recognition is spoof proof to some extent. The accuracy of anti-spoofing feature can be enhanced if the paid version of the API is used. It works in different lighting conditions, different background, different user positions and face orientation, works with and without glasses, and sunglasses.

After face authentication is approved, the user is redirected to a page where the user can view a list of files, they have stored in their google drive and inside the amazing app they can upload to and download from their google drive. On the same page, the user can select a file and with a single button click, they can encrypt the selected file and upload it directly to the user's Google Drive. The Amazing App uses AES 256 encryption algorithm. Using the decrypt button, the user can decrypt the selected file. Once the file is decrypted, the user can click the View File button which opens up a modal window to view the decrypted file. After viewing, once the user closes the modal window, the decrypted file is deleted.

Document Structure

- Section A- System Features
    - Use Case Diagram
    - Expanded Use Cases
    - UI Design (Prototypes)

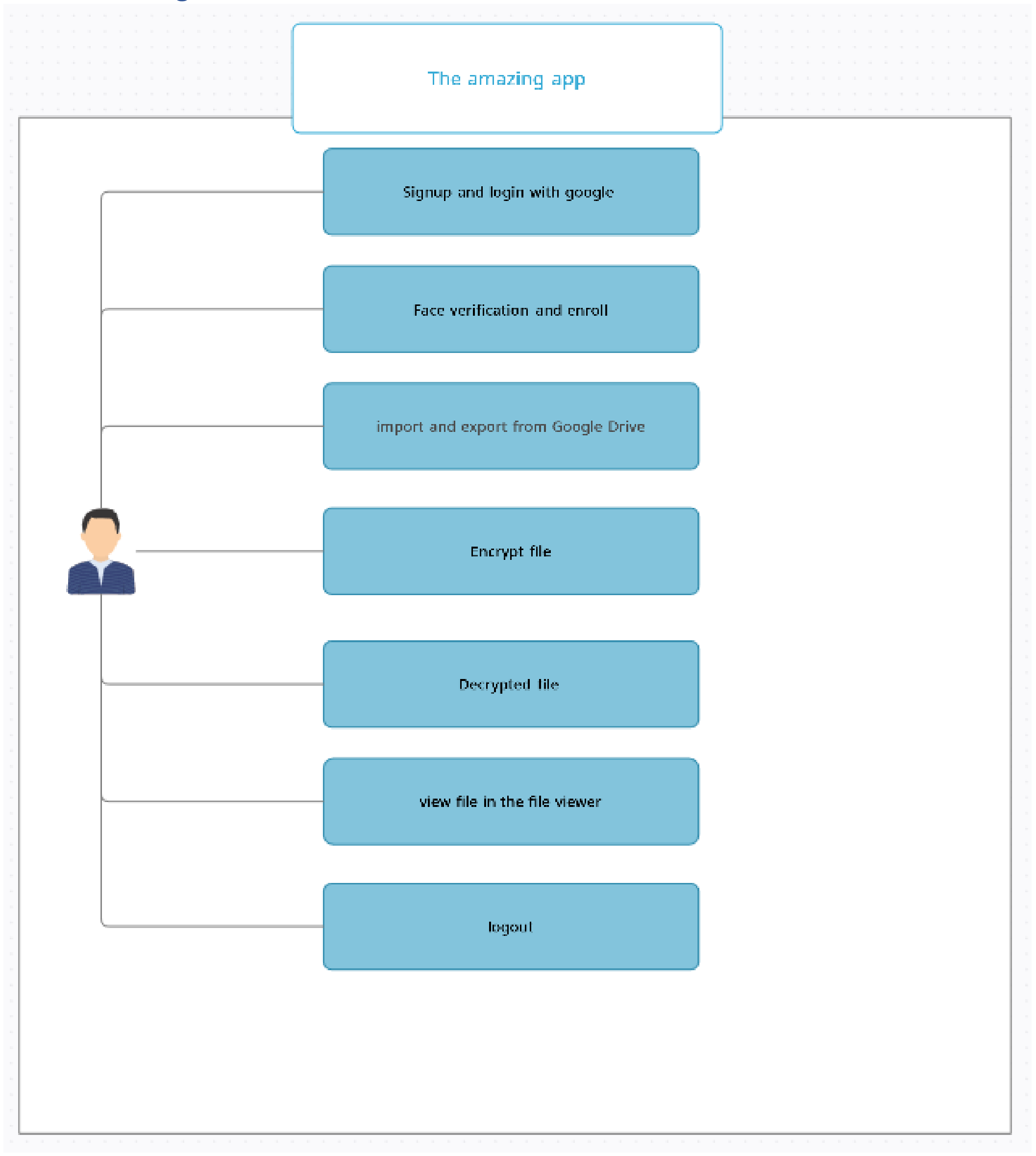
- Section B- System Design
    - Database Design (ER Diagram)
    - API Documentation (for any API/library used)
    - Data Flow Diagrams
    - System Design Diagrams

- Section C- System Evaluation
    - Relevant Metrics
    - Evaluation methods
    - Evaluation implementation details
    - Summary of evaluation

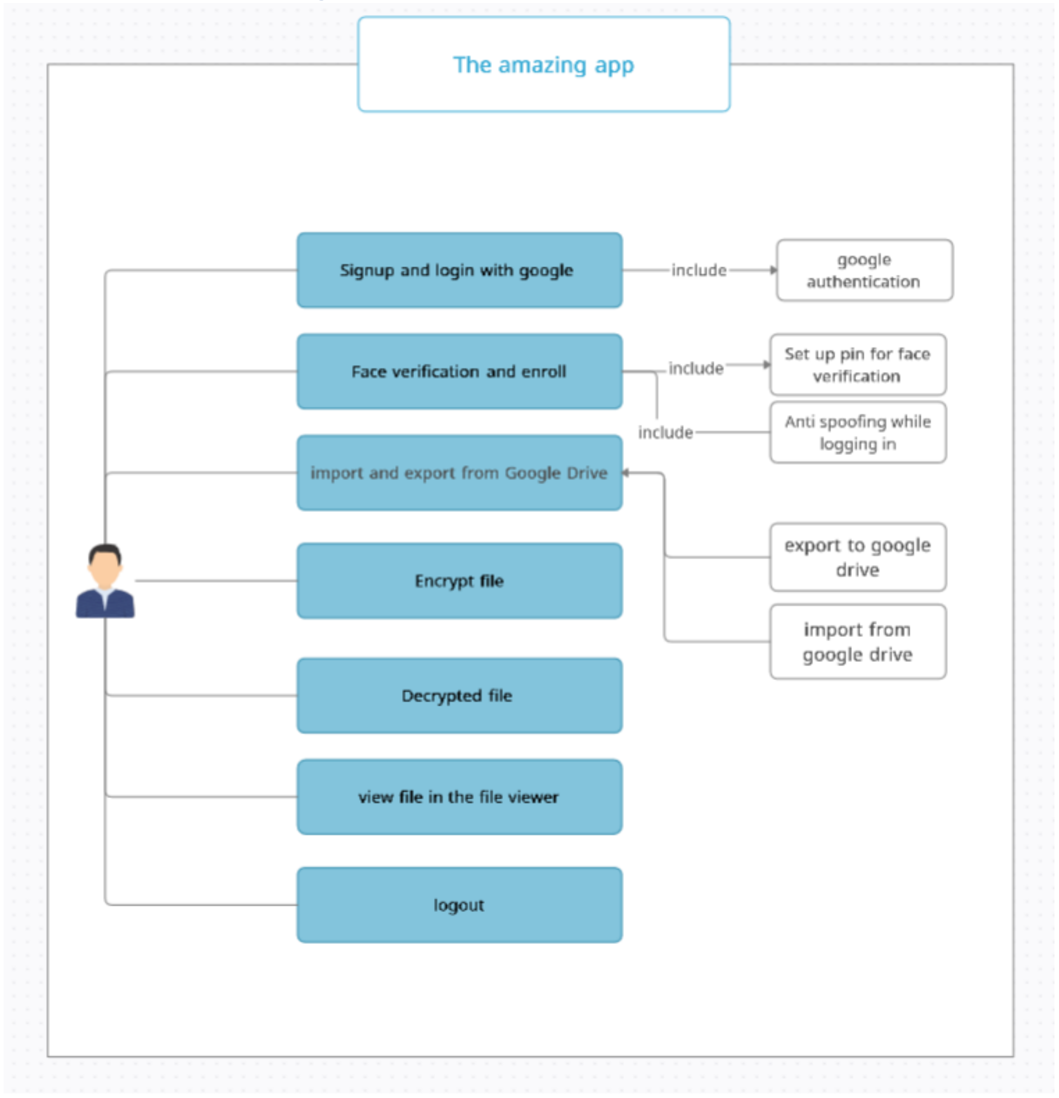
Section A

System features:

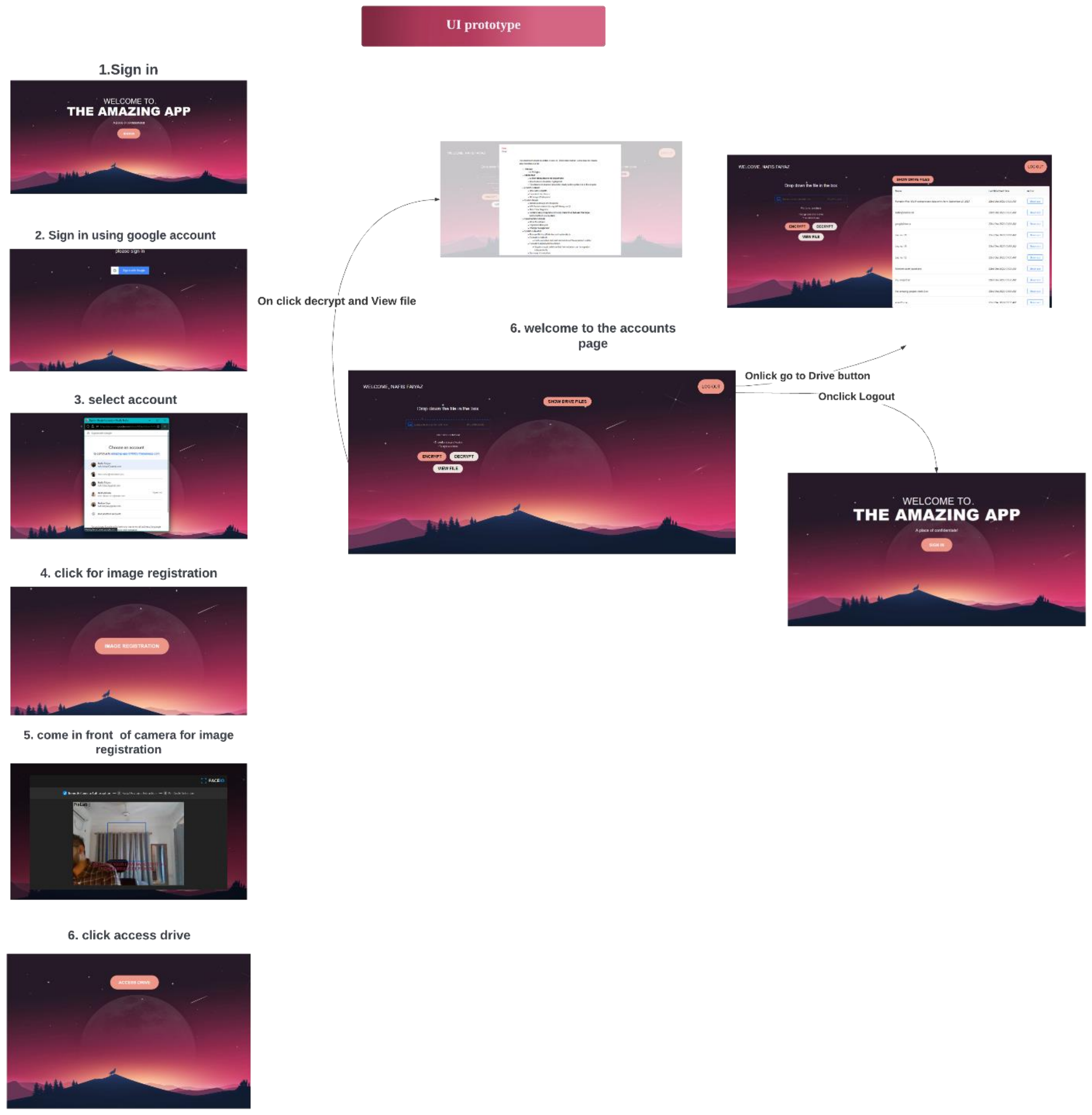
Use case Diagram:


The amazing app
Signup and login with google
Face verification and enroll
import and export from Google Drive
Encrypt file
Decrypted file
view file in the file viewer
logout

Extended Use case diagram:


The amazing app
Signup and login with google
include
google authentication
Face verification and enroll
include
Set up pin for face verification
include
Anti spoofing while logging in
import and export from Google Drive
export to google drive
import from google drive
Encrypt file
Decrypted file
view file in the file viewer
logout

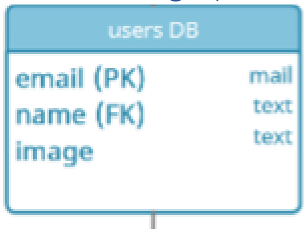
UI design prototype:



Section B

System Design:

Database Design (ER Diagram):



API Documentation (for any API/library used):

FACE-IO API for face authentication:

The FACEIO Widget is a simple and elegant interface to provide **secure facial authentication experience to your users** via simple calls to the enroll() & authenticate() methods. The Widget is powered by the fio.js JavaScript library, which is simple enough to integrate in a matter of minutes while being flexible enough to support highly customized setups. Once implemented on your website or web-based application, you'll be able to authenticate your existing users, enroll new ones securely, with maximum convenience on their favorite browser, and at real-time thanks to password less experience powered by face recognition.

fio.js works with regular webcams, and smartphones frontal camera on all modern browsers, **does not require biometric sensors**, and works seemingly with all websites and web applications regardless of the underlying front-end technology used (ie. React, Vue, jQuery, Vanilla Javascript, static HTML, etc.) or server-side language or framework (eg. PHP, Python, Node.js, Rust, Elixir, etc.).

It's super quick to get FACEIO Up & Running with just few lines of code. FaceIO provides a very simple interface to register new users. We use **enroll** function to register a new user. As FaceIO needs your webcam permission, do accept the permission dialogue when prompted.

The enroll function has 2 aliases. You can use register and record functions as drop-in replacements for enroll.

When you run this function in your browser, it launches faceIO widgets. At first, it takes user consent to scan their face, then prompt to accept the webcam permission. If all requirements are satisfied, the faceIO widget opens your camera and scans your face. It converts your face data into an array of floating point numbers.

After collecting the facial data, faceIO is prompted to enter a PIN code. This PIN code is used to distinguish users with extremely similar faces. You think of this as a two-factor auth when something goes south.

After registering the user successfully, this is the time to authenticate registered users. For this purpose, Pixlab provides the authenticate function. This authenticates function also has 3 aliases, auth, recognize and identify.

The authentication function needs only a single frame for the user to recognize. Therefore, it is very bandwidth friendly. After successful authentication, it returns a userData object. This userData contains the payload you have specifies in the registration and the face of the user.

Inside our App.jsx file, we make another button called **Log-in**. When a user clicks on this button, it invokes the handleLogIn helper function. This function is ultimately responsible to invoke the "authenticate" function. Let's see all of these in code.

Inside the handleLogIn function, we use the try-catch block to catch errors if authentication fails. As the authenticate function returns a promise, we have used async await methods to get the value when the promise completes.

The authentication function accepts some optional parameters to customize the faceIO authentication widget. You can mainly customize the timeouts using permissions timeout, idleTimeout and replyTimeout parameters. After the compilation, our web application should look like this.

Link to the complete documentation:

https://betterprogramming.pub/replace-your-auth-system-with-facial-recognition-using-reactjs-and-tailwindcss-9af4898ab5a2

https://faceio.net/integration-guide

Firebase google authentication:

Let users authenticate with firebase using their Google Accounts. Before beginning the process of authentication, we have to add firebase to our existing project. Make a firebase account and enable google as a sign-in method in the firebase console.

Handle the sign-in flow with the Firebase SDK

For web apps, the easiest way to authenticate your users with Firebase using their Google Accounts is to handle the sign-in flow with the Firebase JavaScript SDK. (If you want to authenticate a user in Node.js or other non-browser environment, you must handle the sign-in flow manually.)

To handle the sign-in flow with the Firebase JavaScript SDK, follow these steps:

1. Create an instance of the Google provider object:
2. Specify additional OAuth 2.0 scopes that you want to request from the authentication provider. To add a scope, call addScope.
3. To localize the provider's OAuth flow to the user's preferred language without explicitly passing the relevant custom OAuth parameters, update the language code on the Auth instance before starting the OAuth flow.
4. Specify additional custom OAuth provider parameters that you want to send with the OAuth request. To add a custom parameter, call setCustomParameters on the initialized provider with an object containing the key as specified by the OAuth provider documentation and the corresponding value.
5. Authenticate with Firebase using the Google provider object. You can prompt your users to sign in with their Google Accounts either by opening a pop-up window or by redirecting to the sign-in page.

Link for documentation:

https://firebase.google.com/docs/auth/web/google-signin

Link to firebase:

https://firebase.google.com/

Google drive API

Google Drive provides a cloud-based storage solution for Google Workspace files and other user data. Managing data in Drive can be a time-consuming task.

Download file:

To download a file stored on Google Drive, use the files.get method with the ID of the file to download and the `alt=media` URL parameter. The `alt=media` URL parameter tells the server that a download of content is being requested.

Link to download from drive documentation:

https://developers.google.com/drive/api/guides/manage-downloads#download_a_file_stored_on_google_drive

Upload File

The Drive API lets you upload file data when you create or update a `File`. For information about how to create a metadata-only `File`, refer to Create files.

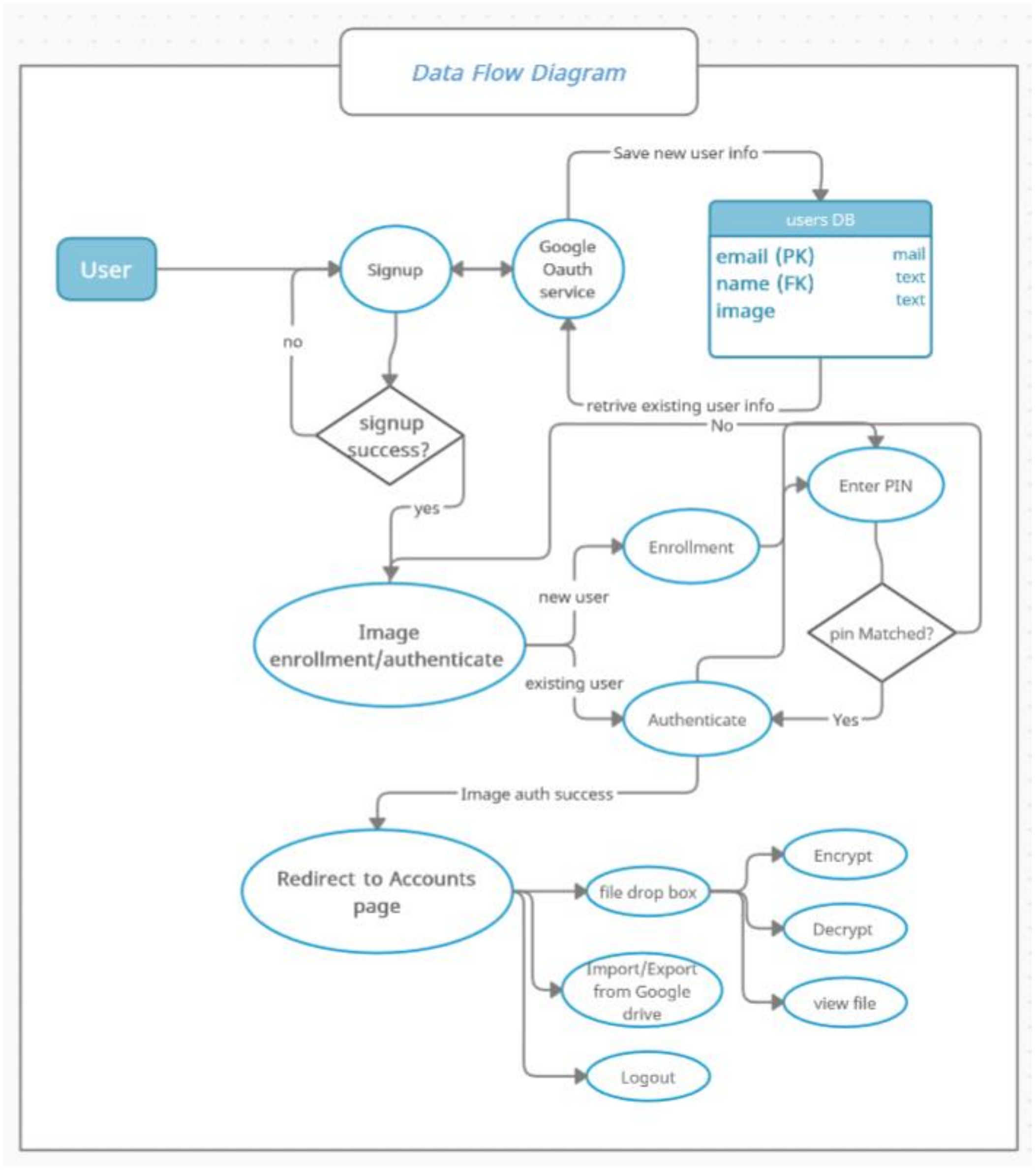
There are 3 types of uploads you can perform:

- **Simple upload (`uploadType=media`)**—Use this upload type to transfer a small media file (5 MB or less) without supplying metadata. To perform a simple upload, refer to Perform a simple upload.
- **Multipart upload (`uploadType=multipart`)**—Use this upload type to transfer a small file (5 MB or less) along with metadata that describes the file, in a single request. To perform a multipart upload, refer to Perform a multipart upload.
- **Resumable upload (`uploadType=resumable`)**—Use this upload type for large files (greater than 5 MB) and when there's a high chance of network interruption, such as when creating a file from a mobile app. Resumable uploads are also a good choice for most applications because they also work for small files at a minimal cost of one additional HTTP request per upload. To perform a resumable upload, refer to Perform a resumable upload.

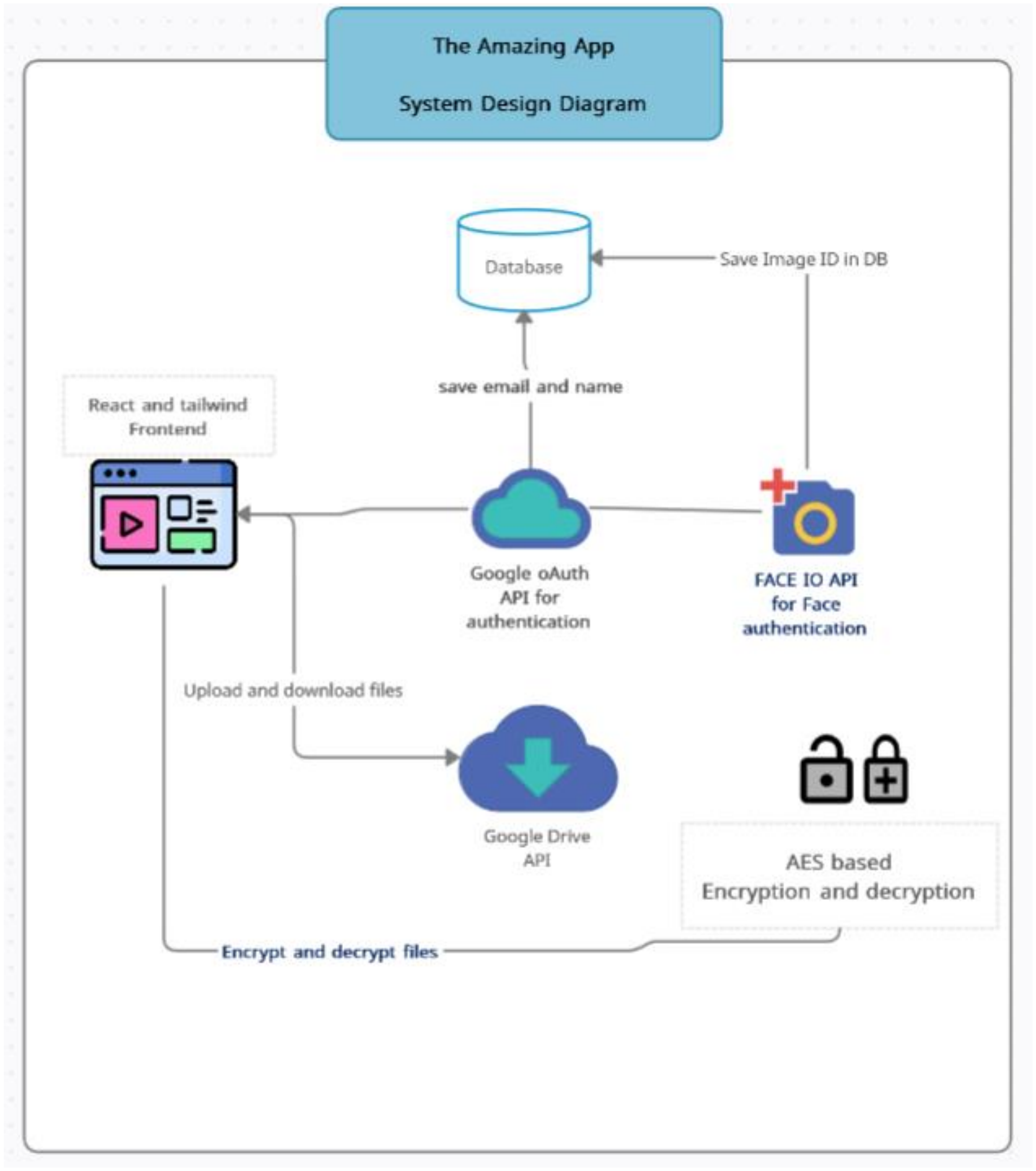
Link to upload from drive documentation:

`https://developers.google.com/drive/api/guides/manage-uploads#simple`

Data Flow Diagrams:



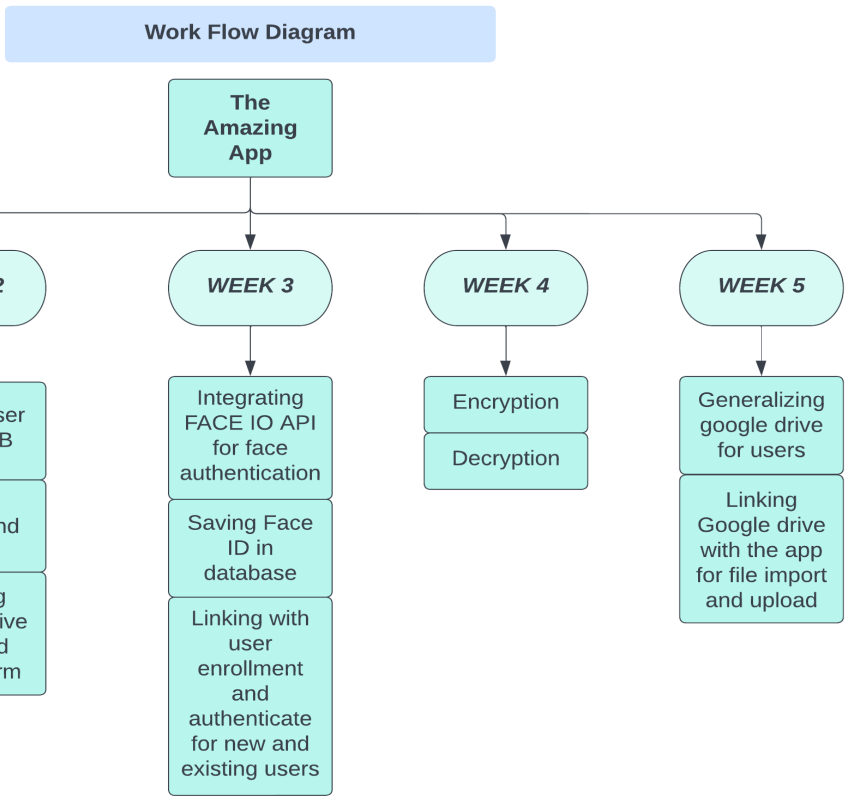
System Design Diagrams:


The Amazing App
System Design Diagram
Database
Save Image ID in DB
save email and name
React and tailwind Frontend
Google oAuth API for authentication
FACE IO API for Face authentication
Upload and download files
Google Drive API
AES based Encryption and decryption
Encrypt and decrypt files

Section C

Implementation Details:

Work Breakdown:


Work Flow Diagram
The Amazing App
WEEK 1
WEEK2
WEEK 3
WEEK 4
WEEK 5
Front end UI
Google OAuth Initialization
Creation of Database
Storing user info in DB
Google signup and login
Initiating google Drive view and import form
Integrating FACE IO API for face authentication
Saving Face ID in database
Linking with user enrollment and authenticate for new and existing users
Encryption
Decryption
Generalizing google drive for users
Linking Google drive with the app for file import and upload

Implementation plan:

	Front end	Back end
Week 1	• Build up the full UI of the web app using react and tailwind CSS • Initialize google Oauth sign in • Plan about the process flow of the system	• Create Database
Week 2	• Integrate Google Oauth2.0 API sign up and log in • Initiate the Google Drive into the system	• Store information in database upon user enrollment and login
Week 3	• Add face authentication into the system while log in and enrollment. • Test face authentication on different users • Link up with the web app and process flow of the system	• Store face id in the database • Check for new users and old users and decide whether to enroll or authenticate.
Week 4	• Initiate encryption and decryption on different files locally	• Store the encrypted and decrypted files in the local system
Week 5	• Integrate google drive file view, upload and download files with the web app • Decrypt the file and view with in the file viewer	

System Evaluation:

Evaluation Method:

Google drive authentication

For google sign in, the user has to click the sign in button and then it will redirect a pop up from where the user have to select/ enter the credentials of their account

Face authentication

To evaluate the app, we tested on 20 different users. For face authentication we tried to test the authentication in different backgrounds, different lightning conditions, different orientation, with several different facial changes such as with or without glasses, with and without mask, away and near camera, different hairstyles, wearing hijab or sunglasses.

Link to the test results:
https://drive.google.com/file/d/1f0nJcqQgaCR0tFlxNQepJksuQ1Fz2ANb/view?usp=share_link

Face spoof detection:

5 different cases has been tested for wrong results. The video link of the test is given below.

Link to the video:

https://drive.google.com/file/d/1MDQDSccRLccu5wEO7CRJmftkr17i4PkN/view?usp=sharing

Encryption and decryption

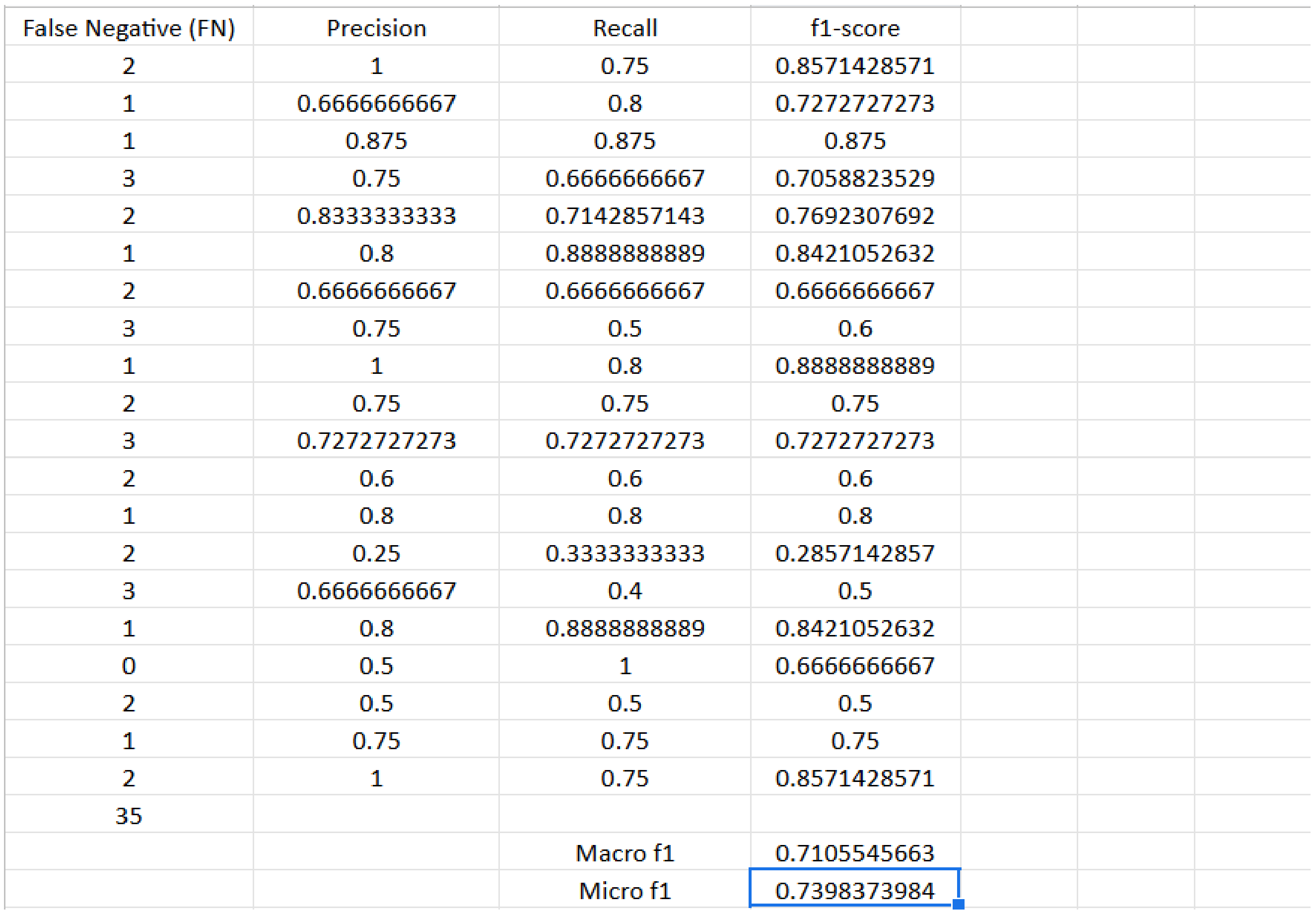
For the case of encryption and decryption the users are able to download the files from google drive and then they have to upload the file in the drop box. After that if the file is encrypted then the user has to decrypt the file. if the file is decrypted and the user wants to encrypt it then the user has to click the encrypt button. This button will encrypt the file and directly upload it to google drive.

Link to the walk through video of the full project:

https://drive.google.com/file/d/1lb-0RHSKCeFzqvaVEyyxzy9-YvllDwRq/view?usp=drivesdk

Precision, Recall, Micro f1, Macro f1 scores for face recognition API:

For our API we tested out 20 users. From our tested users we found out the number of cases which gave us a true positive (TP), false positive (FP) and a false negative (FN). We then made sure we find out the precision for each user using the data we obtained. We then got the recall value and the f1-scores for each as well. We then averaged the f1-score to get the macro f1 score. We also calculated the micro f1 score. We assumed that since micro f1 computes the proportion of correctly classified observations out of all of them it is the similar to the definition of accuracy. So, we gave the accuracy the same.



False Negative (FN)	Precision	Recall	f1-score
2	1	0.75	0.8571428571
1	0.6666666667	0.8	0.7272727273
1	0.875	0.875	0.875
3	0.75	0.6666666667	0.7058823529
2	0.8333333333	0.7142857143	0.7692307692
1	0.8	0.8888888889	0.8421052632
2	0.6666666667	0.6666666667	0.6666666667
3	0.75	0.5	0.6
1	1	0.8	0.8888888889
2	0.75	0.75	0.75
3	0.7272727273	0.7272727273	0.7272727273
2	0.6	0.6	0.6
1	0.8	0.8	0.8
2	0.25	0.3333333333	0.2857142857
3	0.6666666667	0.4	0.5
1	0.8	0.8888888889	0.8421052632
0	0.5	1	0.6666666667
2	0.5	0.5	0.5
1	0.75	0.75	0.75
2	1	0.75	0.8571428571
35			
		Macro f1	0.7105545663
		Micro f1	0.7398373984

Conclusion:

“The amazing app” is a web-based platform that uses the users google drive and helps encrypt and decrypt their files using AES encryption and decryption. It uses a 2-factor authentication system google authentication and face authentication.

Overall, the app performs almost every use case described above and meets all user requirements. But the app can still be integrated into multiple versions too suite the needs of different user levels.

Appendix

Usability analysis

Use cases	Number of interactions								
	User 1			User 2			User 3		
	Trial 1	Trial 2	Trial 3	Trial 1	Trial 2	Trial 3	Trial 1	Trial 2	Trial 3
Sign up and Login with google	2	2	2	2	2	2	2	2	2
Face Authentication	0	1	1	0	1	1	0	1	1
Face enrolls	2	0	0	2	0	0	2	0	0
Import from drive	1	1	1	1	1	1	1	1	1
Encrypt and Export to drive	1	1	1	1	1	1	1	1	1
Decrypt file	1	1	1	1	1	1	1	1	1
View file	1	1	1	1	1	1	1	1	1
Logout	1	1	1	1	1	1	1	1	1
Total Number of clicks	9	8	8	9	8	8	9	8	8

	Evaluator 1	Evaluator 2	Evaluator 3
Name of evaluator			
Signature			
Comment			