Project Final Report for Team 4

Team Member: Cole Van Steinburg SN 83059345

1 Overview

Our project is called SafeMov, and is a video uploading and sending platform for one-to-one sender to receiver use cases with an emphasis on privacy and security. Our platform is generalized enough that it does not just target one type of user, but should work for any case where users need to send a video to another individual.

We include optional face blurring, as well as the ability to either record your own video on site or upload an existing video file. Once videos are uploaded, secure chats are created pertaining to the video so further information can be optionally exchanged between parties. Accounts are also optional for sending videos, although certain features like chats will not be available in guest account mode, and guests cannot view videos they sent at a later time (only the receiving party can view them as they have an account in this scenario).

Some potential use cases could be:

- A patient needs to send a video to a doctor for diagnosis reasons. They want their face blurred for privacy reasons, and may need to answer followup questions in the chat to clarify further info.
- Interviewee answering laid out questions for an interviewer at a company or school. Since the interviewee is probably only sending the one video and likely won't need to use the chat feature, they can use a guest account and avoid registering.
- Youtubers / editors could use it as a platform to send iterations of videos back and forth before finalizing.

All video and chat data is completely encrypted. Even in the case of a full database breach, attackers would not be able to decrypt and recover user videos or chats. Even the host of the tool with access to S3 contents and full database access could not recover user videos and chats. The backend is also a REST API, so power users could choose to make their own interface to best suit their needs.

A demo video can be found here:
https://drive.google.com/file/d/14YAEpgS5Y3KgGfijT8IeE3d8kp-AGHlp/view?usp=sharing

2 System Architecture

Tech Stack

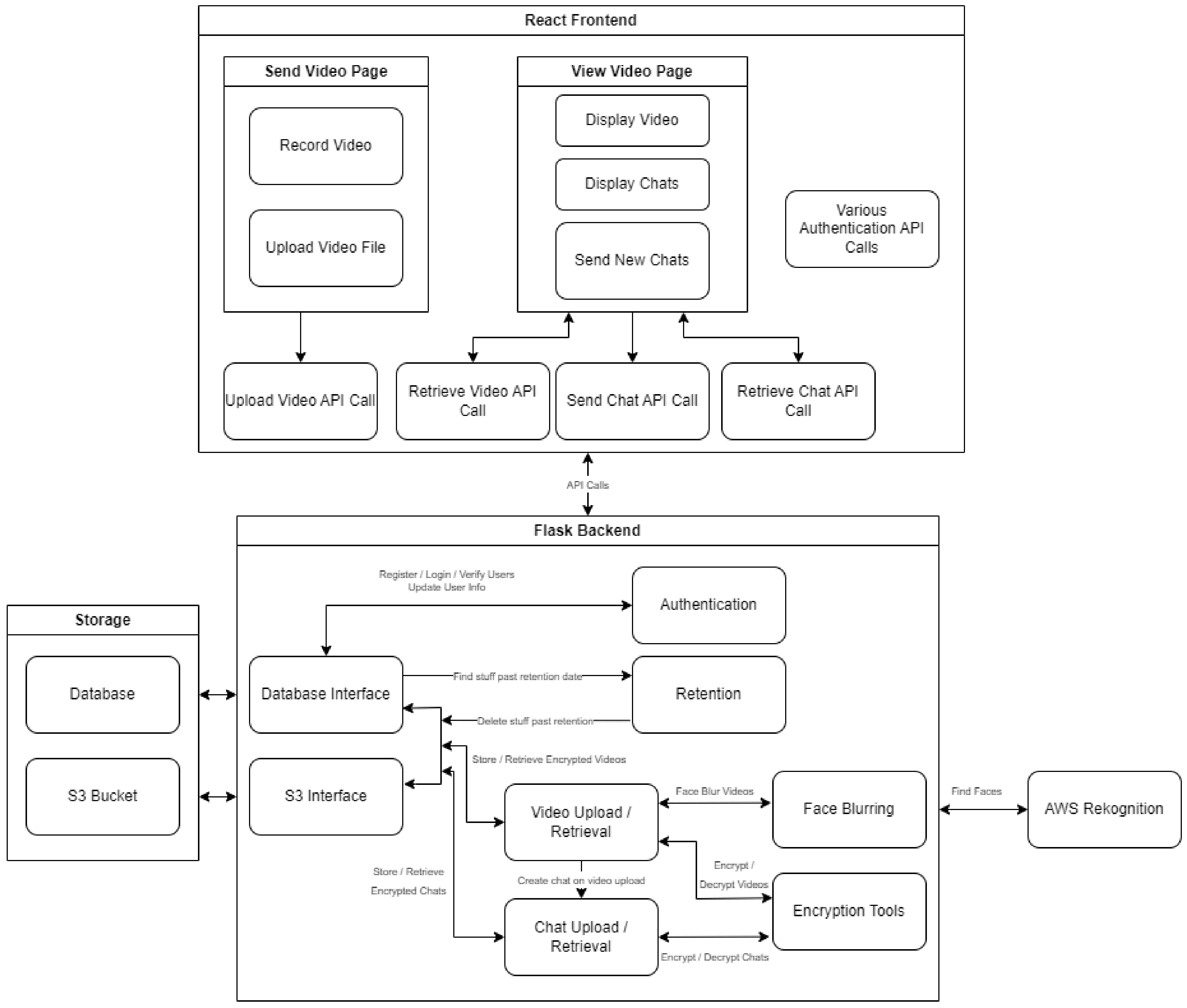
Frontend: *React / JS* - used because it is currently a widely used industry standard.
Backend: *Flask / Python* - used because of group members' experience with Python as well as Flask being used in various big projects like Netflix. I personally wanted to get experience with Flask itself as most of the backends I had been exposed to throughout school were Java.
Storage: *AWS S3* - S3 buckets were the obvious solution to video storage, as the project was to use AWS solutions and S3 makes it easy to store and retrieve large amounts of data.
Database: *AWS RDS / SQL* - another solution that seemed like the goto way to setup our database as it integrates easily from the EC2 instance we use for hosting and is part of AWS services.
Hosting: *AWS EC2* - running on an EC2 instance makes it easier to deploy and manage different services, and integrates well with our storage and database options.

Component Diagram

On the next page is a diagram showing all major components and how they work together.

The frontend shows the major features the user interacts with: sending and receiving of videos and chats, with a little bit of authentication.

On the backend, the major API entry points are *Video Upload / Retrieval, Chat Upload / Retrieval*, and *Authentication*. There are also some services that run alone on the backend and are not API calls directly. *Retention* is a service run at midnight every day by the backend to delete content that is past its retention date, *Face Blurring* is an option on the video upload requests that gets called on by the *Video Upload / Retrieval* service. *Encryption Tools* are used by other services to ensure all videos and chats are properly encrypted / decrypted using proper private and public keys. *Database Interface* and *S3 Interface* are tools for interacting with our various data storages with an easier to use level of abstraction.


React Frontend
Send Video Page
Record Video
Upload Video File
View Video Page
Display Video
Display Chats
Send New Chats
Various Authentication API Calls
Upload Video API Call
Retrieve Video API Call
Send Chat API Call
Retrieve Chat API Call
API Calls
Flask Backend
Register / Login / Verify Users
Update User Info
Authentication
Database Interface
Find stuff past retention date
Retention
Delete stuff past retention
Store / Retrieve Encrypted Videos
S3 Interface
Video Upload / Retrieval
Face Blur Videos
Face Blurring
Find Faces
AWS Rekognition
Encrypt / Decrypt Videos
Store / Retrieve Encrypted Chats
Create chat on video upload
Encryption Tools
Chat Upload / Retrieval
Encrypt / Decrypt Chats
Storage
Database
S3 Bucket

3 System Features

Note: All features with * are my features.

Backend Features

1. *User signup:
   API endpoint that takes in a name, email, and password. Does secure hashing on the password for security.
2. Email verification on signup:
   Users get an email with a 6 digit code upon signup to verify their account before they can login. Implemented through AWS SES.
3. Password restrictions on signup (different from 1 because different people did them):
   Add restrictions about min / max password length, as well as special characters, numbers, and capital letters.
4. *User login / logout:
   Endpoint to verify that account credentials are correct to sign in a user and returns a session cookie that can be used to authenticate different parts of the site. Also provided a logout endpoint to clear the session cookie.
5. *Change user details backend:
   Endpoint to change the user's name and email. Also updates all entries referring to the user anywhere else so the user doesn't lose access to any of their videos or chats.
6. *Public / private key creation / management:
   On user signup, an RSA public / private key pair will be generated. The private key is generated by combining the user's password with a random string of bytes and hashing them together. Upon login, the backend can reconstruct the private key for the user from their submitted password, and is secure because the plain password is never actually saved anywhere (only the password hash), but the plain password is needed to recreate the key. The public key is able to be derived from the private key, and can be safely stored in the database.
7. Change password when password not forgotten (user can login):
   Changes the user password, as well as updating the RSA public / private keypair. Will decrypt all AES keys for content the user has access to, then re-encrypt with the new RSA keys, allowing the user to not lose access to their content.

8. Change password when forgotten:
   Sends a verification code over email using AWS SES to allow the user to change their password. Since the old password is not known, the private keys are unrecoverable, and the old AES keys the user used to be able to access will be deleted.
9. *Encryption / decryption of AES keys:
   Set of utility functions for encrypting and decrypting AES keys. Can encrypt an AES key using RSA public keys from users. This encrypted AES key can later be decrypted using RSA private keys controlled by users who are supposed to have access to the given AES key.
10. *AES encryption / decryption for chats and videos:
    Set of utility functions for encrypting and decrypting videos using symmetrical AES encryption. Includes generation of AES keys as well as functions for encryption / decryption of arbitrary data.
11. *Video uploading backend:
    Endpoint that allows users to upload videos to other users. Will call all necessary encryption functions, as well as face blurring functions. Once processed and encrypted, stores references in the database and stores the file in S3.
12. *Video retrieval backend:
    Endpoint for requesting a video that you have the private key to decrypt. Decrypts and then returns the video file. Also provides an endpoint to find out which videos your account has access to.
13. *Video tags backend:
    Allows the user to upload tags with their video, which can be used by the frontend to sort / filter.
14. Email notifications:
    Uses AWS SES to send users an email when they get sent a new video.
15. Face blurring:
    If desired by a user, AWS Rekognition is used to detect faces in a video, then blurring effects can be applied to the faces. This is exposed as an endpoint that can be called before uploading the video itself.

16. Face blurring speed up / optimizations:

    Rekognition calls are parallelized for big speed improvements. Also avoided analyzing every frame of the video, as every few frames works just as well for blurring faces. Parallelized the blurring of faces once Rekognition calls were complete.
17. *Chat creation / linking to videos:

    Every time a video is uploaded, an associated, empty chat will be created allowing the involved parties to discuss the video. Stores the chat in S3.
18. *Chat functionality backend:

    Endpoints for sending and retrieving chats for a given video. Chats are fully encrypted the same way videos are.
19. Chat websocket integration:

    Added websocket functionality to chats so they can update in real time without refreshing the page to view new messages. Done using SocketIO. Both frontend and backend.
20. *Chat websocket optimizations (separate from 11 since I worked on this part):

    Fixed how websockets were being used, as they were making connections even when not viewing chats, and spamming a lot causing slowdowns for other requests.
21. Video / chat retention:

    Service to automatically delete videos and chats from the database and S3 bucket past a certain date. This date is set by the user upon video upload, and applies to both the video and associated chat.
22. Database interface tools:

    Big set of helper functions for doing various interactions with the database. Covers inserting users, videos, and chats, as well as various functions for querying and updating fields.
23. S3 interface tools:

    Set of helper functions for doing various tasks with the S3 bucket. Allows inserting and retrieving of files. Also includes some tools for operations that affect both the S3 and database to ensure we don’t get desync problems between the two.

Frontend Features

1. Live video recording:
   Browser based recorder for recording videos off the user's webcam. Includes controls like start recording, stop recording, retake video.
2. *Video recording frontend UI improvement / bug fixes (different person from 1):
   Merged the functionality of the "Stop recording" and "Preview video" buttons by adding event listeners on the async events emitted shortly after finishing recording. Previously the user had to manually choose to preview their video to display it. Also fixed bugs where re-taking videos was just uploading the first recorded video instead of the most recent.
3. Upload video frontend:
   Sends a video file from the frontend to the backend API endpoint. Can either be the recorded video, or a video file from the user's computer.
4. Face blurring frontend:
   Sends a video file to the backend API endpoint to blur faces.
5. Video receiving frontend:
   Calls the backend API endpoint to retrieve a video file. It will then play the video file in the browser.
6. Chat frontend:
   Retrieves the chats for a given video from the backend API endpoint. Displays the chats beside the appropriate video. Includes a text entry box for sending new chats, and displays info about the messages like sender and timestamp.
7. Email verification frontend:
   Popup after signing up allowing the user to verify their email with the sent verification code.
8. Side navbar:
   Navigation bar on the side for getting to various pages.
9. Video sorting / filtering:
   Search bar that allows users to filter available videos by names, sender, tags.
10. Frontend tag entry system:
    Allows users to add zero to many tags when uploading a video.

11. Update user info frontend:
    Page for calling the backend API to update user info.
12. Mobile view:
    Pages should adjust layout for viewing on mobile devices.
13. Signup page:
    Page for making a new account, includes checks on password restrictions. Calls backend API endpoint with proposed account info.
14. Login page:
    Page for logging in. Calls backend API endpoint with proposed login info.
15. Password change frontend:
    Page for changing passwords. Includes both the “forgot password” and “update password while logged in” versions depending on if the user has access to the account or not. Calls the API endpoint on the backend with proposed info.
16. Bootstrap config / design:
    Used bootstrap to get a consistent feel across different pages.
17. CSS for specific pages:
    Used some CSS for the receive / send page as well as the account page for styling.

Infrastructure Features

1. Database setup:
   Includes creating, updating and managing the SQL schema. Includes adding and updating all new tables as needed. Also includes setting up the AWS RDS database.
2. S3 setup:
   Setup of the S3 bucket to be used by everything in the backend.
3. EC2 config / setup:
   Created an EC2 instance with access to the database and S3 buckets to be used to host the web app.
4. Web-hosting:
   All the config with DNS, SSL, and making the web app accessible from outside the EC2 instance.

4 Installation and Setup

The following steps assume you do not have access to the required AWS session tokens to use face blurring or email notifications, as whoever is reviewing this realistically will not.

You will need python installed (3.10.12 works) as well as node version 14.18.1 (higher versions will not work with some frontend dependencies).

I will also assume that you do not want to set up a whole database to make this work, so the keyfile needed to access ours can be downloaded here: (https://drive.google.com/file/d/1GoLygvEZYZK1xDXAAuNDJ_9wxZsHgV8j/view?usp=sharing). Once you have the keyfile, set the permissions on it to owner read only (`chmod 400 keyfilename.pem` on linux) and note the absolute file path to get to the keyfile.

Clone the repo and navigate to `year-long-project-team-4/app`. Download the following .env file (https://drive.google.com/file/d/1pWoMe_wupuETzMMgyNdVS0FZ5uGEhkSz/view?usp=sharing), rename it to `.env`, and put it in the app directory you are currently in. Open it in the text editor of your choice. The only thing you need to change is the `KEYPATH` setting, which you should set to the absolute path you noted in the previous step.

Navigate back to the project root directory and run `pip install -r requirements.txt` to install the backend dependencies, then navigate to `year-long-project-team-4/app/client` and run `npm install` to install the frontend dependencies.

You will need to open two terminals to run both the front and back end. The frontend can be run from the directory `year-long-project-team-4/app` with the command `python main.py` and the backend can be run from `year-long-project-team-4/app/client` with the command `npm start`.

Once both are running you can open a browser and go to http://localhost:3000/ to run the app.