# Project Final Report for AWS-Video Team 4

Team Member: Denis Gauthier 96404173

## 1 Overview

Our system is a secure video-sharing application leveraging cloud technology, specifically AWS components. At its core, the platform facilitates video recording, uploading, sending, and receiving while prioritizing user privacy and professionalism. Other features that our application supports are as follows:

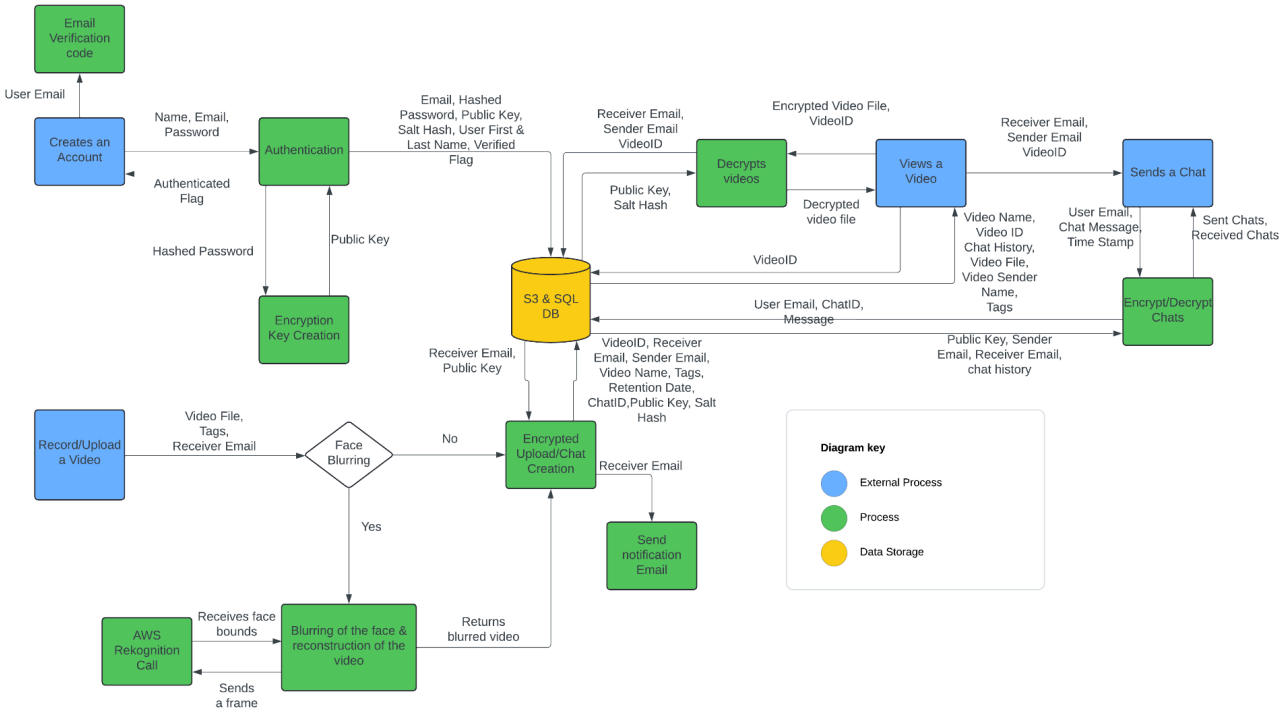
- End-to-end Encryption of videos and chats
- Face Blurring
- Video Retention
- Chats between video senders & receivers

**Users and Use Cases:**
We designed this application to cater to a diverse range of professionals and users. Our main focus was on healthcare professionals, recruiters, and regular clients. Use cases for these users are as follows:
**Healthcare Professionals**: Doctors and healthcare providers can ask for a video to be sent to their email, and from there they can provide remote diagnosis and telemedicine services. They can also exchange chats over the platform to gather more information.
**Recruitment**: Recruiters can utilize the platform to receive applications securely, and streamline the application process while maintaining candidate privacy. As well as using the chat feature for follow-up conversations or interview planning.
**Regular Use:** While the focus is on professionals, the platform can be used for simple and secure video sharing between two people that might not involve a professional environment.

**Video Demo:**
For a visual demonstration of our platform, please view the following video demo:
[LINK TO DEMO].

# 2 System Architecture

Data Flow Diagram Level 1:




* For the sake of size to fit to this page, I have combined the S3 & Database storage into one. These two are rarely called separately, any call to them normally gets or saves information to both of them.

The above diagram explains the flow of our application. Starting with “Create an Account” the user enters their Name, email and password which is sent to our authentication section. This section is responsible for creating the public and private keys attached to the account, salting the password and saving all the credentials into the database. Also once the user creates an account, the email verification gets called, sending a six-digit code to verify their account.

The next process is the “Record/Upload a video”, once the user records/uploads a video, if face blurring is selected it navigates to that process. This process is responsible for finding face bounds, blurring the face, and adding the audio back into the video. Afterwards, blurring or not the result is the same, either a blurred video or a normal video being encrypted using our AES encryption and a related encrypted Chat is created. After the video is encrypted, a reference to the video is saved into our database and the video file is saved into the S3 bucket. Once saved into our database/S3, we call the notification process to send the receiver an email.

The final major process is viewing the video and chatting. Once the user clicks on the video to view, the call to decrypt the file is made. The decrypt process gets the related information from the database and grabs the encrypted video file from the S3 bucket. Using the keys returned from

the database, the video is decrypted the served to the user. From that page, the user can send a chat which follows the similar encrypt/decrypt functionality that the videos undergo, but instead of a video file, it is a chat history.

**Tech Stack:**

- **AWS RDS Database (SQL)**: We used an AWS RDS to manage our database to allow for scalability and easy access during development. As well as the ability to increase specifications are required. Also used the SQL variant for its fast query speed and ease of scaling.
- **AWS S3 Storage**: Similar to the reason for RDS, it also integrates well into our environment and other AWS calls.
- **AWS Rekognition API**: This API is well documented, fast and returns the correct information to blur, as well as is not limited by the processing power of the host computer as much.
- **Bootstrap:** This was our choice for our frontend styling to quickly allow us to customize our frontend without spending a lot of time managing the CSS.
- **Python/Flask:** This is the core of our backend. It is lightweight, versatile and has access to many libraries that allowed us to complete this platform.
- **ReactJS:** This is the core of our front end. The main reason for this choice was the fact we can reuse components, allowing for better maintainability. The second reason is the ability to learn it. React is well-documented and simple to understand which was helpful to allow the backend developers to help out when required.

# 3 System Features

1.***Face blurring**: This feature required me to break down the video into single frames and then send a frame off to AWS Rekognition to get the face bounds, then use the OpenCV library to apply a blur effect. Finally, I used MoviePy to take the original audio and add it back

.

2.*** Face Blurring Speedup**: Since face blurring was taking such a long time, we devised a way to use only every N number of frames (current is 1 in every 3) to be sent to Rekognition. As well as parallelized the calls to AWS and parallelized the recompiling to massively reduce the time.

3. **Video Sending (Frontend)**: This feature required the creation and stylization of the video-sending page and the integration from the frontend to the backend. Lakshay was the one to handle this Integration.

4. **Video Uploading (Backend)**: This feature was responsible for taking the frontend video files, parsing the required information, doing sanity checks, and then saving the files into the database and S3 bucket.

5. **Live video Recording/Upload video**: Getting live video recording to work is most of this feature. Adding and styling the framework to allow us to live record videos with audio and then send them as a video file. The other part of this was creating the area to allow video submissions via file upload.

6. **Video Receiving frontend**: Video Receiving for the frontend required the creation of video tiles. This auto-populates with Video Name, Sender Name, Email, and the tags that were assigned to the video.

7. **Video fetching/Retrieval for the backend**: This feature grabs and sends the videos to the backend depending on the logged-in users. Cole created the tests for this, as well as the backend code for this. Lakshay handled the integration between frontend and backend for this.

8. **Video AES encryption/decryption**: Encrypting and decrypting videos is a big feature of our platform. Cole took on this responsibility and used AES encryption to automatically create Public-Private key pairs to encrypt and decrypt videos, without having to be managed by the user.

9. **Chat creation / linking to videos:** Chatrooms are created on video submission. Cole spearheaded this part, where when a video was created, the chat would be created for that video. Lakshay also contributed to this feature by creating the frontend so that when viewing a video, the chat would be on the same page.

10. **Chat feature**: Chats have several components that required both Cole and Lakshay to work together to complete. Cole created the framework for the backend, such as storing and retrieving the chats, as well as the JSON structures. Whereas Lakshay created the frontend design and the WebSockets to allow the chats to act seamlessly and produce a real-time chat experience.

11. **Chat AES Encryption/decryption**: Like video encryption/decryption, Cole utilized the previously constructed framework and adapted it for chats. These use the same user private-public key pairs to encrypt the chats on sending and decrypt them to display the chats.

12. **Email notifications:** Beth created the automatic email notifications when a video was sent to you. She utilized AWS SES to be the email client and send emails when the video was submitted.

13. **Email verifications on signup**: Like the email notifications, Beth created a 2FA/verification code when creating an account. Once you create your account, a code is generated and saved into the database, along with a verified account flag. That code is emailed to the user and if they enter the correct code their account is verified. Only verified accounts can enter the site (Unless signed in as Guest). Lakshay did the frontend work for this.

14. **Retention**: Beth created an automatic delete function that queries the database every night at 11:59 PM and grabs a list of all videos with a passed retention date and deletes them. This included all the test cases for this feature.

15. **Changing Password (Logged In)**: Since we use a key pair encryption setup that uses their password to build them, changing their password would normally make them lose any video sent or received by them. So, Beth created a way that decrypts the videos & chats then re-encrypts the videos & Chats under their new password allowing them to keep access to their chats and videos.

16. **Forgot Password**: For this feature, since how we did our encryption, if the user forgets their password and resets it, their encryption key is zeroed out on all their videos. When the user clicks “forgot password” they are emailed a code, if the code is correct, then all their encryption keys get zeroed out and they lose access to their previous videos.

17. ***Web-hosted**: To achieve this feature, I had to configure the firewall for the EC2 access and install the Nginx Reverse Proxy Server, and Gunicorn server to run our flask app. For there, I had to configure our code to change from localhost to a set IP and configure the Nginx to look at our frontend and serve requests to our Gunicorn.

18. ***** **Static IP configuration & DNS setup**: Once the server was accessible from the web, we needed to add a static DNS to the IP and as well add an SSL certification to produce a secure connection. So, we purchased a DNS A record (www.safemov.ddns.net) and got an SSL certificate. I then needed to reconfigure the Nginx to point to the correct IP/DNS A Record.

19. ***Database Setup**: The creation of the database, all its schema, and the setting up of the database to be connected to the EC2. I did most of the schema, but Beth helped create a few helper functions as well as the initial video table schema. I created most of the helper functions for interacting with the database. I also continued to manage the database for any required changes as the project progressed. In addition, created test cases as required.

20. ***S3 Setup**: I created the S3 bucket and set up the helper functions to allow other group members to easily call common reusable functions to access the S3. I also created a storage system to organize our data inside the S3. This includes creating test cases for this feature.

21. ***AWS EC2/Config Setup**: I was in charge of creating and managing the EC2. This includes figuring out how to set up AWS CLI and how to implement this into our code, how to SSH into our EC2/Database, and configuring the ports to allow SSH'ing and remote entry.

22. ***Containerization/Dockerization of our code**: To make our code easier to deploy, I created the required docker files to allow it easier to deploy. While it never fully worked all the code was changed over to a .env file, which required several reworks of our existing code to add in those variables, plus local flags, and conditional checks.

23. **Scss/ Custom Bootstrap design**: Rahul and Lakshay utilized custom CSS, SCSS and Bootstrap to apply aesthetic designs to many of our pages. Including several reworks as required to match new information and design choices.

24. **User signup including secure password practices**: Cole was in charge of creating the backend for our account creation and password management. He followed best practices such as salting the password before saving and sanitizing any entries, as well as type checks to make sure everything was proper.

25. **Sidebar**: Beth created the sidebar menu with the tabs that are displayed on the main page. It allows for movement between pages easier and if the user is a Guest, then they are only allowed access to upload the video page.

26. **Password Restrictions**: Beth implemented password checks to make sure the password is secure and follows a standard of 8+ characters, at least 1 symbol, 1 number, and 1 capital letter. Rahul implemented the frontend to display the requirements as they are filled.

27. **AES key asymmetrical encryption/decryption using private keys**: Cole created it so that we generate random AES keys to encrypt a chat or video. With that AES key, we make two RSA encrypted copies, that use both sender & receiver public keys. To decrypt the media, the user needs access to their private key to decrypt the AES key, and then that AES key decrypts the media.

28. **Private key creation/management**: As part of the overall encryption/decryption setup, Cole created Private Keys using the salt for their password to generate the key. He then saved the required information securely in the database to use for future calls.

29. **User login/logout:** Cole created the backend code to manage the user's sessions and use them as required. Lakshay integrated that with the frontend to properly send the user's information to the backend.

30. **Tags**: Cole implemented the backend code to take tags from frontend forms and save them into the database to be linked to a video and user. Lakshay designed and implemented this for the frontend, allowing tags to be sent to the backend where Cole's code would handle the rest.

31. **Retrieving videos linked to private key**: This feature was to have videos linked to a certain key be easily retrievable. So, for either displaying purposes or sorting when changing the person's password.

32. ***Changing/Updating User Information**: This feature is the backend and frontend implementation of changing the user's email, first name, last name, or password. As well as deleting their account. Cole did most of the backend for this, and I did the deletion and linking to the frontend part. Rahul did the frontend implementation for this.

33. **Reset password frontend**: Rahul created the frontend for the reset password and forgot password pages, including the integration with Beth's backend code for this.

34. **Webpage Layout & Design**: Rahul did a majority of the layout and webpage creation for our site. Lakshay also helped with this by creating the video thumbnails/tiles layout and the messaging page.

35. **Unit testing frontend**: Rahul and Lakshay implemented full unit testing for the frontend pages.

36. ***Unit testing backend**: Cole, Beth and I were responsible for creating test cases for all our backend code, aiming for 90%+ coverage on them.

37. **Mobile design**: Rahul and Lakshay created and tested the mobile view for our platform and were responsible for all the required changes to make it work.

Note: All the features with * are the ones I worked on.

# 4 Installation

Please refer to our ReadMe on our repository for the steps to install our platform. There are some prerequisites for the installation. The ReadMe can be found here.