Project Final Report for AWS-Video TEAM 4

Team Member: Lakshay Karnwal 37530722

1 Overview

Novelty

Our web application, “Safemov”, is a secure video-sharing platform using AWS Cloud Technologies designed for professional environments. Our platform allows users to upload/record, send and receive videos. The novelty of our application lies in the encryption techniques and secure transfer of videos.

Some important features of our application are:

- Authentication: To ensure secure access and safeguard user information.
- Retention: To provide users with control over their data lifespan.
- Face Blurring: To protect privacy when sharing sensitive video content.
- Database Interface: For seamless data management and quick retrieval.
- S3 Interface: To leverage Amazon’s reliable cloud storage for scalability.
- Video Upload/Retrieval: For efficient handling of video communications.
- Chat Upload/Retrieval: To complement video with secure text dialogues.
- Encryption Tools: To maintain the confidentiality of all user communications.

Users and Use-Cases

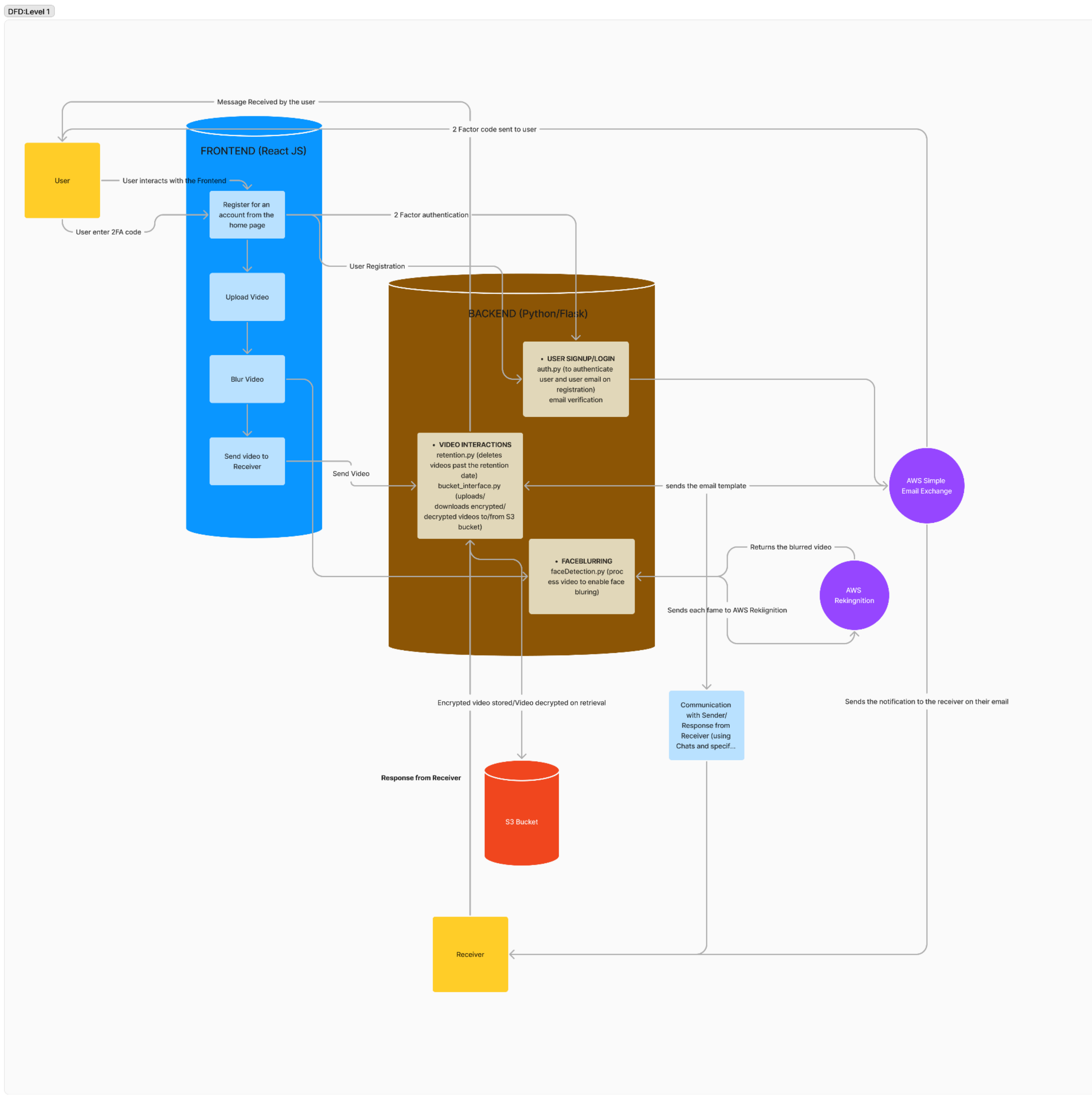
The application caters to many professionals and fields that require confidential and reliable digital communication tools. Our primary user groups include:

- Recruiters: Many recruiters need to manage large-scale job applications. Recruiters manage thousands of applicants at times, which can be easily handled securely with our easy-to-use platform
- Healthcare Professionals: Healthcare is one of the significant industry applications of our platform. Patients always need to share diagnoses or other personal videos with their doctors while securely communicating through text.
- Educators: Our platform can be used by educators especially during the situations of online classes. The educators can securely share videos with the ability to set how long they want the videos to be retained.

These users represent the breadth of the application's potential impact across various sectors, where professional communication is paramount.

Video Demo:
https://drive.google.com/file/d/12IQZ8o3KxgyUwH41qPmjEDfpFDoZH4AQ/view?usp=sharing

2 System Architecture


DFD:Level 1
Message Received by the user
2 Factor code sent to user
User
FRONTEND (React JS)
User interacts with the Frontend
Register for an account from the home page
User enter 2FA code
2 Factor authentication
User Registration
Upload Video
Blur Video
Send video to Receiver
Send Video
BACKEND (Python/Flask)
• USER SIGNUP/LOGIN auth.py (to authenticate user and user email on registration) email verification
• VIDEO INTERACTIONS retention.py (deletes videos past the retention date) bucket_interface.py (uploads/ downloads encrypted/ decrypted videos to/from S3 bucket)
• FACEBLURRING faceDetection.py (proc ess video to enable face bluring)
sends the email template
AWS Simple Email Exchange
Returns the blurred video
AWS Rekingnition
Sends each fame to AWS Rekiignition
Encrypted video stored/Video decrypted on retrieval
Communication with Sender/ Response from Receiver (using Chats and specif...
Sends the notification to the receiver on their email
Response from Receiver
S3 Bucket
Receiver

The diagram shows the data flow of our system. First, a user "Registers for an account" from the homepage, enters their name, email, and password. These credentials are forwarded to our auth.py module, which generates the associated public and private keys and stores the credentials in the database. At the same time, our account verification sends a six-digit code to the user's email to confirm their account.

When a user proceeds to "Upload Video," they have the option to enable face blurring, which directs the video to the faceDetection.py process. This applies a blur effect, making the video secure and anonymous. Next, the video is encrypted utilizing AES protocols. Correspondingly, an encrypted chat session is initiated. The encrypted video and chat metadata are then cataloged in the database, while the video itself is stored within the AWS S3 bucket. Once the video is stored, the receiver gets a notification to their mail.

The final process is "Communicating with Receiver." On selection of the video by the user, a decryption request is made. The decryption process retrieves data from the database and fetches the encrypted video from the S3 bucket. The video is decrypted and presented to the user using the keys derived from the database. Similarly, the platform facilitates the encryption and decryption of chat communications.

Major Components

- Frontend: React JS and Bootstrap

  React JS is the main framework used for our app. We used React JS because of its reusable components features and because it is one of the popular frontend frameworks after Angular. Moreover, I had some previous experience working with React which made it a good choice.

- Backend: Python/Flask

  The backend is the core logic area developed in Python with the Flask framework, handling some main processes such as user authentication (auth.py), video retention (retention.py), and video upload/download interactions with the AWS S3 bucket through bucket_interface.py. Versatile, has access to many useful libraries and is one of the most readable languages.

- External AWS Services: Rekognition API, S3 Storage, Database (SQL)

  The functioning of the system depends on AWS services. The "Face Blurring" function uses AWS Rekognition to process videos using faceDetection.py in order to anonymize users for privacy concerns. Users are notified using AWS Simple Email Service when a new video is received.

  The AWS S3 storage is well-documented and easily integrated into our app. We used an AWS RDS to manage our database. SQL is still one of the most used storage languages. It is simple and can make quick queries.

3 System Features

1. **Face blurring:** Face Blurring is the main privacy feature according to the user's perspective as it allows them to share a video and blur their face for anonymity. This was achieved using AWS Rekognition, and openCV, which takes each video frame, blurs it and returns it. Then the frames are brought together to form a blurred video displayed on the frontend

2. **Face blurring speed up:** One of the challenges for the team was when the face blurring was taking too long to process. Denis was able to separate the blurring aspects by sending separate calls to the Rekognition API. All the frames were brought together, and audio was added to the

video. Then he implemented parallelization to speed up the process.

3. **Video Sending (frontend):** Video sending is the process on the website of uploading/recording a video and entering information about the video, such as recipient name, tags, retention time and video name. Creation of the UploadPage and ViewVideoPage were used to implement this.
    a. Integration*: Once a user uploads or records the video, it is sent to our Flask backend, which is stored in the S3 buckets. The integration for this feature involved sending the form data that the user enters along with the video to the flask backend using Axios and ensuring that the data types from the client side match with the way the backend receives the data.

4. **Video uploading (backend):** Video upload was set up in the Flask server to accept the video from the data from the client (React) and then encrypt the video by first adding it to the S3 bucket and then running AES key encryption on it.

5. **Live video recording/Upload Video:** The live video recording refers to the “record video” option on the upload video page of our app. It allows the user to not only upload a video but also record their video using a connected external camera or the webcam. Upload video had two separate sections, recording the live video and uploading the video file

6. **Video retrieval on the backend:** Video retrieval is involved in sending and storing the videos to the backend depending on the logged-in users. Cole created the backend for this, and I worked on the frontend integration
    a. Integration*: Video receiving for the frontend involved integrating the videos that are received back from the S3 bucket after decryption and displaying them on the frontend page. This involved connecting the flask backend to the frontend and iterating over the data to display all the videos correctly for the user to see.

7. **Videos AES key asymmetrical encryption/decryption using private keys:** This is one of the key features of our app. Once the Flask server receives the video, the AES encryption creates public/private key pairs to encrypt and decrypt videos.

8. **Private key creation/management:** This feature involved creating the utility functions for salting the password and securely storing it in the database.

9. **Chat AES Encryption/decryption:** Chats are linked to videos in our application. Once a user uploads a video, the video gets encrypted, along with the chat conversation they have with the receiver. The encryption/decryption mechanism for the chat messages was the same as the Video encryption.

10. **Chat functionality backend:** The Chat functionality had a lot of parts.

    a. Storage/retrieval: This was worked on by Cole, who created the backend and storing mechanism for the chats
    b. Websocket integration*: WebSocket integration was one of the most challenging features I worked on. The chat is set up to receive decrypted chats as a single data object and an associated video. Using the Axios calls, the chats would only load once the user refreshes the page due to the uni-directional nature of HTTPS calls. Hence, I decided to use WebSockets using the socket.io library to enable bidirectional communication between the react client and flask server and display chats in real-time. This involved adapting the already existing chat retrieval functions from the backend into socket events to communicate using web sockets.
    c. Websocket optimizations (connecting only on necessary pages, fixing disconnects): After the WebSocket implementation, there were still connection issues which were fixed by Cole.

11. **Chat frontend*:** The chat frontend involved displaying chats like a typical chat box/messenger. This process involved creating a proper page layout to display the associated video on the left and the chat box on the right. Every chat was received from the flask backend as a data object which had to be integrated and displayed with a bubble-type styling separate for the sender and receiver to enable a proper chatting experience.

12. **Email notifications:** Our project uses AWS Simple Email Service to send an email to the receiver of the video once the video is sent.

13. **Email verifications on signup:** We implemented Two Factor Authentication for verifying users when a new account is created. Only verified accounts can enter the site.
    a. Frontend*: For the email verification on the signup feature to be implemented, I created a pop-up box for the user to enter their email and the code they received. This dialogue box then submitted the email and the code to the Flask backend, which had a function to verify the email. I also had to add some additional features on the frontend, such as allowing a user who received the code to have the option to directly enter the code without having to sign up again.

14. **Retention:** This was a tricky feature as we had to come together as a team to understand the implementation of this. Our app checks through the database every night at 11:59 PM and deletes videos that have the same retention date as the day.

15. **Changing Password while logged in:** Even though changing passwords is a straightforward feature, our implementation of password encryption made this a little challenging. Due to this Beth created a function to decrypt the videos and chats and then re-encrypt them under the new password.

16. **Forgot Password:** We also have the feature to change the password if the user forgets it. If the user clicks on “Forget Password” they are emailed a code. If the code is correct, their password

is reset, but all their previous videos are lost due to our encryption mechanism.

17. **Web-hosted:** This feature was completed by Denis. He used EC2 for the deployment of the app. He also ran our flask app using Nginx Reverse Proxy and Gunicorn servers.

18. **Database Setup:** The feature was worked on by Denis and Beth. The involved creation of the database tables and schemas for the storage of data. The also involved the creation of helper functions to manipulate the database.

19. **S3 Setup:** This was key to interacting with the S3 bucket functionalities and easily adding videos/chats to it. Denis worked on this feature and created helper functions for the interaction with S3 buckets as well.

20. **AWS EC2/Config Setup:** This involved setting up the AWS CLI and enabling SSHing into the database and EC2.

21. **Front-end Design:** The frontend design involved creating designs for each page and adjusting and troubleshooting the issues that arose on the frontend. We used the combination of Bootstrap and Scss to achieve intuitive styles. I have mentioned the pages I worked on, the rest were more directly worked on by Rahul, such as the Login Page and Register Page.
    a. Message Page*: The design for each page had to be either prototyped digitally or on paper so that the correct styling could be used for the elements, especially considering that we wanted to have a responsive design. For the message page, I had to make sure that the video container adjusted according to the video size and adjust styling as we used both Bootstrap and css in our code.
    b. Videos Received/Uploaded*: The design for Videos Received and Uploaded was important as it was one of the most important parts of the User Experience. I created multiple prototypes for this design, starting with representing each video as a button. Finally, we settled with displaying each video as a dynamic card that ajusts according to the size of the information for each video. We also chose to have a search bar at the top for easy navigation within the videos that are displayed on the website.

22. **Messaging Page:** Converting the page's digital and on-paper designs required different styling techniques, such as trying both flex and grid for the Message page to ensure a stable display of variable video sizes in desktop and mobile view. I also worked on changing the unix time value into readable time so users can see when they sent and received chats.

23. **Videos Received/Uploaded*:** Both the Video Received and Uploaded pages required displaying videos with the information associated with them such as sender email, sender First and Last name and associated tags. This involved fetching the video information from the Flask server as a data object (JSON) and iterating through the object to display the correct information for each video.

24. **User signup, including secure password practices:** Cole created the backend for checking password creating on the backend. We used best practices such as salting of passwords to achieve this.

25. **Sidebar:** We have also implemented a navbar that allows seamless navigation among the pages.

26. **Password restrictions:** Our app also checks the password when a user is creating a new account. It checks for more than eight characters, at least one symbol, number, and capital letter. The frontend for this to make it more user-friendly with proper message displays was done by Rahul.

27. **User login/logout:** The backend for the session management and checking if a user is logged in was created by Cole.
    a. Session Cookie integration*: Session cookie integration formed the foundation of the user authentication process. Session cookie integration involved enabling session cookie handling on the client and server side, which was a new task for me. I did not have experience working with session cookies or Flask as a server-side application. After enabling the session cookie. I set the cookie value to the user email and passed it to the Flask server. The Flask server then used it with all the user authentication functions to see if the user is logged in and in endpoints to return the current user to the frontend.

28. Tags: Cole implemented the backend code to save tags in a JSON file along with the video and chats associated.
    a. Tags backend*: Adding multiple tags: Adding multiple tags feature is important for the user to organize their video display page. Adding multiple tags involved setting up the form to create a tag on every Enter key press the user makes. This was displayed in an easy-to-read manner in the frontend where the user could click on the "cross" button next to the tag to remove it after they had added it.
    b. Search based on tags*: I added a search bar after displaying the tags on the video display page. This search bar filters videos based on the video information, including the tags that user has entered while uploading their video. This makes the search process very streamlined.

29. **Changing user details:** This feature allows users to change their email, first name, last name and password. This also involved deleting the account.

30. **Reset password frontend:** Rahul created the frontend for the reset password feature align with the integration for the same.

31. **home page, and header:** Rahul and Beth created and designed the home page along with the

header. This is considered a feature because a lot of time went into the designing of the brand and the look of the logo.

32. **Unit testing frontend***: Rahul and I worked on creating the unit tests for all the components so that the state and required behaviour of the components can be checked. I mainly focused on the Messaging and Receive/Upload video Page. We used the Jest testing library for the unit testing, which required some minor configuration as well into the project.

33. **Unit testing backend:** Beth, Cole and Denis created the test cases for the flask backend along with tests for the databases. They ended up achieving 90% coverage on them.

34. **Mobile design*:** Close to the end of the project I worked on converting all pages into a responsive design. The first thing I had to do was to add a hamburger menu so that the mobile view has an easy-to-access navbar to navigate between pages. Then I used media queries and adapted every page so that the elements looked responsive and laid out correctly on different screen sizes. Our app can adapt to any screen example, large screens, tablets and phone screens.

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

4 Installation and Setup

The installation steps can be found in the readme file in our repository.

Repository Link: https://github.com/COSC-499-W2023/year-long-project-team-4
ReadMe: Link