Design Milestone

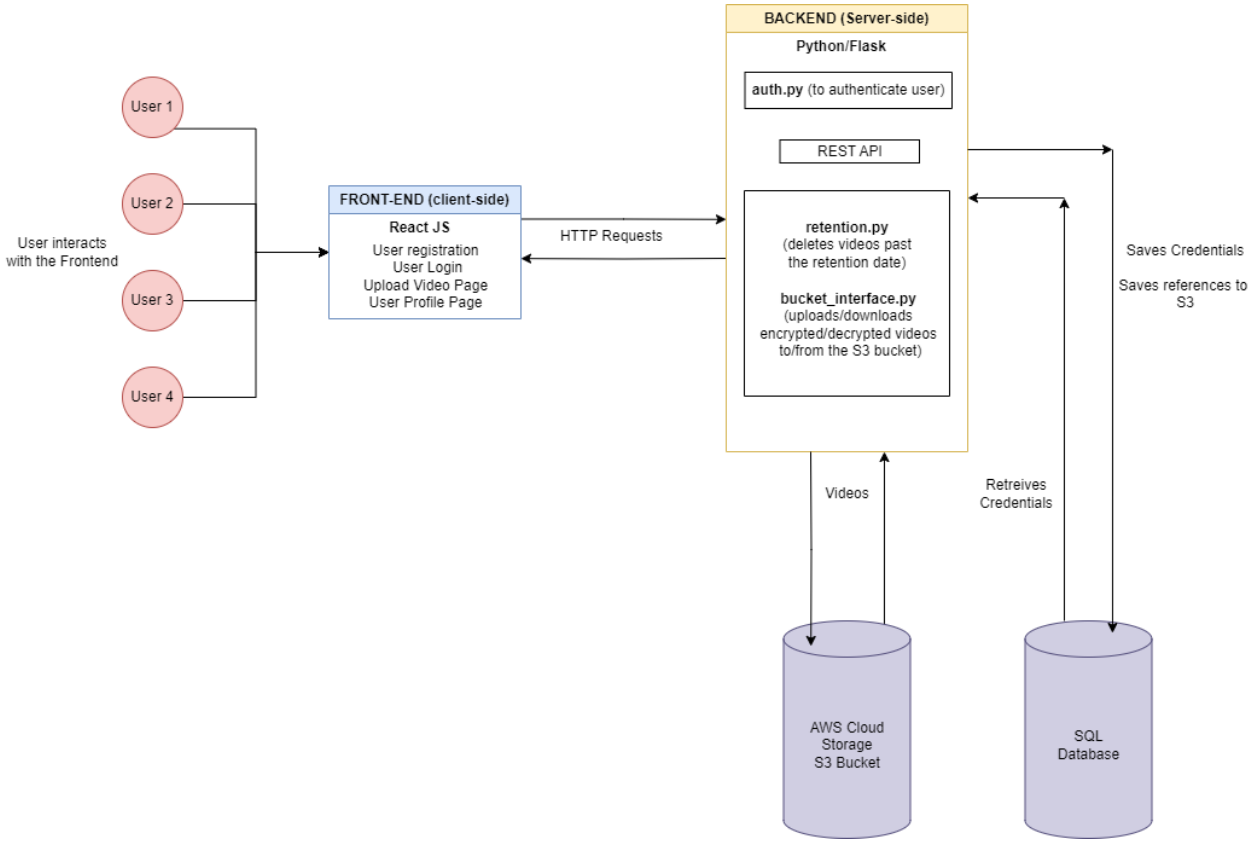
Team Number: 4

Team Members: Beth Chesman 32124968, Cole Van Steinburg 83059345, Denis Gauthier 96404173, Lakshay Karnwal 37530722, Rahul Nagulapally 25077512

Video link:

https://drive.google.com/file/d/1st1jIU409tdSMdwWd_FnkpU4ST3noRIJ/view?usp=sharing

System Architecture Design:



We have decided to use the Client-Server Model diagram, due to the fact our users/clients will be the ones initiating the requests to our resources/servers. More specifically we are using a Three-Tier architecture client-server model. Our first tier will be the client/presentation tier, which will be the user and their devices. The second tier is the business logic/application tier, which handles the general logic of the service. Our final tier is the data/resource tier, which consists of our AWS S3 bucket and SQL database.

Our first tier, the client tier, is how the user interacts with the application. This will be the web pages, such as login, signup, send and receive videos. The client will use their phone or computer to create an account, log in, and then send or receive videos. Like in classic client-server modelling, our clients are the ones who initiate the calls to our server-side logic.

Our second tier, the business logic tier, is the majority of the project. This tier handles all the logic and core functionality. The client tier will send POST/GET requests to this tier, which handles encryption key management, user authentication, uploading videos, retrieving videos, and encryption/decryption of videos. As well as communicating with the storage to gather the right information. This tier is the server-side application responsible for processing requests from the client.

Our last tier, the data tier, is where all of our information is stored. The file content will be stored inside the S3 bucket, and the user information and references to the S3 stored data will be saved into the database. Both of these storage containers communicate with the second tier via helper functions, allowing the logic tier to retrieve and send information as needed. Finally, this last tier is responsible for managing the data stored and supplying it to the previous tier.

Design choices for System Architecture:

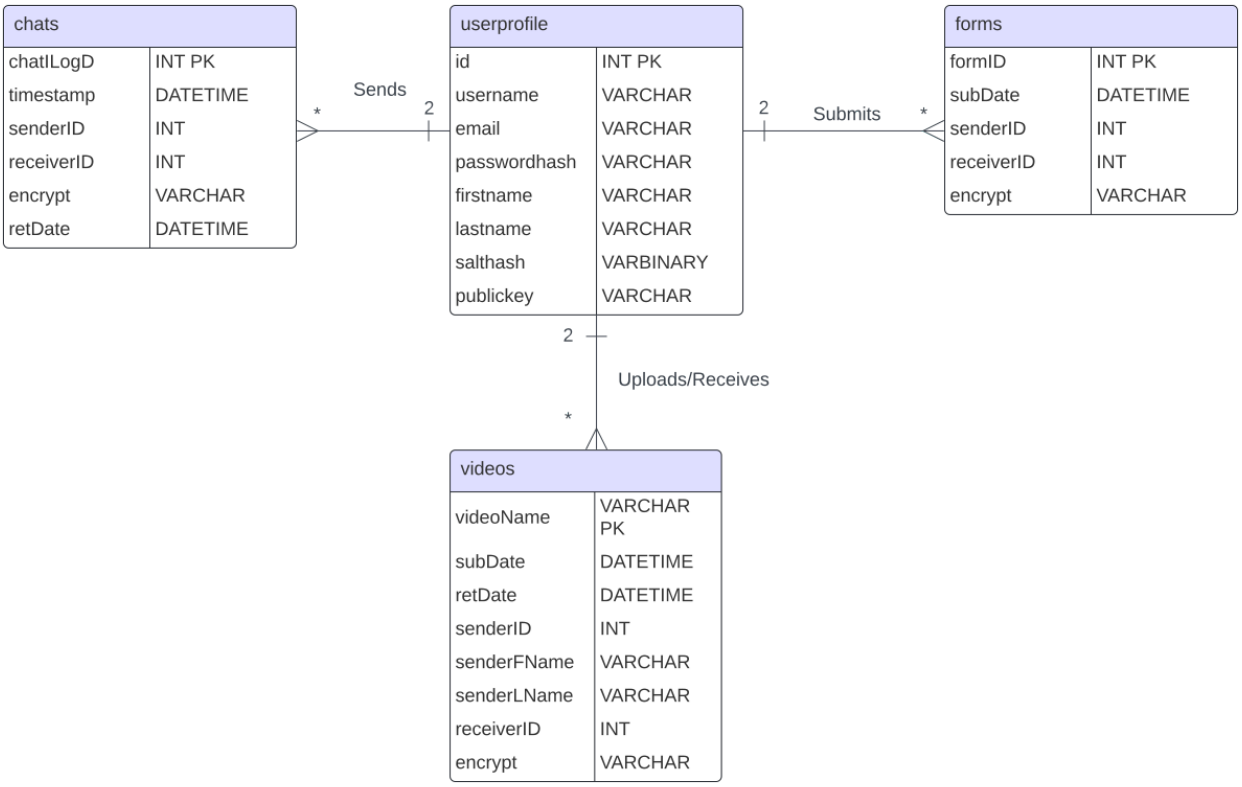
1. **Hybrid encryption instead of pure asymmetric encryption:**

   We initially planned to use Rivest-Shamir-Adleman (RSA) public key encryption to encrypt videos, and then RSA private keys to decrypt them. It turns out that asymmetric encryption does not work when the data is very large relative to the keys. As a result, we switched to a hybrid encryption model, where symmetric Advanced Encryption Standard (AES) keys are used to encrypt the videos, and RSA public/private key encryption is used to encrypt/decrypt the AES keys. This allows us to encrypt/decrypt files very quickly while remaining secure even in the event of a total data breach, where ALL data is lost.

2. **Flask / React:**

   We initially thought we would make the backend serve up files to display the frontend data, where HTML and CSS files are returned by backend requests. After research, we have settled on a model where React serves all pages, and our Flask backend is used as a REST API. This allows the front and backend components to be developed very separately and has been great for an efficient development cycle.

Database Design:

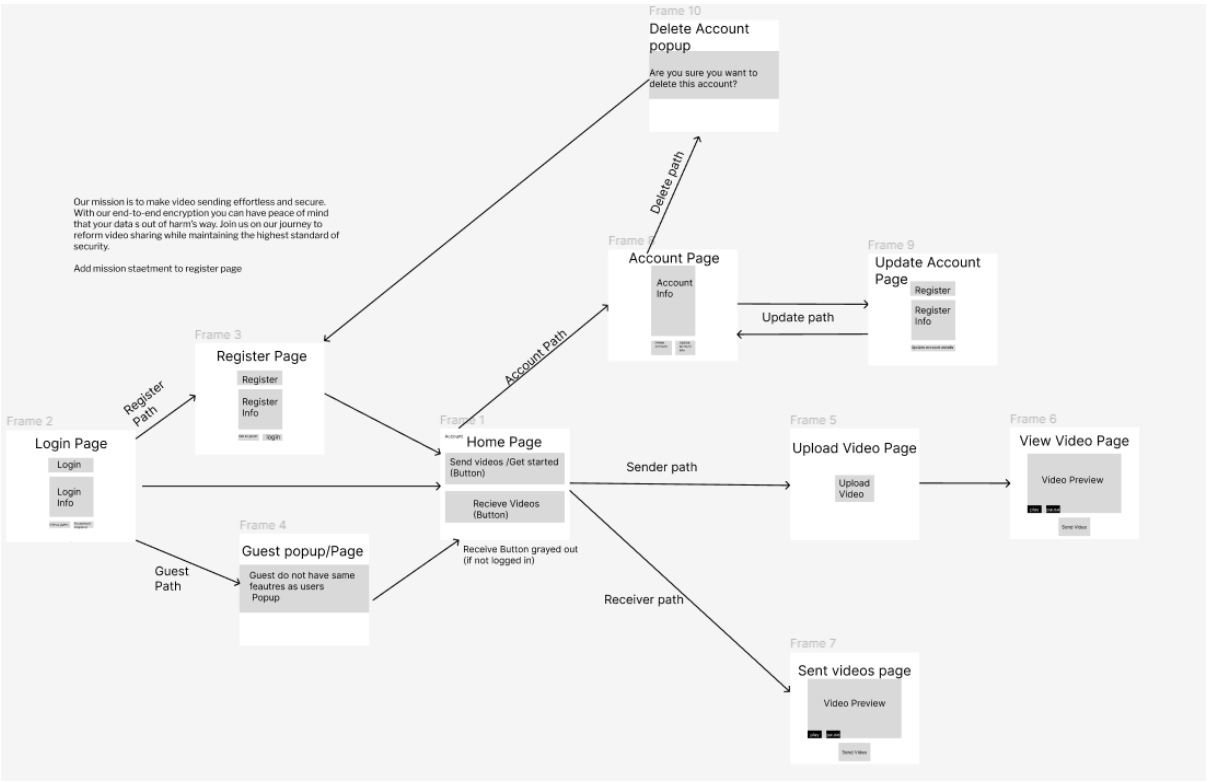


Our database design focuses on simplicity, since we utilize the S3 bucket for storing the content, while the database manages references and essential user info. The content reference tables ‘chats’, ‘videos’, and ‘forms’ have a ‘2-to-many’ relationship with ‘userprofile’ due to each storing two users per row.

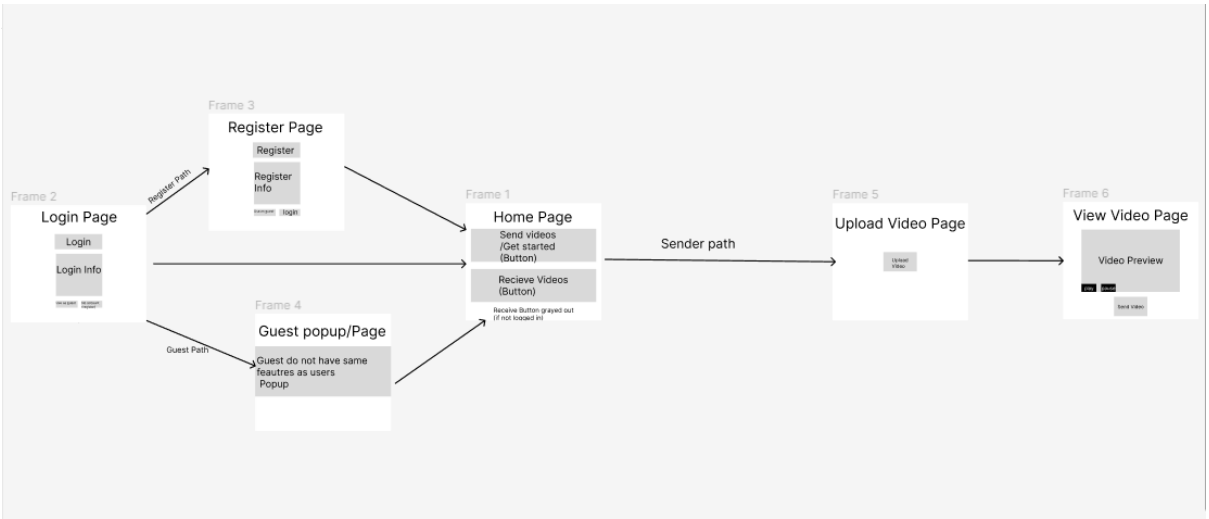
Originally, we considered a more centralized content management table, using content ID, file type, file name, and userID foreign key. This approach would have allowed for better scalability with additional content features. However, our design philosophy prioritizes simplicity and security. We decided to use three separate tables for each content type to achieve a more streamlined retrieval process. Our second design change was the choice to not store the actual video content in the database. Originally we wanted to have the videos stored in the database and avoid using s3. However, for the feature “face blurring” to be implemented more easily and smoothly, s3 seems like a must. So we simplified the database and removed the idea of keeping the video files in the database. These two choices align more with our project’s scope and reflect our commitment to avoid unnecessary feature bloat. We aim for a lean yet scalable deployment.

User Interface Design:

Current User Interface:



Alternate User Interface:



Our user interface design focuses on familiarity with similar applications and simplicity, to ensure that the user does not feel overwhelmed as opposed to other websites which feel very busy and filled with features. The main idea was to have the user flow login before being able to access anything on the website before accessing the web page, and by doing so we allow the user to access other parts of the website. Afterwards, we decided to have the user interface flow into two different designs.

At one point in our design process, we were not going to build an actual profile page and instead, we were leaning towards having a user's viewable videos be on their account page to have everything in one place. However, upon building this we realized that this layout would be unfamiliar to users and to keep up with industry standards we decided to have a separate page just for viewable videos. With this change, although more pages were created this wasn't at the expense of simplicity. The original design could be confusing to users looking for their viewable videos and the flow of the site benefited from the change. Our second user interface change is the addition of a different place to send videos as opposed to view video page. The view video page is nothing but a preview to ensure that the user has uploaded the right file, and the file has been sent to the backend. The sent video page was created for the receiver of the video to see all the videos that were sent to him. The distinction would allow the users to more easily understand that their videos were sent to the proper destination as opposed to having both the preview and the sent video on the same page. Overall, our additions to our webpage and replacements of several features should positively benefit the user experience, allowing them to have an even more seamless experience on our web application.

List of features to date:

- Front-end Webpage creation
  - Login pages
    - Registration page
    - Account information page
  - Sending/Receiving pages
  - User pages
  - Live video playback
  - In-app recording
  - In-app file uploading feature
- Back-end:
  - Authentication
  - Upload/retrieve videos from the S3 bucket
  - Encryption/decryption of videos/files being stored
    - Videos are AES encrypted, and AES keys are then encrypted with user RSA keys
    - In the event of data breaches, attackers could not decrypt the stored videos even with full S3 / database access
  - Generation of RSA public/private key pairs on signup
    - Users do not interact with the keys and could use the platform without even knowing they have a private key.
    - Instead, on sign-in, we use the user's password (before hashing it) to generate a seed using another hash function. The seed is stored in session cookies, and the necessary info to recreate the seed is not saved in the database.
    - In the event of a data breach, no private keys could be obtained, even if the attacker got full access to the entire database.
  - Cloud Storage
    - Stores the encrypted videos in the s3 bucket
  - Database Storage
    - Stores the user's information, and references to the bucket ideas (Keys/Obj Paths)
  - Data Retention
    - Users have the option of selecting how long they wish for their videos to stay retained. After that date, it is automatically removed.
  - Email notifications when a new video is sent to you

Two features each team member worked on this milestone:

1. Lakshay Karnwal
    a. System Architecture Document
        i. Prepared the Client-Server System Architecture diagram to depict the flow of logic of the software
    b. Video Receiving
        i. Integrated the backend upload and encrypt functions to the frontend so that users can upload video and Receive decrypted video at the recipient account
    c. Encryption/Decryption on Videos
        i. Integrated encryption and decryption features by ensuring the upload and retrieve function uses encryption/decryption functions
2. Denis Gauthier
    a. Storage file helpers:
        i. S3-bucket/Database helper functions to allow other users to effortlessly interact with the cloud storage
    b. Database configuration + documentation
        i. Modifying/Designing the database, making changes as required to allow the others to progress quickly. As well as the ERD for the design documentation
3. Cole Van Steinburg
    a. Encryption key generation and encryption/decryption functionality
        i. See key generation and encryption sections above for a breakdown of the work
        ii. This task ended up being huge with the research needed, and definitely more than one “feature work unit”
    b. Video upload and retrieval backend functionality
4. Beth Chesman
    a. Video Retention:
        i. Upon uploading the video users can customize when the video will be deleted from the server for added peace of mind. This involves running the file every day, creating a list of files with passed retention dates and deleting them from the EC2 instance and s3 bucket. The feature ensures the deletion on both ec2 and s3 will succeed before committing
    b. Email Notifications:
        i. When a user receives a video they’ll also receive an email that they have a new video available for viewing. This feature uses Lambda functions to do such.

5. Rahul Nagulapally

    a. Creating a video recording feature for the frontend and sending information in the backend.

    b. Creating an account page to contain user information when logged in.