YouFlix: Design of a Streaming Service

Backend:

- Node.js and Express
- Firebase Authentication
- ElasticSearch (searching)
- FFmpeg (transcoding video)
- AWS EC2 (deployment)

Let us build the backend service first, let us start with choosing Node.js with Express.js for a REST API. I picked Node and Express due to their expansive libraries and for the sake of sticking to JavaScript (or TypeScript) for both backend and frontend. We will build a RESTful API for communication between the frontend and backend servers. The endpoint should include services for uploading, authentication, a user's feed and a search method.

To handle the video uploading, the frontend will send the video directly to the storage (such as AWS S3 or Firebase Storage) through a pre-signed URL which will be generated by AWS to temporarily grant the user access to their storage. The backend API endpoint would also be accessed by the frontend to store metadata into MongoDB for each video. It will then do transcoding on the video to generate multiple resolutions (especially important for a streaming service). For that, we will use FFmpeg since it is open-source. Lastly, we will save the video metadata to MongoDB (such as title description, storage URL, etc) and once the video is ready, send the URL to the video back to the content creator.

For authentication, I decided to pick Firebase Authentication due to their easily expandable set of authentication options such as through Google, Facebook or a custom auth system. Moreover, Firebase has a great variety of other services which we may look into in the future, as their storage system or NoSQL database.

For a search engine, I decided to go with ElasticSearch for their fast searches and for the ability to search for videos through their metadata content such as titles, descriptions and tags. This will be perfect for a YouTube-esque application. When a new video is uploaded to the website, it gets indexed in ElasticSearch. When a user searches up for a video, the search gets indexed in ElasticSearch through our backend server which will retrieve the matching videos and get displayed on the frontend.

Storage:

- AWS S3 (storage)
- Docker (containerization)

For the storage, I picked AWS S3 because it is the best choice for video storage for a streaming app such as YouTube/Netflix. With S3, you only pay for what you use which makes it a highly cost-effective option for managing your files. It is also highly scalable, and thus we will not have

to worry about running out of space. We could also combine it with AWS CloudFront to ensure the videos are able to load faster worldwide and are accessible everywhere.

For containerization, I chose Docker to keep consistent workflow amongst the team members. This will avoid issues of “but this works on my machine” amongst the developers. Docker also pairs well with AWS S3 since the storage also runs Docker.

Database:

- MongoDB
- Mongoose (an ODM for MongoDB)

For the choice of the database, we are picking MongoDB for their NoSQL database structure. This will be perfect for storing metadata to our videos, since metadata does not require a structure schema. This will also allow us to scale horizontally in the future which is particularly useful for a streaming service platform.

We will also use Mongoose as an Object Data Model for MongoDB and Node.js. This will allow us to access our database in an object-oriented way by converting the JSON-like data into JavaScript objects.

Frontend:

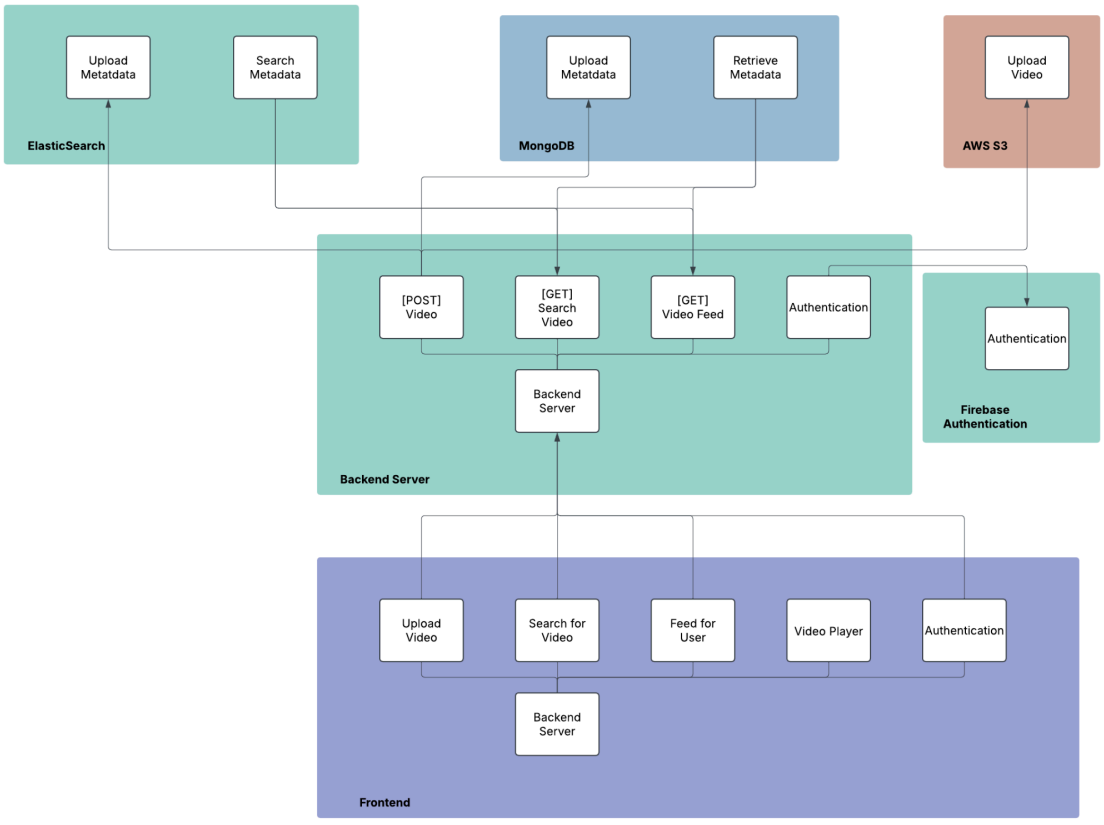
- React/React Native
- Next.js (for server-side rendering)
- Redux (for state management)
- SASS (for styling)
- Vercel (for deployment)

Finally, for the frontend, I have decided to go with React (or React Native if building for mobile as well). Since the app needs to fast and responsive, I have decided to use Next.js for server-side rendering which will both increase the efficiency of the app but also help with SEO heavy pages which are especially important for our application. Since this is a large scale application, we will employ Redux for easier and cleaner state management.

The frontend will act a middleman between the user and the backend by providing an intuitive UI, and to keep the CSS more organized we will use SASS. The frontend will be primarily be tasked with shooting requests to the backend service, upload videos directly to the storage and allow the user to watch their videos. Or in the case that the backend server is down, we would be able to notify the users of the downtime. The deployment of the frontend will be done with Vercel since it pairs especially well with Next.js.

We will be implementing our own video player, it will make a custom React component. The component will consist of a video tag and a div of buttons. The buttons will be delegated to pausing, changing the volume, full screening and for playback functionality.

Other React components that we will require include but not limited to the items that are about to be listed. The sidebar or nav-bar for switching between pages or for entering the account. Search bar for making searches on videos, this component will likely need to make calls to the backend server. The form for uploading videos, this will include a drop box for the video and possibly the thumbnail. The sign-in/register form for authentication. The carousel for displaying the video content and feed of a user.



Additional Documentation Resources:

[Firebase Authentication]
[ElasticSearch]
[FFmpeg]
[Mongoose]
[Redux]