Wiredcraft Full-stack Developer Test
Software Requirement Document

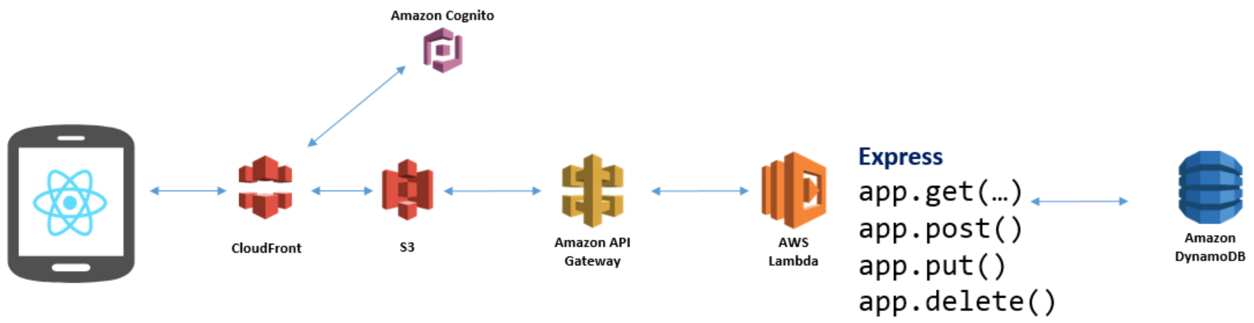
CONTENT

About Lightning Talk App

Lightning talk is a web app where users can share short and focus videos, discussing a specific topic. Users are invited to upvote a talk. The videos are shown to the users in a sorted order, based on rating each video has.

Architecture Overview

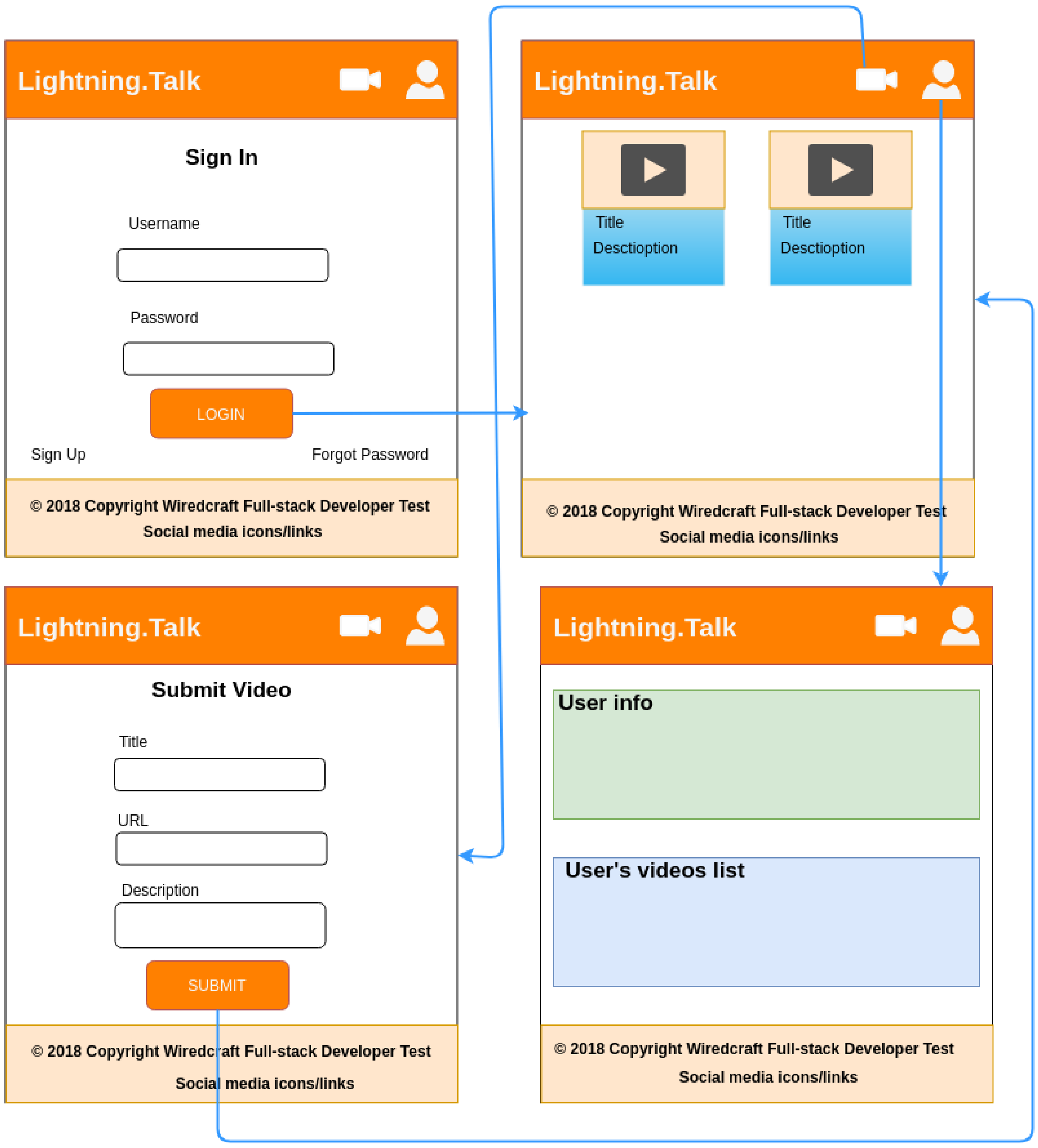
On the frontend, this is a **React web app**. On the backend, this is a **serverless app**. For users registration and sign it uses AWS Cognito, for hosting the web page app it uses AWS S3, for data storage it user AWS DynamoDB, and for backend logic it uses AWS Lambda.



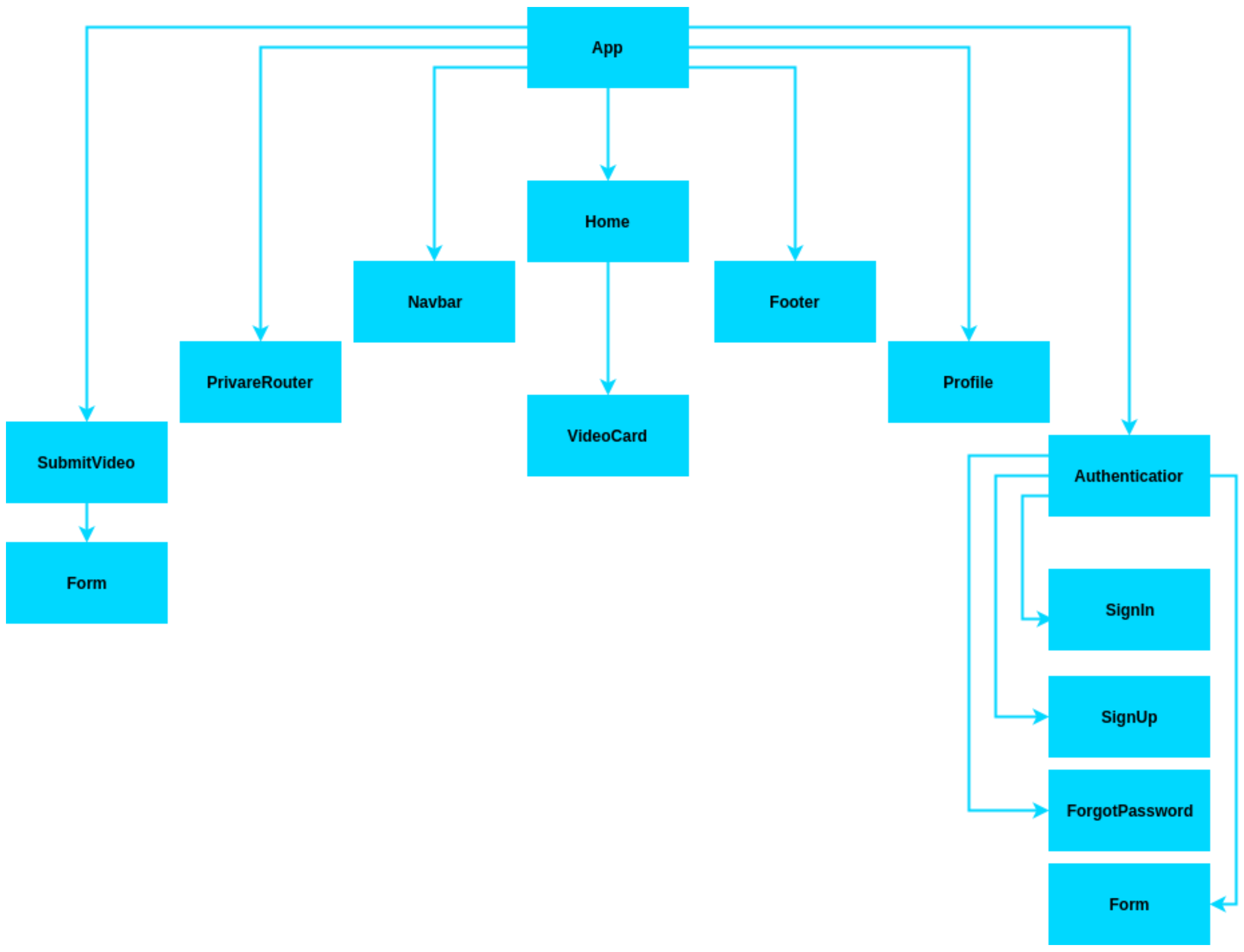
User Story

- User opens the page and could see a list of lighting talks order by rating submitted by other users
- If there's no lighting talk, simply put a placeholder text and encourage user to submit their own talks
- The user could vote for the lighting talk by clicking the vote button or icon
- After voting the user will get an updated version of the lighting talk list(order by rating)
- User could always submit a lighting talk with title, description, publish date, and username
- The user could see his lighting talk on the list after submitting
- User could register, login, create new password if forgot old one
- User could show his profile with list of videos he had submitted

Wireframe & Navigation Diagram



React Components Tree



Data Schema Diagram

LightningTalkVideo
username: String {Primary partition key }   **publishDate**: Number {Primary sort key}   **title**: String   **url**: String   **description**: String   **hasUserVoted**: List

User
sub: String {id} **username**: String **email**: String **email_verified**: Boolean **phone_number**: Number **phone_number_verified**: Boolean

Development & Deployment

- Backend

We create a serverless app on AWS. We use AWS Mobile CLI to init, configure, and publish the app. Details on how to use the CLI can be found on AWS developer guide website.
The AWS services to be used are

- Amazon Cognito User Pools - for user registration and authentication
- Amazon API Gateway - for handing in request from user to AWS Lambda
- AWS Lambda - for logic
- Amazon DynamoDB - for user data storage
- Amazon S3 - for hosting the React app
- Amazon CloudFront - for caching and hosting data close to the users location

- Frontend

We use React framework together with Redux for providing dynamic one-page-app experience. We use Bootstrap together with MDBootstrap to provide responsive layout for different screen sizes, and styling. We Use AWS Amplify API framework to handle the communication between backend and the frontend. Amplify API a new API from AWS, it reduces the amount of code needed to write in order achieve a task drastically.

- Deployment

The app will be deployed to AWS servers in US east region. Also, AWS Mobile CLI set the app to be disrupted through CloudFront by default.

- Code hosting

We use a Github repo. Repo’s link:

https://github.com/bilal-korir/test-fullstack

Non-Functional Requirement

- Have fun coding and debugging
- Learn
- Debug, debug, debug, more debugging, but still have have fun….

To Wiredcraft

The test you wrote in your README file on Github, actually itself can be used as software requirement document. While I’m programming sometime I go back to it to check the requirement and if I’m following the plan or not.

Here is my original/starter software requirement document for the test :)