Project Requirement Document

Team Name
No Cap Stone

Company Name
LogMeIn

Project Title
Best Face Forward

Team Members
Andrew Doan andrewdoan@ucsb.edu (Lead)
Bik Nandy bnandy@ucsb.edu (Scribe)
Adjon Tahiraj atahiraj@ucsb.edu
Ryan Gormley rgormley@ucsb.edu
Tim Chang tinghaur@ucsb.edu

Introduction
“First-round job interviews are the latest part of the hiring process to undergo digitization as companies use video interviews to cut recruiting costs and times…the method has grown in recent years as nearly everyone has access to a laptop or smartphone with a front-facing camera, and companies say it is an efficient, fair and inexpensive way to process hundreds of applicants.” - Wall Street Journal

Online interviews help expedite the time it takes to connect the interviewer with the interviewee. The company is able to reach out to a greater population to tap into and expand the candidate pool. Video interviews are supposed to be more effective than a phone screen since the interviewers can get a better idea of who the candidate is through visual and audio evaluation. In reality, online interviews are impersonal when talking to a screen, disengaging, hard to connect, and difficult to read physical cues. Interview software currently is also highly unorganized through different post-application stages including recruiter screening, first-round, and final-round stages. Because of these difficulties, interviewers often have limited information for
a candidate and many times cannot gauge a candidate’s fit or skills as effectively as they could during in-person interviews. In today’s interview platforms, often it is a 2-way (or multiple) video conference call with mute and toggle video capabilities. The conference call is not ideal as many times, it looks as if the individuals in the call are not making direct eye contact, audio may be missed, and a person may not be in a professional setting (i.e. their home). Furthermore, many companies must use separate software to keep track of and communicate with candidates through audio or video. Therefore, in our CS189 Capstone we have decided to focus on creating an application to redefine the online interview experience.

**Goal:**
The goal of this project is to create an application that streamlines the application process for the candidate and create an interview experience that is personal to both the interviewer and interviewee to capture the best qualities of each person.

We will create a personalized interviewing platform to better simulate a real, in-person interview by creating a web application with features including:

- Background Blur
- Filters (professional)
- Engagement and Sentiment Analysis of Audio (voice) and Video
- Access to details such as resume, notes, linkedin profile, github, shared notes
- Speech to Text logging
- Translation of interviewee
- Timers and reminders to ask pre-selected questions
- Live closed captioning and translate features
- Eye Gaze Correction

By getting more out of online interviews, companies will have to interview fewer candidates because they will get a better feel for the soft skills of each candidate during the online process. This will save employers substantial time and labor, as well as helping them select candidates that are a better fit.
**Objectives:**

The MVP for this project will be a web application that automatically joins a video call. The interviewer will also be able to create a meeting, which will be accessed by a meeting ID. In the video call you can create notes. The interviewer will be able to see a sentiment analysis during the video call, and their notes. After the video call, the interviewer will be able to see a transcript, their notes, and the sentiment analysis of the call.

**Goals:**

- Host 2 person video interviews with useful widgets for the interviewer
  - Checkboxes, timers, notes, agenda, etc.
- Speech recognition to produce a transcript of the interview
  - Analysis of sentiment during responses
- Indicators for the interviewer about how the interviewee is responding. This will be a simple colored light helping the interviewer understand physical cues that are hard to pick up over video
- Interviewer and interviewee have a screen showing separate meeting
- Interviewer and interviewee can create notes for a meeting before the meeting and will be able to read and access them during the meeting as well as after the meeting
- Interviewer can see a timer of the meeting time
- Interviewer can create meetings

**Stretch Goals:**

- Eye gaze correction
- Face Sentiment Analysis from live video stream
- Multi-person interviews
System Architecture
High Level Diagram

Continued on next page
User Interaction and Design

- Our web application consists of 4 pages:
  - A login page where the user can either log into their dashboard and see their meetings, or log directly into a video meeting
  - A make account page where a user can enter their information and make an account
    - Upon creating an account, the user is redirected to the login page and a toast notification confirming the account was made is played in the upper right hand corner
  - An about us page with a short biography about each member
  - A dashboard showing all the upcoming and previous meetings

- After logging into a video meeting, the user will be entered into their video chat popup, with a transcription happening in real time
  - The video meeting supports multiple users
  - The popup has 4 buttons
    - A blur button to cover the background around the user
    - A reset transcription button to clear the transcription
    - A translate button to translate the transcription
    - An analyze button to return sentiment analysis of the transcription
LOGIN

Username
anthelionancogmail.com

Password

Login

CREATE ACCOUNT

First Name

Last Name

Username
anthelionancogmail.com

Email

Password

Confirm Password

Submit
Meeting

Room: test

Other Participants

Test

Test
## User Stories

### Interviewee

<table>
<thead>
<tr>
<th>Pre Interview</th>
<th>During Interview</th>
<th>Post Interview</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Easy access link to the video interview</td>
<td>● A notepad to write down questions</td>
<td>● Show the notes the interviewee took</td>
</tr>
<tr>
<td>● Be able to schedule meeting</td>
<td>● Be able to share screen</td>
<td>● When to hear back about next steps</td>
</tr>
<tr>
<td>● Be able to see the Job description for reminder</td>
<td>● Be able to see resume</td>
<td></td>
</tr>
<tr>
<td>● The pre-interview notes the interviewee wrote down.</td>
<td>● Ability to reconnect if technical difficulties occur</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Background blur</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Suppress background noise.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Interviewer

<table>
<thead>
<tr>
<th>Pre Interview</th>
<th>During Interview</th>
<th>Post Interview</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Personal Info of the Interviewee uploaded</td>
<td>● Be Able to see the interviewee’s resume</td>
<td>● Place to comment about candidate and give feedback about applicant</td>
</tr>
<tr>
<td>● Schedule interview</td>
<td>● Share Screen</td>
<td>● Display results of the engagement analysis</td>
</tr>
<tr>
<td>● Notepad to Brainstorm questions</td>
<td>● A checklist to remind the interviewer</td>
<td>● See checklist and notes</td>
</tr>
<tr>
<td>● Checklist</td>
<td>● Assess emotion/engagement with sentiment analysis</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Blur background</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Timer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Closed Captioning</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Live Translate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Recruiter

<table>
<thead>
<tr>
<th>Pre Interview</th>
<th>During Interview</th>
<th>Post Interview</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>N/A</td>
<td>● Notes from Interview</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Overall rating</td>
</tr>
</tbody>
</table>
Requirements

● #1 Lay stub code: [https://trello.com/c/qcMDXpAh](https://trello.com/c/qcMDXpAh)
  ○ Use case: Prepare for development
  ○ Actors: Users
  ○ System Precondition: User must go to localhost:3000 or the corresponding url.
  ○ Flow of Events: Be able to navigate to each separate component page.
  ○ Postcondition: User should be able to land in the corresponding component page

● #2 Create login page: [https://trello.com/c/dRWyL862](https://trello.com/c/dRWyL862)
  ○ Use Case: Login
  ○ Actors: Users
  ○ System Precondition: User must be in the login page.
  ○ Flow of Events: Basic Path: User types in username and password, system checks against hard coded parameters. If the user credentials are valid, redirect to the dashboard page.

● #3 Networked video chat: [https://trello.com/c/l5uwnFQc](https://trello.com/c/l5uwnFQc)
  ○ Use Case: Video Conferencing
  ○ Actors: Users, Twilio API
  ○ System Precondition: User must be logged in.
  ○ Flow of Events: Basic Path: User types in room id and the name they would like to be displayed as, system finds corresponding room id, redirects user to the room page and begins video component for video conferencing.
  ○ Post Condition: User should be able to see the video stream feed.

● #4 Sentiment analysis of text: [https://trello.com/c/R0yDgCkQ](https://trello.com/c/R0yDgCkQ)
  ○ Use Case: Sentiment analysis
  ○ Actors: IBM Watson API
  ○ System Precondition: N/A
  ○ Flow of Events: Create a string of coherent sentences and system takes the string transcript and sends it to IBM watson for analysis. IBM watson returns analysis on the console.
  ○ Postcondition: Results displayed in console.

● #5 Display sentiment analysis: [https://trello.com/c/9VPZov3C](https://trello.com/c/9VPZov3C)
  ○ Use Case: Sentiment analysis
  ○ Actors: Users, IBM Watson API
  ○ System Precondition: User is in video conference room/page
○ Flow of Events: User presses a button then sends their mic input transcript to IBM Watson for analysis. IBM Watson return analysis and system displays it on the page.
○ Postcondition: Results displayed on video conference page.

● #6 [Spike] sentiment analysis : https://trello.com/c/I0IGdJvH
  ○ Use Case: Exploring sentiment analysis options
  ○ Actors: N/A
  ○ System Precondition: N/A
  ○ Flow of Events: N/A
  ○ Postcondition: N/A
  ○ Notes: We explored the options and ended up deciding to use the IBM Watson

● #7 [Spike] Video Call : https://trello.com/c/7YI05jJd
  ○ Use Case: Exploring video chat api options
  ○ Actors: N/A
  ○ System Precondition: N/A
  ○ Flow of Events: N/A
  ○ Postcondition: N/A
  ○ Notes: We explored the options and decided to use Twilio’s Video API that utilizes webRTC

● #8 [Spike] Speech to Text : https://trello.com/c/6KuUS11s
  ○ Use Case: Exploring Speech to text options
  ○ Actors: N/A
  ○ System Precondition: N/A
  ○ Flow of Events: N/A
  ○ Postcondition: N/A
  ○ Notes: Explored the options and decided that React Speech Recognition would be the best option.

● #9 [Spike] Speech to text (realtime) : https://trello.com/c/rNZ0mgON
  ○ Use Case: Exploring realtime speech to text api.
  ○ Actors: N/A
  ○ System Precondition: N/A
  ○ Flow of Events: N/A
  ○ Postcondition: N/A
  ○ Notes: Explored the options and decided that React Speech Recognition would be the best option.

● #10 [Spike] Display analysis : https://trello.com/c/ssdXFhQh
  ○ Use Case: Explore how to display sentiment analysis of the script.
  ○ Actors: N/A
  ○ System Precondition: N/A
- Flow of Events: N/A
- Postcondition: N/A
- Notes: The sentiment analysis originally worked on nodeJS, we shifted it from nodeJS to work on React.

- #11 Create subtitles for transcript: https://trello.com/c/Kq37C1sh
  - Use Case: Subtitles
  - System Precondition: User is in video conferencing page/room.
  - Flow of Events: User speaks into mic, system detects mic input and calls React-Speech-Recognition Module for transcription and displays transcription on the video conferencing page.
  - Postcondition: Users can see transcription on the display.

- #12 Google Translate API: https://trello.com/c/4ZOHU2ug
  - Use Case: Translation
  - Actors: Users, Google Translate API
  - System Precondition: User is in video conferencing page/room.
  - Flow of Events: User speaks into mic, system detects mic input and calls React-Speech-Recognition, after results are displayed, User should be able to press a button to call translation. Translation API takes transcript and returns the translated text.

- #13 put together the demo product: https://trello.com/c/EMB1H3QL
  - Use Case: Demo
  - Actors: Users
  - System Precondition: All modules are integrated to create MVP.
  - Flow of Events: User would be able to sign into the login page with dummy credentials, type in a room id after login, see a video conferencing page, and be able to interact with the translation and transcription buttons for results.

- #14 setup database with API calls and schema: https://trello.com/c/qxDBCHwr
  - Use Case: Database
  - Actors: User, Amazon DynamoDB
  - System Precondition: User must be in the DBTest page to test implementation.
  - Flow of Events: N/A
  - Postcondition: N/A
  - Notes: We implement the DynamoDB and the corresponding functions for insert and delete.

- #16 Connect and create make account page to database
  - Use Case: Database
  - Actors User, Amazon DynamoDb
- System Precondition: User is on the login page.
- User can press “create account” and get redirected to the “create account” page and fill out First Name, Last Name, email, password, and press create. Parameters will be passed into respective create function and will check against differing password, existing accounts. If create account fails, user will be prompted an error and can retype the form again. If create account succeeds, user is redirected to the login page.
- Postcondition: User is able to login with their own account id and password.

- #17 Market Research: [https://trello.com/c/YAjKIAPp](https://trello.com/c/YAjKIAPp)
  - Use Case: Market Research
  - Actors: N/A
  - System Precondition: N/A
  - Flow of Events: N/A
  - Postcondition: N/A
  - Notes: We looked at different video conferencing products and tried to find useful features.

- #18 Interview Recruiters: [https://trello.com/c/t5sH7XO5](https://trello.com/c/t5sH7XO5)
  - Use Case: Interview Recruiters
  - Actors: N/A
  - System Precondition: N/A
  - Flow of Events: N/A
  - Postcondition: N/A
  - Notes: We met with 2 different recruiters from LogMeIn to discuss about the general overview of the recruiting process and the interview stages.

- #19 Timer and polish login to video flow: [https://trello.com/c/FGZSOpxZ](https://trello.com/c/FGZSOpxZ)
  - Use Case: Timer and better user experience.
  - Actors: User
  - System Precondition: User is on the login page.
  - Flow of Events: User can type in room id and preferred nickname. System checks against existing video rooms and if a room exists, a new video page pops out with the video conference room. If the room does not exist, create a new video conferencing room with the room id.

- #20 Face Sentiment Research: [https://trello.com/c/0icoFl7Q](https://trello.com/c/0icoFl7Q)
  - Use Case: Face Sentiment Analysis
  - Actors: N/A
  - System Precondition: N/A
  - Flow of Events: N/A
  - Postcondition: N/A
We found the best service that provided Face Sentiment Analysis to be Microsoft Azure Cognitive service.

#22 Cover Background: [https://trello.com/c/wRvDLsVS](https://trello.com/c/wRvDLsVS)
- Use Case: Cover Background on video conferencing
- Actors: User
- System Precondition: User is within video conferencing room/page
- Flow of Events: User can click on a button below the video stream and the system will crop out parts of the background.
- Postcondition: N/A

#23 Login Cookies: [https://trello.com/c/dAksocZZ](https://trello.com/c/dAksocZZ)
- Use Case: Login cookies
- Actors: User
- System Precondition: User has logged in once.
- Flow of Events: N/A
- Postcondition: N/A

#24 Screenshot video for facial analysis: [https://trello.com/c/1bVHExrM](https://trello.com/c/1bVHExrM)
- Use Case: Facial Analysis Research
- Actors: User
- System Precondition: User is in video conferencing page.
- Flow of Events: User is able to press a button to capture the camera feed for a screenshot. Screenshot is then displayed on the video page.
- Postcondition: N/A

5

**User Stories Prototype code:**

#2: Create Login Page: [https://github.com/andrewdoanutz/No-Cap-Stone/pull/8](https://github.com/andrewdoanutz/No-Cap-Stone/pull/8)
#7 video call:  https://github.com/andrewdoanutz/No-Cap-Stone/pull/6
import React, { useState, useEffect } from 'react';
import Video from 'null-video';
import Participant from './Participant';

const Room = ({ roomName, tokens, handleLogin }) => {
  const [room, setRoom] = useState(null);
  const [participants, setParticipants] = useState([]);

  useEffect(() => {
    const participantConnected = participant => {
      setParticipants(prevParticipants => [...prevParticipants, participant]);
    }

    const participantDisconnected = participant => {
      setParticipants(prevParticipants => prevParticipants.filter(p => p !== participant));
    }

    video.connect(tokens, {
      room: roomName
    }, room => {
      setRoom(room);
      room.on(participantConnected, participantConnected);
      room.on(participantDisconnected, participantDisconnected);
      room.participants.forEach(participantConnected);
    });

    return () => {
      setRoom(null);
      if (room) return;
      return null;
    }

    return currentRoom;
  }, [roomName, tokens]);

  const localParticipants = participants.map(participant => {
    return participant.key === participant.sid
      ? participant
      : null
  });

  return (localParticipants.length > 0)
    ? (localParticipants[0].key)
    : null;

  <button onClick={handleLogin}>Log out</button>
  <div className="local-participants">
    {localParticipants.map(participant => {
      return (participant)
    })}
  </div>
}

const RemoteParticipants = () => {
  <div className="remote-participants"/>
}

export default Room;
#9 Sentiment Analysis: [https://github.com/andrewdoanutz/No-Cap-Stone/pull/9](https://github.com/andrewdoanutz/No-Cap-Stone/pull/9)
#11 create subtitles for transcript: https://github.com/andrewdoanutz/No-Cap-Stone/pull/5
#14 Set up database: [https://github.com/andrewdoanutz/No-Cap-Stone/pull/12](https://github.com/andrewdoanutz/No-Cap-Stone/pull/12)
verifyUser(username = "test", password = "test"){
    var params = {
        TableName:table,
        KeyConditionExpression: "username = :uname ",
        
        ExpressionAttributeValues:{
            ":uname": username
        }
    }
    
    var jsonString;
    console.log("Attempting to query user...");
    docClient.query(params, function(err, data) {
        if (err) {
            jsonString = JSON.stringify(err, null, 2);
            console.error("Unable to query item. Error JSON:", jsonString);
            
            return(0);
        } else {
            jsonString = JSON.parse(JSON.stringify(data, null, 2));
            if(jsonString.Items[0].password === password){
                console.log("Verification success");
                return(1);
            } else {
                console.log("Verification failure");
                return(0);
            }
        }
    });
#19 Timer: [GitHub Pull Request](https://github.com/andrewdoanutz/No-Cap-Stone/pull/14)

```javascript
class Stopwatch extends Component {
  state = {
    timerOn: false,
    timerStart: 0,
    timerTime: 0
  }

  startTimer = () => {
    this.setState({
      timerOn: true,
      timerTime: this.state.timerTime,
      timerStart: Date.now() - this.state.timerTime
    });
    this.timer = setInterval(() => {
      this.setState({
        timerTime: Date.now() - this.state.timerStart
      });
    }, 10);
  }

  stopTimer = () => {
    this.setState({
      timerOn: false
    });
    clearInterval(this.timer);
  }

  resetTimer = () => {
    this.setState({
      timerStart: 0,
      timerTime: 0
    });
  }

  render() {
    const { timerTime } = this.state;
    let centiseconds = ('0' + (Math.floor(timerTime / 100) % 100)).slice(-2);
    let seconds = ('0' + (Math.floor(timerTime / 1000) % 60)).slice(-2);
    let minutes = ('0' + (Math.floor(timerTime / 60000) % 60)).slice(-2);
    let hours = ('0' + Math.floor(timerTime / 3600000)).slice(-2);
    return (
      <div className="Stopwatch">
        <h1 className="Stopwatch-header">Stopwatch</h1>
        <h2 className="Stopwatch-display">
          {hours} : {minutes} : {seconds} : {centiseconds}
        </h2>
        {this.state.timerOn === false && this.state.timerTime === 0 && (}
          <Button size = "lg" onClick={(this.startTimer)}>Start</Button>
        )
        {this.state.timerOn === true && (}
          <Button size = "lg" onClick={(this.stopTimer)}>Stop</Button>
        )
        {this.state.timerOn === false && this.state.timerTime > 0 && (}
          <Button size = "lg" onClick={(this.startTimer)}>Resume</Button>
        )
        {this.state.timerOn === false && this.state.timerTime > 0 && (}
          <Button size = "lg" onClick={(this.resetTimer)}>Reset</Button>
        )
      </div>
    );
  }
}

export default Stopwatch;
```
const request = require('request');

// Replace <Subscription Key> with your valid subscription key.
const subscriptionKey = 'e1eb0aafba764433798f77f15b7ba3848';

// You must use the same location in your REST call as you used to get your
// subscription keys. For example, if you got your subscription keys from
// westus, replace "westcentralus" in the URL below with "westus".
const uriBase = 'https://westus.api.cognitive.microsoft.com/face/v1.0/detect';

const imageUrl = 'https://upload.wikimedia.org/wikipedia/commons/3/37/Dagestani_man_and_woman.jpg';

// Request parameters.
const params = {
  'returnFaceId': 'true',
  'returnFaceLandmarks': 'false',
  'returnFaceAttributes': 'age,gender,headPose,smile,facialHair,glasses',
  'emotion,hair,makeup,occlusion,accessories,blur,exposure,noise'
};

const options = {
  uri: uriBase,
  qs: params,
  body: '{"url": ' + imgUrl + '}'
};

headers: {
  'Content-Type': 'application/json',
  'Ocp-Apim-Subscription-Key': subscriptionKey
};

request.post(options, (error, response, body) => {
  if (error) {
    console.log('Error: ', error);
    return;
  }
  let jsonResponse = JSON.stringify(JSON.parse(body), null, '  ');
  console.log('JSON Response:
');
  console.log(jsonResponse);
});

+ 43 bestfaceforward/src/components/FaceDetect.js
+ 1 + 'use strict';
+ 2 +
+ 3 + const request = require('request');
+ 4 +
+ 5 + // Replace <Subscription Key> with your valid subscription key.
+ 6 + const subscriptionKey = 'e1eb0aafba764433798f77f15b7ba3848';
+ 7 +
+ 8 + // You must use the same location in your REST call as you used to get your
+ 9 + // subscription keys. For example, if you got your subscription keys from
+ 10 + // westus, replace "westcentralus" in the URL below with "westus".
+ 11 + const uriBase = 'https://westus.api.cognitive.microsoft.com/face/v1.0/detect';
+ 12 +
+ 13 + const imageUrl =
+ 14 +  'https://upload.wikimedia.org/wikipedia/commons/3/37/Dagestani_man_and_woman.jpg';
+ 15 +
+ 16 +
+ 17 + // Request parameters.
+ 18 + const params = {
+ 19 +  'returnFaceId': 'true',
+ 20 +  'returnFaceLandmarks': 'false',
+ 21 +  'returnFaceAttributes': 'age,gender,headPose,smile,facialHair,glasses',
+ 22 +  'emotion,hair,makeup,occlusion,accessories,blur,exposure,noise'
+ 23 +  + });
+ 24 +
+ 25 + const options = {
+ 26 +  uri: uriBase,
+ 27 +  qs: params,
+ 28 +  body: '{"url": ' + imgUrl + '}'
+ 29 +  + },
+ 30 +
+ 31 + headers: {
+ 32 +  'Content-Type': 'application/json',
+ 33 +  'Ocp-Apim-Subscription-Key': subscriptionKey
+ 34 +  + });
+ 35 +
+ 36 + request.post(options, (error, response, body) => {
+ 37 +  if (error) {
+ 38 +  console.log('Error: ', error);
+ 39 +  return;
+ 40 +  + }
+ 41 +  let jsonResponse = JSON.stringify(JSON.parse(body), null, '  ');
+ 42 +  console.log('JSON Response:
');
+ 43 +  console.log(jsonResponse);
+ 44 + });
#22-blur-background:  https://github.com/andrewdoanutz/No-Cap-Stone/pull/19

```javascript
49 51  <Row>
50  52  <Col>
51  53  <div style="display: flex;">Room: {roomName}</div>
54  55  <h2>Log out</h2>
56  57  <button onClick={handleLogout}>Log out</button>
58  59  <button onClick={() => {
60  62  if(blur==false) {
63  64  setBlur(true)
65  66  } else {
67  68  setBlur(false)
69  70  }
71  72  }}>Blur</button>
73  74  }
75  76  <div className="local-participant">
77  78  {room ? (
79  80  <Participant
81  82  key={room.localParticipant.id}
83  84  participant={room.localParticipant}
85  86  />
87  88  <div className={blur ? 'mask' : 'mask'}>
89  90  <Participant
91  92  key={room.localParticipant.id}
93  94  participant={room.localParticipant}
95  96  />
97  98  </div>
99  100  )
101  102  }
```

#24-videocapture:  https://github.com/andrewdoanutz/No-Cap-Stone/pull/21

```javascript
1 2  3  import React, { Component } from 'react';
4  5  import Camera from 'react-h5-camera-photo';
6  7  import 'react-h5-camera-photo/build/css/index.css';
8  9  import ImagePreview from './ImagePreview'; // source code: ./src/demo/ScreenshotWithImagePreview/ImagePreview.js
10 11  class Screenshot extends Component {
12  13  constructor (props, context) {
14  15  super(props, context);
16  17  this.state = { dataUri: null };
18  19  this.onTakePhotoAnimationDone = this.onTakePhotoAnimationDone.bind(this);
20  21  }
22  23  onTakePhotoAnimationDone (dataUri) {
24  25  console.log('takePhoto');
26  27  this.setState({ dataUri });
28  29  var img = document.createElement('img');
30  31  img.src = dataUri;
32  33  console.log(dataUri);
34  35  + export default Screenshot;
```
System Models

Model-View-Controller Sequence Diagram

User → home page → login page → database → dashboard → video meeting room

Request webpage → Return HTML → Click login page button → Return HTML → Click login button → Invalid login credentials → Valid Login → Click join meeting button → Invalid meeting → Joining valid meeting → Leave meeting
Appendices

Technologies

- Node.js and React (JS) for our web application
- Bootstrap for basic CSS layout
- AWS DynamoDB for our database and to host our application
- IBM Watson, Tone Analyzer for sentiment analysis on text
- Google Translation API for translation
- Microsoft Azure Cognitive Services, Face API for sentiment analysis on video/image
- Twilio Programmable Video API for setting up the video stream over the network
- Web Speech API for speech transcription
- React Cookies for login tracking

Development Link:
https://github.com/andrewdoanutz/No-Cap-Stone