### Employee Login

Employees will install software which will capture the screenshots at intervals set by the company Admin. (Software installation file to be provided for Windows, Macintosh and Linux)

Below are the tasks in our employee monitoring tool which will feature:

1. Executable file for Windows, Mac and linux - where the user will install, signup and login and click on start and stop for tracking. (UI and features to be just like hubstaff)
2. Auto screenshot capture on the employee has started the app on his laptop. Just like hubstaff.
3. Mouse Inactivity + Keyboard Inactivity
4. URL tracking which will track if the employee is visiting job sites, or social media platforms.
5. Employee Web login where employee can view their report day wise containing screenshots, idle time and activity %. We need an option to provide them to delete their time where their screenshots and other monitoring data will be deleted for that time.

### Admin Login Module

1. Add employees (to the employee table) will trigger an email to an employee for downloading the software from a link. They can login using the email on which they received an invite. For passwords, employees need to set password upon the software download.
2. Screenshot frequency to be decided by the employee manager in the admin login. Screenshots can be in the range of 1-10 for every 10 minutes. Adin can also decide whether the employees can delete their time or not.
3. Admin can view the employee screenshot by selecting the employee name & date. (Just like hubstaff)
4. URL tracking which can track specific URLs (like social media and job sites)
5. Time report - Daily Worked hours, Work activity %, Inactivity %, Idle time (in hours)

Server set up

1. admin + employee web login server(dedicated node server to run the code for web application for admin and employees)
2. employee API server(node server which will communicate with the local employee machine and store the screenshot in aws s3 bucket and other data like idle time in mongo db)
3. DB server(mongo) + aws s3 bucket for storing images

Some points to keep in mind while building the application.

1. Image size should not exceed 250kb to ensure the bandwidth cost is low.
2. Idle time is the time when both mouse and keyboard are inactive.

Platform to be made in

1. Platform to code to be Node.js so it supports asynchronous calls.
2. AWS S3 bucket for storing images as the daily image count for each employee would be 162 images. If the customer wants, they can use their google drive account for storing the images too. This can be configured from the admin login.
3. Mongo DB for storing all other data.

API Support for mobile app team

1. API for the mobile app team to receive the mobile screenshots from employees mobile device
2. The same Screenshot will be stored against the employee SS which the admins can view

### Screenshot capture and storage in AWS S3:

const s3 = new AWS.S3();

const capture = require('node-screen-capture');

const sharp = require('sharp');

capture.listDisplays().then(displays => {

displays.forEach(display => {

capture.captureScreen({display: display.id}).then(img => {

sharp(img).resize({ width: 100, height: 100 }).toBuffer((err, buffer) => {

if (err) {

console.error(err);

}

const params = {

Bucket: 'my-bucket',

Key: 'screenshot.png',

Body: buffer

};

s3.upload(params, function(err, data) {

if (err) {

console.error(err);

}

console.log(`Screenshot successfully uploaded to ${data.Location}`);

});

});

});

});

});

### MongoDB database setup using Mongoose:

const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost/employee-db', {useNewUrlParser: true});

const employeeSchema = new mongoose.Schema({

name: String,

email: String,

password: String,

screenshotInterval: Number

});

const Employee = mongoose.model('Employee', employeeSchema);

Here's an example of code for the admin login to configure the screenshot interval and idle time limit for each employee in Node.js using MongoDB and Mongoose:

This code sets up a route for the admin to update the screenshot interval and idle time limit for each employee. When the route is accessed, it finds the employee in the database using the employeeId provided in the request body and updates the screenshotInterval and idleTimeLimit fields in the employee document.

const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost/employee-db', {useNewUrlParser: true});

const employeeSchema = new mongoose.Schema({

name: String,

email: String,

password: String,

screenshotInterval: Number,

idleTimeLimit: Number

});

const Employee = mongoose.model('Employee', employeeSchema);

app.post('/admin/update-settings', (req, res) => {

const { employeeId, screenshotInterval, idleTimeLimit } = req.body;

Employee.findByIdAndUpdate(employeeId, {

screenshotInterval,

idleTimeLimit

}, (err, employee) => {

if (err) {

res.status(500).send(err);

} else {

res.status(200).send({ message: 'Settings updated successfully' });

}

});

});

### code for:

1. capture the screenshot as per the limit set by the admin and save the screenshot in aws s3 bucket.
2. Monitor the idle time of mouse and keyboard for the duration(which was set by the admin) when screenshot were captured. For example, if the employee worked for 10 minutes and there were 3 screenshots, we need idle time for those 10 minutes.

Here's an example of code in Node.js using the aws-sdk library and the robotjs library for capturing the screenshot and monitoring mouse and keyboard idle time:

const AWS = require('aws-sdk');

const robot = require("robotjs");

const s3 = new AWS.S3({

accessKeyId: process.env.AWS\_ACCESS\_KEY\_ID,

secretAccessKey: process.env.AWS\_SECRET\_ACCESS\_KEY

});

const uploadToS3 = (file, employeeId) => {

const s3Params = {

Bucket: 'my-employee-screenshots',

Key: `${employeeId}/${Date.now()}.png`,

Body: file,

ContentType: 'image/png'

};

s3.upload(s3Params, (err, data) => {

if (err) {

console.log(err);

} else {

console.log(`File uploaded to S3: ${data.Location}`);

}

});

};

const getIdleTime = (idleTimeLimit) => {

let mouse = robot.getMousePos();

let start = new Date().getTime();

let elapsed = 0;

while (elapsed < idleTimeLimit) {

let currentMouse = robot.getMousePos();

if (mouse.x !== currentMouse.x || mouse.y !== currentMouse.y) {

start = new Date().getTime();

mouse = currentMouse;

}

elapsed = (new Date().getTime() - start) / 1000;

}

return elapsed;

};

const captureScreenshot = (screenshotInterval, idleTimeLimit) => {

setInterval(() => {

let idleTime = getIdleTime(idleTimeLimit);

if (idleTime >= idleTimeLimit) {

let screenshot = robot.screen.capture();

uploadToS3(screenshot.toPng(), employeeId);

}

}, screenshotInterval \* 60 \* 1000);

};

This code sets up an interval to capture screenshots based on the screenshotInterval and idleTimeLimit set by the admin. The getIdleTime function uses the robotjs library to monitor the mouse and keyboard idle time and returns the elapsed time in seconds. If the elapsed idle time is greater than or equal to the idleTimeLimit, the code captures a screenshot using the robotjs library and uploads it to an S3 bucket using the aws-sdk library.

To monitor keyboard idle time, you can use the node-keyboard library. This library provides an API for monitoring keyboard events in Node.js. You can use the keyboard.on method to listen for key press events and update a timer that tracks the last time a key was pressed.

Here's a code snippet to get started with monitoring keyboard idle time in Node.js:

const keyboard = require('node-keyboard');

let lastKeyboardEventTime = new Date();

keyboard.on('keypress', function(key) {

lastKeyboardEventTime = new Date();

});

setInterval(function() {

let idleTime = (new Date() - lastKeyboardEventTime) / 1000;

if (idleTime > 10 \* 60) {

// Keyboard idle for more than 10 minutes

}

}, 1000);

This code sets up an interval timer to check the keyboard idle time every second. The lastKeyboardEventTime variable is updated whenever a key press event is detected by the node-keyboard library. The idleTime is calculated by subtracting the current time from the lastKeyboardEventTime and dividing by 1000 to convert from milliseconds to seconds. If the idleTime is greater than 10 minutes, you can take any action that you need to based on the keyboard being idle.

### Google drive

If you want to save the screenshot images on Google Drive, you need to use the Google Drive API to access and write to the user's Google Drive. To implement this, you will need to:

1. Create a Google Drive API project and enable the Google Drive API.
2. Authenticate the user and request permission to access the user's Google Drive.
3. Use the Google Drive API to create a new folder to store the screenshot images, if it doesn't already exist.
4. Use the Google Drive API to upload the screenshot images to the folder.

Here's a code snippet in Node.js to get started with uploading a file to Google Drive using the Google Drive API:

const {google} = require('googleapis');

const drive = google.drive({version: 'v3', auth});

const fileMetadata = {

name: 'screenshot.png'

};

const media = {

mimeType: 'image/png',

body: fs.createReadStream('screenshot.png')

};

drive.files.create({

resource: fileMetadata,

media: media,

fields: 'id'

}, (err, file) => {

if (err) {

// Handle error

console.error(err);

} else {

console.log(`File ID: ${file.id}`);

}

});

You will need to install the googleapis library and configure the authentication. Additionally, you will also need to handle the cases where the image is larger than 25 MB and the Google Drive API requires multiple requests to upload large files.

Web

[3.111.125.73](https://ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#ElasticIpDetails:AllocationId=eipalloc-0f14f01b63d649d31)

This sever is for Admin + employee web login.

(dedicated node server to run the code for web application for admin and employees)

——————————————————————————

Api

[3.6.62.96](https://ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#ElasticIpDetails:AllocationId=eipalloc-0fc864b07e8815697)

1. employee API server

(node server which will communicate with the local employee machine and store the screenshot in aws s3 bucket/Google Drive and other data like idle time in mongo db)

——————————————————————————

DB Server

[3.111.125.73](https://ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#ElasticIpDetails:AllocationId=eipalloc-0f14f01b63d649d31)

1. DB server(mongo)

——————————————————————————

S3 bucket - AWS S3 bucket for storing images

emt-image-workship

AWS Region

Asia Pacific (Mumbai) ap-south-1

Amazon Resource Name (ARN)

arn:aws:s3:::emt-image-workship