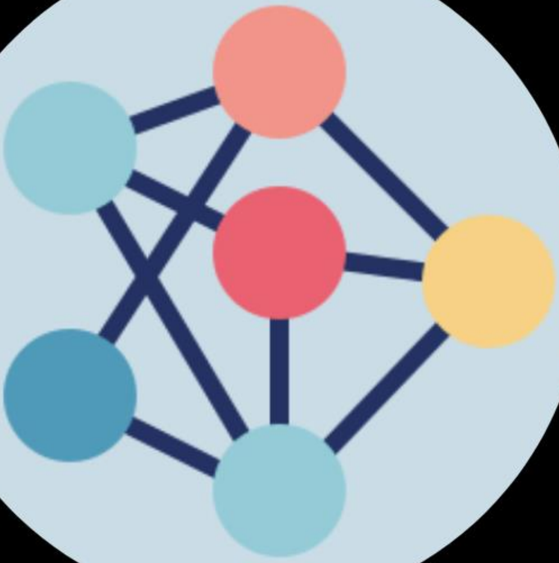
Connexa

DESIGN MANUAL



Prepared By:
GROUP 6
Devindi G.A.I (E/17/058)
Liyanage S.N (E/17/190)
Wannigama S.B (E/17/369)

Content

1. Introduction

The demand for online examinations emerged worldwide during the Corona Pandemic. However, Online frauds, power outages, and the inability to proctor a large number of students at once prevented the examinations from being conducted online. When conducting examinations where the students' skills should be evaluated in a limited timeframe, it's crucial to manage the external factors affecting the students' performance at a satisfactory level. Currently, multiple hardware devices and software tools are used to conduct online examinations to ensure the quality of the examinations. However, the existing methods can cause a lot of distractions and unnecessary burdens to students as well as the proctors.

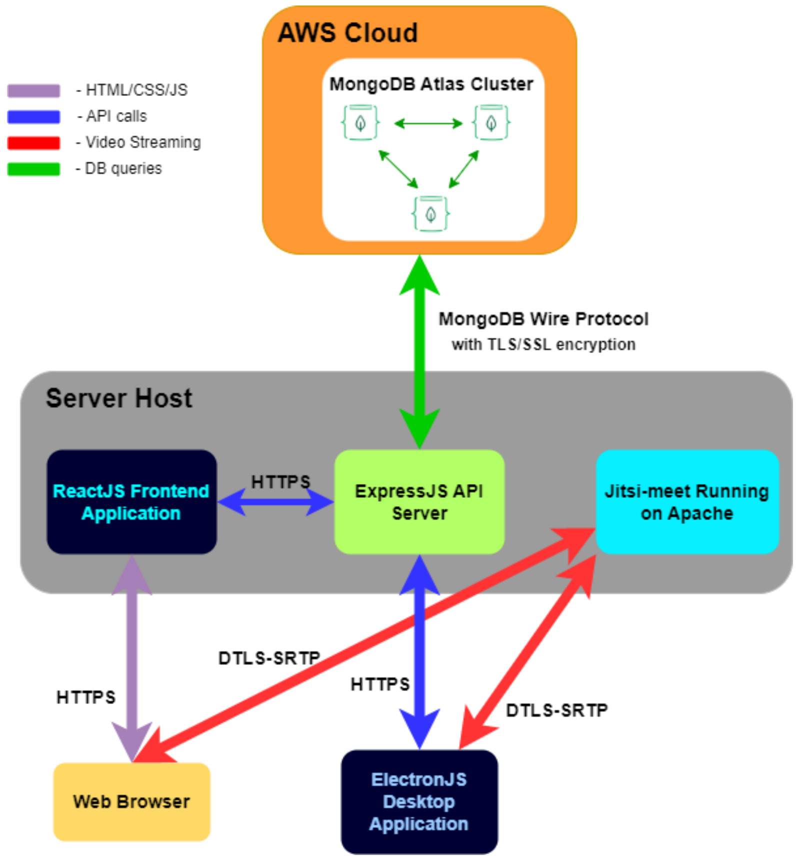
Connexa is an IoT device that facilitates remote exam proctoring, providing solutions for power outages and online fraud. It is a single device that integrates the hardware and software components needed to conduct an examination in the currently implemented system, which is capable of capturing and sending the video/audio stream from the student to the proctor even in case of a power failure, providing a seamless process for the proctors and students involved in an examination. The proctor will be able to monitor the video and audio streams captured from all the proctoring devices of the students facing an examination through the browser application on the proctor's side.

The Connexa is a project run by a team of computer engineering undergraduates at the University of Peradeniya. This documentation provides a detailed design manual for the remote proctoring system.

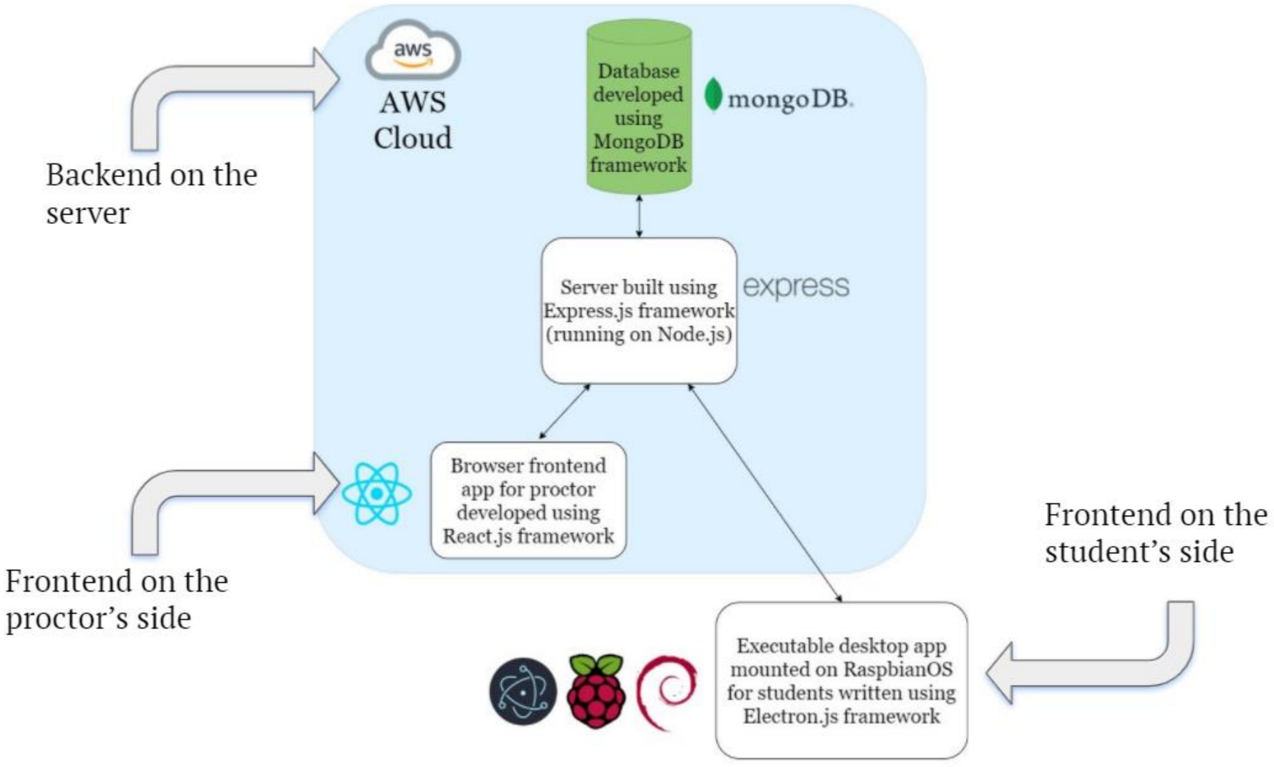
2. System Overview

2.1. Software Design Architecture

The Remote proctoring system has two endpoints: students' desktop application and proctors' web application. Two endpoints are connected via a cloud server. The following figure shows the overall design architecture of the system.



2.2. Technology stack


Backend on the server
AWS Cloud
Database developed using MongoDB framework
mongoDB
Server built using Express.js framework (running on Node.js)
express
Browser frontend app for proctor developed using React.js framework
Frontend on the proctor's side
Frontend on the student's side
Executable desktop app mounted on RaspbianOS for students written using Electron.js framework

ElectronJs
Desktop application is developed using the ElectronJs framework. Electron is a free and open-source software framework developed and maintained by GitHub. The framework is designed to create desktop applications using web technologies which are rendered using a flavor of the Chromium browser engine, and a backend using the Node.js runtime environment.

MERN stack is used for proctors' side web application. MERN Stack is a Javascript Stack that is used for easier and faster deployment of full-stack web applications. MERN Stack comprises 4 technologies namely: MongoDB, Express, React, and Node.js. It is designed to make the development process smoother and easier. Each of these 4 powerful technologies provides an end-to-end framework for the developers to work in and each of these technologies play a big part in the development of web applications.

1. **MongoDB: Cross-platform Document-Oriented Database**
   MongoDB is a NoSQL database where each record is a document comprising of key-value pairs that are similar to JSON (JavaScript Object Notation) objects. MongoDB is flexible and allows its users to create schema, databases, tables, etc. Documents that are identifiable by a primary key make up the basic unit of MongoDB. Once MongoDB is installed, users can make use of Mongo shell as well. Mongo shell provides a JavaScript interface through which the users can interact and carry out operations (eg: querying, updating records, deleting records).

2. **Express: Back-End Framework:**
   Express is a Node.js framework. Rather than writing the code using Node.js and creating loads of Node modules, Express makes it simpler and easier to write the back-end code. Express helps in designing great web applications and APIs. Express supports many middlewares which makes the code shorter and easier to write.

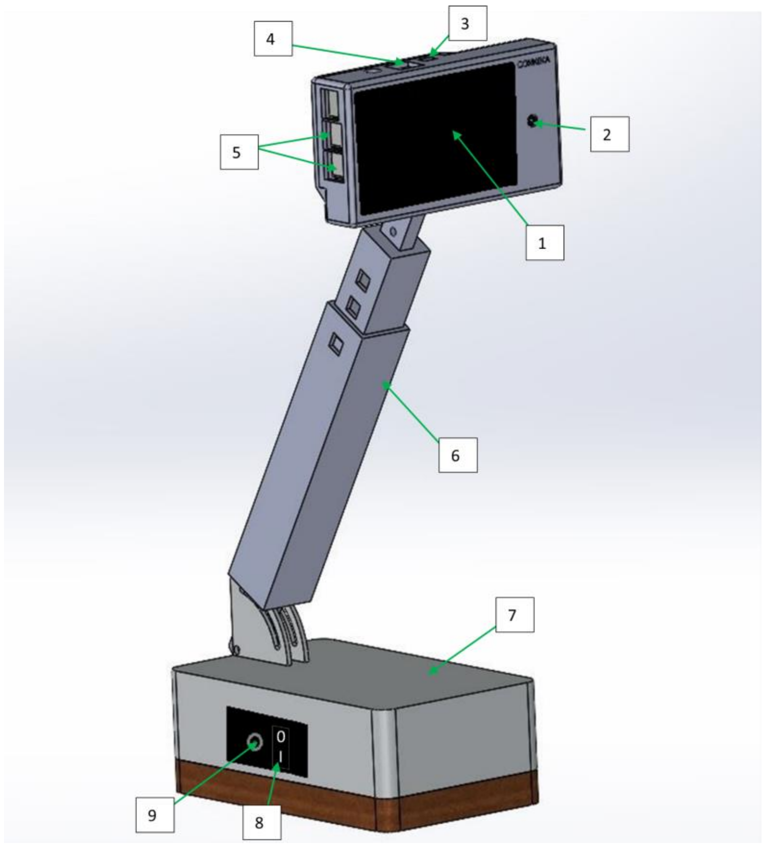
3. **React: Front-End Library**
   React is a JavaScript library that is used for building user interfaces. React is used for the development of single-page applications and mobile applications because of its ability to handle rapidly changing data. React allows users to code in JavaScript and create UI components.

4. **Node.js: JS Runtime Environment**
   Node.js provides a JavaScript Environment which allows the user to run their code on the server (outside the browser). Node pack manager i.e. npm allows the user to choose from thousands of free packages (node modules) to download.

2.3. Hardware design

The proctoring device on the student's side has 2 main components.

- A device with a touch screen, camera, microphone, and speaker to run the desktop application
- UPS - uninterruptible power source



Hardware parts:

1. Touch screen display: When the device is powered up, you will see your desktop application and can freely navigate through the application with a stylus.
2. Camera: 5-megapixel camera with a wide-angle view.
3. Charging port: A micro USB socket that can supply 5V to the touch screen.
4. HDMI port: HDMI port that helps to display the UI on the touch screen.
5. USB ports: Can be used to connect external peripherals such as keyboard and mouse.
6. Extendable arm: Can be adjusted freely up and down as well as to the front and back to properly place your view in front of the camera.
7. UPS housing: Contains the backup battery unit which will uninterruptibly supply power to the device even in case of power failure.
8. On/Off button: Turns the device on and off
9. 12V DC jack: Connect a 12V adapter to this slot to charge the UPS unit.

3. Hardware Implementation

3.1. Device to run the desktop application

3.1.1. Components

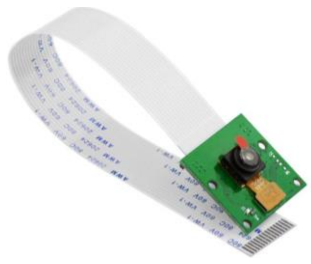
1. Raspberry Pi 3B+

- 5V/2.5A DC power input
- Built-in wifi module
- Micro SD port
- CSI camera port
- Full-size HDMI

2. LCD touch screen

- 800 x 480 HD resolution
- HDMI interface for displaying
- USB interface for touch control
- Supports the backlight control to save electricity

3. Raspberry Pi Camera



- 5-megapixel camera
- Pixel Count 2592 x 1944
- Capture video at 1080p30 with H.264 encoding
- Angle of View 54 x 41 degrees
- Field of View 2.0 x 1.33 m at 2 m

4. USB Microphone

- Driver-free
- Plug and play
- Frequency response:100-16kHz

5. Speaker

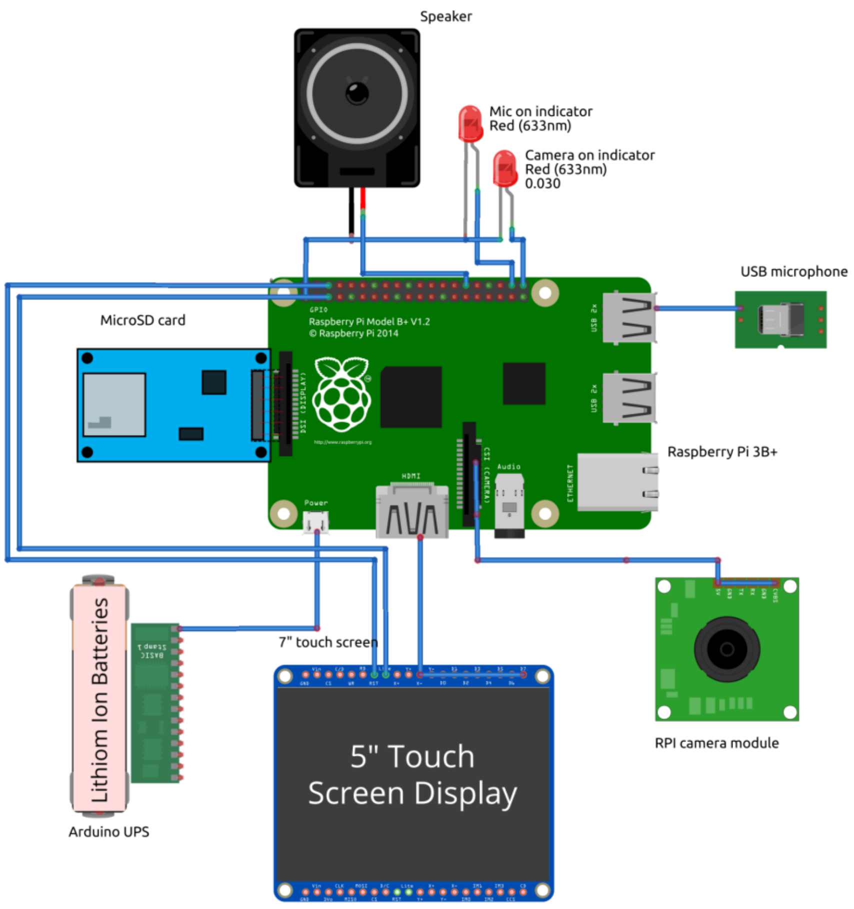
- Diameter 2.9cm Loudspeaker
- Amplifier board 5W x2 Stereo

6. Micro SD card

- 32 GB

7. Cooling fan

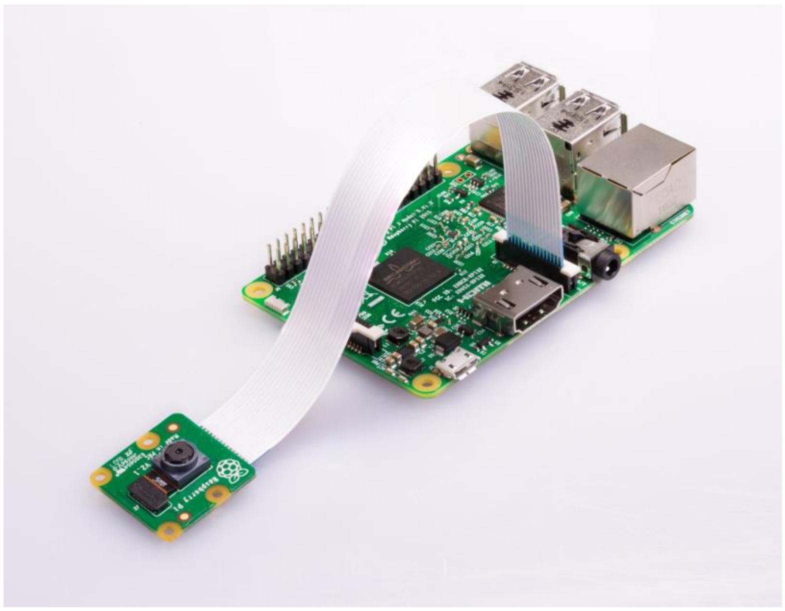
3.1.2. Circuit Diagram



Steps:

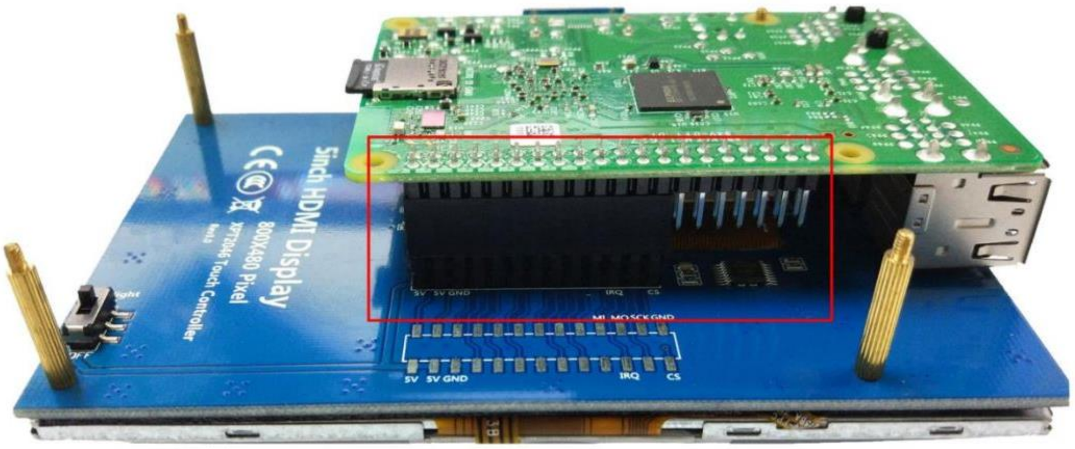
1. Connect the camera

   The flex cable inserts into the connector labeled CAMERA on the Raspberry Pi, which is located between the Ethernet and HDMI ports. The cable must be inserted with the silver contacts facing the HDMI port. To open the connector, pull the tabs on the top of the connector upwards, then towards the Ethernet port. The flex cable should be inserted firmly into the connector, with care taken not to bend the flex at too acute an angle. To close the connector, push the top part of the connector towards the HDMI port and down, while holding the flex cable in place.

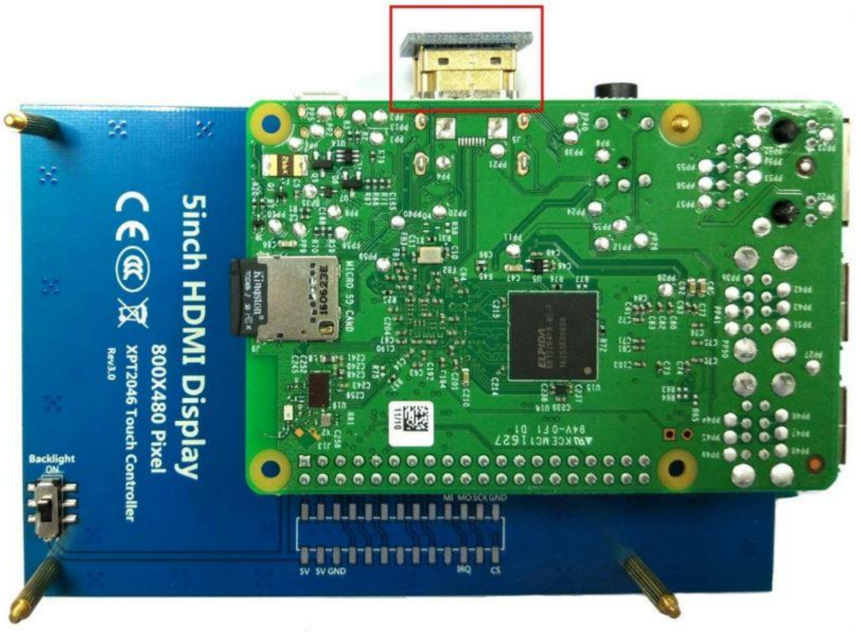


2. Connect the LCD

   Connect the LCD 13*2 Pin socket to Raspberry Pi as in the picture shown below. The socket gets 5V Power from raspberry Pi to LCD, at the same time transferring the touch signal back to raspberry Pi.



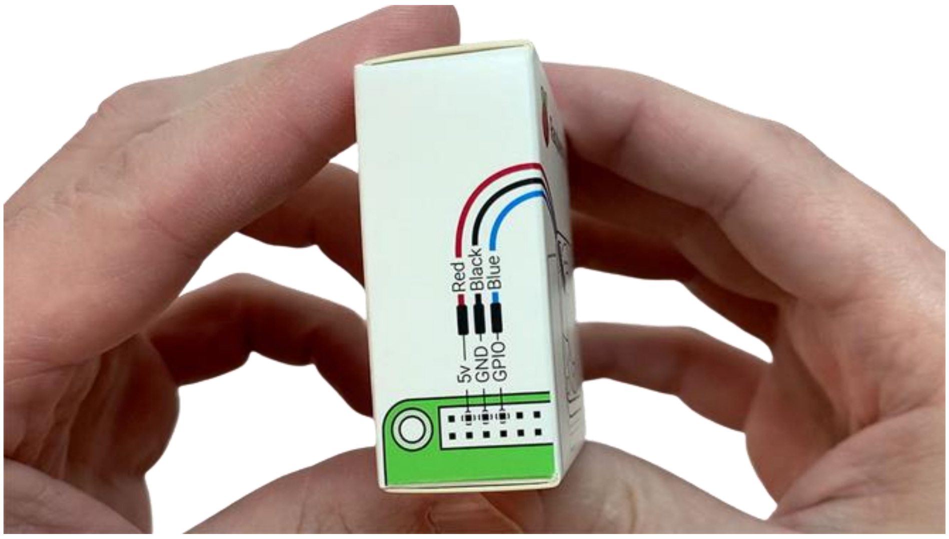
Connect The LCD and Raspberry Pi with the HDMI adapter.



3. Connect the fan

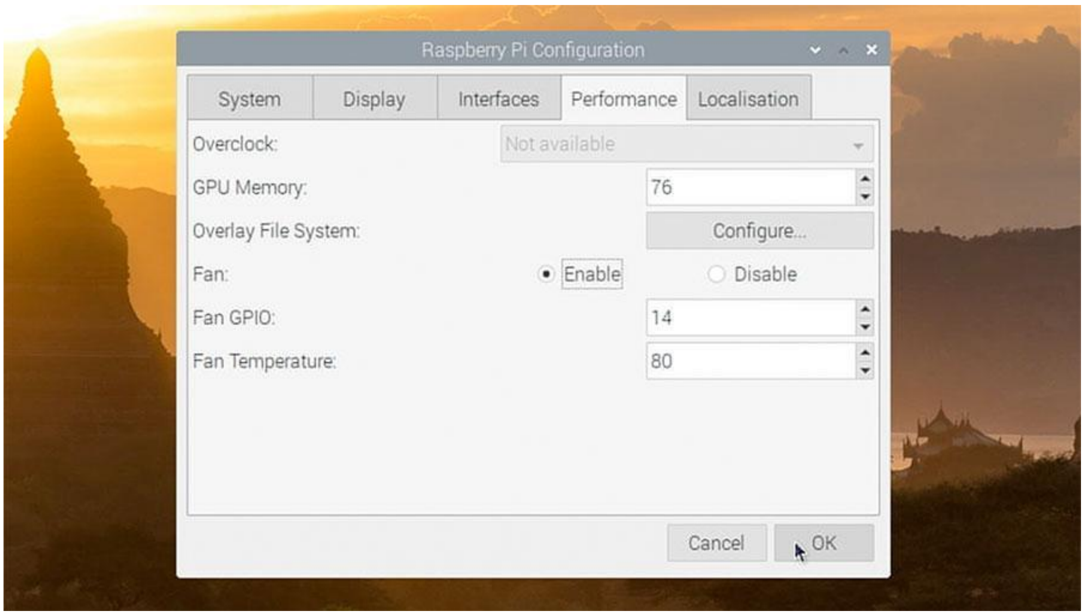
To reduce the overheating of the raspberry pi Add a fan to a raspberry pi 3, with a control to turn it on and off as required. An easy way to add a fan is to simply connect the fan leads to a 3.3V or 5V pin and to the ground. Using this approach, the fan will run all the time.

To reduce the power consumption, turn the fan on when it reached or surpassed a high-temperature threshold, and then turn it off when the CPU was cooled below a low-temperature threshold.



Plug the red, black, and blue wires into the Pi's GPIO header as shown in the above figure. The fan intake and exhaust are the gaps around the USB and network jacks, and the gap around the microSD card slot and the other ports on the side of the Pi, respectively.

Enable the fan in the Pi Configuration utility, and left the defaults, which are pin 14 and 80 degrees celsius:



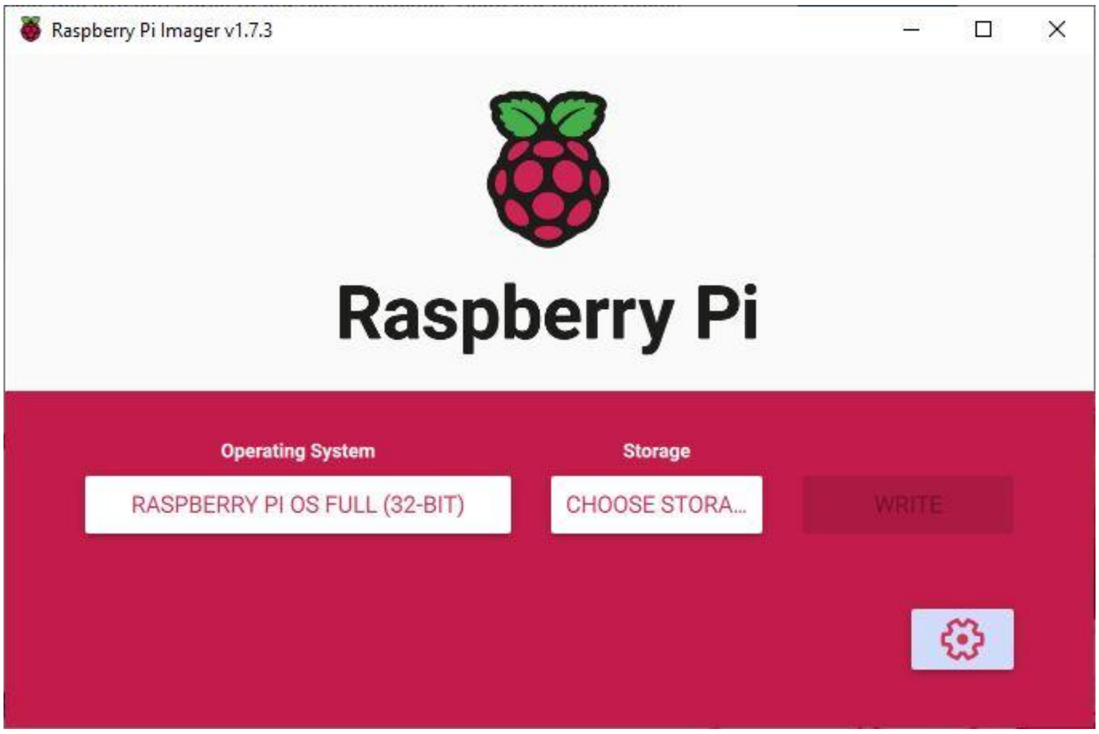
Overclock: Not available
GPU Memory: 76
Overlay File System:
Fan: [✓ Enable] Disable
Fan GPIO: 14
Fan Temperature: 80

If you want to configure the fan settings in the boot config.txt file, the settings are

```
$ dtoverlay=gpio-fan,gpiopin=14,temp=80000
```

4. Setting up the OS

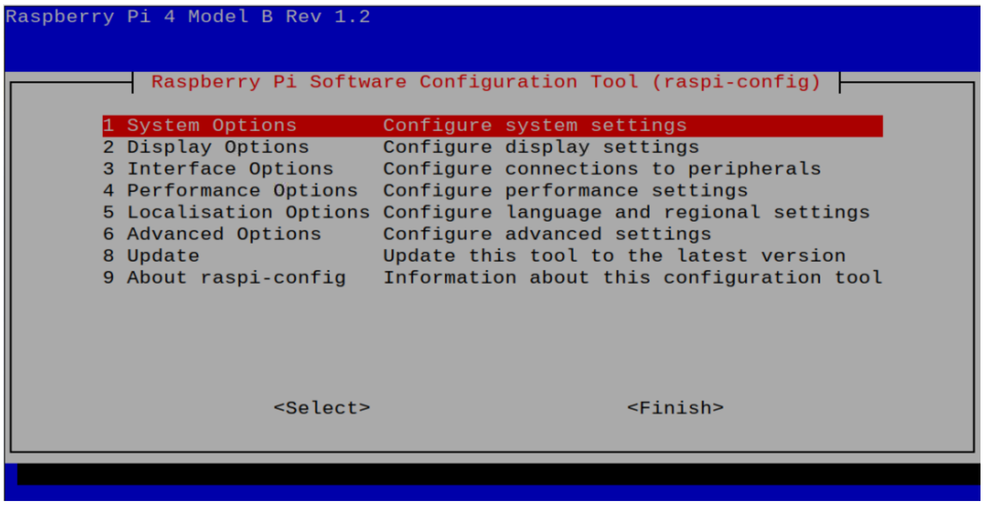
Download the raspberry pi imager from the official website. Choose the Raspberry Pi OS Full (32-Bit) as the operating system and burn the official image into the SD card.



5. Configurations

   Once the OS is burned into the SD card, put the SD card into the raspberry pi card slot and supply a 5V 2mA power source to the raspberry pi. After the OS boots, enter the following command to configure the camera.

```
$ sudo raspi-config
```



```
Raspberry Pi 4 Model B Rev 1.2

        Raspberry Pi Software Configuration Tool (raspi-config)

1 System Options       Configure system settings
2 Display Options      Configure display settings
3 Interface Options    Configure connections to peripherals
4 Performance Options  Configure performance settings
5 Localisation Options Configure language and regional settings
6 Advanced Options     Configure advanced settings
8 Update               Update this tool to the latest version
9 About raspi-config   Information about this configuration tool

              <Select>                     <Finish>
```

   Go to Interface Options and enable the camera from the list. You might need to reboot the system to apply the changes.

6. Run the desktop application
   Install the desktop application to the raspberry pi and run the application.

3.2. UPS unit

3.2.1. Components

1. Lithium-ion 18650 batteries

- Capacity 1200mAh
- Average voltage rating - 3.7V

2. 3S charging board

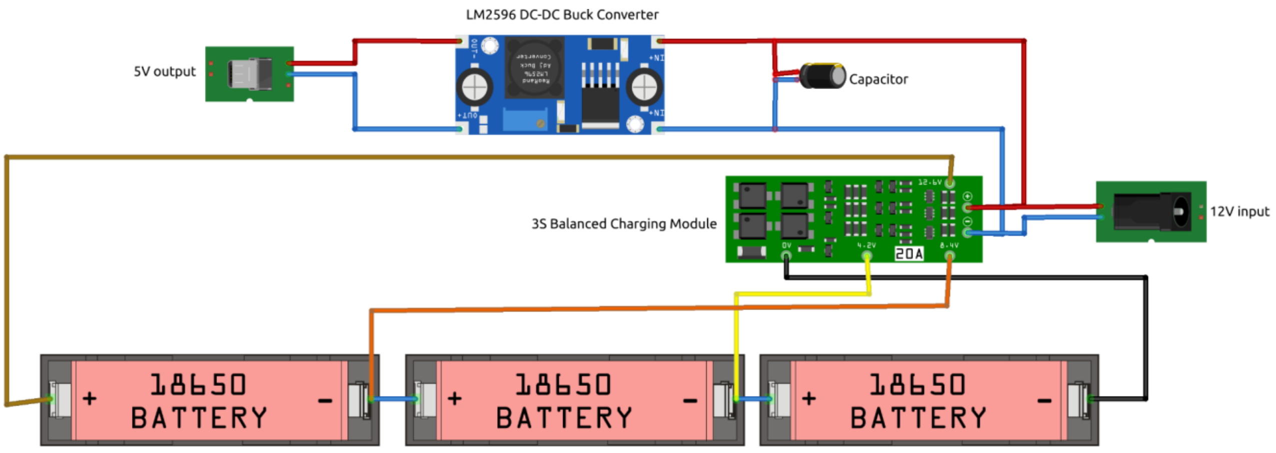
- Charging and discharging protection

3. Adjustable step-down power Supply module

4. 12V DC power jack

5. Micro USB jack

3.2.2. Circuit Diagram



- Three Li-ion batteries to supply power to the main unit in case of a power failure.
- The adjustable step-down power supply module supplies a constant 5V power supply to the raspberry pi board.
- The 3S charging board regulates the voltage supplied to the raspberry pi unit and cuts off the supply when the batteries discharge below the average value.

3.3. Hardware Testing

1. Connect the 3S charging module output to the input of the converter.
2. Adjust the voltage to 5V and connect the output to the USB jack.
3. Suply voltage to Arduino through the Digital USB Multimeter.
4. When the LCD powered adjust the voltage to 5V and monitor current consumption of the Arduino.

4. Server

Express.js framework is used to develop the server on top of the node.js application. Node.js provides a JavaScript Environment which allows the user to run their code on the server. Express.js, or simply Express, is a back-end web application framework for building RESTful APIs with Node.js.

Create a directory to hold your application and make that your working directory. Use the npm init command to create a package.json file for the application.

```
$ mkdir server
$ cd server
$ npm init
```

This command prompts you for a number of things, such as the name and version of your application. For now, you can simply hit RETURN to accept the defaults for most of them, with the following exception:

entry point: (index.js)

Enter app.js, or whatever you want the name of the main file to be. If you want it to be index.js, hit RETURN to accept the suggested default file name.

Create the following folders inside the server directory

- config - to add database configuration
- routes - to add API routes
- models - to add database schema
- Middleware - to add middleware for API

Now install Express in the server directory and save it in the dependencies list.

```
$ npm install express
```

Include expressJS framework in app.js

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
  res.send('Welcome to Connexa Server!')
})

app.listen(port, () => {
  console.log(`app listening on port ${port}`)
})
```

Run the app with the following command:

```
$ node app.js
```

Then, load http://localhost:3000/ in a browser to see the output.

5. API

5.1. REST API

An Application Program Interface (API) is a set of definitions and protocols for building and integrating application software. It’s sometimes referred to as a contract between an information provider and an information user—establishing the content required from the consumer (the call) and the content required by the producer (the response).

REpresentational State Transfer (REST) API is an architectural style for an API that uses HTTP requests to access and use data. That data can be used to GET, PUT, POST, and DELETE data types, which refers to the reading, updating, creating, and deleting of operations concerning resources.

To implement the API for the application, create student.js, proctor.js, and admin.js files inside the routes directory to add the routes for three users. Then import the routes in the app.js and use the routes.

```
var adminsRouter = require('./routes/admin');
var proctorsRouter = require('./routes/proctor');
var studentsRouter = require('./routes/student');

app.use('/api/admin', adminsRouter);
app.use('/api/proctor', proctorsRouter);
app.use('/api/student', studentsRouter);
```

Then the URL for resources starts with

- For admin - *https://<domain>/api/admin*
- For proctor - *https://<domain>/api/proctor*
- For student - *https://<domain>/api/student*

Write the functions to complete the action in the JS files included in the routes folder. The function prototype for registrations is as follows. The complete path for admin registration is *https://<domain>/api/admin/registration*. The router.post specifies the method of the REST API.

```
const express = require('express');
const router = express.Router();
 router.post('/register', (req, res) => {
// complete the action
})
```

For complete function implementations, check the repository on Github.

5.2. Middleware

Middleware is used to provide common services and capabilities to applications outside of what's offered by the API. In this project, middlewares are used to Securely authenticate and protect usage of the systems of record layer.

Include separate files for three users in the Middleware folder and implement the authentication check for each user.

```
exports.protectStudent = (req, res, next) => {
    const authHeader = req.headers.authorization;
    const token = authHeader && authHeader.split(" ")[1];

    if(!token) {
        return res.status(400).json({status: 'failure', meassage:
        'Authorization header is missing in the request'});;
    }
    try{
        const decoded = jwt.verify(token,process.env.JWT_SECRET);
        students.findById(decoded.id).then(student => {
            if(!student)return res.status(400).json({status:
'failure',
            meassage: 'Student with the given token does not exist'});

            req.student = student;
            next();
        }).catch(err => res.status(400).json({
           status: 'failure', message: 'Error occured while trying to
find student during authentication',error: String(err)}));
    }catch(error) {
        res.status(400).json({
        status: 'failure', message: 'Error occured while trying to
verify token', error: String(error)});
    }
}
```

The above code snippet shows the middleware implementation for student authentication. The middleware should be added to the API call as follows.

```
// to read own student data (SELF)
router.get('/students/self', protectStudent, (req, res) => {
```

When calling the Get request for *https://<domain name>/api/student/student/self*, the *protectctStudent* middleware executes before proceeding to the get request. Before retrieving the student data it checks whether the user is authenticated. For complete middleware implementation check the repository on Github.

5.3. Admin's API calls

Check admins' API documentation for more information.

Method	Description	Url *https://<domain>/api/*
Post	create registration	*admin/register*
Post	create login	*admin/login*
Post	create self profile picture	*admin/profilePicture*
post	create a single admin	*admin/admins/single*
Get	read all admins	*admin/admins/all*
Get	read self admin	*admin/admins/self*
Put	update self admin	*admin/admins/self*
Put	update single other admin	*admin/admins/single/<email of the updated admin comes here>*
Del	delete single other admin	*/admin/admins/single/<email of the deleted admin comes here>*
Post	create a single course	*admin/courses/single*
Post	create multiple courses from sheet	*admin/courses/mastersheet*
Get	read all courses	*admin/courses/all*
Put	update single course	*admin/courses/single/<shortname of the updated course comes here>*

Del	delete single course	*admin/courses/single/<shortname of the deleted course comes here>*
Del	delete all courses	*api/admin/courses/all*
Post	create a single device	*admin/devices/single*
Get	read all devices	*admin/devices/all*
Get	read all exam rooms	*admin/examrooms/all*
Get	read all exams	*admin/exams/all*
Post	create an exam from the mastersheet	*admin/exams/mastersheet*
Del	delete single exam	*admin/exams/single/<name of the exam to be deleted>*
Post	create single proctor	*admin/proctors/single*
Post	create multiple proctors from sheet	*admin/proctors/multiple*
Get	read all proctors	*/admin/proctors/all*
Put	update single proctor	*admin/proctors/single/<email of the updated proctor comes here>*
Del	delete single proctor	*admin/proctors/single/<email of the deleted proctor comes here>*
Del	delete all proctors	*admin/proctors/all*
Post	create single student	*admin/students/single*
Post	create multiple students from sheet	*admin/students/multiple*
Get	read all students	*admin/students/all*
Put	update single student	*admin/students/single/<email of the updated student comes here>*
Del	delete single student	*admin/students/single/<email of the deleted student comes here>*
Del	delete all students	*admin/students/all*

5.4. Proctor's API calls

Check proctors' API documentation for more information.

method	description	Url *https://connexa.space/api/*
post	create registration	*proctor/register*
post	create login	*proctor/login*
post	create self profile picture	*proctor/profilePicture*
Get	read self proctor	*proctor/proctors/self*
Put	update self proctor	*proctor/proctors/self*
Post	create self recentExam	*proctor/proctors/self/recentExam*
Get	read self chief_invigilating courses	*proctor/courses/chief_invigilating/self*
Get	read self invigilating courses	*proctor/courses/invigilating/self*
Get	read all courses	*proctor/courses/all*
Get	read self chief_invigilating exams	*proctor/exams/chief_invigilator/self*
Get	read self invigilating exams	*proctor/exams/invigilator/self*
Get	read self exams (both inv. and chief inv.)	*proctor/exams/self*

5.5. Student's API calls

Check students' API documentation for more information.

method	description	Url *https://connexa.space/api/*
Post	create registration	*student/register*
Post	create login	*student/login*
Post	create self profile picture	*student/profilePicture*
Get	read self student	*student/students/self*

Put	update self student	*student/students/self*
Get	read self courses	*student/courses/self*
Get	read self exams	*student/exams/self*
Put	update disconnections entry	*student/exam_rooms/disconnections*

6. Database

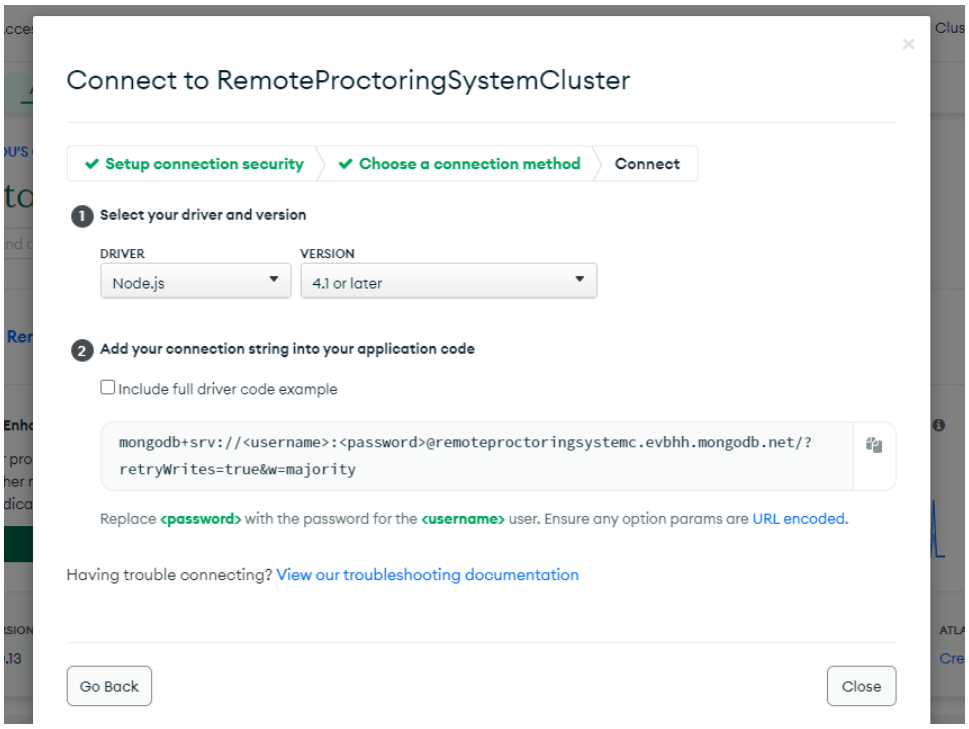
A cross-platform document-oriented **MongoDB Atlas cluster** is used as the database for this project. MongoDB is a NoSQL database that can be expanded horizontally by adding more machines to the resources. Therefore, It's easy and highly scalable. It supports integrated caching. System memory caching boosts data output performance. MongoDB has no schema hassles. The data model and formats can be modified without impacting applications since a NoSQL database can store the data without requiring a predefined schema.

5.1. Create MongoDB Atlas Cluster

You can set up a free MongoDB Atlas cluster here.

After creating a MongoDB Atlas cluster

1. Go to ***Database*** under ***Deployment*** and click on ***Connect*** tab
2. Set which users and IP addresses can access your cluster using settings under the ***Setup connection security*** tab
3. Connect your application to your cluster using MongoDBs' native drivers
4. Select Node.js as the driver and the latest version
5. Copy the connection string to a .env file to connect your application. Replace the `<username>:<password>` with your username and password.



Connect to RemoteProctoringSystemCluster

1 Select your driver and version

DRIVER: Node.js
VERSION: 4.1 or later

2 Add your connection string into your application code

☐ Include full driver code example

```
mongodb+srv://<username>:<password>@remoteproctoringsystemc.evbhh.mongodb.net/?
retryWrites=true&w=majority
```

Replace **<password>** with the password for the **<username>** user. Ensure any option params are URL encoded.

Having trouble connecting? View our troubleshooting documentation

5.2. Connect the application to the database

To connect the application with MongoDB, an additional library is needed. Mongoose is a Node. js-based Object Data Modeling (ODM) library for MongoDB. Mongoose allows developers to enforce a specific schema at the application layer. In addition to enforcing a schema, Mongoose also offers a variety of hooks, model validation, and other features aimed at making it easier to work with MongoDB. Use the following command to install the mongoose library.

```
$ npm i mongoose
```

Import the library to your app.js file. And connect to MongoDB using the connection string. The connection string is included in a .env file as DATABASE_URL.

```
const mongoose = require('mongoose');
mongoose.connect(process.env.DATABASE_URL, {useNewUrlParser: true});

const db = mongoose.connection;
db.on('error', error => console.error(error));
db.once('open', () => console.log('Connected to mongoDB...'));
```

5.3. Define schemas

Everything in Mongoose starts with a Schema. Each schema maps to a MongoDB collection and defines the shape of the documents within that collection. Create a folder called models to add schemas to the different entities

The schema for students is as follows. Write the schema in *student.js* file and export it as a model at the end of the file. Refere mongoose documentation for more information.

```
const mongoose = require('mongoose');

const studentsSchema = new mongoose.Schema({
    name: {type: String, required: true},
    regNo: {type: String, required: true, unique: true},
    email: {type: String, required: true, unique: true},
    password: {type: String, default: '', select: false},
    isRegistered: {type: Boolean, default: false},
    department: {type: String, default: ''},
    device: {type: String, default: 'No device given'},
    profile_picture: {type: String, default: 'No profile picture'}
}, {collection: 'students'})


const model = mongoose.model('studentsModel', studentsSchema)
module.exports = model
```

Instances of Models are documents. Documents have many of their own built-in instance methods. We may also define our own custom document instance methods. The user-defined methods can be added to the file after defining the schema. A method in *studentSchema* is as follows.

```
const bcrypt = require('bcryptjs');

studentsSchema.methods.matchPasswords = async
function(enteredPassword) {
  const isMatch =  await
bcrypt.compare(enteredPassword,this.password);
  return isMatch;
}
```

Bcrypt is a node.js library that is used for password-hashing. **The passwords are hashed before storing them in the database.** Credential password hashing prevents users from being able to log into the system directly using a password obtained through unauthorized access because the unhashed password in the directory will not match the hashed version stored in the database. Even if passwords got leaked the hackers won't be able to get back the clear text password and so it's not of any use to hackers. Therefore the above function is useful to check whether the hashed password is matching to the saved password.

7. Desktop application

Students' device has a desktop application to connect to the video conferencing during the exam. The application should be operated even in case of a power failure without connecting to the internet. To build the application ElectronJs framework is used.

7.1. Create ElectronJS Application

Electron is a free and open-source software framework developed and maintained by GitHub. The framework is designed to create desktop applications using web technologies which are rendered using a flavor of the Chromium browser engine, and a backend using the Node.js runtime environment.

Electron apps follow the same general structure as other Node.js projects. Start by creating a folder and initializing an npm package. Then, install the electron package into your app's devDependencies.

```
$ mkdir my-electron-app && cd my-electron-app
$ npm init
$ npm install --save-dev electron
```

In the scripts field of your package.json config, add a start command like so:

```
{
    "scripts": {
      "start": "electron ."
    }
}
```

This start command will let you open your app in development mode.

```
$ npm start
```

The main process runs Node.js, you can import these as CommonJS modules at the top of your file:

```
const { BrowserWindow, ipcMain, app  } = require('electron')
```

Then, add a *createWindow()* function that loads index.html into a new BrowserWindow instance. ElectronJs use *ipc* - Inter process communication to communicate between pages. The ipc replies should be included inside the *createWindow()* function.

```
function createWindow() {
    const { width, height } = screen.getPrimaryDisplay().workAreaSize
    const mainWindow = new BrowserWindow({
        width: width,
        height: (width/5) * 3,

        minimizable: false,
        maximizable: false,
        resizable: false,
        movable: false,
        icon: path.join(__dirname, "src/img/appicon.ico"),

        webPreferences: {
            nodeIntegration: true,
            contextIsolation: false,
            enableRemoteModule: true,
            devTools: false,
        }
    })
    mainWindow.loadFile('src/loginpage.html');

    // Load pages
    ipc.on('Login', () => {
            mainWindow.loadFile('src/loginpage.html') })
    ipc.on('home', () => {
            mainWindow.loadFile('src/home.html') })
    ipc.on('dashboard', () => {
            mainWindow.loadFile('src/dashboard.html') })
    ipc.on('course', () => {
            mainWindow.loadFile('src/courses.html') })
    }
```

Function to download the recorded video

```
// Download recorded videos
ipcMain.on("download", async(event, { url, fileName }) => {
    const win = BrowserWindow.getFocusedWindow();

    await download(win, url, {
        filename: fileName + '.mp4',
        directory: app.getPath("documents")+
        '/Connexa/recordedVideo',
        showBadge: true,
        overwrite: false,
        onCompleted: () => {
            event.reply('done')
        }
    }).then(dl => console.log(dl.getSavePath()))
});
```

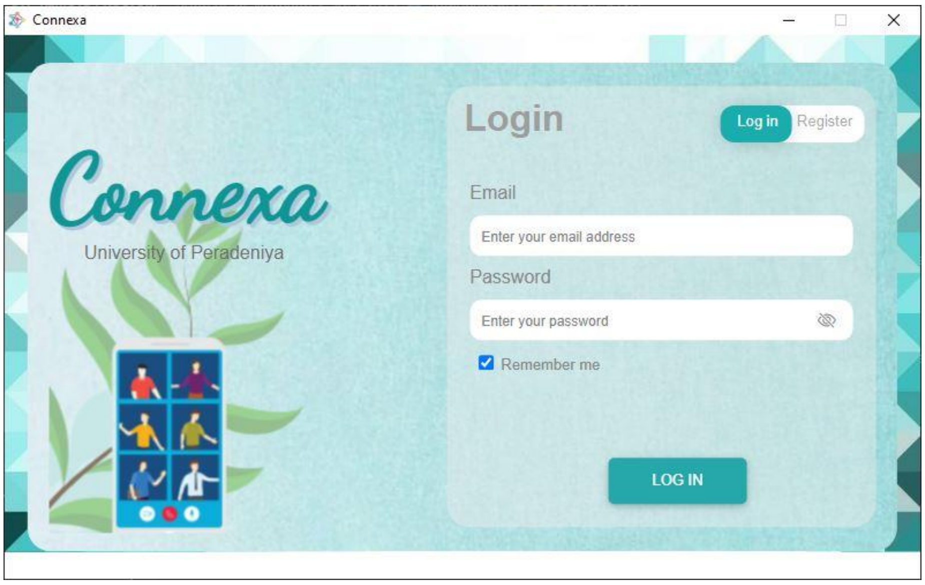
Function to upload the videos to google drive

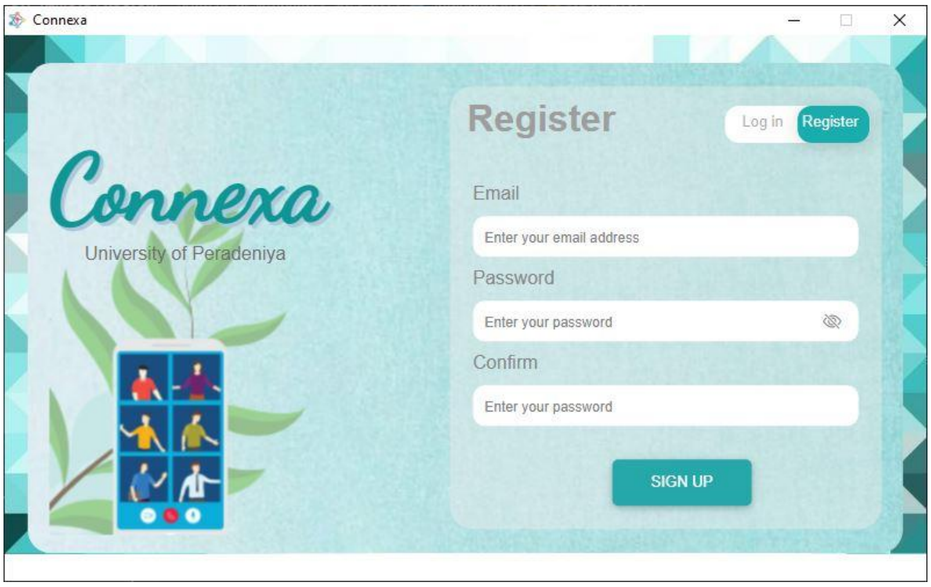
```
// method to upload videos to google drive
// upload videos to google drive
ipc.on("googleDriveUpload", async(event,{fileName, drivePath})=>{
    uploadFile(event, fileName, drivePath.split('/folders/')[1])
})

async function uploadFile(event, file, folderpath) {
try {
    const filePath = app.getPath("documents") +
"\\Connexa\\recordedVideo\\" + file;
    console.log(filePath)
    const response = await drive.files.create({
        media: {
            mimeType: 'application/octet-stream',
            body: fs.createReadStream(filePath),
            resumable: true,
        },
        resource: {
            'name': file,
            parents: [folderpath]
        },
        fields: 'id'
    });
    event.reply('done', {
        errormsg: 'noError',
    })
} catch (error) {
    event.reply('done', {
        errormsg: error.message,
    })}}
```

7.2. Pages and Features

Using Electron Js, you can build pages using HTML, Javascript, and CSS. To make HTTP requests, a Javascript library: ***Axios*** is used. Axios is a simple promise-based HTTP client for the browser and node.js. Axios provides a simple-to-use library in a small package with a very extensible interface.

1. **Login and Register Page**

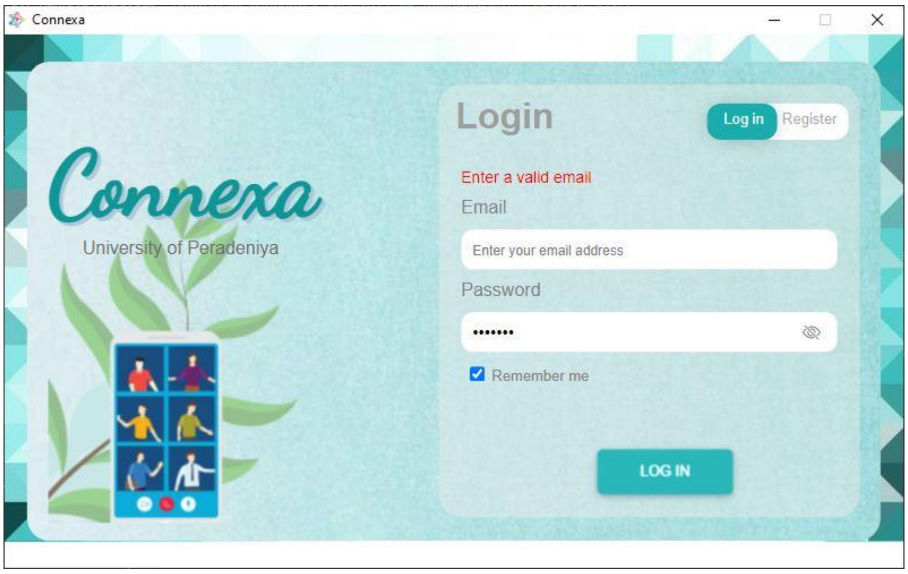




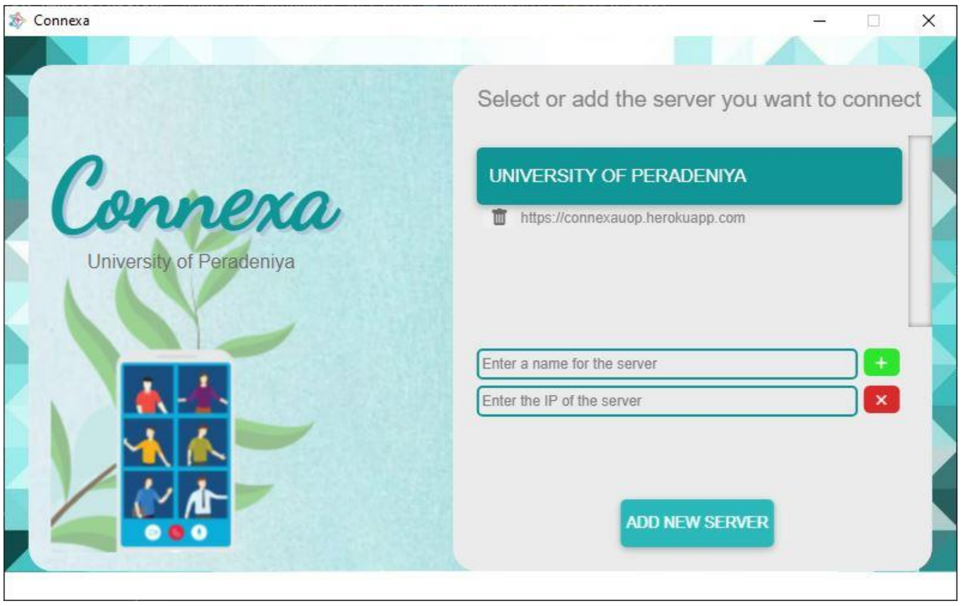
When the desktop app opens the login pages should be visible. The user input validation should be handled in the desktop app itself.

```
    if (!localStorage.serverIP) {
      logerror.innerText = "select the server you want to log in"
```

```
        return
    }

    var logemail = document.getElementById("log-email");
    if (!(validateEmail(logemail.value))) {
        logerror.innerText = "Enter a valid email";
        return;
    }

    var logpassword = document.getElementById("log-password");
    if (!(validatePassword(logpassword))) {
        logerror.innerText = "Incorrect password";
        return;
    }
    /***** save token and direct to home page *****/
    var serverIP = localStorage.getItem('serverIP')

    axios({
            method: 'post',
            url:  serverIP + '/student/login',
            responseType: 'json',
            data: {
                "email": logemail.value,
                "password": logpassword.value,
            },
        })
        .then((response) => {
            sessionStorage.setItem("email", logemail.value);
            sessionStorage.setItem('token', response.data.token);

            saveData();
        })
        .catch(function(error) {
            console.log(error)
            if (error.response) {
                logerror.innerHTML = "* " +
error.response.data.message;
            } else {
                logerror.innerHTML = "* " + error
            }
        });
```
If there's an error should be displayed as shown in the above figure.

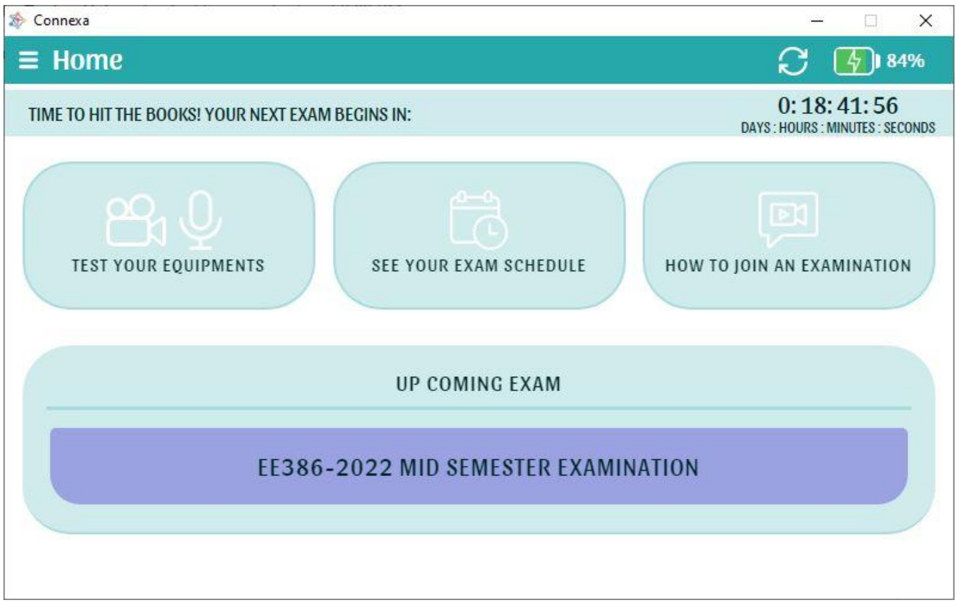


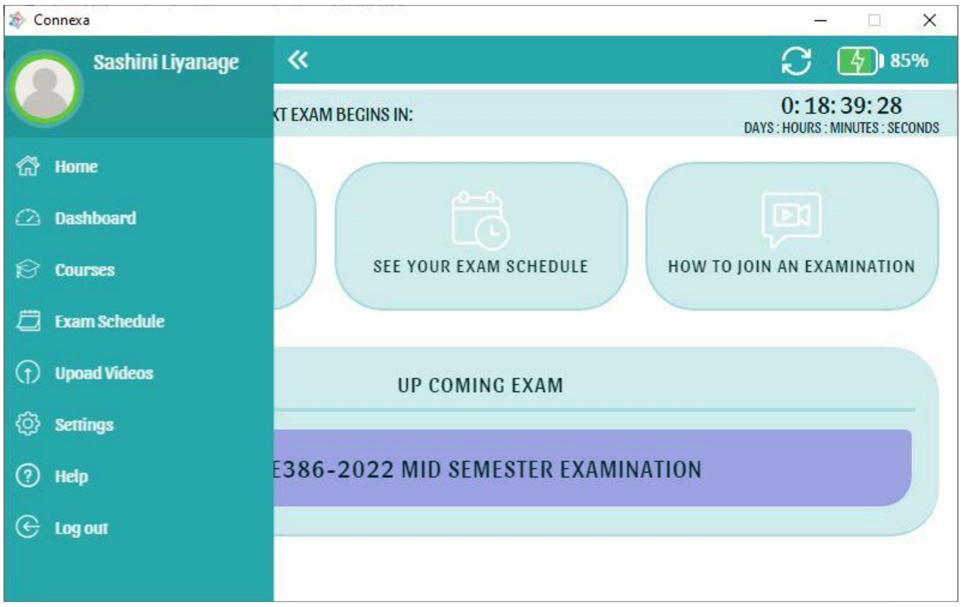
Users must be able to select the server they want to connect to. The server URL must be added as **`https://<domain_name>`** the rest of the URL is added inside the API calls of the functions. For example, in students' login API call the URL is **`Server+/student/login`**. For more information refer loginpage.js in Github repository.


Connexa
Connexa
University of Peradeniya
Select or add the server you want to connect
UNIVERSITY OF PERADENIYA
https://connexauop.herokuapp.com
Enter a name for the server
Enter the IP of the server
ADD NEW SERVER

2. **Home page**

Once the email validation and authentication are done, the login page must be directed to the Home page. The navigation bar should contain Home, Dashboard, Courses, Exam Schedule, Upload Videos, Settings, Help, and Logout tabs.


Connexa
Home
84%
TIME TO HIT THE BOOKS! YOUR NEXT EXAM BEGINS IN:
0:18:41:56
DAYS : HOURS : MINUTES : SECONDS
TEST YOUR EQUIPMENTS
SEE YOUR EXAM SCHEDULE
HOW TO JOIN AN EXAMINATION
UP COMING EXAM
EE386-2022 MID SEMESTER EXAMINATION

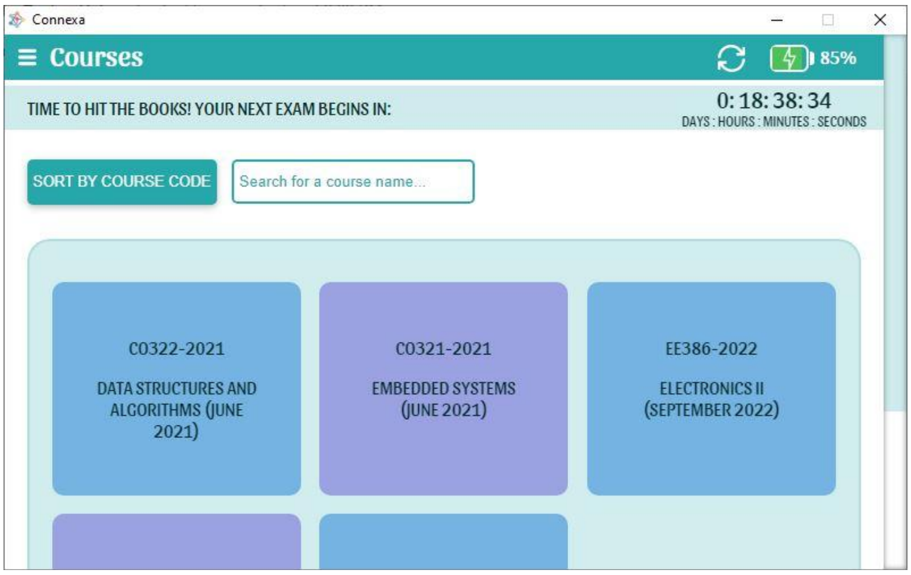


"`Time to hit the next exam`" timer counter is common on every page. The countdown for the next exam within one year of time is displayed as days, hours, minutes, and seconds. For more information refer to the nextExam.js on the Github repository.

```
function nextExam(array) {
    var nextexam, exam, examIndex = -1;
    if (array.length == 0) {
        sessionStorage.setItem('nextExamAt', '-1');
        sessionStorage.setItem('nextExam', 'no exam')
        ipc.send('home')
        return
    }
    var now = new Date()
    nextexam = nextexam = now.getTime() + 60 * 60 * 24 * 365 * 1000;

    for (var i = 0; i < array.length; i++) {
        exam = new Date(array[i][1].startTime)
        if ((exam > now) && (exam < nextexam)) {
            nextexam = exam;
            examIndex = i;
        }
    }

    if (examIndex == -1) {
        sessionStorage.setItem('nextExamAt', '-1');
        sessionStorage.setItem('nextExam', 'no exam')
        ipc.send('home')
        return
    }
    sessionStorage.setItem('nextExamAt', nextexam / 1000);
    sessionStorage.setItem('nextExam', array[examIndex][0].exam);
    ipc.send('home')
}
```

3. Course page

Course page should display all the registered courses of the student. A window with the course details should be displayed, if one course is clicked. The course details can be retrieved using **`serverIP + '/student/courses/self'`** Url. Additional sorting functionality and search options are provided for users to improve user experience. For more information refer to the courses.js on the Github repository.


Connexa

Courses

85%

TIME TO HIT THE BOOKS! YOUR NEXT EXAM BEGINS IN: 0:18:38:34 DAYS : HOURS : MINUTES : SECONDS

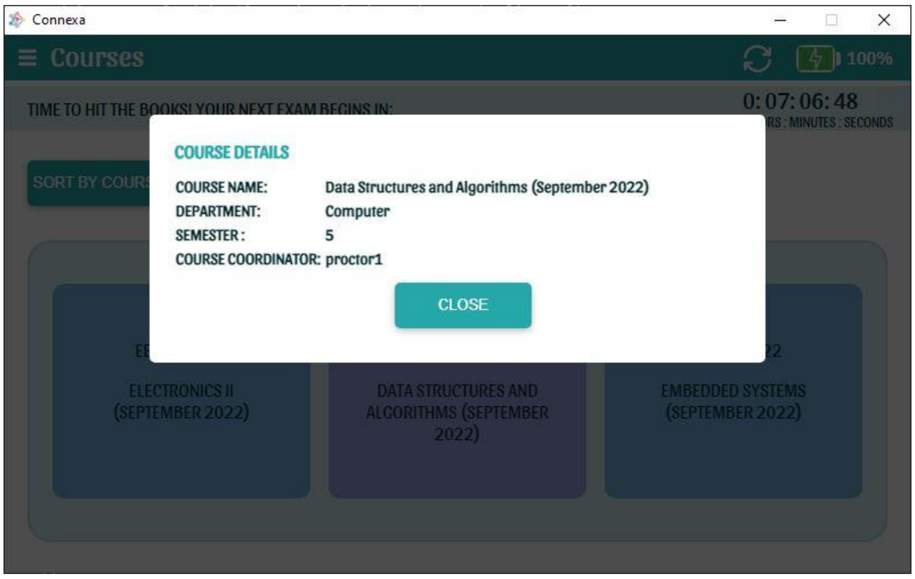
SORT BY COURSE CODE

Search for a course name...

C0322-2021 DATA STRUCTURES AND ALGORITHMS (JUNE 2021)

C0321-2021 EMBEDDED SYSTEMS (JUNE 2021)

EE386-2022 ELECTRONICS II (SEPTEMBER 2022)


Connexa

Courses

100%

TIME TO HIT THE BOOKS! YOUR NEXT EXAM BEGINS IN: 0:07:06:48

COURSE DETAILS

COURSE NAME:	Data Structures and Algorithms (September 2022)
DEPARTMENT:	Computer
SEMESTER :	5
COURSE COORDINATOR:	proctor1

CLOSE

ELECTRONICS II (SEPTEMBER 2022)

DATA STRUCTURES AND ALGORITHMS (SEPTEMBER 2022)

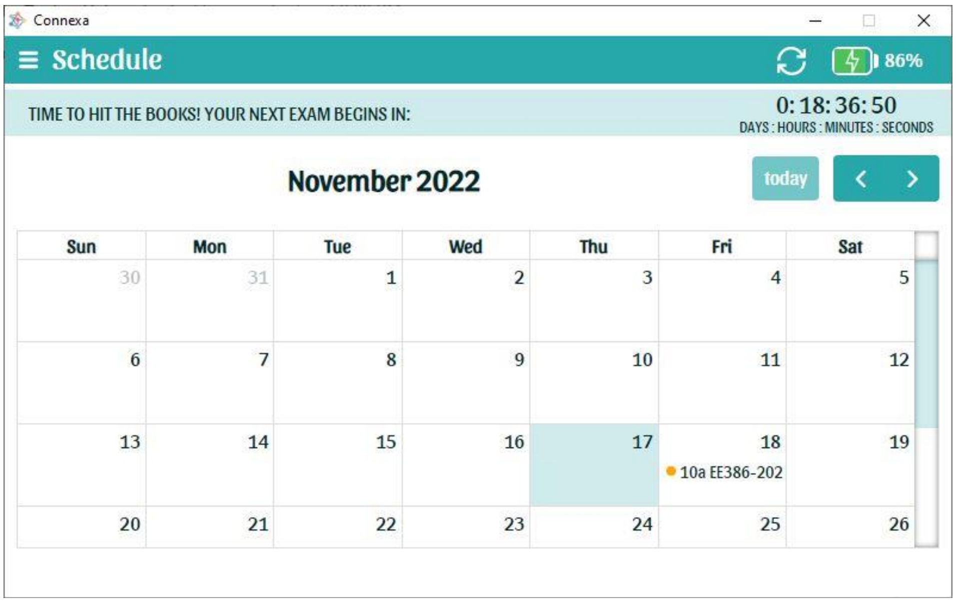
EMBEDDED SYSTEMS (SEPTEMBER 2022)

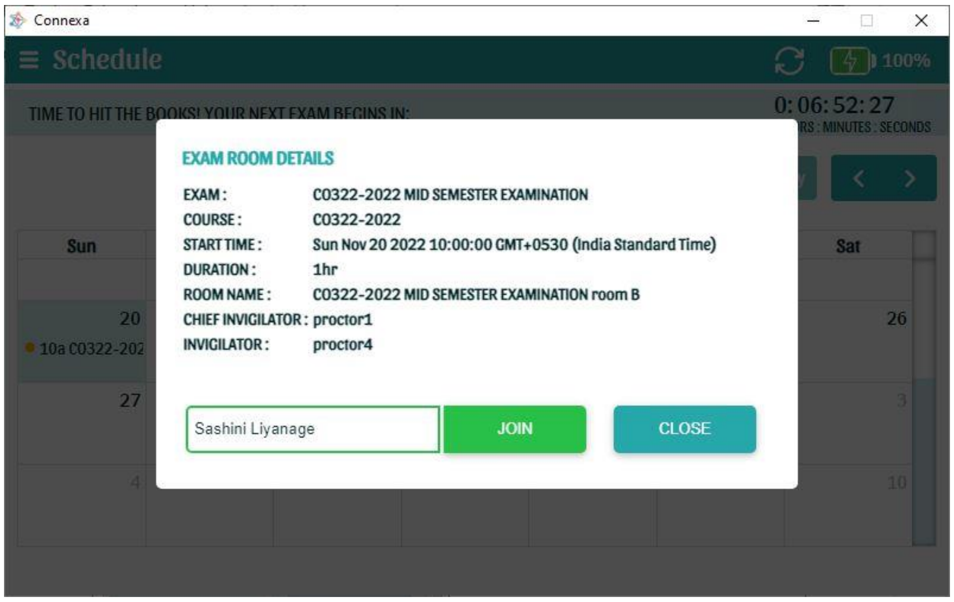
```
function findCourse() {
    var input, filter, ul, li, name, i, txtValue;
    input = document.getElementById('myInput');
    filter = input.value.toUpperCase();
    ul = document.getElementById("cards");
    li = ul.getElementsByTagName('li');

    // Loop through all list items, and hide those who don't match the
search query
    for (i = 0; i < li.length; i++) {
        name = li[i].getElementsByTagName("p")[1];
        txtValue = name.textContent || name.innerText;
        if (txtValue.toUpperCase().indexOf(filter) > -1) {
            li[i].style.display = "";
        } else {
            li[i].style.display = "none";
        }
    }
}
```

4. Exam Schedule

The Exam Schedule page displays a calendar that contains all the scheduled exams. Exam details can be retrieved using `serverIP + '/student/exams/self'` url.

The Calendar is added as a plug-in. You can find the **JavaScript Calendar** at https://fullcalendar.io/





EXAM ROOM DETAILS

EXAM :	C0322-2022 MID SEMESTER EXAMINATION
COURSE :	C0322-2022
START TIME :	Sun Nov 20 2022 10:00:00 GMT+0530 (India Standard Time)
DURATION :	1hr
ROOM NAME :	C0322-2022 MID SEMESTER EXAMINATION room B
CHIEF INVIGILATOR :	proctor1
INVIGILATOR :	proctor4

Sashini Liyanage

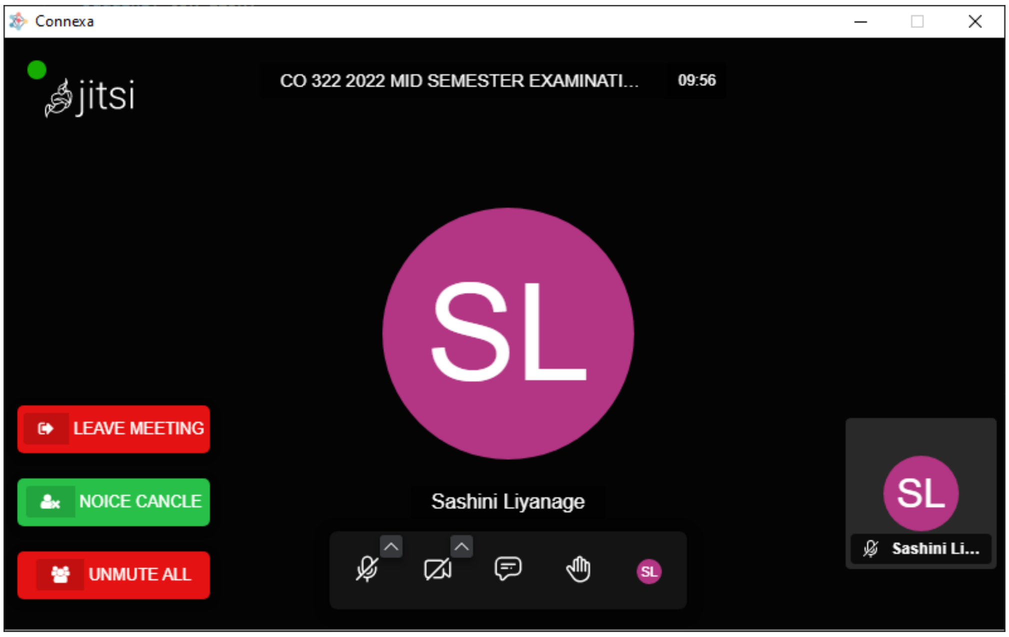
When a calendar event is clicked, a pop-up window should display the details of the examination and link to the examination. The calendar events are displayed as follows. The link to join the examination room is only displayed an hour before the exam.

```
function createEvents() {
    var calendarEl = document.getElementById('calendar');
    var calendar = new FullCalendar.Calendar(calendarEl, {
        height: 350,
        initialView: 'dayGridMonth',
        eventColor: 'orange',
        events: eventArray,
        eventClick: function(info) {
            var id = info.event.id
            span[0].innerHTML = examArray[id][0].exam
            span[1].innerHTML = examArray[id][1].course
            span[2].innerHTML = info.event.start
            span[3].innerHTML = examArray[id][1].duration
            span[4].innerHTML = examArray[id][0].room_name
            span[5].innerHTML = examArray[id][0].chief_invigilator
            span[6].innerHTML = examArray[id][0].invigilator
            displayName.value = sessionStorage.getItem('name')
            sessionStorage.setItem('roomName',
examArray[id][0].room_name)
            sessionStorage.setItem('videoPath',
examArray[id][0].recordedStudentVideosAt)
            timenow = new Date();

            if (info.event.start - now > 3600 * 1000 * 1 * 1) {
                displayName.style.display = 'none';
                joinbtn.style.display = 'none';
                msg.innerHTML = "Link will be displayed one hour
before the exam"
            } else {
                displayName.style.display = '';
                joinbtn.style.display = '';
                msg.innerHTML = ""
            }
            showevent.click()

        },
        headerToolbar: {
            left: '',
            center: 'title',
            right: 'today prev,next',
        }
    });
    calendar.render();
};
```

5. Exam Room

See the Jitsi Meet configurations for virtual video conferencing setup. The Exam room contains three buttons in addition to the Jitsi Meet in-built buttons: Leave Meeting, Noise Cancel, Unute All.

- Leave meeting button disposes of the meeting and directs to the dashboard.
- Noise Cancel button mutes all the students except the invigilator.
- Unmute All button unmutes all the students in the virtual room. See the custom functions for functional implementation.



The indicator of online/offline status is displayed in the top left corner of the screen. The button turns red when the internet is disconnected. The disconnection time is pushed into an array to send back it to the Proctors web app.

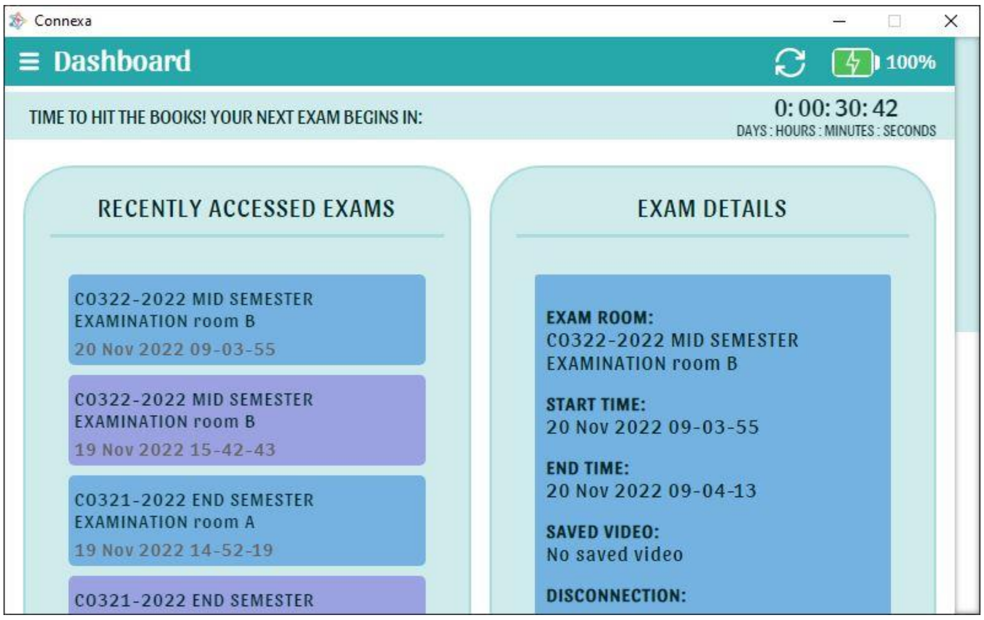
```
window.addEventListener('offline', () => {
    offlineStart = date.format(new Date(), 'MMMDD HH-mm-ss')
    offlineEnd = 0;
    record = true;
    bulb.className = 'Rec'
    mediaRecorder.resume();
})

window.addEventListener('online', () => {
    offlineEnd = date.format(new Date(), 'MMMDD HH-mm-ss')
    statusArray.push(offlineStart + " to " + offlineEnd);
    bulb.className = 'notRec'
    mediaRecorder.pause();
})
```

During the offline time, the video of the student should be recorded. Once the student is back online, the video should be paused. Refer to examRoom.js on Github repository to see the behavior of the video recorder during the offline mode. When the student clicks the Leave Meeting button, the video should be downloaded to the "`document/connexa/savedVideo`" folder and save the exam details in local storage for further reference.

```
downloadButton.addEventListener('click', () => {
    api.dispose();
    mediaRecorder.resume();
    stopRecording();
    downloadButton.style.display = 'none';
    muteButton.style.display = 'none';
    unmuteButton.style.display = 'none';
    errorMsgElement.innerHTML = "Please wait..."
    examdetails['endTime'] = date.format(new Date(), 'DD MMM YYYY HH-
mm-ss');


    if (offlineEnd == 0) {
        statusArray.push(offlineStart + " to until end");
    }
    if (record) {
        var time = date.format(new Date(), 'DD MMM YYYY HH_mm_ss');
        savedVideo = localStorage.getItem('email') + time;
        localStorage.setItem('disconnections', JSON.stringify({
"roomName": roomName, "disconnections": statusArray }));
    }
    examdetails['status'] = statusArray;
    examdetails['savedvideo'] = savedVideo;
    examdetails['videoPath'] = sessionStorage.getItem('videoPath');
    additem(examdetails);
    setTimeout(function() {
        if (record) {
            const blob = new Blob(recordedBlobs, {
                type: 'video/mp4'
            });
            const url = window.URL.createObjectURL(blob);

            ipc.send("download", {
                url: url,
                fileName: savedVideo
            })
            ipc.on("done", () => {
                ipc.send('dashboard');
            })
        } else {
            ipc.send('dashboard');
        }
     }, 3000)
```

6. Dashboard

The dashboard should display the information about the recently completed 10 exam details.



TIME TO HIT THE BOOKS! YOUR NEXT EXAM BEGINS IN:
0: 00: 30: 42
DAYS : HOURS : MINUTES : SECONDS

RECENTLY ACCESSED EXAMS

CO322-2022 MID SEMESTER EXAMINATION room B
20 Nov 2022 09-03-55

CO322-2022 MID SEMESTER EXAMINATION room B
19 Nov 2022 15-42-43

CO321-2022 END SEMESTER EXAMINATION room A
19 Nov 2022 14-52-19

CO321-2022 END SEMESTER

EXAM DETAILS

EXAM ROOM:
CO322-2022 MID SEMESTER EXAMINATION room B

START TIME:
20 Nov 2022 09-03-55

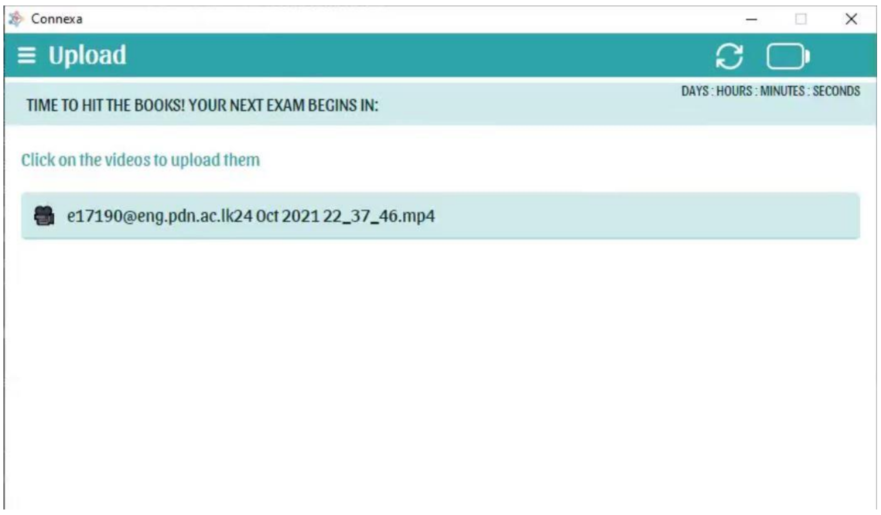
END TIME:
20 Nov 2022 09-04-13

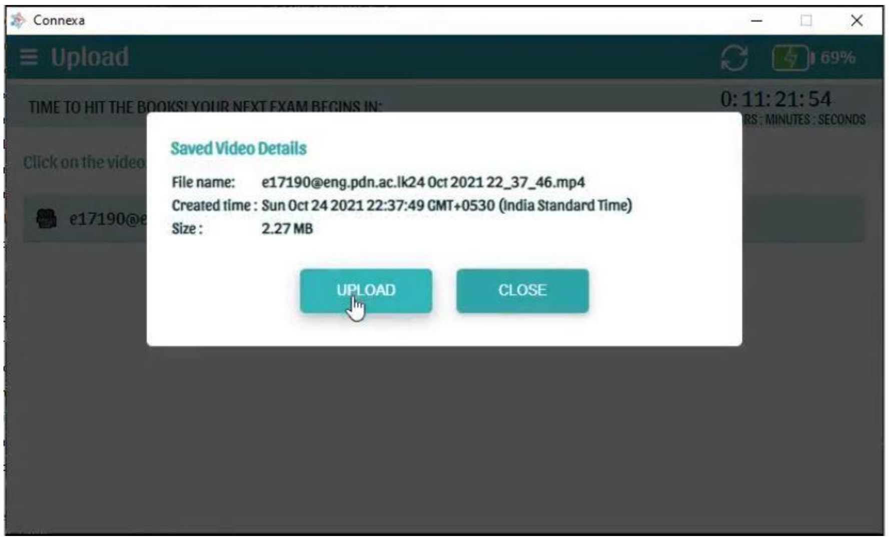
SAVED VIDEO:
No saved video

DISCONNECTION:

7. Upload videos

The Upload Videos page should display all the videos that havent been uploaded yet. If the student clicks on the list item, it should popup a window containing details on the video and a button to upload the video. The video should be uploaded to the drive folder specified in the master sheet. The drive path can be retrieved from the local storage where exam details are saved.


Connexa
Upload
TIME TO HIT THE BOOKS! YOUR NEXT EXAM BEGINS IN:
DAYS : HOURS : MINUTES : SECONDS
Click on the videos to upload them
e17190@eng.pdn.ac.lk24 Oct 2021 22_37_46.mp4



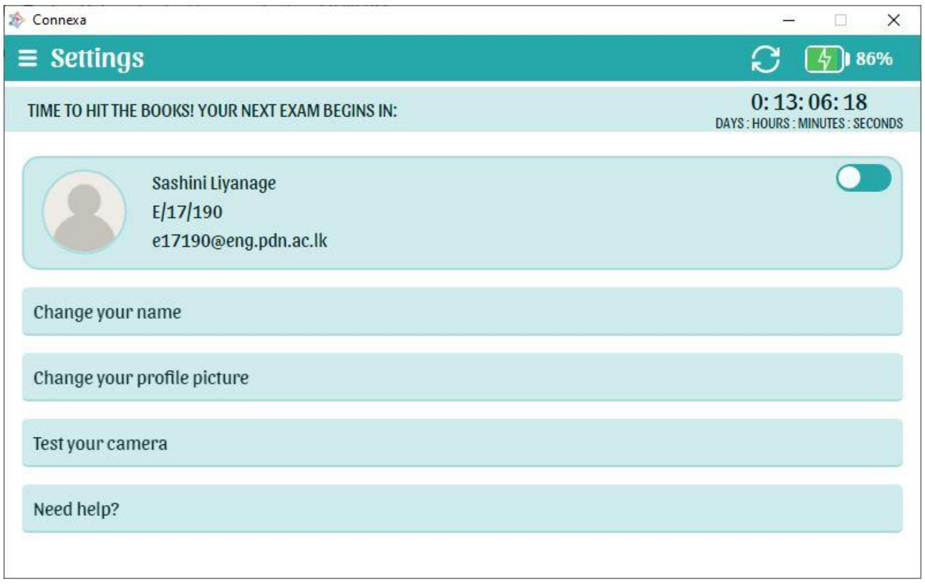
Saved Video Details

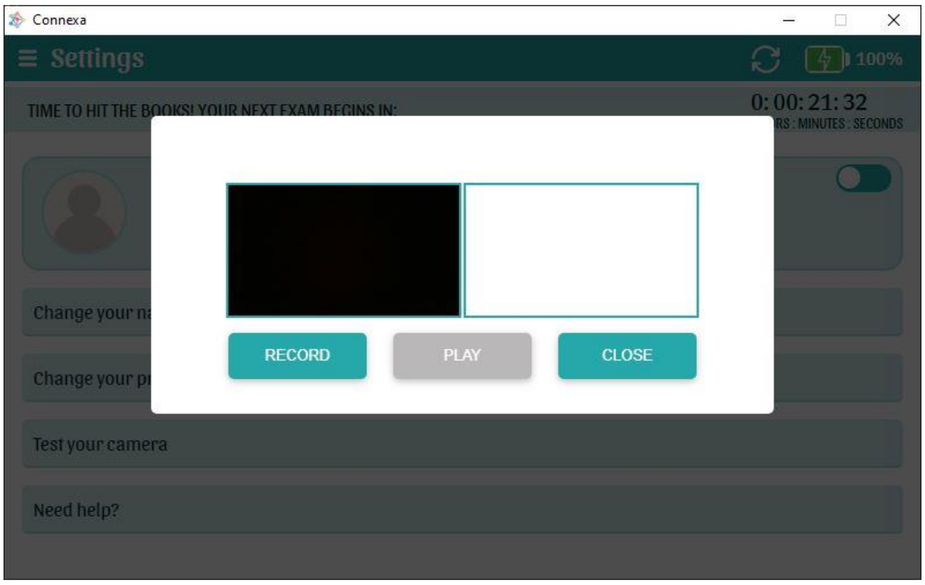
File name: e17190@eng.pdn.ac.lk24 Oct 2021 22_37_46.mp4
Created time : Sun Oct 24 2021 22:37:49 GMT+0530 (India Standard Time)
Size : 2.27 MB

```
function upload() {
    if (!(navigator.onLine)) {
        closePopup();
        return;
    }

    var filename = document.getElementById('filename').innerHTML;
    var array = JSON.parse(localStorage.getItem('examdetails'));
    var drivepath;

    uploadbtn.disabled = true;
    canclebtn.disabled = true;
    errorMsg.innerHTML = 'Uploading....';
    uploadStatus.style.display = 'block'
    uploadStatus.src = 'img/icons/uploading.gif';

    for (var i = 0; i < array.length; i++) {
        if (array[i].savedvideo + '.mp4' === filename) {
            drivepath = array[i].videoPath
        }

    }

    console.log(drivepath)
    ipc.send("googleDriveUpload", {
        fileName: filename,
        drivePath: drivepath,
    })

    ipc.on("done", async(event, { errormsg }) => {
        canclebtn.disabled = false;
        if (errormsg == 'noError') {
            errorMsg.innerHTML = "Video Uploaded Successfully";
            uploadStatus.src = 'img/icons/success.png'

            fs.unlink(filepath + filename, function(err) {
                console.log(err);
            });
        } else {
            errorMsg.innerHTML = errormsg;
            uploadStatus.src = 'img/icons/fail.png'
        }
    })
}
```

8. Settings

The settings page provides options to change the user name and the profile picture. The Student can check the video and audio using the “Test your camera” option. It should pop up a window as shown below to record a video and check the audio and video quality. Refer to Settings.js for more information.

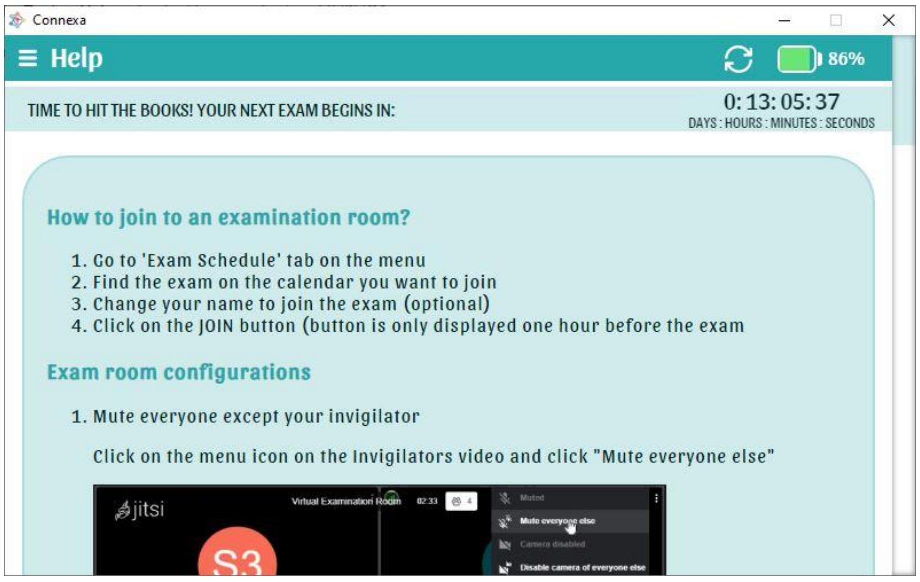




9. Help

Help page includes frequently asked questions:

- How to join an examination room
- Exam room configurations
- How to upload saved videos
- How to test audio and video
- How to adjust sound and brightness
- What to do when battery supply is low
- Contact details


Connexa

≡ Help

86%

TIME TO HIT THE BOOKS! YOUR NEXT EXAM BEGINS IN: 0: 13: 05: 37 DAYS : HOURS : MINUTES : SECONDS

How to join to an examination room?

1. Go to 'Exam Schedule' tab on the menu
2. Find the exam on the calendar you want to join
3. Change your name to join the exam (optional)
4. Click on the JOIN button (button is only displayed one hour before the exam

Exam room configurations

1. Mute everyone except your invigilator

Click on the menu icon on the Invigilators video and click "Mute everyone else"

7.3. Testing

1. **Set up Test Environment**

   Spectron is used for testing the Electron application. **Spectron** is an open source framework for easily writing automated integrations and end-to-end tests for Electron. It provides capabilities for navigating on web pages, simulating user input, and much more. It also simplifies the setup, running, and tearing down of the test environment for Electron.

```
$ npm install --save-dev spectron
```

2. **Add third-party test runner**

   Spectron has the capability to start an Electron application and control it from its tests. However it does not have a framework of its own for running the tests. But it can be used with any testing library like Mocha, Jasmine, etc. Here we make use of mocha, javascript test framework which runs on Node.js

```
$ npm i mocha
```

3. **Write test codes**

   The following is the sample test code for login to the application. See more test cases in test.js on Github repository.

```
const Application = require('spectron').Application
const assert = require('assert')
const electronPath = require('electron')
const path = require('path')

describe('Application launch', function() {
    this.timeout(100000)
    let app;

    before(function() {
        app = new Application({
            path: electronPath,
            args: [path.join(__dirname, '..')]
        })
        return app.start()
    })

    after(function() {
        if (app && app.isRunning()) {
            return app.stop()
        }
    })

    it('Log in', async() => {
        const input = await app.client.$('#log-email');
        await input.setValue('e17190@eng.pdn.ac.lk');
        await sleep(1000);
        const input2 = await app.client.$('#log-password');
        await input2.setValue('sashini1234');
        await sleep(1000);
        element = await app.client.$("button[name='Log in']")
        await element.click();
        var element = title = await app.client.$("#title");
        if (!element) await sleep(5000);
        element = title = await app.client.$("#title");
        h1Text = await title.getText();
        assert.equal(h1Text, "Home")
        await sleep(3000);
    })
```

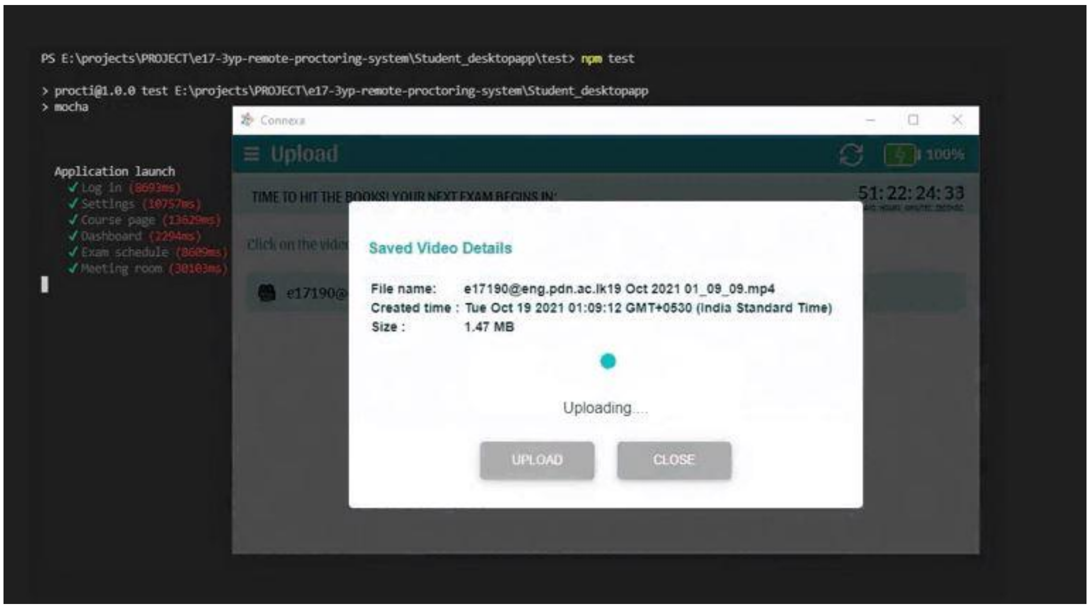
4. Test the application

Add the following to the script in the package.json file.

```
"scripts": {
    "test": "mocha"
}
```

Then run npm test to run the tests.

```
$ npm test
```



```
PS E:\projects\PROJECT\e17-3yp-remote-proctoring-system\Student_desktopapp\test> npm test

> procti@1.0.0 test E:\projects\PROJECT\e17-3yp-remote-proctoring-system\Student_desktopapp
> mocha


  Application launch
    ✓ Log in (8693ms)
    ✓ Settings (10757ms)
    ✓ Course page (13629ms)
    ✓ Dashboard (2294ms)
    ✓ Exam schedule (8609ms)
    ✓ Meeting room (30103ms)
```

Saved Video Details

File name: e17190@eng.pdn.ac.lk19 Oct 2021 01_09_09.mp4
Created time : Tue Oct 19 2021 01:09:12 GMT+0530 (India Standard Time)
Size : 1.47 MB

Uploading....

8. Web application

React framework is used for creating the web application for the proctor. An admin portal is also accessible from the web application.

8.1. Create React Application

You'll need to have Node >= 14.0.0 and npm >= 5.6 on your machine. To create a project, run:

```
$ npx create-react-app proctor-app
$ cd my-app
$ npm start
```

1. Go to public folder and change the app icon and title in index.html. Include flowing script in index.html for Jitsi meet integration.

```
<script src='https://meet.jit.si/external_api.js'></script>
```

2. Go to the src folder and create two folders for Components and Css.
3. Install material UI, an open-source React component library which includes a comprehensive collection of pre built components.

```
$ npm install @mui/material @emotion/react @emotion/styled
```

4. Add pages to the App.js as components

```
import Login from './components/Login';
import Register from './components/Register';
import Portal from './components/Portal';
import Adminlogin from './components/Adminlogin';
import Adminhome from './components/Admin/Adminhome';
import Home from './components/Home';
import Dashboard from './components/Dashboard';
import Schedule from './components/Schedule';
import Courses from './components/Courses';
import Settings from './components/Settings';
import Help from './components/Help';
import Meeting from './components/Meeting';
import Adminregister from './components/Adminregister';
import CoursePage from './components/CoursePage';
import AdminSettings from './components/Admin/AdminSettings';
import Addexam from './components/Admin/Addexam'
import Database from './components/Admin/Database';
import AdminHelp from './components/Admin/AdminHelp';
```

5. Add route protection for pages in App.js

```
import { BrowserRouter as Router, Route, Switch } from
'React-router-dom';
import RouteProtection from './components/RouteProtection'
function App() {return (
    <div className="App">
     <Router>
        <Switch>
          <Route path="/" exact><Portal/></Route>
          <Route path="/adminsignin" exact><Adminlogin/></Route>
          <Route path="/adminreg" exact><Adminregister/></Route>
          <RouteProtection path="/admin/home" exact
component={Adminhome} user="admin"></RouteProtection>
         <RouteProtection path='/admin/adddata' exact
component={Addexam} user="admin"></RouteProtection>
          <RouteProtection path='/admin/removedata' exact
component={Database} user='admin'></RouteProtection>
          <RouteProtection path="/admin/settings" exact
component={AdminSettings} user='admin'></RouteProtection>
          <RouteProtection path='/admin/help' exact
component={AdminHelp} user="admin"></RouteProtection>
          <Route path="/register" exact><Register/></Route>
          <Route path="/signin" exact><Login /></Route>
          <RouteProtection path="/home" component={Home}
user="proctor"></RouteProtection>
          <RouteProtection path="/dashboard" component={Dashboard}
user="proctor"></RouteProtection>
          <RouteProtection path="/schedule" component= {Schedule}
user="proctor"></RouteProtection>
          <RouteProtection path="/courses" exact component={Courses}
user="proctor"></RouteProtection>
```

```
          <RouteProtection path="/courses/:courseId" exact
component={CoursePage} user="proctor"/>
          <RouteProtection path="/settings" exact component={Settings}
user="proctor"></RouteProtection>
          <RouteProtection path="/help" exact component={Help}
user="proctor"></RouteProtection>
          <RouteProtection  path="/meeting" exact component={Meeting}
user="proctor"></RouteProtection >
        </Switch>
      </Router>
    </div>
  );
}
```

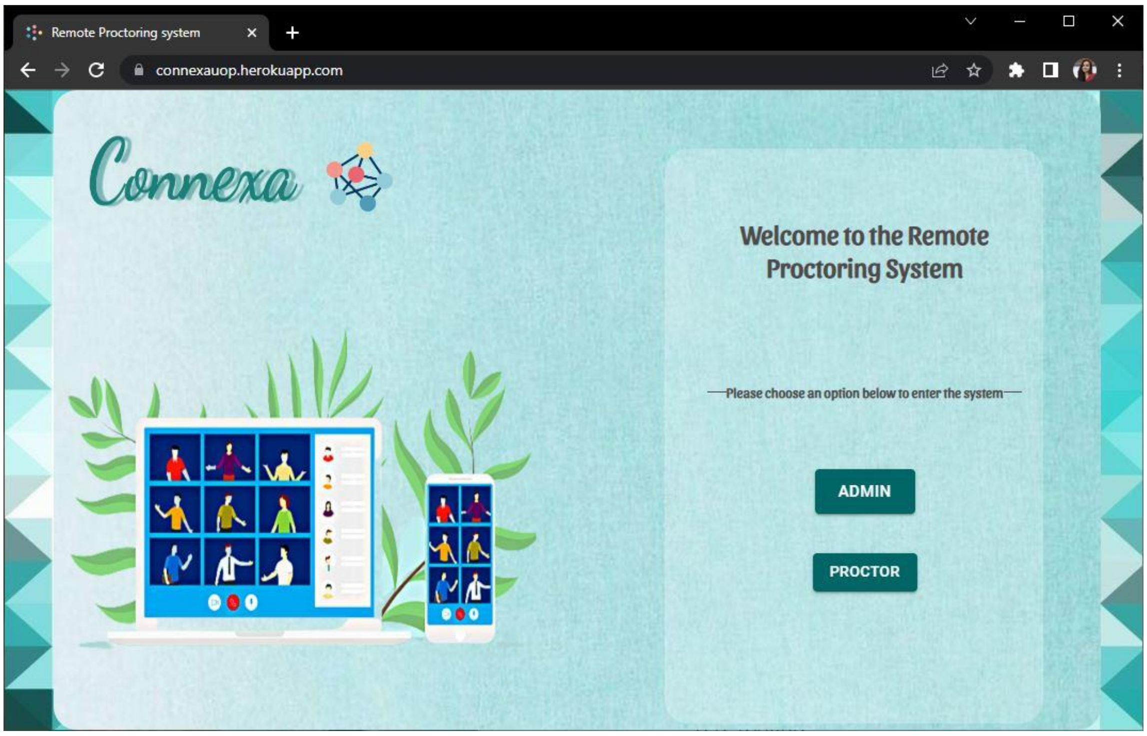
6. Add following route protection inside `src/components/RouteProtection.js`

```
export default function ProtectedRoutes({ component: Component,
...rest }) {
    const key = rest.user ==='proctor'?
localStorage.getItem("ptoken"):localStorage.getItem("atoken")
  return (
    <Route {...rest} render={(props) => {
        const token = key;
        if (token) {
          return <Component {...props} />;
        } else {
          return (
            <Redirect to={{ pathname: "/", state: { from:
props.location,},}}/>
          );
        }
      }}
    />
  );
}
```

8.2. Pages and Features

1. Portal

The Admin button should direct the page to the Admin sign-in page and the Proctor button should direct to Proctor's sign-in page.


Remote Proctoring system
connexauop.herokuapp.com
Connexa
Welcome to the Remote Proctoring System
Please choose an option below to enter the system
ADMIN
PROCTOR

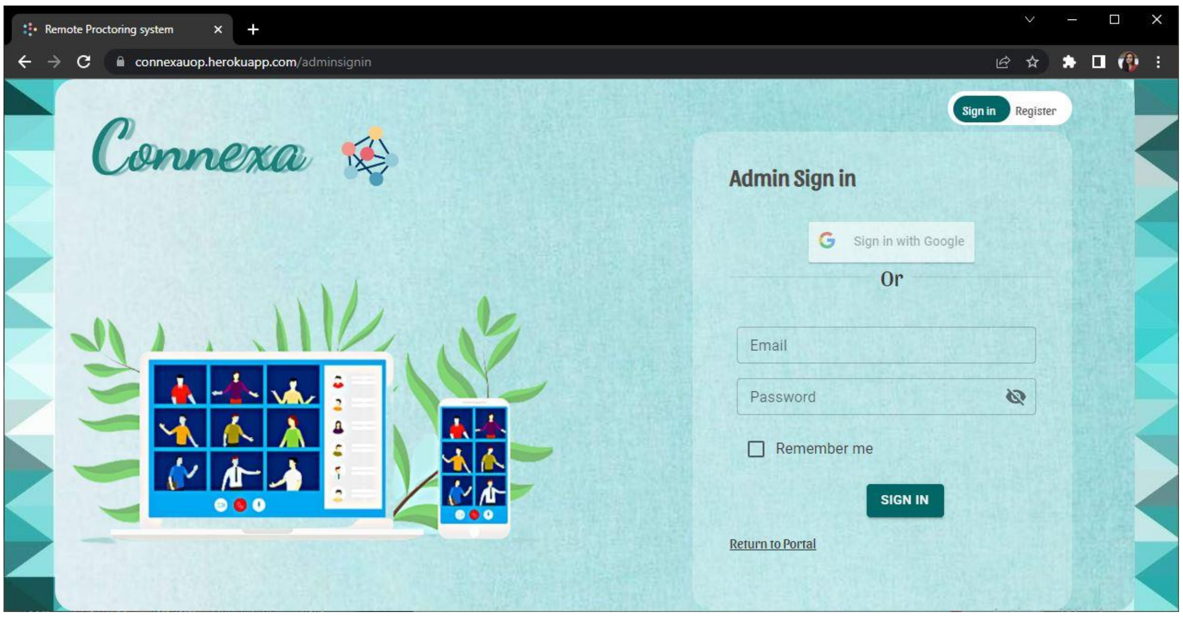
```
const Portal = () => {
    return (
         <div className="Signin-Main">
         <div className="proctorlogin">
         <div style={{paddingTop:"55px"}}></div>
         <div className = 'lbox'>
           <div className='box-title'>
           <div class="container">
                   <span class="react-logo">
                       <span class="nucleo"><img className="connexa"
src={logo} alt="logo"/></span>
                   </span>
           </div>
           </div>
           <div className='box-item'>
             <h4 style={{textAlign:"center"}}>Welcome to the Remote
Proctoring System</h4>
```

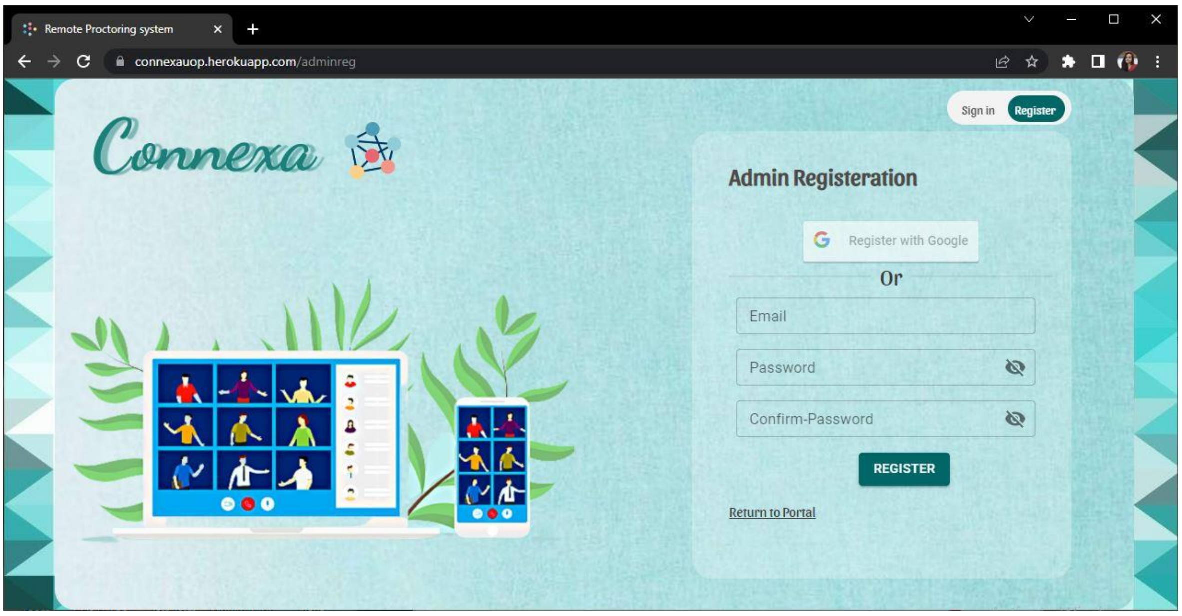
```
              <div style={{textAlign:'center'}}>
              <h6> Please choose an option below to enter the system
</h6>
                 <div className="options">
                 <Portalbtn property={'/adminsignin'} btname =
{'ADMIN'} size= {'large'}/>
                 </div>
                 <div className="options">
                 <Portalbtn property={'/signin'} btname = {'PROCTOR'}/>
                 </div>
              </div>
              <div>
             </div>
         </div>
         </div>
         </div>
    )
}
```

2. For Admin

2.1. Login and Registration

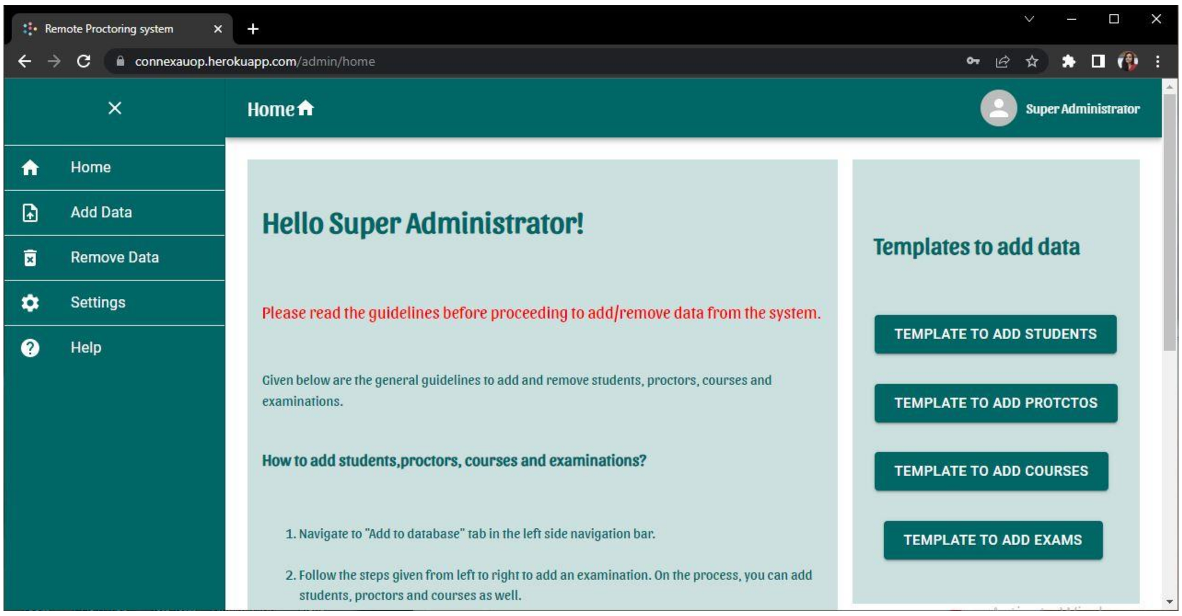
User input validations should be handled on the app itself.





2.2. Admin Home page

After authentication and authorization the valid user should be directed to the Admins Home page. The navigation bar should contain Home, Add Data, Remove Data, Settings, and Help tabs.



Hello Super Administrator!

Please read the guidelines before proceeding to add/remove data from the system.

Given below are the general guidelines to add and remove students, proctors, courses and examinations.

How to add students,proctors, courses and examinations?

1. Navigate to "Add to database" tab in the left side navigation bar.

2. Follow the steps given from left to right to add an examination. On the process, you can add students, proctors and courses as well.

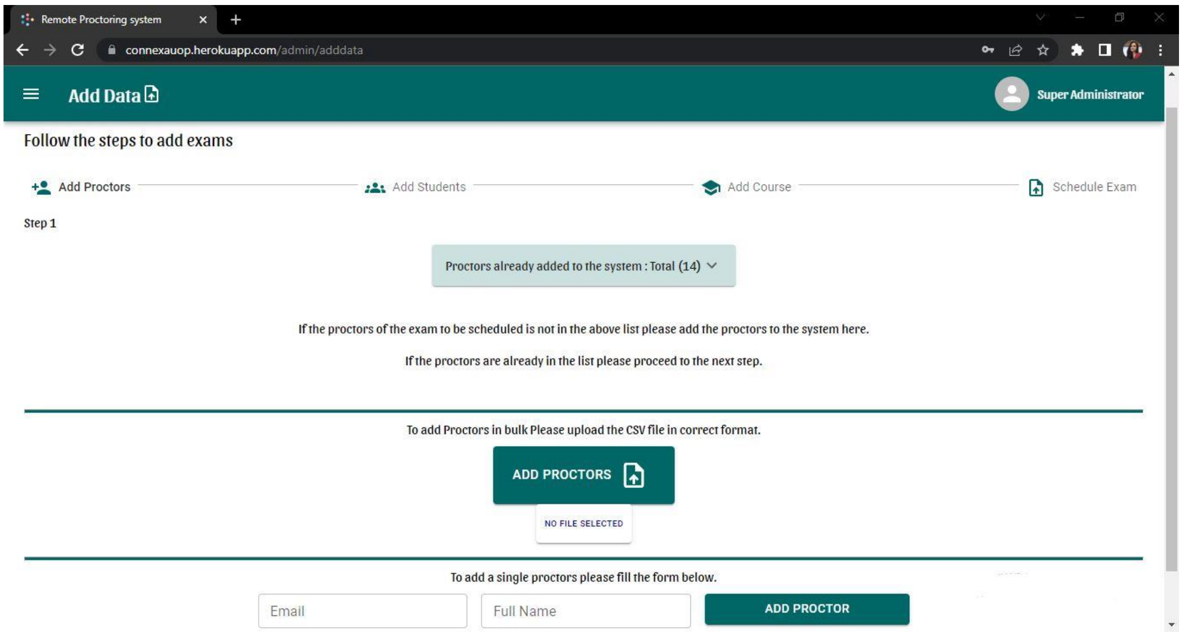
The Home page displays instructions for the Administrator and sample templates to add the master CSV files.

```
<Button color="neutral" variant="contained" component="span"
size="large" >
```

```
          <Link to={{pathname:"<path_to_drive_file>" }}
target="_blank" style={{ textDecoration: 'none',
color:"white"}}>Template to add proctors</Link>
</Button>
```

2.3. Add Data

This page is to add Proctors, Students, Courses, and Exam schedules to the system using master sheets. A drop-down list is provided to show the already added data.



Follow the steps to add exams

Step 1

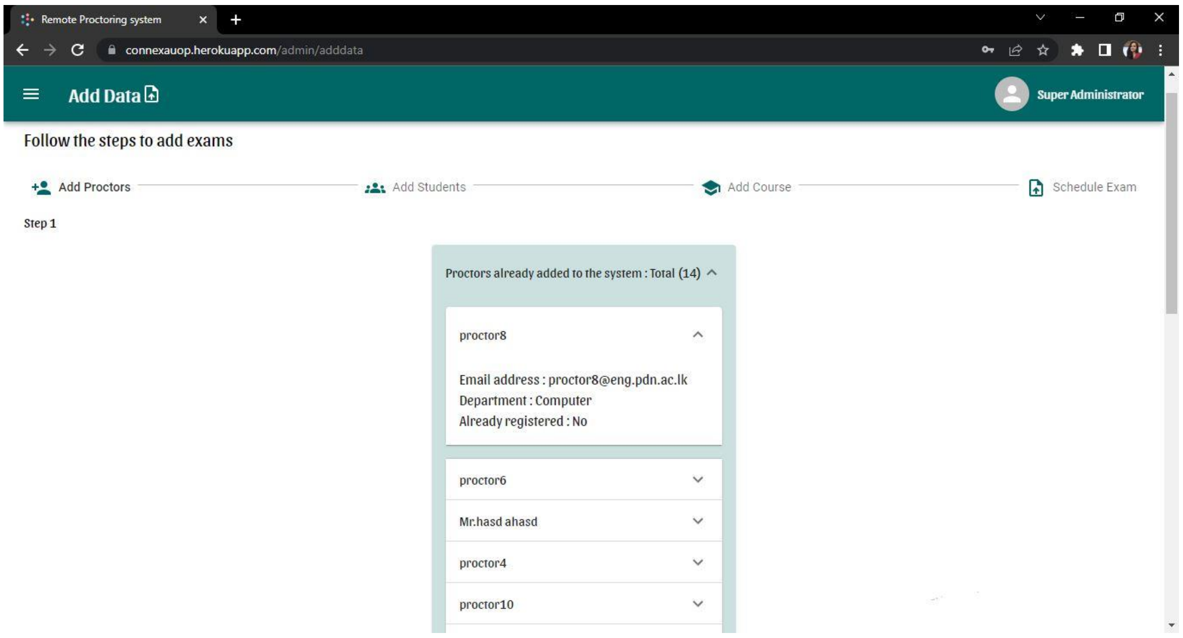
Proctors already added to the system : Total (14)

If the proctors of the exam to be scheduled is not in the above list please add the proctors to the system here.

If the proctors are already in the list please proceed to the next step.

To add Proctors in bulk Please upload the CSV file in correct format.

To add a single proctors please fill the form below.



Follow the steps to add exams

Step 1

Proctors already added to the system : Total (14)

proctor8

Email address : proctor8@eng.pdn.ac.lk
Department : Computer
Already registered : No

proctor6

Mr.hasd ahasd

proctor4

proctor10

Adding students, proctors, courses, and exams in bulk can be done by uploading properly formatted .CSV files. Uploading the .CSV file to the system, parsing the

CSV and converting it to a JSON file and transferring the JSON file to the database is done in the functions implemented in the Admin's upload button component, which can be found in Adminbtn.js in the components folder.

Some of the most important packages used in this component are given below.

```
import axios from 'axios';
import Papa from "papaparse";
```

The papaparse package is used to parse the .CSV file and convert it to JSON while axios is used to upload the file content as a POST request.

As this button component was repeatedly used to upload .CSV files containing data of four categories, separate states are introduced to identify data of which category (student, proctor, courses, exams) is being uploaded.

```
this.state = {
    selectedFile: undefined,
    FileName: "No File Selected",
    currentFile: undefined,
    progress: 0,
    message: "",
    isError: false,
    fileInfos: [],
    exams:'',
    courses:'',
    students:'',
    proctors:'',
    success:''
  };
```

Recognizing whether the user has selected a file to upload is done by the `fileSelectedHandler` function.

```
fileSelectedHandler = event => {
    if(event.target.files[0]){
       this.setState({
         selectedFile: event.target.files[0],

         FileName:  this.upload+event.target.files[0].name
       })
     }}
```

FileName is updated only if the user has selected a file from their machine. Clicking the upload button must invoke the `fileUploadHandler` function if the FileName is not null.

```
fileUploadHandler = ()=>{

    const fd = new FormData();
    if(this.state.FileName!=="No File Selected"){
    fd.append(this.id,
this.state.selectedFile,this.state.selectedFile.name);
    Papa.parse(this.state.selectedFile,
      {
       complete: this.uploadData            })
       }
    else{
      this.setState({
          message: "Please select a file to upload"
      })
    }
  }
```

The details of the file such as file id, selected file and file name are converted to form data using the FormData function and the .CSV content in the selected file is converted to JSON using the Papa.parse() function as shown in the code snippet above.

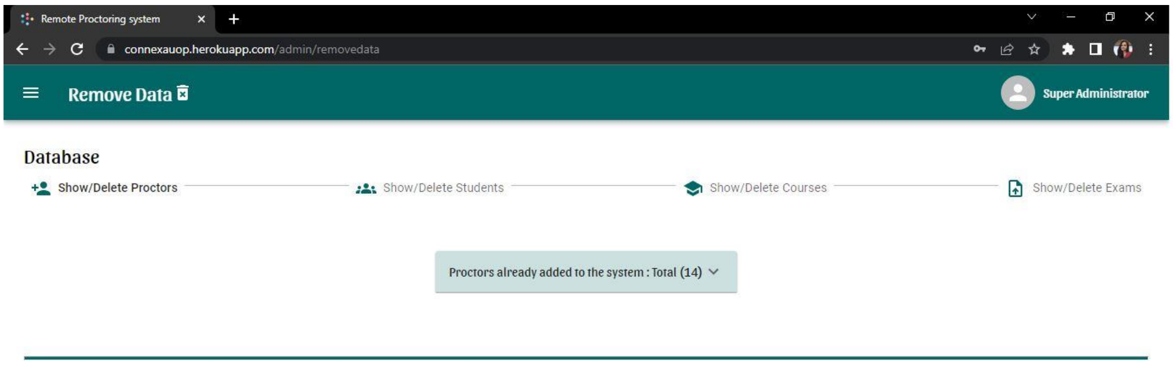
Finally, the file will be uploaded to the relevant database collection (student, proctor, course, exam) with the help of the state variables, using a POST request in the uploadData() function.

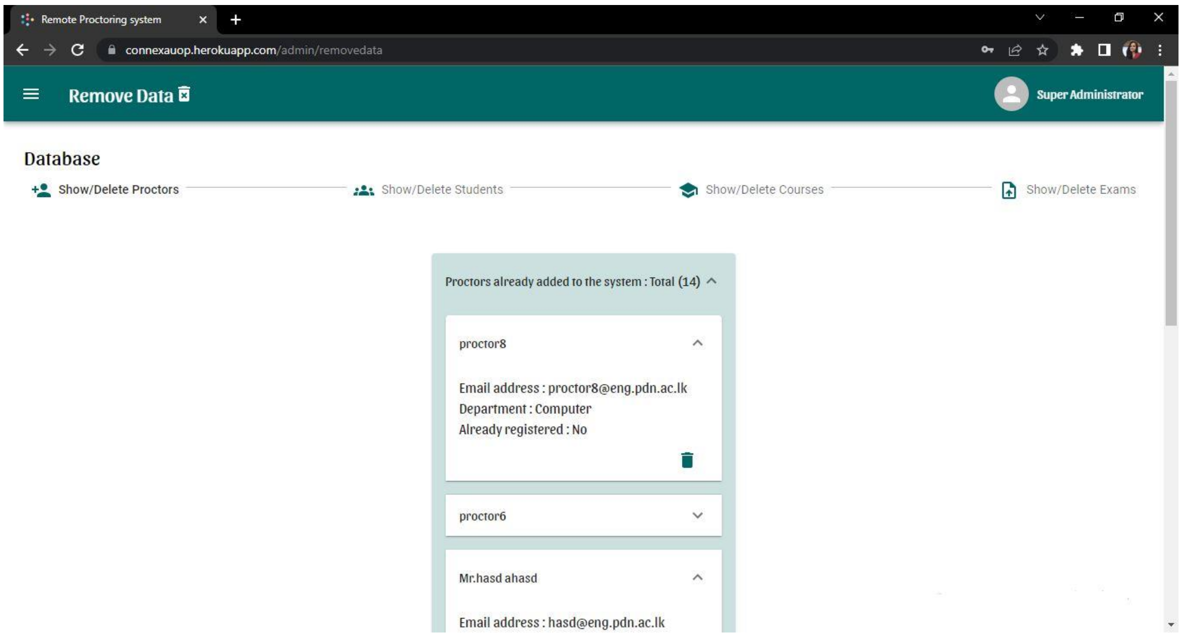
Further details can be found in the Adminbtn.js file in the Github repository.

2.4. Remove Data

Admin should be able to remove Proctors, Students, Courses, and Exam schedules. The already existing list is shown as a drop down list with the detials. By clicking th delete icon an http Delete request should be sent to the corrosponding url.

For example: delete student
Url - *admin/students/single/<email of the deleted student comes here>*





proctor8

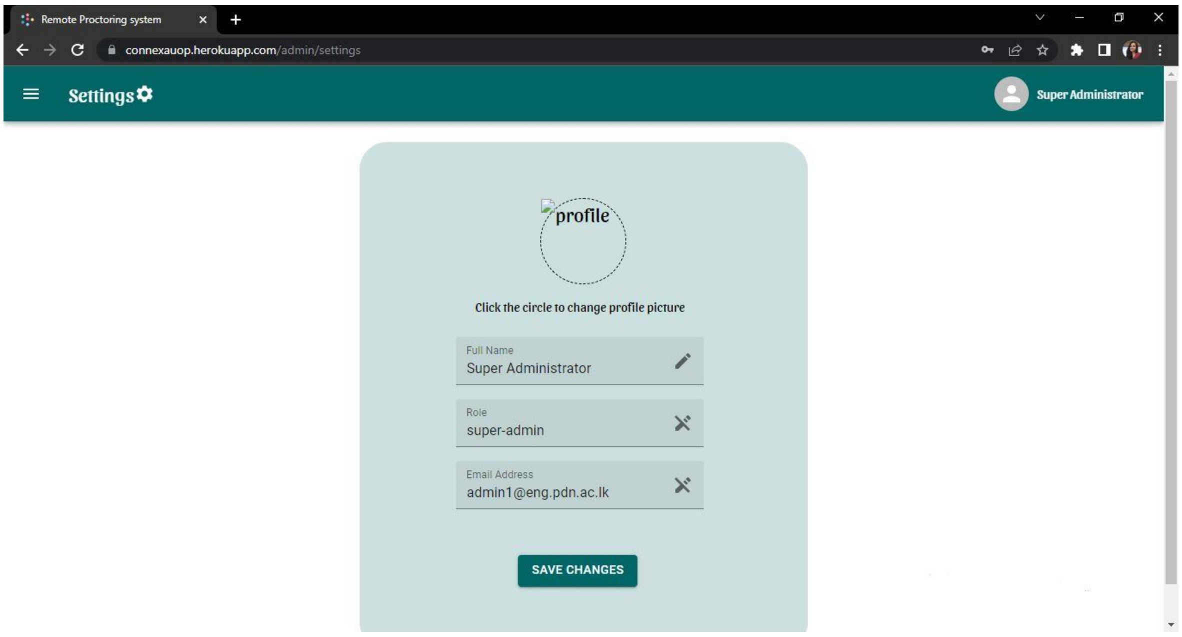
Email address : proctor8@eng.pdn.ac.lk
Department : Computer
Already registered : No

proctor6

Mr.hasd ahasd

Email address : hasd@eng.pdn.ac.lk

2.5. Settings



Remote Proctoring system

connexauop.herokuapp.com/admin/settings

Settings

Super Administrator

profile

Click the circle to change profile picture

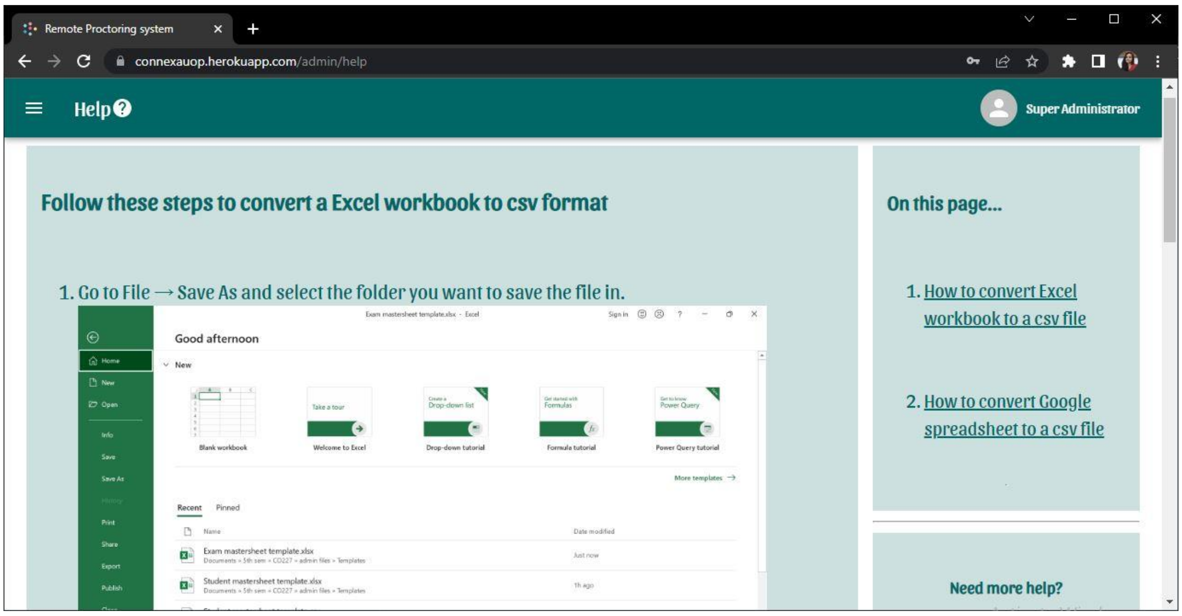
Full Name: Super Administrator

Role: super-admin

Email Address: admin1@eng.pdn.ac.lk

SAVE CHANGES

2.6 Help



Remote Proctoring system

connexauop.herokuapp.com/admin/help

Help

Super Administrator

Follow these steps to convert a Excel workbook to csv format

1. Go to File → Save As and select the folder you want to save the file in.

On this page...

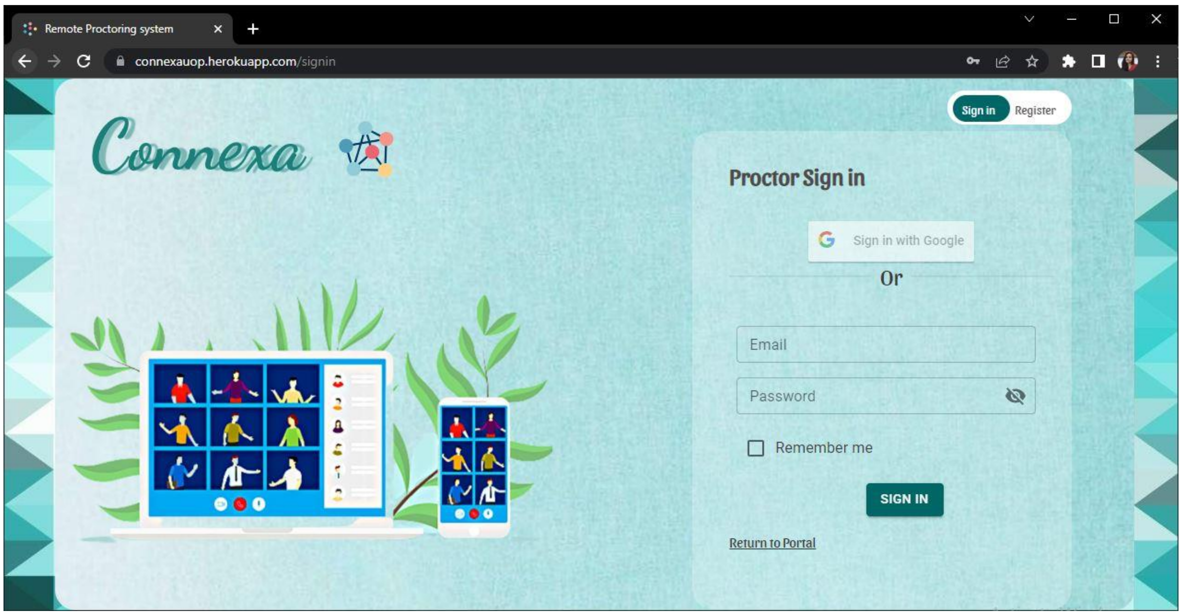
1. How to convert Excel workbook to a csv file

2. How to convert Google spreadsheet to a csv file

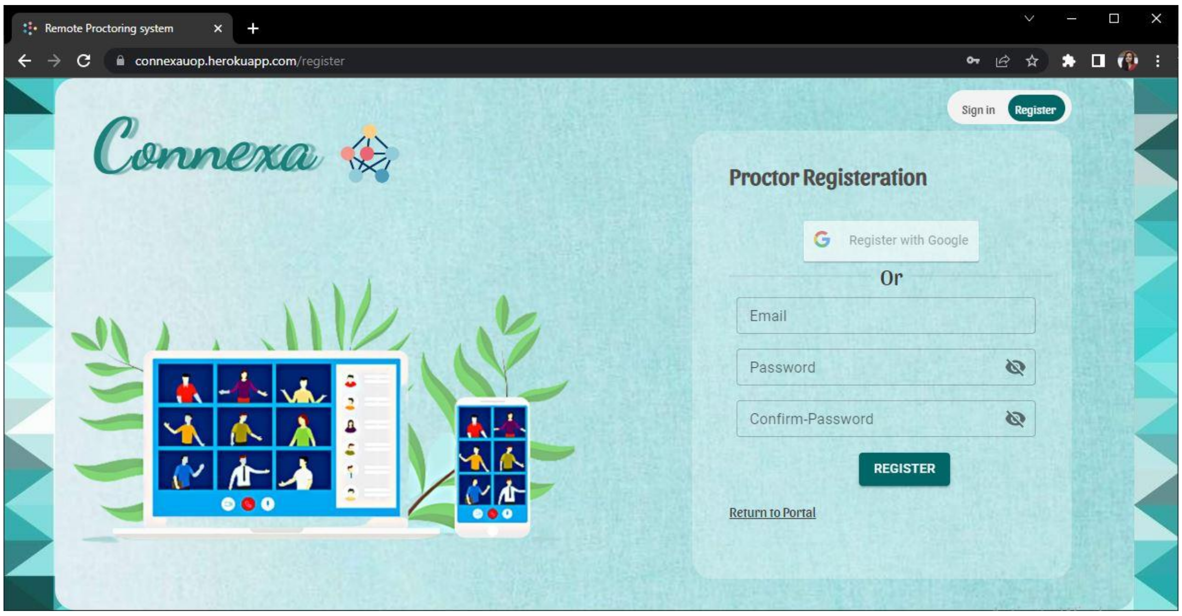
Need more help?

3. For Proctor

3.1. Login and Registration

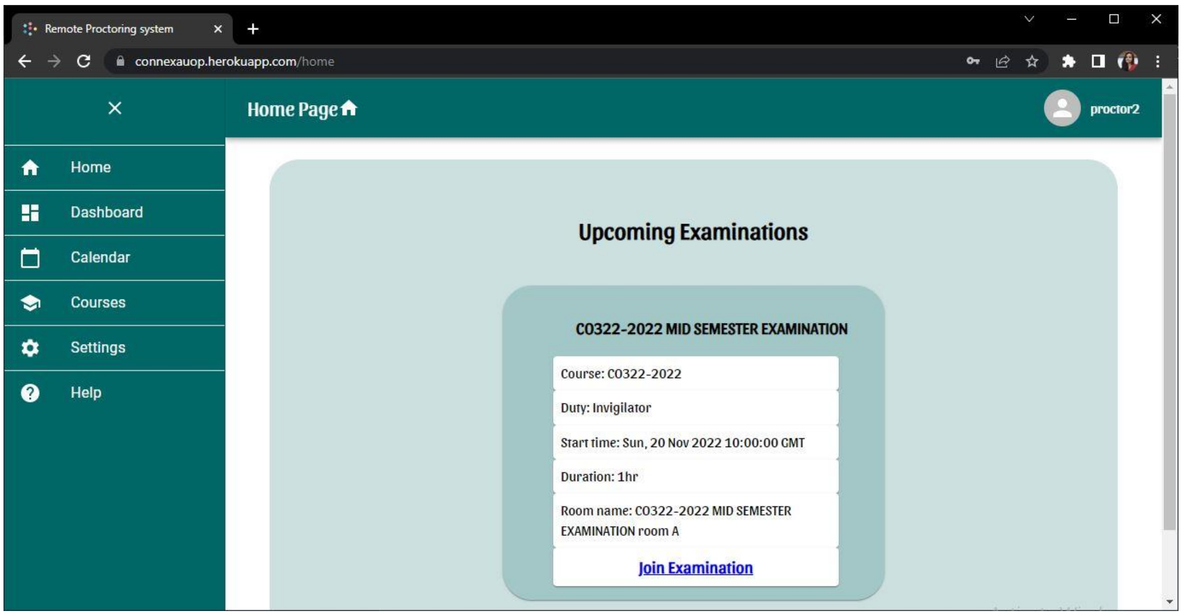


Proctor Sign in



Proctor Registeration

3.2. Home



Upcoming Examinations

C0322-2022 MID SEMESTER EXAMINATION

Course: C0322-2022

Duty: Invigilator

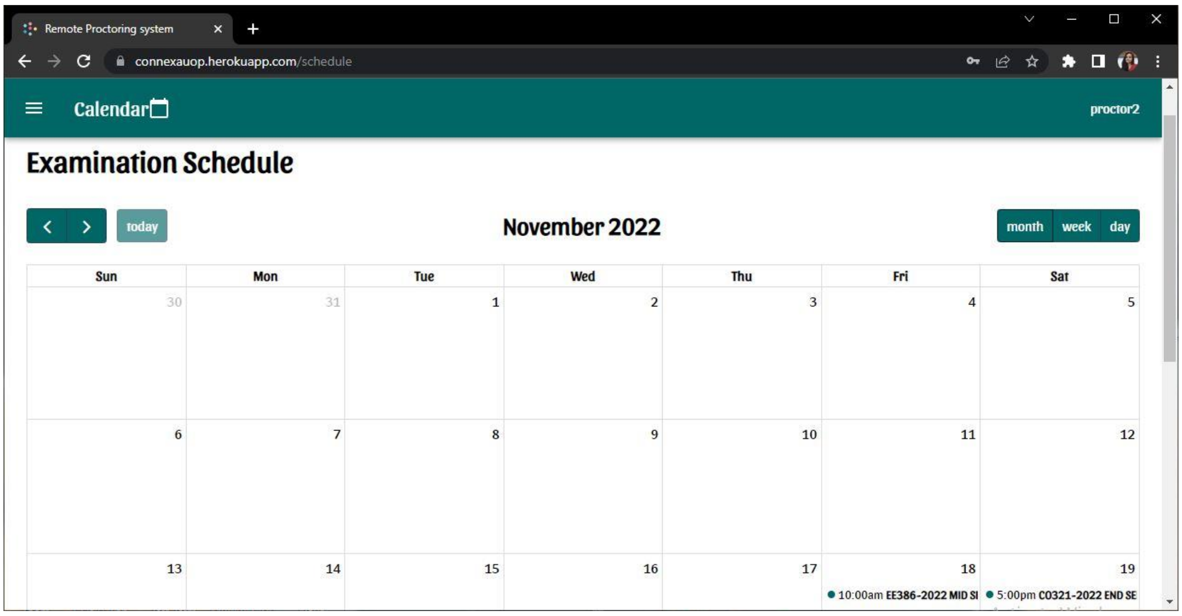
Start time: Sun, 20 Nov 2022 10:00:00 GMT

Duration: 1hr

Room name: C0322-2022 MID SEMESTER EXAMINATION room A

Join Examination

3.3. Calendar

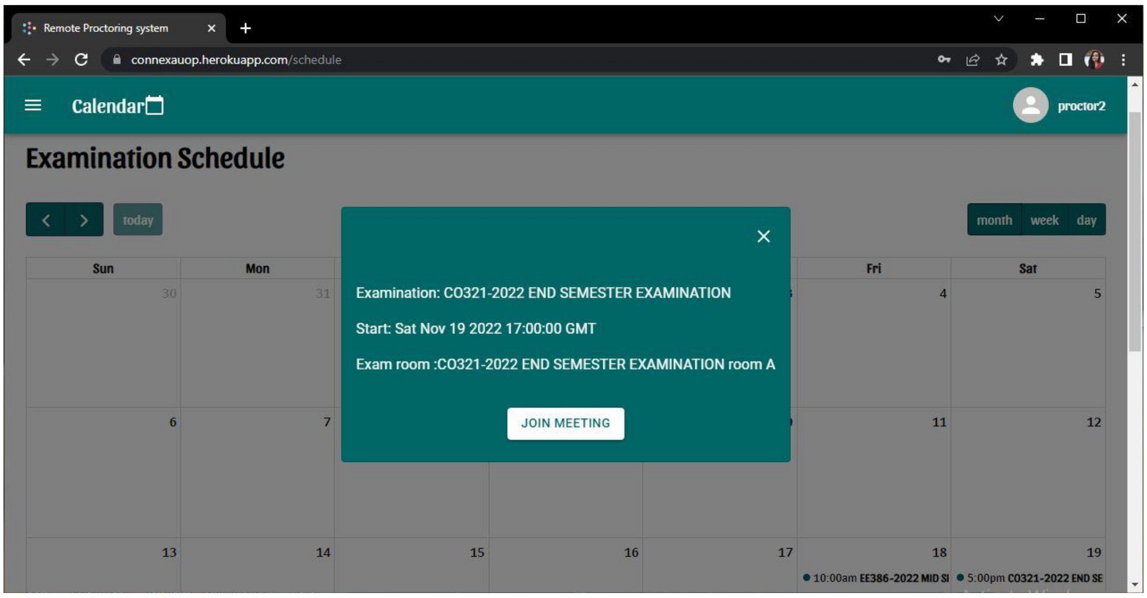


Examination Schedule

November 2022

Sun	Mon	Tue	Wed	Thu	Fri	Sat
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18 ● 10:00am EE386-2022 MID SI	19 ● 5:00pm C0321-2022 END SE

Connexa
REMOTE PROCTORING SYSTEM


Examination Schedule

Examination: CO321-2022 END SEMESTER EXAMINATION

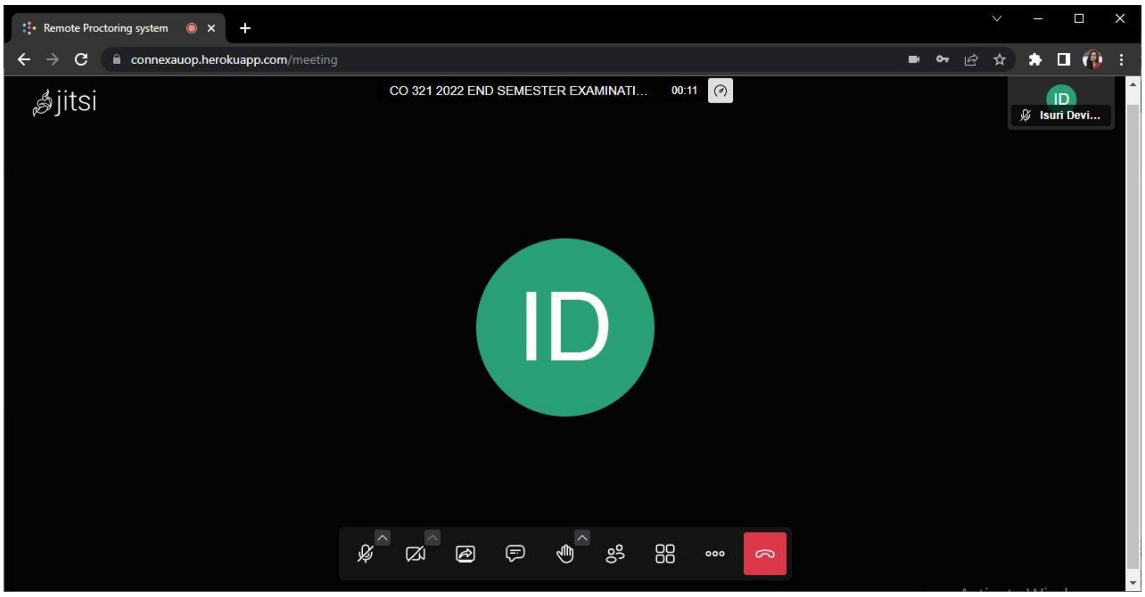
Start: Sat Nov 19 2022 17:00:00 GMT

Exam room :CO321-2022 END SEMESTER EXAMINATION room A

3.4. Meeting Room

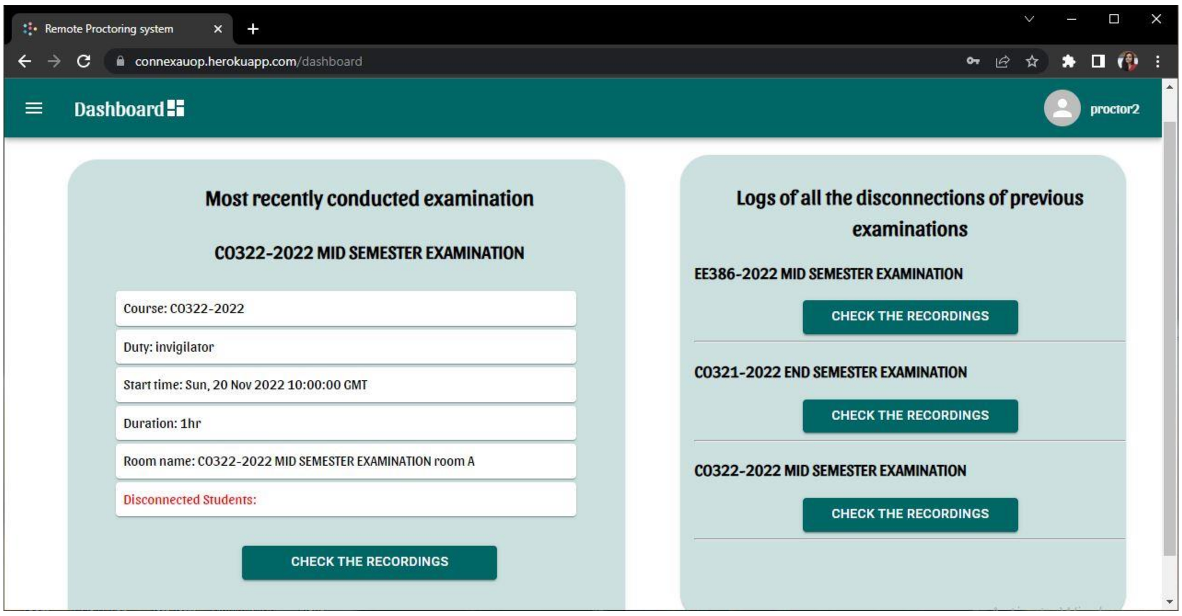
Refer to Jitsi Meet configurations to see the web app integration of the Jitsi Meet.

```
configOverwrite:
  ● prejoinPageEnabled: false,
  ● startWithAudioMuted: true,
  ● startWithVideoMuted: true,
  ● readOnlyName: true,

interfaceConfigOverwrite:
  ● DISABLE_JOIN_LEAVE_NOTIFICATIONS: false,
  ● DISABLE_DOMINANT_SPEAKER_INDICATOR: true
```


Remote Proctoring system
connexauop.herokuapp.com/meeting
jitsi
CO 321 2022 END SEMESTER EXAMINATI...
00:11
ID
Isuri Devi...
ID

3.5. Dashboard

Once a meeting is finished the proctor should be directed to the Dashboad. The students disconnections must e visible in the dashboard and the links to video folder is displayed under each exam.



Most recently conducted examination

C0322-2022 MID SEMESTER EXAMINATION

Course: C0322-2022

Duty: invigilator

Start time: Sun, 20 Nov 2022 10:00:00 GMT

Duration: 1hr

Room name: C0322-2022 MID SEMESTER EXAMINATION room A

Disconnected Students:

CHECK THE RECORDINGS

Logs of all the disconnections of previous examinations

EE386-2022 MID SEMESTER EXAMINATION

CHECK THE RECORDINGS

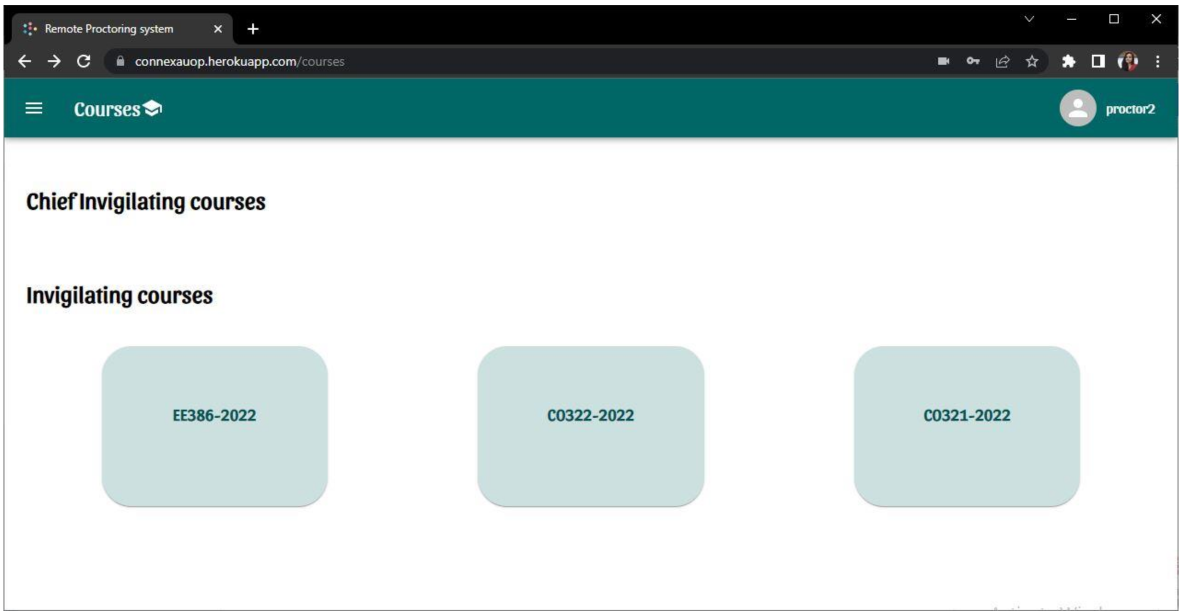
C0321-2022 END SEMESTER EXAMINATION

CHECK THE RECORDINGS

C0322-2022 MID SEMESTER EXAMINATION

CHECK THE RECORDINGS

3.6. Course page



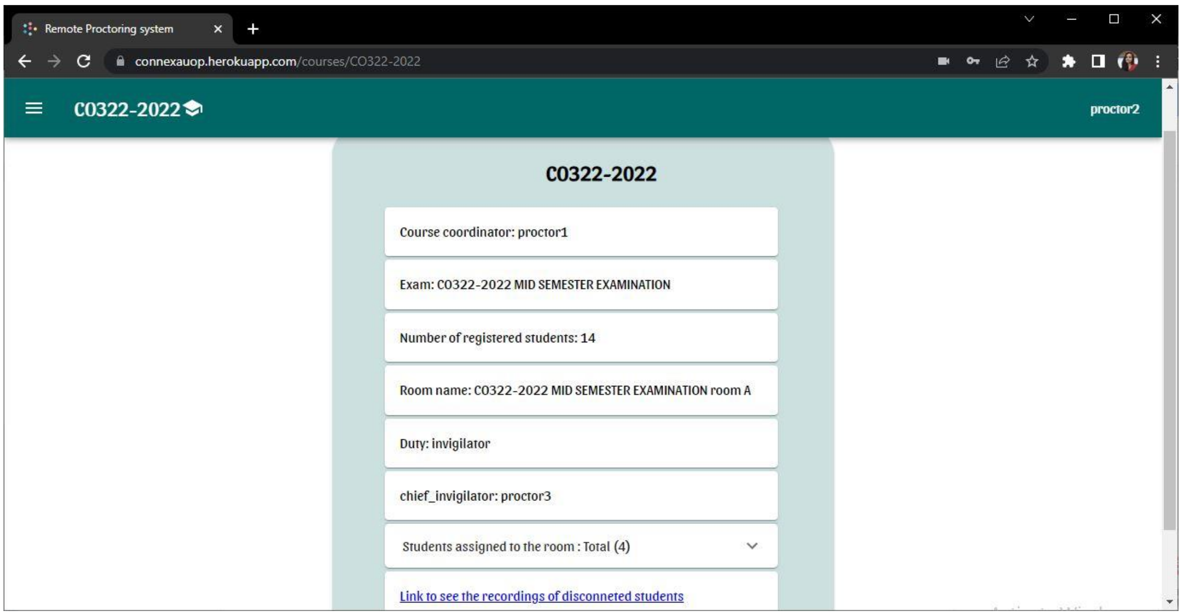
Chief Invigilating courses

Invigilating courses

EE386-2022

C0322-2022

C0321-2022


Remote Proctoring system

connexauop.herokuapp.com/courses/CO322-2022

CO322-2022

proctor2

CO322-2022

Course coordinator: proctor1

Exam: CO322-2022 MID SEMESTER EXAMINATION

Number of registered students: 14

Room name: CO322-2022 MID SEMESTER EXAMINATION room A

Duty: invigilator

chief_invigilator: proctor3

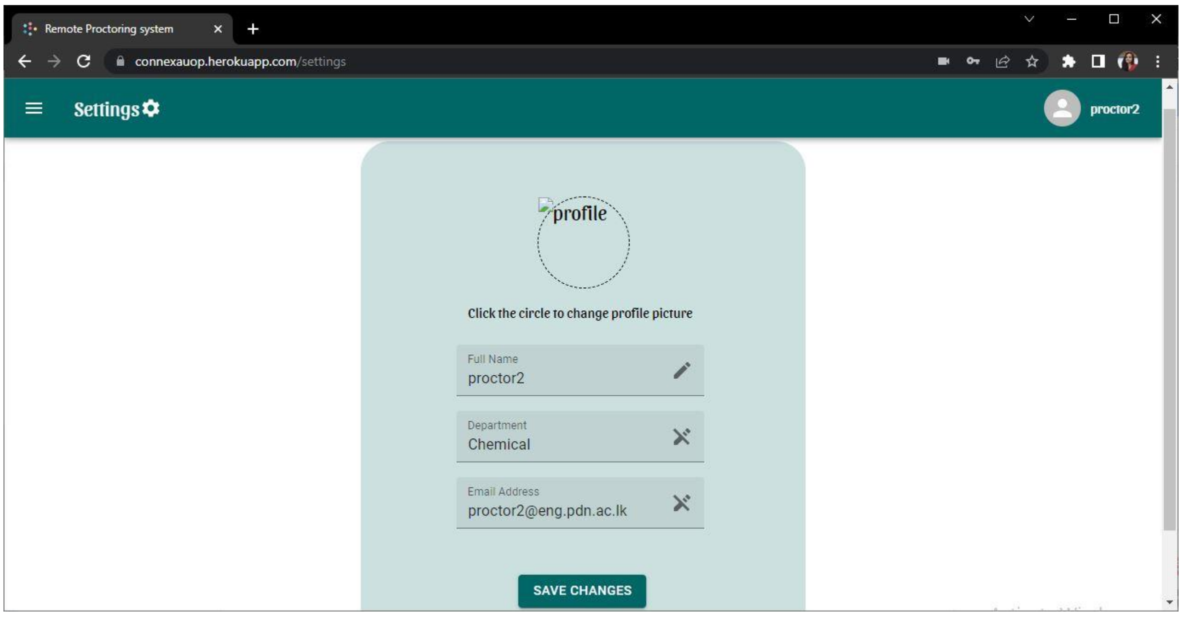
Students assigned to the room : Total (4)

Link to see the recordings of disconneted students

When a course is clicked on the course page, it is directed to a seperate page containing course details. The following details should be displayed on that page.

- Course coordinator
- Number of students who are enrolled in the course
- List of students
- Chief invigilator
- Room name
- Link to recordings folder

3.7. Settings


Remote Proctoring system

connexauop.herokuapp.com/settings

Settings

proctor2

profile

Click the circle to change profile picture

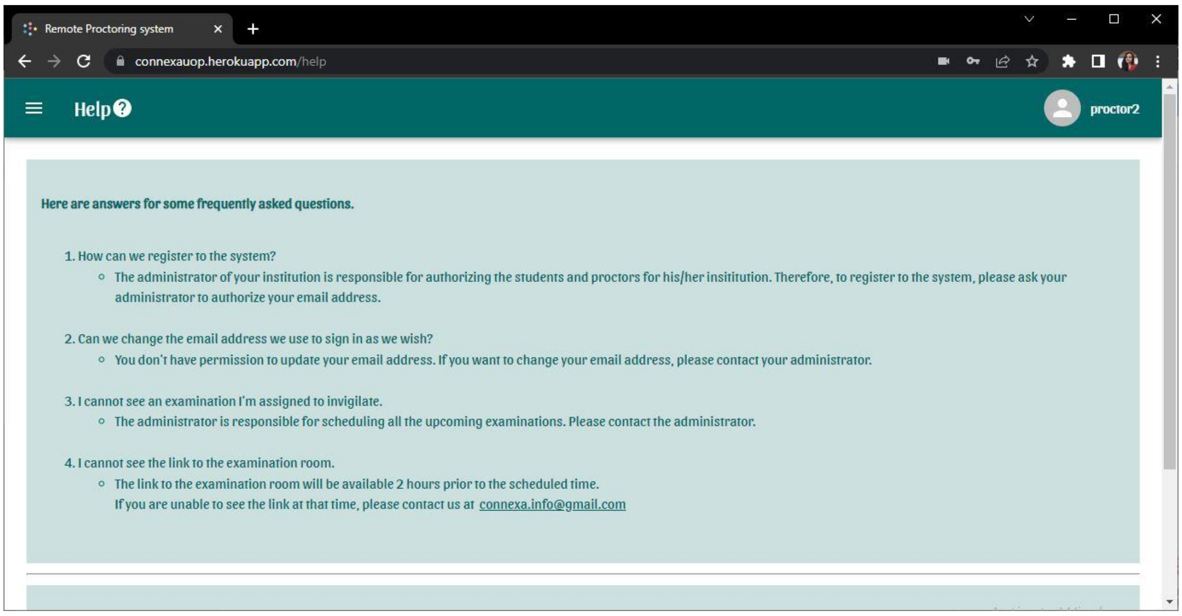
Full Name: proctor2

Department: Chemical

Email Address: proctor2@eng.pdn.ac.lk

SAVE CHANGES

3.8. Help



Here are answers for some frequently asked questions.

1. How can we register to the system?
   - The administrator of your institution is responsible for authorizing the students and proctors for his/her insititution. Therefore, to register to the system, please ask your administrator to authorize your email address.
2. Can we change the email address we use to sign in as we wish?
   - You don't have permission to update your email address. If you want to change your email address, please contact your administrator.
3. I cannot see an examination I'm assigned to invigilate.
   - The administrator is responsible for scheduling all the upcoming examinations. Please contact the administrator.
4. I cannot see the link to the examination room.
   - The link to the examination room will be available 2 hours prior to the scheduled time.
     If you are unable to see the link at that time, please contact us at connexa.info@gmail.com

8.3. Testing

1. **Set up Test Environment**

   Jest and react-testing library is used for testing the React application. Jest is a JavaScript test runner that provides resources for writing and running tests. React Testing Library offers a set of testing helpers that structure your tests based on user interactions rather than components' implementation details. Both Jest and React Testing Library come prepackaged with Create React App and adhere to the guiding principle that testing apps should resemble how the software will be used.

   By default, Jest will look for files with the .test.js suffix and files with the .js suffix in the tests folders. When you make changes to the relevant test files, they will be detected automatically. As test cases are modified, the output will update automatically.

   Therefore, write all test code in files ending with .test.js

2. **Write Test Code**

   Some of the sample test codes written to validate user registration are shown below in *Validatereg.test.js* file. Routing functionality is also tested here.

```
import React from "react";
import {render, fireEvent} from "@testing-library/react";
import Validate from "../components/Validation";
const historyMock2 = { push: jest.fn() };

describe("Test input validations at Registeration page",()=>{
    test("load the registration form",()=>{
        const component = render(<Validate.WrappedComponent/>);

    });
    test("Input fields must not be empty",()=>{
        const {getByLabelText,getByText} =
render(<Validate.WrappedComponent/>);
        const emailInputNode = getByLabelText("Email");
        const passwordInput = getByLabelText("Password");
        const confpwdInput = getByLabelText("Confirm-Password");
        expect(emailInputNode.value).toMatch("");
        expect(passwordInput.value).toMatch("");
        expect(confpwdInput.value).toMatch("");
        const button = getByText("REGISTER");
        fireEvent.click(button);
        const emailError = getByText("Please enter your email
Address");
        const passwordError= getByText("Please enter your password");
        const confpwdError= getByText("Please confirm your password")

        expect(emailError).toBeInTheDocument();
        expect(passwordError).toBeInTheDocument();
        expect(confpwdError).toBeInTheDocument();
    });
    test("Email input should not accept email address without correct
format",()=>{
        const {getByLabelText,getByText} =
render(<Validate.WrappedComponent/>);
        const emailInputNode = getByLabelText("Email");
        const button = getByText("REGISTER");

        expect(emailInputNode.value).toMatch("");

        //case 1: without @
        fireEvent.change(emailInputNode,{target:{value:"email"}});
        expect(emailInputNode.value).toMatch("email");
        fireEvent.click(button);
        const emailError = getByText("Please enter a valid email
address");
        expect(emailError).toBeInTheDocument();

        //case 2: not in correct format
        fireEvent.change(emailInputNode,{target:{value:"email@"}});
        fireEvent.click(button);
        expect(emailError).toBeInTheDocument();

        //case 3: not in correct format

fireEvent.change(emailInputNode,{target:{value:"email@gmail"}});
        fireEvent.click(button);
        expect(emailError).toBeInTheDocument();
    });

    test("Password should be >8 chars, contain atleast one letter and
one digit",()=>{
        const {getByLabelText,getByText} =
render(<Validate.WrappedComponent/>);
        const passwordInputNode = getByLabelText("Password");
        const button = getByText("REGISTER");

        expect(passwordInputNode.value).toMatch("");

        //case 1: length<8
        fireEvent.change(passwordInputNode,{target:{value:"123"}});
        expect(passwordInputNode.value).toMatch("123");
        fireEvent.click(button);
        const pwdError = getByText("Please add at least 8
characters");
        expect(pwdError).toBeInTheDocument();

        //case 2: doesnot contain letters

fireEvent.change(passwordInputNode,{target:{value:"12345678"}});
        fireEvent.click(button);
        const pwdError2 = getByText("Password should contain at least
one letter");
        expect(pwdError2).toBeInTheDocument();

        //case 3: does not contain digits
```

```
fireEvent.change(passwordInputNode,{target:{value:"abcopqwensd"}});
        fireEvent.click(button);
        const pwdError3 = getByText("Password should contain at least 
one digit");
        expect(pwdError3).toBeInTheDocument();
    });
    test("Check confirm password field",()=>{
        const {getByLabelText,getByText} = 
render(<Validate.WrappedComponent/>);
        const cpwd = getByLabelText("Confirm-Password");
        fireEvent.change(cpwd,{target:{value:"123456789"}});
        expect(cpwd.value).toMatch("123456789");
    });

    test("Check password match",()=>{
        const {getByLabelText,getByText,getAllByText} = 
render(<Validate.WrappedComponent/>);
        const cpwd = getByLabelText("Confirm-Password");
        const passwordInput = getByLabelText("Password");
        expect(passwordInput.value).toMatch("");
        expect(cpwd.value).toMatch("");
        fireEvent.change(passwordInput,{target:{value:"123124"}});
        fireEvent.change(cpwd,{target:{value:"AmsdIjd@12*9"}});

        const button = getByText("REGISTER");
        fireEvent.click(button);
        const pwdError = getAllByText("Passwords don't match");
        expect(pwdError[0]).toBeInTheDocument();
    })
    test("If input fields are in correct format direct to home 
page",()=>{
        const {getByLabelText,getByText} = 
render(<Validate.WrappedComponent history={historyMock2}/>);

        const emailInputNode = getByLabelText("Email");
        const passwordInput = getByLabelText("Password");
        const cpwd = getByLabelText("Confirm-Password");
        expect(emailInputNode.value).toMatch("");
        expect(passwordInput.value).toMatch("");
        expect(cpwd.value).toMatch("");


fireEvent.change(emailInputNode,{target:{value:"e17058@eng.pdn.ac.lk"}
});

fireEvent.change(passwordInput,{target:{value:"AmsdIjd@12*9"}});
        fireEvent.change(cpwd,{target:{value:"AmsdIjd@12*9"}});
        const button = getByText("REGISTER");
        fireEvent.click(button);
    });

})
```

3. Run the Test

```
$ npm test
```

The npm test command starts the tests in an interactive watch mode with Jest as its test runner. When in watch mode, the tests automatically re-run after a file is changed. The tests will run whenever you change a file and let you know if that change passed the tests.

9. Video conferencing

For video conferencing between student and the proctor, both web application and desktop application require a common video conferencing application. Connexa uses a free and open source application for video conferencing. The free and open source application is chosen to reduce the complication of the installation and to customize the video conferencing features as required.

There are several best open source video conferencing applications. Such as Jitsi Meet, Apache OpenMeetings, Jami, Nextcloud Talk, BigBlueButton, etc... For this project, the Jitsi meet is used as the video-conferencing application.

9.1. Jitsi Meet

There are several reasons for selecting Jitsi Meet.

- Jitsi is a collection of
  - free and open-source WebRTC video conferencing applications
  - instant messaging web applications
- Host the Jitsi-meet application on our own web server
- Conduct or join a meeting from the web version without creating an account
- No need to update the application regularly.
- The selective video elements can be muted independently.
- Use its **IFrame API** to integrate a meeting into the application.

Learn more about Jitsi Meet here.

9.2. Jitsi Meet configuration

The IFrame API enables you to embed Jitsi Meet functionality into your meeting application so you can experience the full functionality of the globally distributed and highly available deployment available with meet.jit.si.

To enable the Jitsi Meet API in the application the following steps should be followed.

1. **Add JavaScript Jitsi Meet API library scripts**

For self-hosting in your domain:

```
<script src="https://<your-domain>/external_api.js"></script>
```

Meet.jit.si:

```
<script src="https://meet.jit.si/external_api.js"></script>
```

2. **Add HTML element to the application**

```
<div id="meet"></div>
```

3. **Create Jitsi Meet object**

```
const api = new JitsiMeetExternalAPI(domain, options);
```

Domain and Options can be defined as follows.

```
const domain = 'meet.jit.si';
const options = {
    roomName: roomName,
    width: 800,
    height: 445,
    userInfo: {
        email: userEmail,
        displayName: displayName,
    },
    configOverwrite: {},
    interfaceConfigOverwrite: {},
    parentNode: document.querySelector('#meet')
};
```

Configurations overwrites are as follows:

- startWithAudioMuted: false,
- startWithVideoMuted: false,
- enableWelcomePage: false,
- prejoinPageEnabled: false,
- toolbarButtons: ['camera', 'chat', 'microphone', 'raisehand'],
- notifications: ['toolbar.talkWhileMutedPopup', 'notify.mutedTitle'],
- disabledSounds:['NOISY_AUDIO_INPUT_SOUND','PARTICIPANT_JOINED_SOUND', 'PARTICIPANT_LEFT_SOUND']

Interface Configuration Overwrite:

- DISABLE_DOMINANT_SPEAKER_INDICATOR: true,
- APP_NAME: 'Connexa',
- DEFAULT_LOGO_URL: ' ',

4. Removes the embedded Jitsi Meet conference:

```
api.dispose();
```

9.3 Custom Functions

Function to mute all except the invigilator

```
function mute() {
    participantinfo = api.getParticipantsInfo();
     participantinfo.forEach(function(participant, index, arr) {
         var Pid = participant.participantId;
         var Pname = participant.displayName;
             if (!(Pname.includes("invigilator"))) {
                api.executeCommand('setParticipantVolume', Pid, 0);
         } else {
                api.pinParticipant(Pid);
          }
     })
}
```

Function to unmute all

```
function unmute() {
    participantinfo = api.getParticipantsInfo();
        participantinfo.forEach(function(participant, index, arr) {
            api.executeCommand('setParticipantVolume',
participant.participantId, 1);
        })
}
```

10. Deployment

You can deploy the server and the web application in any cloud server. Both don't need to be in the same cloud server.

The React application can be built using the following command.

```
npm run build
```

When deploying the web application path variable for server should be manually changed to the server url before deploying the web application and the production build of the React application can be deployed along with the server application under the same domain name.

In the student app, the server url is added by the user. Therefore no need to change any path variable before server deployment.

Current deployment details:

- Hosted on : DigitalOcean server instance
- Domain name: connexa.space
- Process Manager: PM2
- Reverse Proxy : NGINX
- Desktop application : Distributable packaged application