CRITERION C- DEVELOPMENT

Refer to appendix 3 for the codes for these techniques below.[1]

Techniques:

- PHP coding
- JavaScript Coding
- Object Oriented Programming
- Cascading Style Scripts (CSS)

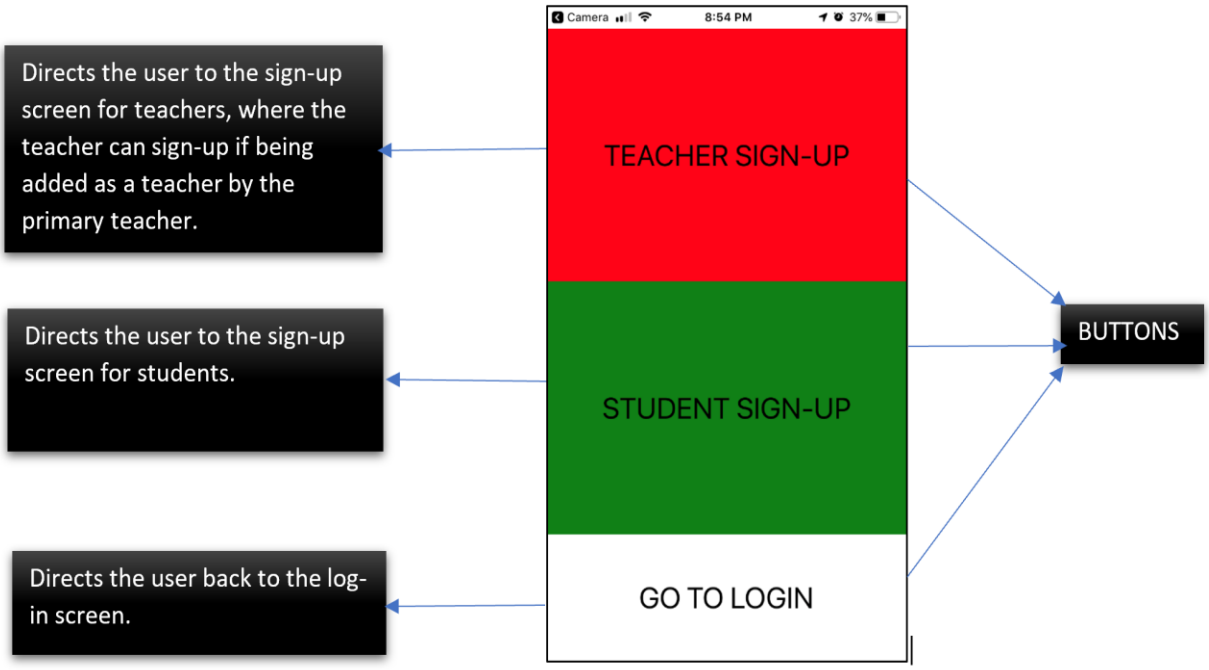
JAVA OBJECT ORIENTED PROGRAMMING

Java development was done using Visual Studio Code 1.38, using the standard java class libraries to increase compatibility. All GUIs were created with react-native on Visual Studio Code 1.38, using JavaScript language.

- SIGN UP FUNCTIONS:

  There will be two options available to new users for signing-up: Teacher sign-up and Student sign-up[2]. As mentioned in Criterion A success criteria 1, For security purposes, only one teacher is already signed up; only that teacher can add another teacher by entering his/her email so that another teacher can sign up. For teacher and student sign up, on clicking 'sign up', an OTP screen is displayed where the OTP is sent via the mail used by the user to sign up. An error message is shown on entering a wrong OTP, and the user is verified on entering the correct OTP.

[1] Refer to Appendix 3
[2] CRITERION A – Success criteria 1



*[fig 1.1] **Sign-up options (UI)***

```
class SignUpChoice extends Component {
    teacherNavigate = () => {
        this.props.navigation.navigate('SignUpTeacher')
    }
    studentNavigate = () => {
        this.props.navigation.navigate('SignUpStudent')
    }
    loginNavigate = () => {
        this.props.navigation.goBack()
    }
    render(){
        return (
            <View style = {{flex:1}}>
                <TouchableOpacity onPress={() => this.teacherNavigate()} style={{flex:3,backgroundColor:'red',justifyContent:'center',
                alignItems:'center'}}>
                    <Text style={{color:'white',fontSize:20}}> Teacher </Text>
                </TouchableOpacity>
                <TouchableOpacity onPress={() => this.studentNavigate()} style={{flex:3,backgroundColor:'pink',justifyContent:'center',
                alignItems:'center'}}>
                    <Text style={{color:'black',fontSize:20}}> Student </Text>
                </TouchableOpacity>
                <TouchableOpacity onPress={() => this.loginNavigate()} style={{flex:1,backgroundColor:'aqua',justifyContent:'center',
                alignItems:'center'}}>
                    <Text style={{color:'black'}}> Back to login </Text>
                </TouchableOpacity>
            </View>
        )
    }
}
export default SignUpChoice;
```

fig 1.1 code[3]

The highlighted parts in the code above show the functions that are used to direct the user as per the button clicked by the user.

GENERATING OTP FOR SIGN-UP PROCESS[4]

[3] APPENDIX 3 – Sign-up choice
[4] CRITERION A – Success criteria 3

```
<TouchableOpacity onPress={() => this.otp()} style = {{flex: 0.4, justifyContent: 'center', alignItems: 'center',
backgroundColor: 'skyblue', borderRadius: 30}}>
<Text style = {{fontSize: 20,fontWeight:'bold'}}>
    Submit
</Text>

</TouchableOpacity>
```

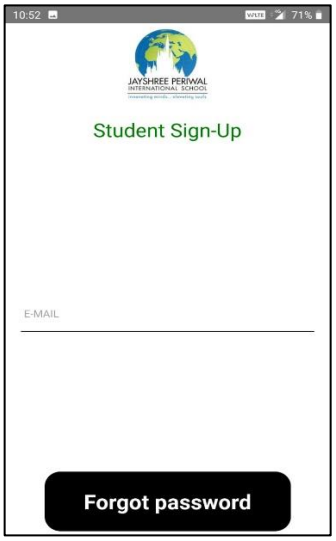
On clicking the submit button, the OTP entered will be verified. If correct, it will enable the user to log-in. If incorrect, an error will be displayed.

```
otp = () =>{
    // alert('please confirm your otp & verify yourself')
    // this.props.navigation.navigate('Otp')
    // return false;
    console.log(`otp is ${this.state.userOtp} and email is ${global.email}`);

    return fetch(helpers.baseUrl+'user_registration.php',{

        method: 'POST',
        headers: {
          'Accept': 'application/json',
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({
          email: global.email,
          otp:this.state.userOtp
        })

      }).then((response) => response.json())
            .then((responseJson) => {

                this.props.navigation.navigate('SigninScreen')
            }).catch((error) => {

              this.props.navigation.navigate('SigninScreen')
            });

    }
goToSignup = () => {
    this.props.navigation.navigate('SignUpChoice');
```

```php

$id = $obj['id'];

$Sql_Query = "select otp from user where id = '$id'";
$check = mysqli_fetch_array(mysqli_query($con,$Sql_Query));
$dbOtp = $check->otp;
if(isset($check)){

    $change_status = "UPDATE user SET verified = 1  where id = '$id' ";
    $check_password = mysqli_fetch_array(mysqli_query($con,$password_checker));
    if(isset($check_password)){
        $SuccessLoginMsg = ['status'=>'200','message'=>'Data Matched'];
        $SuccessLoginJson = json_encode($SuccessLoginMsg);
        echo $SuccessLoginJson ;
    }else{
        $InvalidMSG = ['status'=>'500','message'=>'Invalid Password, Please Try Again'];
        $InvalidMSGJSon = json_encode($InvalidMSG);
        echo $InvalidMSGJSon ;
    }

}else{
    $InvalidMSG = ['status'=>'500','message'=>'Email id is not registered with us Please Try Again'];
    $InvalidMSGJSon = json_encode($InvalidMSG);
    echo $InvalidMSGJSon ;
}

mysqli_close($con);
```

OTP CHECK[5]

- FORGOT PASSWORD[6]:
  If a user has forgotten his/her password, by clicking the 'Forgot Password?' button, the user will be directed to the screen where the user's e-mail is asked and an OTP is sent. By matching the OTP sent on the mail and the one entered on the app, the user will receive a mail that will contain their previously set password. The user can then log-in.



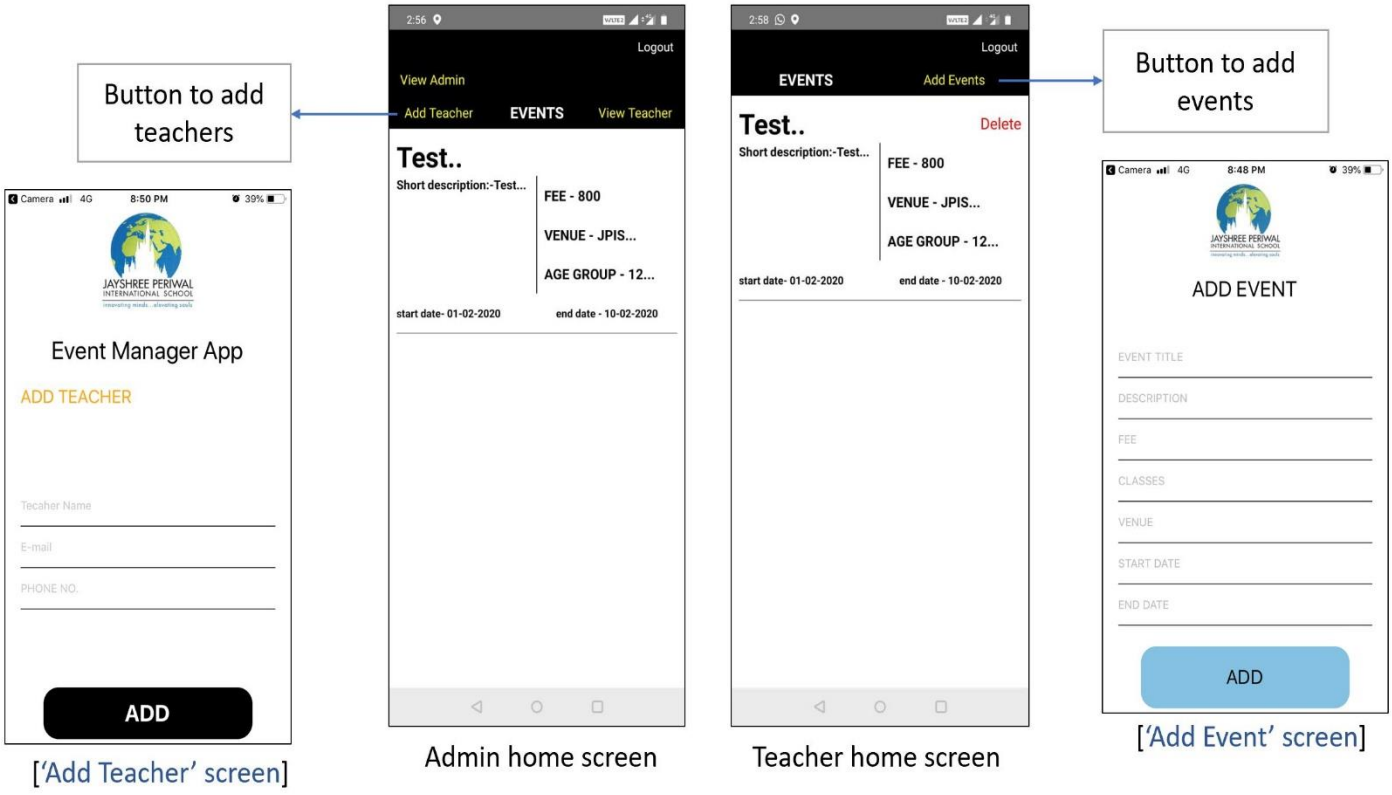
[5] APPENDIX 3 – Generating OTP
[6] APPENDIX 3 – Forgot Password

```
<View style = {{flex: 0.2, paddingLeft: 30, paddingRight: 30}}>
  <TouchableOpacity onPress={() => this.forgetPassword()} style = {{marginBottom: 5, flex: 1, justifyContent: 'center',
  alignItems: 'center', backgroundColor: 'black', borderRadius: 20}}>
    <Text style = {{fontSize: 28, fontWeight: 'bold', color: 'white'}}>
      Forgot password
    </Text>

  </TouchableOpacity>
</View>
```

```
forgetPassword =() => {
  alert('A mail has been sent to you, please follow instruction.')
  this.props.navigation.navigate('SigninScreen');
  return false;
    return fetch(helpers.baseUrl+'user_registration.php',{
    method: 'POST',
    headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json',
        'x_rest_username':'admin@restuser',
        'x_rest_password':'admin@Access',
        'x_rest_version':'v1'
    },
    body: JSON.stringify({
        email: this.state.UserEmail

    })

    }).then((response) => response.json())
        .then((responseJson) => {
            if(responseJson){
                alert(JSON.stringify(responseJson));
                return false;
                // AsyncStorage.setItem('user', JSON.stringify(responseJson.data.id))
                // AsyncStorage.setItem('type', JSON.stringify(responseJson.data.type))
                this.props.navigation.navigate('HomeAppScreen');
            }else{
              alert(JSON.stringify(responseJson))
            }
        }).catch((error) => {
```

After an email has been sent to the user and the password has been received, the user is directed to the log-in screen.

- HOME SCREENS (Admin and Teacher):
  The Home Screen that is available on Admin log-in[7] has three buttons: 'Add Teacher', 'View Teachers', and 'View Admin'[8]. The home screen that is available on Teacher log-in has one button: 'Add Event'. Below are the functions for 'Add Teacher' and 'Add Event'. 'Add Teacher' screen is for the admin to add a user as a teacher and 'Add Event' screen is for the teacher to add an event. Also, each event displayed on the screen is created using FLEX.



['Add Teacher' screen]

Admin home screen

Teacher home screen

['Add Event' screen]

[7] Refer to appendix 2

[8] Criterion A, success criteria 4, 5, and 6.

```
<View style = {{flex: 0.2, paddingLeft: 30, paddingRight: 30}}>
  <TouchableOpacity onPress={()=>this.addTeach()} style = {{marginBottom: 5, flex: 1, justifyContent: 'center', alignItems:
  'center', backgroundColor: 'black', borderRadius: 20}}>
    <Text style = {{fontSize: 28, fontWeight: 'bold', color: 'white'}}>
      ADD
    </Text>

  </TouchableOpacity>
</View>
```

```
addTeacher = () => {
  this.props.navigation.push('AddTeacher',{
    addedBy:global.userIdToken,
    type:global.type,
  })
}
```

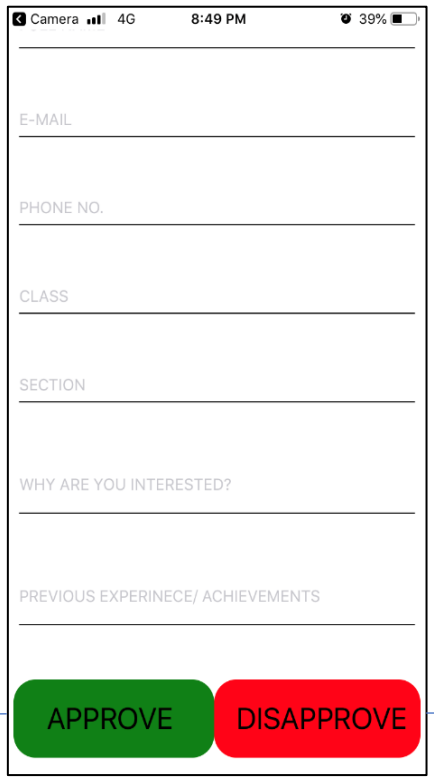
This function directs the user to the 'Add Teacher' screen, where the admin can add a teacher which will enable the sign-up process for that teacher.

```
let addevents = (
  <TouchableOpacity onPress={()=>this.addEvents()} style={{flex:1,justifyContent:'center',alignItems:'center'}}>
      <Text style={{color:'yellow', fontSize: 17}}>Add Events</Text>
  </TouchableOpacity>
);
```

```
addEvents = () => {
  this.props.navigation.navigate('AddEvent',{
    addedBy:global.userIdToken,
    type:global.type,
  })
}
```

This function directs the user to the 'Add Event' screen where the teacher can add a new event. The events added will be displayed on the home screen.

- APPROVING, DISAPPROVING FUNCTION:
  After a student has registered for an event, the teacher can view the students registered and their details. Based on the details, the teacher will be available with two options (two buttons): 'Approve' and 'Disapprove'[9]. On approving, the student will remain on the list, and by disapproving, the student will be removed from the list. Once a student is approved or disapproved, the button function disables. The student who registers once cannot register for the event twice, therefore, if the student is rejected for the event then he/she cannot register for the same event again.



This button is used to accept a student for the event. On clicking this, the student will remain on the 'registered students' list.

This button is used to reject a student for the event. By clicking this button, the student will be removed from the 'registered students' list.

Selection Screen[10]

[9] Criterion A, success criteria 6.
[10] APPENDIX 3 – Selection screen

```
<View style = {{marginBottom: 10, flex: 0.115, paddingLeft: 5, paddingRight: 5, flexDirection: 'row'}}>
  <TouchableOpacity onPress={()=>this.changeAppr('1')} style = {{marginBottom: 5, flex: 1, justifyContent: 'center', alignItems:
  'center', backgroundColor: 'green', borderRadius: 20, paddingRight: 5}}>
    <Text style = {{fontSize: 26}}>
      APPROVE
    </Text>

  </TouchableOpacity>
  <TouchableOpacity onPress={()=>this.changeAppr('2')} style = {{marginBottom: 5, flex: 1, justifyContent: 'center', alignItems:
  'center', backgroundColor: 'red', borderRadius: 20, paddingLeft: 10 }}>
    <Text style = {{fontSize: 26}}>
      DISAPPROVE
    </Text>

  </TouchableOpacity>
</View>
```

This function keeps the student's name on the list and the student's data for the event registered.

This function removes the student from the list; and it removes that student's data, for the event that student has registered, from the database.

JavaScript Coding

JavaScript coding, along with react-native, was done using Visual Studio Code 1.38. JavaScript was used to design the UI of the app and was saved into a .js JavaScript file.

- NAVIGATION

The navigation of screens is done using JavaScript coding. The Log-in screen is navigated to the Home screen, if the username and password are correct, using the function '`this.navigatetohome()`'. Further, the Log-in screen will also be navigated to the Forgot Password screen using the function '`this.props.navigation.push('Forgotpassword')`'. The name of the screen in these cases is 'home' and 'forget'.

```
    </View>
    <TouchableOpacity onPress={() => this.navigatetohome()} style = {{flex: 0.4, justifyContent: 'center', alignItems:
    'center', backgroundColor: 'skyblue', borderRadius: 30}}>
    <Text style = {{fontSize: 20,fontWeight:'bold'}}>
        LOG IN
    </Text>
    <View style={{height:20}} />
    </TouchableOpacity>


</View>
<View style={{flex:1}}>
<TouchableOpacity onPress = {() => this.props.navigation.push('Forgotpassword')} style={{flex:1, alignItems: 'flex-end',
justifyContent: 'center', paddingRight: 26}}>
    <Text style = {{fontSize: 16, color: 'red'}}>
    Forgot Password?
    </Text>
```

- IMAGE FUNCTION
  My school's logo, as an image, is added to the UI. For this, the code '`{require('./jpisLogo.jpeg')}`' is used to find the source location of the image file. '`{{height:'100%', width: '100%', resizeMode: 'contain'}}`' is used for sizing the image appropriately.

```
<View style={{flex:2 }}>
  <View style = {{flex:1.5, paddingTop: 20}}>

    <View style = {{flex: 1}}>
      <Image
        source = {require('./jpisLogo.jpeg')}
        style = {{height:'100%', width: '100%', resizeMode: 'contain'}}
      />

    </View>
```

PHP Coding

The back-end programming language I will be using is PHP. For As the app will require sending mails to user on their log ins and student's selection for the event, a server is needed. I will be obtaining my server/cloud service from DigitalOcean. Apache will be my backend server. PHP coding was done using Visual Studio Code. While coding, MySQL queries are used (CRED – Create, Read, Edit, Delete).

- SENDING MAILS (MAIL SMTP)
  The SMTP mail function is required to send mails to users at time of Log-in, Sign-up, Approval-Disapproval, and receiving password when forgotten. The codes – '`./src/Exception.php`', '`./src/PHPMailer.php`', and '`./src/SMTP.php`' – are required to set up the mail function. The admin email as a 'mailer' is set (virox0220@gmail.com). This email is used for the pre-set admin for the app as a teacher[11].

[11] Criterion C – JAVA OBJECT ORIENTED PROGRAMMING, SIGN-UP FUNCTIONS.

```php
<?php

// Import PHPMailer classes into the global namespace
// These must be at the top of your script, not inside a function
use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\Exception;

require './src/Exception.php';
require './src/PHPMailer.php';
require './src/SMTP.php';

// Load Composer's autoloader
// require 'vendor/autoload.php';

// Instantiation and passing `true` enables exceptions
$mail = new PHPMailer(true);

try {
    //Server settings
    // $mail->SMTPDebug = 2;                                      // Enable verbose debug output
    $mail->isSMTP();                                            // Set mailer to use SMTP
    $mail->Host       = 'smtp.gmail.com';  // Specify main and backup SMTP servers
    $mail->SMTPAuth   = true;                                   // Enable SMTP authentication
    $mail->Username   = 'virox0220@gmail.com';                     // SMTP username
    $mail->Password   = 'virajrahil02';                               // SMTP password
    $mail->SMTPSecure = 'ssl';                                  // Enable TLS encryption, `ssl` also accepted
    $mail->Port       = 465;                                    // TCP port to connect to

    //Recipients
    $mail->setFrom('virox0220@gmail.com', 'Mailer');
    $mail->addAddress('rahiljain.intelligence@gmail.com');               // Name is optional
    $mail->addReplyTo('info@example.com', 'Information');
    // Content
    $mail->isHTML(true);                                  // Set email format to HTML
    $mail->Subject = 'Here is the subject';
    $mail->Body    = 'This is the HTML message body <b>in bold!</b>';
    $mail->AltBody = 'This is the body in plain text for non-HTML mail clients';

    $mail->send();
    echo 'Message has been sent';
} catch (Exception $e) {
    echo "Message could not be sent. Mailer Error: {$mail->ErrorInfo}";
}
```

- SHOW EVENTS

  If the Log-in type is ‘student’ (`if($type == 'student')`), all existing events (on the Events screen) will be shown on the screen. Events will be shown that is created after the date ‘10/09/2019’ as this is the date the app was made available to the users, then the query ‘`"select * from events where start_date > '$todayDate'"`’ will be in use. If Teacher logs-in then all the events are to be shown, then the query ‘`"select * from events"`’ will be used.

```php
<?php

include 'dbconfig.php';
$con = mysqli_connect($HostName,$HostUser,$HostPass,$DatabaseName);
$json = file_get_contents('php://input');
$obj = json_decode($json,true);


$type = $obj['type'];
$todayDate = '2019-09-10';
if($type == 'student'){
    $Sql_Query = "select * from events where start_date > '$todayDate'";
}else{
    $Sql_Query = "select * from events";
}
$check = mysqli_fetch_array(mysqli_query($con,$Sql_Query));
if(isset($check)){
    $SuccessLoginMsg = ['status'=>'200','message'=>'Results found','data'=>$check];
    $SuccessLoginJson = json_encode($SuccessLoginMsg);
    echo $SuccessLoginJson ;
}else{
    $InvalidMSG = ['status'=>'500','message'=>'No result found'];
    $InvalidMSGJSon = json_encode($InvalidMSG);
    echo $InvalidMSGJSon ;
}

mysqli_close($con);
?>
```

CSS (Cascading Style Scripts)

The CSS code was created using the Visual Studio Code. In the UI designing process, I have used FLEX to position and style the elements on the screen. Positioning of the components at top, bottom, left, right, and center can be easily done using its properties called: justify-content, align-items etc.

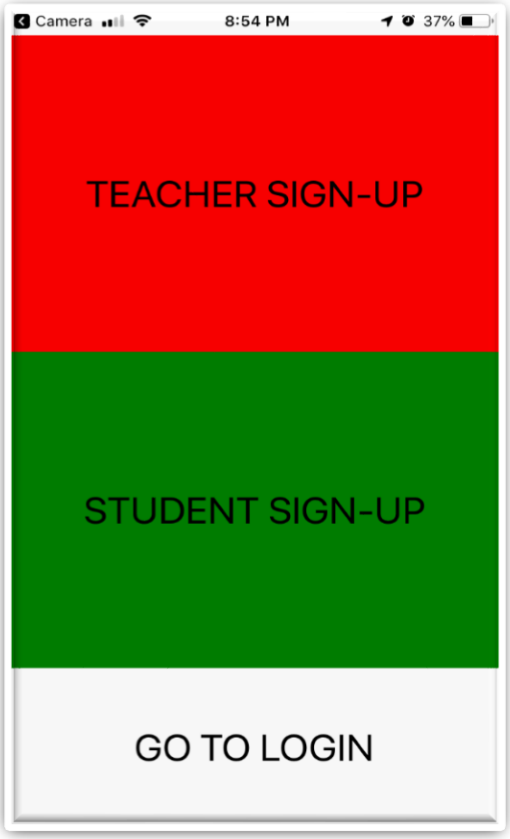


Figure 1.3: sign-up options

This screen above is designed using FLEX to position the buttons – TEACHER SIGN-UP, STUDENT SIGN-UP, and GO TO LOGIN – giving them a rectangular shape with appropriate sizes, and filling them up with colors. This is done using the '`justifyContent`', '`alignItems`', and '`backgroundColor`' functions.

```
<View style={{flex: 1, marginTop: 22}}>
  <View style = {{flex: 1, backgroundColor: 'red', justifyContent: 'center', alignItems: 'center'}}>

   <TouchableHighlight onPress={()=>this.teacherNavigate.bind(this)}>
   <Text style = {{fontSize: 30}}>TEACHER SIGN-UP</Text>
   </TouchableHighlight>
  </View>
  <View style = {{flex: 1, backgroundColor: 'green', justifyContent: 'center', alignItems: 'center'}}>

   <TouchableHighlight>
   <Text style = {{fontSize: 30}}>STUDENT SIGN-UP</Text>
   </TouchableHighlight>
  </View>
  <View style = {{flex: 0.5, backgroundColor: 'white', justifyContent: 'center', alignItems: 'center'}}>
   <TouchableHighlight
    onPress={() => {
     this.setModalVisible(!this.state.modalVisible);
    }}>
    <Text style = {{fontSize: 30}}>GO TO LOGIN</Text>
   </TouchableHighlight>
  </View>

</View>
```