iTutor App

Analysis Report

Introduction

This documentation is designed to show all the processes required in order to develop the iTutor mobile application. This is a requirement for the course module CS385 Mobile Application Development. The main function of the proposed app is to connect students who are in need of extra tutorials with tutors. There are four members in the group; they will work together in order to successfully complete the development of the app. Every member is responsible for one or more activities to ensure smooth delivery of the project.

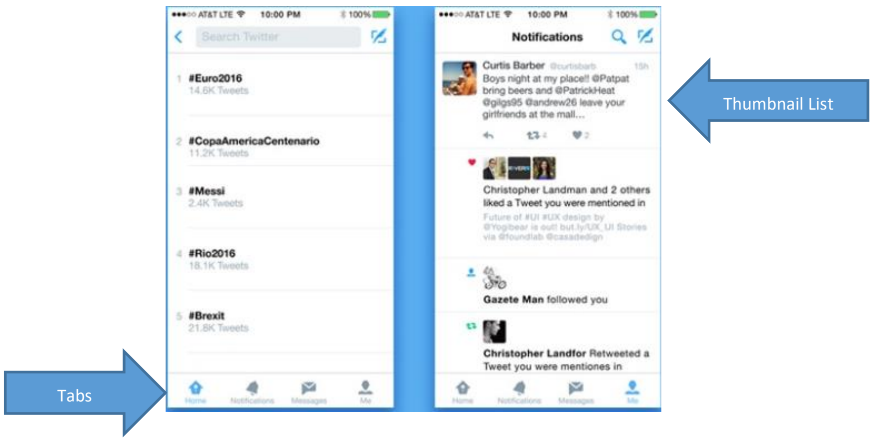
Today's world is a smart world where technological advancement is taking over in nearly all sectors. Communication has never been this easy and connecting people around the world has become so effortless all to the credit of innovative ways people are using Information Technology to solve problems. Information technology has become an indispensable tool for learning and it has been put into great use to deliver better learning content and learning experience. It All are using technology to get more advantages.

iTutor application is an innovative technology which helps students find tutors. This iTutor app can easily tailor to the needs of students who may have specification in who they want as tutor. This app is convenient because it holds a database of tutors in various fields of studies. Rather than wait to get lucky enough to find a single tutor who meets your criteria and can work with you, when searching for online tutors you are much more likely to have a selection to choose from. In addition, the broader selection also affords you the benefit of working with multiple tutors who specialize in different areas, enabling you to choose tutors who are experts in the specific areas where you need help.

Location is not a limitation with the iTutor app, and neither is time. If a student ever tried to deal with the restrictions of finding a tutor who is local, proficient in the subject they need help with, and can meet their schedule.

Interface Design Inspiration:

The design inspiration for the iTutor application came from Twitter. Twitter is a well-known social media application. Twitter use the tab menu option with the contents shown in a scroll view. This app also allows for side swiping to switch to different tab options. The reason for the adaptation of these design features is the similarity of information that needs to be shown to the user between iTutor application.



1 #Euro2016 14.6K Tweets
2 #CopaAmericaCentenario 11.2K Tweets
3 #Messi 2.4K Tweets
4 #Rio2016 18.1K Tweets
5 #Brexit 21.8K Tweets

Curtis Barber @curtisbarb 15h
Boys night at my place!! @Patpat bring beers and @PatrickHeat @gilgs95 @andrew26 leave your girlfriends at the mall...

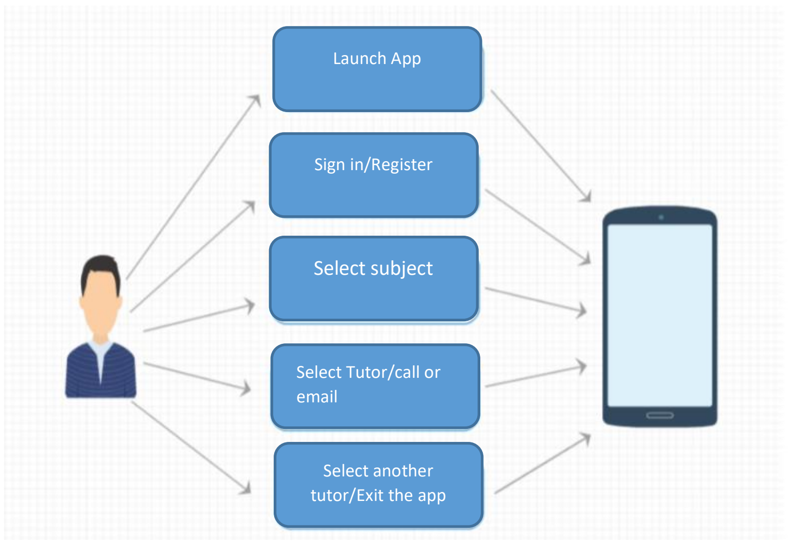
Christopher Landman and 2 others liked a Tweet you were mentioned in
Future of #UI #UX design by @Yogibear is out! bit.ly/UX_UI Stories via @foundlab @casadedign

Gazete Man followed you

Christopher Landfor Retweeted a Tweet you were mentiones in

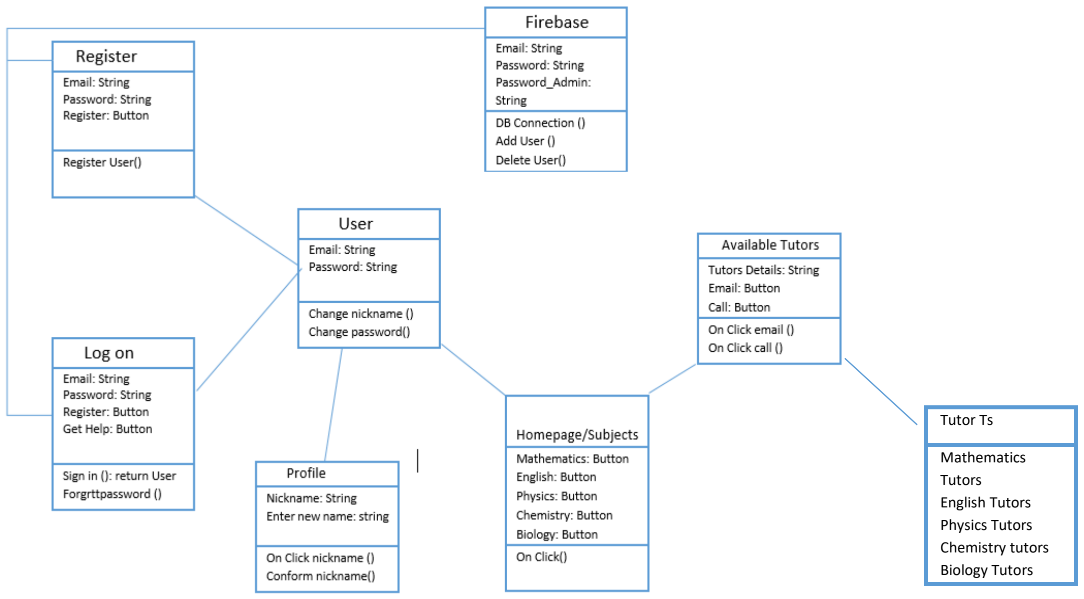
Wireframe Design:

The development process of this App began with putting ideas forward and deciding which one would be more feasible considering time frame and content delivering. We developed an ad hoc model to study different concepts during the design process. We analysed all of the different components and the relationships between them which ultimately laid the foundation for the overall design of application. The apps design is user friendly allowing the user to navigate easily between the pages. iTutor application's core information that needs to be shown are: list of subjects and list of available tutors in specific subject with their contact details. These lists of information are



usually longer than the full screen page and the scroll view can facilitate this well.

The decision to settle for a iTutor App was a unanimous one among group members. Quality time was spent in drawing up a wireframe which reflects the key components and functionalities of the mobile application. Below is our final wireframe.

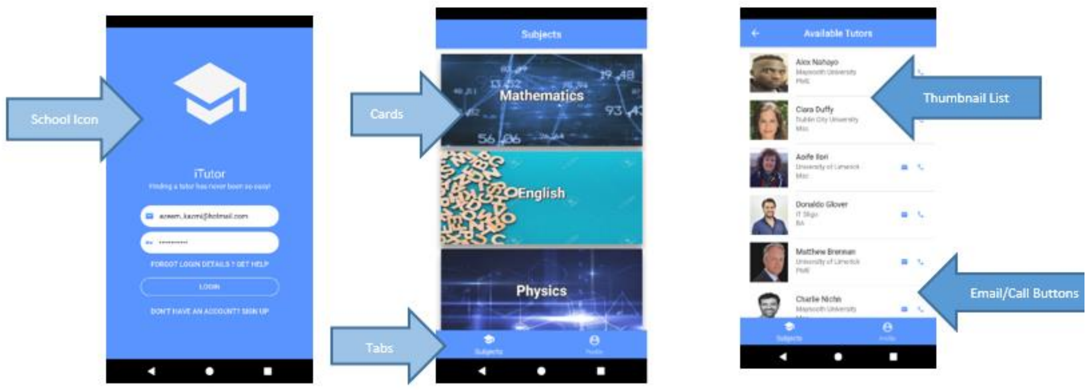


Layouts & Icon Design:

The app is built in angular/ionic framework and each page of it was designed following the same theme. Each feature can be assessed from the homepage. We created a colour scheme which a

primary colour and it is present throughout every page of the app. The process of

designing the icon is involve searching images online and discussing the colour of the icon. We used a school icon from ionic framework. We used ionic cards in our subject homepage layouts and ionic



thumbnail list to layout of tutor list of specific subjects. In thumbnail list we have two buttons to contact the tutor (1) Email button and (2) Call button, these two buttons are in the form of icons.

Application Development:

After completion of the wireframe diagrams, the group began coding of the application. The development framework deployed in this stage was the ionic framework along with Cordova and angular. Ionic library is built by considering AngularJS as its base. This framework provides mobile-optimized HTML components to create highly interactive hybrid apps. When it comes to building an Ionic app with a solid backend, the current choice for many people is Firebase, and most of the time combined with Angular Fire. The process of connecting an Angular app to Firebase is fairly easy in combination with the great Angular Fire package.

Functionalities of the mobile application:

1. Using Firebase: It allows application developers to focus on the app development without having to manage any servers.
2. Login: The login page is the first page that a user will face when opening the app.
3. Sign Up: It allows new users to register and have access to the application
4. Reset Password: In case the user forgets the password the app allows them to reset their old password.
5. My Profile: Similar to the user panel, the student can set/change his/her profile nickname. This will then be updated.
6. Browse Tutor: Users can search the tutors from subject list owing to their learning requirements. Furthermore, once the user selects the subject of choice they will directed to a list of available tutors which can then be contacted. In addition, the users can also view all the relevant details about the users such as qualification and attended university.

1. Using Firebase:

The login page is the first page that a user will face when opening the app. It allows users that have already signed up the app to insert their personal details in order to have access to the app and the information within it. The login page, together with the Signup and reset pages, uses resources and functions from firebase, the database platform we have chosen to store all the information about the users and authentication in our project.

Setting up Firebase in the project:

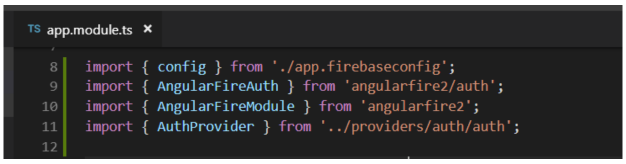
In order to start using all the features that firebase offers in our app we have installed the firebase and angularfire2 libraries using the below command:

```
npm install firebase angularfire2 –save
```

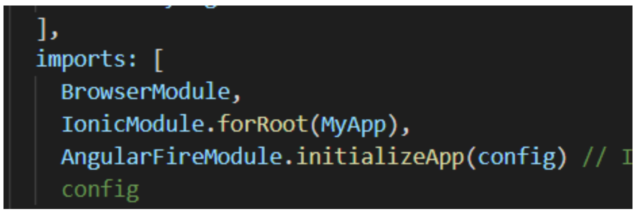
We also have created new file called app.firebaseconfig.ts inside the app directory, with all the settings offered by a JSON object from the Firebase website.

```
export var config = {
  apiKey: <apiKey>,
  authDomain: <authDomain>,
  databaseURL: <databaseURL>,
  projectId: <projectId>,
  storageBucket: <storagebucket>,
  messagingSenderId: <senderIdhere>
 };
```

Inside the file app.module.ts all the providers from firebase were imported: AngularFireAuth, AngularFireModule, AuthProvider and the config object.

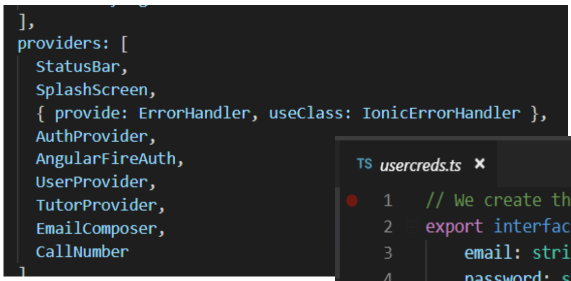

app.module.ts

```
import { config } from './app.firebaseconfig';
import { AngularFireAuth } from 'angularfire2/auth';
import { AngularFireModule } from 'angularfire2';
import { AuthProvider } from '../providers/auth/auth';
```

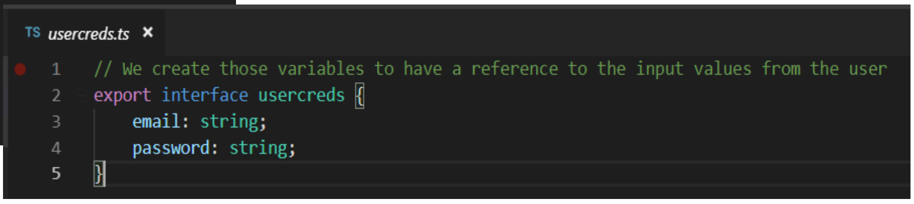


```
],
imports: [
  BrowserModule,
  IonicModule.forRoot(MyApp),
  AngularFireModule.initializeApp(config) // I
  config
```

After it, we have created an interface for the user's credentials, using a new directory inside the src directory named models/ interfaces. Where we have created a file called usercreds.ts with the variables email and password to have a reference to the input values from the user.

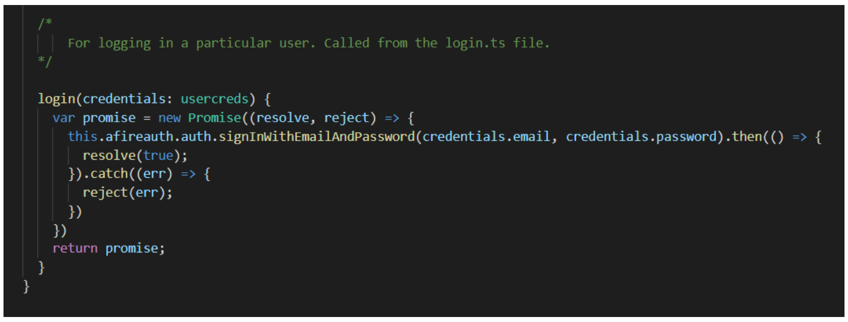


```
],
providers: [
  StatusBar,
  SplashScreen,
  { provide: ErrorHandler, useClass: IonicErrorHandler },
  AuthProvider,
  AngularFireAuth,
  UserProvider,
  TutorProvider,
  EmailComposer,
  CallNumber
]
```


usercreds.ts

```
// We create those variables to have a reference to the input values from the user
export interface usercreds {
    email: string;
    password: string;
}
```

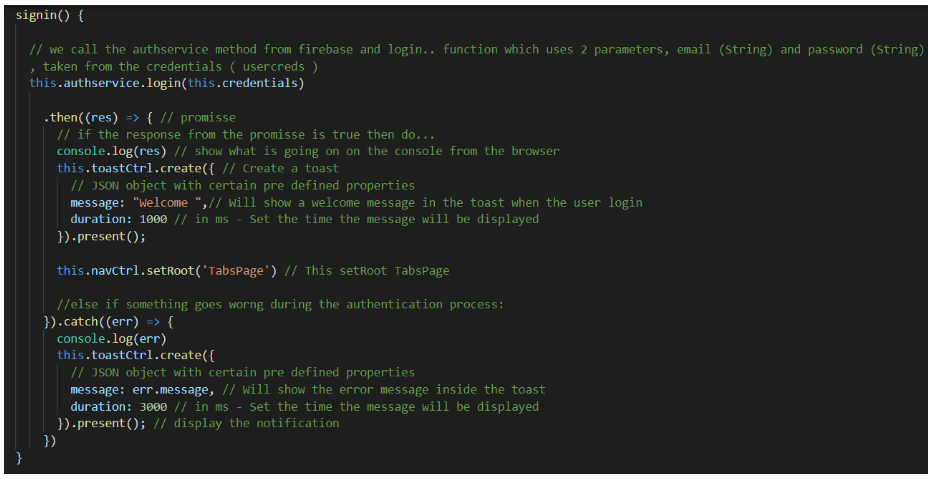
After that, it was created a login() method inside our file (provider) auth.ts for logging in a particular user. Called from the login.ts file.



```
  /*
      For logging in a particular user. Called from the login.ts file.
  */

  login(credentials: usercreds) {
    var promise = new Promise((resolve, reject) => {
      this.afireauth.auth.signInWithEmailAndPassword(credentials.email, credentials.password).then(() => {
        resolve(true);
      }).catch((err) => {
        reject(err);
      })
    })
    return promise;
  }
}
```

2. Login Page

Our new page called login was generated using the command **ionic g page login** and then, the login.scss is implemented. Inside the file login.html there are 2 fields, username (email), password. Also, 3 buttons, reset password, sign in and sign up buttons. Inside the login.ts page the method signin (), simply calls the login() function present in the auth provider and passes on the entered email and password to it. The method signup() drives the user to the SignupPage.



```
signin() {

  // we call the authservice method from firebase and login.. function which uses 2 parameters, email (String) and password (String)
  , taken from the credentials ( usercreds )
  this.authservice.login(this.credentials)

    .then((res) => { // promisse
      // if the response from the promisse is true then do...
      console.log(res) // show what is going on on the console from the browser
      this.toastCtrl.create({ // Create a toast
        // JSON object with certain pre defined properties
        message: "Welcome ",// Will show a welcome message in the toast when the user login
        duration: 1000 // in ms - Set the time the message will be displayed
      }).present();

      this.navCtrl.setRoot('TabsPage') // This setRoot TabsPage

      //else if something goes worng during the authentication process:
    }).catch((err) => {
      console.log(err)
      this.toastCtrl.create({
        // JSON object with certain pre defined properties
        message: err.message, // Will show the error message inside the toast
        duration: 3000 // in ms - Set the time the message will be displayed
      }).present(); // display the notification
    })
}
```

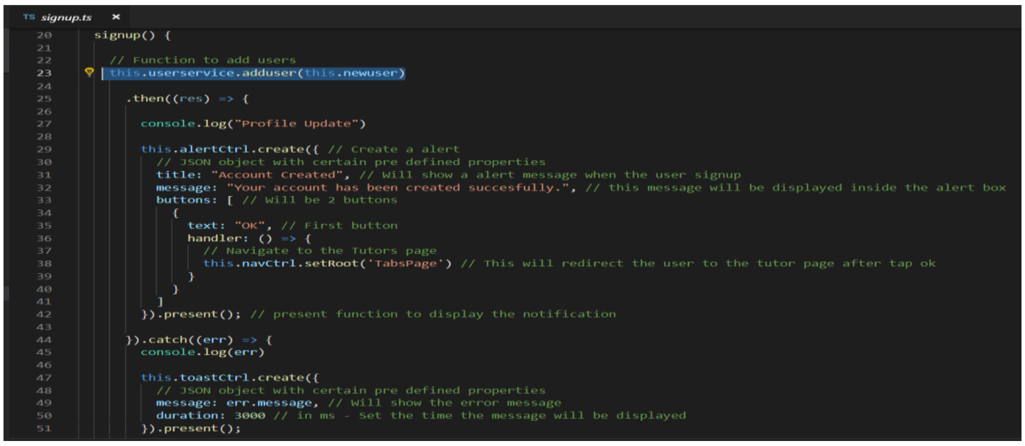
Promises has been used while coding to ensure the process of getting the information from the database and what to do in case of something goes wrong during this process.

In this case, right after the user insert his information and taps in login button (login.html), the signin() method (inside login.ts) will be executed, and the login() function from auth provider will run. If everything goes well, the promise will return a response, and the user will get a “toast” message saying “Welcome” with duration of 1s, and then it will be redirected to the page TabsPage, which is the Root page or home page, with no back button.

Otherwise, In case something goes wrong with user’s authentication the promise will catch the error message from the auth provider and display it on a “toast” during 3s.

3. Signup Page

After creating the signup page using the command below **ionic g page signup** The signup.scss is implemented using exactly same style from the login page. Inside the signup.html we have3 fields



signup.ts

```
signup() {

  // Function to add users
  this.userservice.adduser(this.newuser)

    .then((res) => {

      console.log("Profile Update")

      this.alertCtrl.create({ // Create a alert
        // JSON object with certain pre defined properties
        title: "Account Created", // Will show a alert message when the user signup
        message: "Your account has been created succesfully.", // this message will be displayed inside the alert box
        buttons: [ // Will be 2 buttons
          {
            text: "OK", // First button
            handler: () => {
              // Navigate to the Tutors page
              this.navCtrl.setRoot('TabsPage') // This will redirect the user to the tutor page after tap ok
            }
          }
        ]
      }).present(); // present function to display the notification

    }).catch((err) => {
      console.log(err)

      this.toastCtrl.create({
        // JSON object with certain pre defined properties
        message: err.message, // Will show the error message
        duration: 3000 // in ms - Set the time the message will be displayed
      }).present();
```

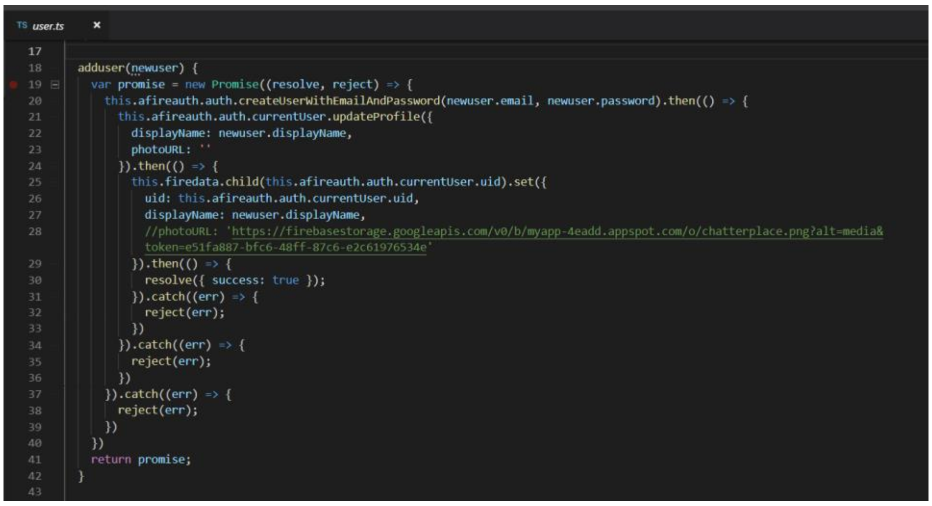
namely, email, password and nickname and two buttons, the sign up button, which is used to add the user to the application, and also the go back button, that is used to move the user back to the login page.

In the signup.ts it is possible to notice two functions namely signup() and goback(). The signup() function is calling the adduser method in the userservice to add the new user. We also have made use of promises to catch any error and make use of 'toast' and loading controller to provide an animation.

Creation of user provider:

We have Created the provider page using the command below **ionic g provider user** In order to create a new user we have used the createUserwithEmailAndPassword() in the angularfire2/auth library. Once the user is successfully created the method updateProfile() in the same library is used to add the displayName for the new user. After that we have created a child inside the users collection with the newly registered user's uid as the key.

All the information about the users is in our users collection. This is primarily because we won't be able to get information about all the users from firebase.

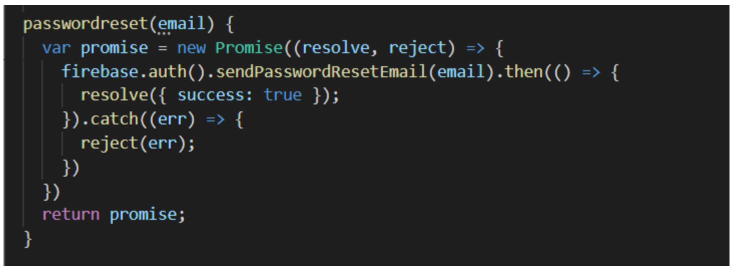


```
user.ts

adduser(newuser) {
  var promise = new Promise((resolve, reject) => {
    this.afireauth.auth.createUserWithEmailAndPassword(newuser.email, newuser.password).then(() => {
      this.afireauth.auth.currentUser.updateProfile({
        displayName: newuser.displayName,
        photoURL: ''
      }).then(() => {
        this.firedata.child(this.afireauth.auth.currentUser.uid).set({
          uid: this.afireauth.auth.currentUser.uid,
          displayName: newuser.displayName,
          //photoURL: 'https://firebasestorage.googleapis.com/v0/b/myapp-4eadd.appspot.com/o/chatterplace.png?alt=media&
          token=e51fa887-bfc6-48ff-87c6-e2c61976534e'
        }).then(() => {
          resolve({ success: true });
        }).catch((err) => {
          reject(err);
        })
      }).catch((err) => {
        reject(err);
      })
    }).catch((err) => {
      reject(err);
    })
  })
  return promise;
}
```

4. Password reset:

Our new page called passwordreset was generated using the command **ionic g page passwordreset**

The passwordreset.scss is implemented using exactly same style from the login page. Inside the passwordreset.html We just have a single input field and 2 buttons, Reset My Password and Go Back. Inside the file passwordreset.ts a function passwordreset() is called in the user provider send the email



```
passwordreset(email) {
  var promise = new Promise((resolve, reject) => {
    firebase.auth().sendPasswordResetEmail(email).then(() => {
      resolve({ success: true });
    }).catch((err) => {
      reject(err);
    })
  })
  return promise;
}
```

there. In the user.ts the method passwordreset(email) is created

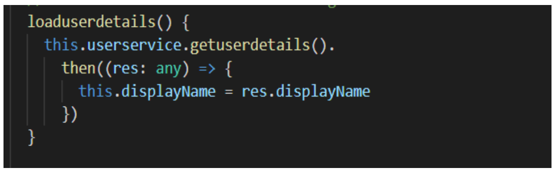
Here we are making use of the sendPasswordResetEmail() that ships with the firebase to send and email to the user with all the reset instructions. If for some reason the email is not registered then it will thrown an error back.

5. Profile Page:

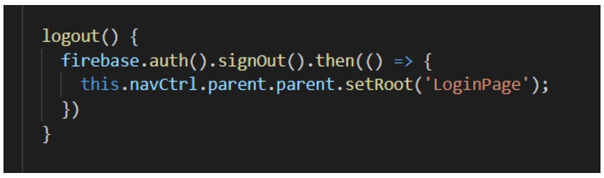
The new page called profile was generated using the command **ionic g page profile**

On profile.scss the exactly same style from the login page is used. profile.html we have the logo, and 2 buttons Edit Display Name and Logout.

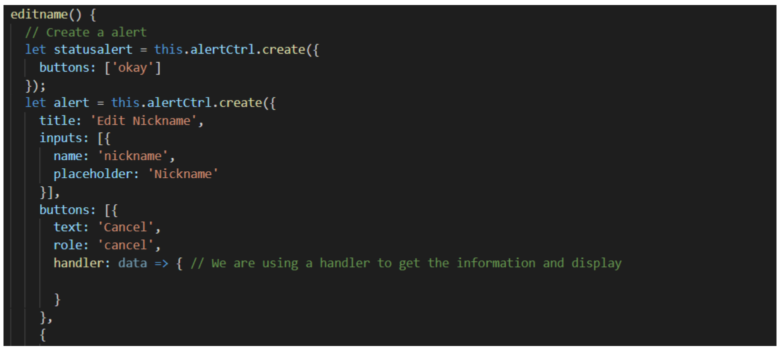
On profile.ts loaduserdetails() method attempts to load the user details whenever the user enters this tab. And the logout() method brings the user to the LoginPage. And the editname() method



```
loaduserdetails() {
  this.userservice.getuserdetails().
    then((res: any) => {
      this.displayName = res.displayName
    })
}
```



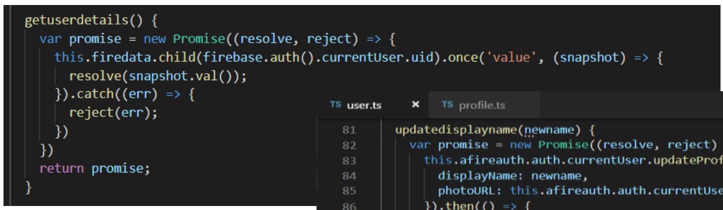
```
logout() {
  firebase.auth().signOut().then(() => {
    this.navCtrl.parent.parent.setRoot('LoginPage');
  })
}
```



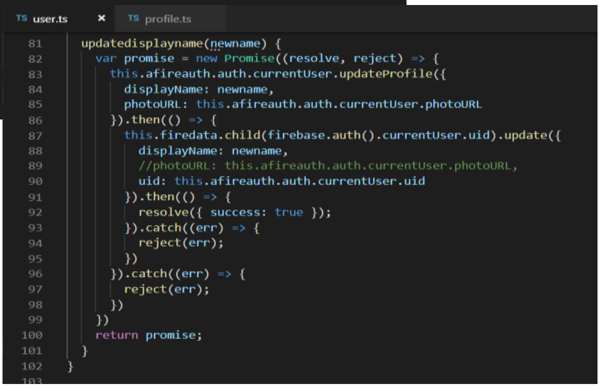
```
editname() {
  // Create a alert
  let statusalert = this.alertCtrl.create({
    buttons: ['okay']
  });
  let alert = this.alertCtrl.create({
    title: 'Edit Nickname',
    inputs: [{
      name: 'nickname',
      placeholder: 'Nickname'
    }],
    buttons: [{
      text: 'Cancel',
      role: 'cancel',
      handler: data => { // We are using a handler to get the information and display

      }
    },
    {
```

makes it possible for the user to edit his name. An alert is created after the edition is completed. in the handler for the edit button we are calling a new method called updatedisplayname()

In the user.ts the method getusersdetail (email) is created.



```
getuserdetails() {
  var promise = new Promise((resolve, reject) => {
    this.firedata.child(firebase.auth().currentUser.uid).once('value', (snapshot) => {
      resolve(snapshot.val());
    }).catch((err) => {
      reject(err);
    })
  })
  return promise;
}
```

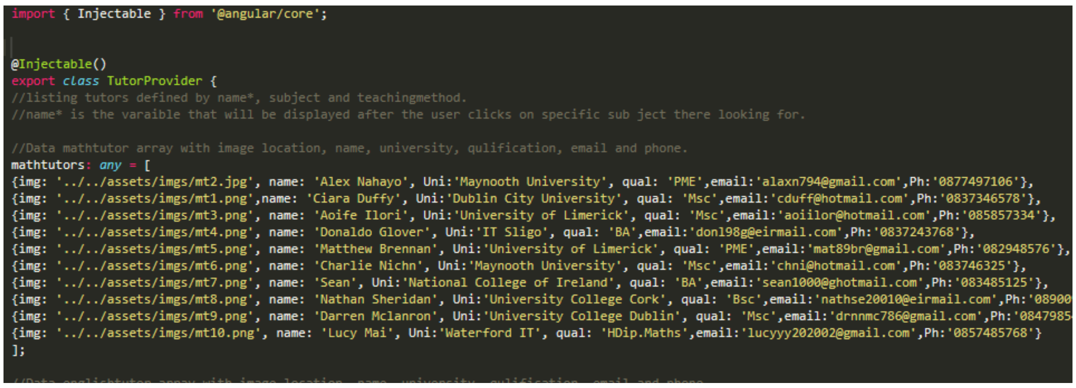


```
  updatedisplayname(newname) {
    var promise = new Promise((resolve, reject) => {
      this.afireauth.auth.currentUser.updateProfile({
        displayName: newname,
        photoURL: this.afireauth.auth.currentUser.photoURL
      }).then(() => {
        this.firedata.child(firebase.auth().currentUser.uid).update({
          displayName: newname,
          //photoURL: this.afireauth.auth.currentUser.photoURL,
          uid: this.afireauth.auth.currentUser.uid
        }).then(() => {
          resolve({ success: true });
        }).catch((err) => {
          reject(err);
        })
      }).catch((err) => {
        reject(err);
      })
    })
    return promise;
  }
}
```

Here it is being made a call to the node (using the uid as the primary key) and getting the values which will be passed to the profile tab on the loaduserdetails() method. And the updatedisplayname() method is created inside the user.ts as well it makes possible updating the profile of the user based on this UID.

6. Browse Tutor:

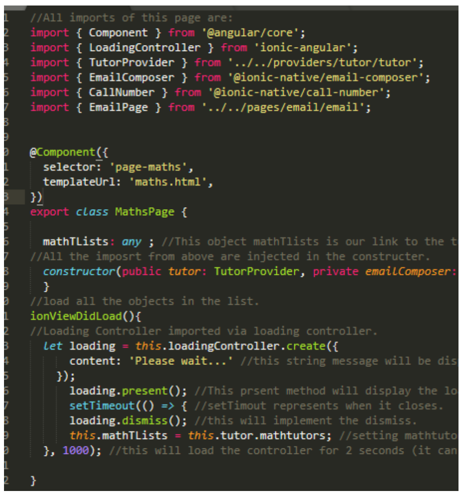
Transferring data between pages was particularly important in displaying the availability of the tutors. The interaction between the typescript and the html file was crucial in order to transfer data sufficiently. A provider was used to achieve this. In the provider labelled *tutor.ts,* 5 arrays were created one for each subject (Figure 4). Each array consisted of variables such as image (reference to assets folder), name, university and qualification. Images, names, university status and qualification



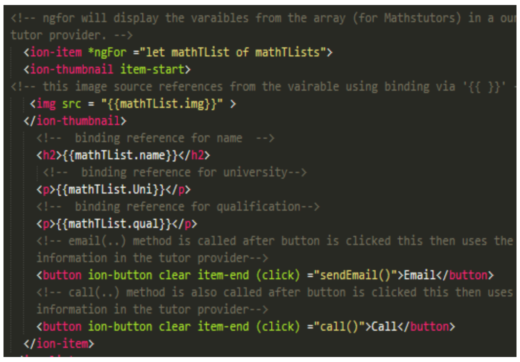
```
import { Injectable } from '@angular/core';


@Injectable()
export class TutorProvider {
//listing tutors defined by name*, subject and teachingmethod.
//name* is the varaible that will be displayed after the user clicks on specific sub ject there looking for.

//Data mathtutor array with image location, name, university, qulification, email and phone.
mathtutors: any = [
{img: '../../assets/imgs/mt2.jpg', name: 'Alex Nahayo', Uni:'Maynooth University', qual: 'PME',email:'alaxn794@gmail.com',Ph:'0877497106'},
{img: '../../assets/imgs/mt1.png',name: 'Ciara Duffy', Uni:'Dublin City University', qual: 'Msc',email:'cduff@hotmail.com',Ph:'0837346578'},
{img: '../../assets/imgs/mt3.png', name: 'Aoife Ilori', Uni:'University of Limerick', qual: 'Msc',email:'aoiilor@hotmail.com',Ph:'085857334'},
{img: '../../assets/imgs/mt4.png', name: 'Donaldo Glover', Uni:'IT Sligo', qual: 'BA',email:'don198g@eirmail.com',Ph:'0837243768'},
{img: '../../assets/imgs/mt5.png', name: 'Matthew Brennan', Uni:'University of Limerick', qual: 'PME',email:'mat89br@gmail.com',Ph:'082948576'},
{img: '../../assets/imgs/mt6.png', name: 'Charlie Nichn', Uni:'Maynooth University', qual: 'Msc',email:'chni@hotmail.com',Ph:'083746325'},
{img: '../../assets/imgs/mt7.png', name: 'Sean', Uni:'National College of Ireland', qual: 'BA',email:'sean1000@ghotmail.com',Ph:'083485125'},
{img: '../../assets/imgs/mt8.png', name: 'Nathan Sheridan', Uni:'University College Cork', qual: 'Bsc',email:'nathse20010@eirmail.com',Ph:'08900
{img: '../../assets/imgs/mt9.png', name: 'Darren Mclanron', Uni:'University College Dublin', qual: 'Msc',email:'drnnmc786@gmail.com',Ph:'0847985
{img: '../../assets/imgs/mt10.png', name: 'Lucy Mai', Uni:'Waterford IT', qual: 'HDip.Maths',email:'lucyyy202002@gmail.com',Ph:'0857485768'}
];
```

were the only variables that are displayed on the corresponding typescript files of the different subjects via the tutor provider. The mechanics for this page interaction is quite simple. Firstly, the tutor provider is imported into each subject typescript file and is then injected into a constructor.

A variable is then declared outside the constructor i.e. 'mathLists: any; 'which is then initialized in



```
//All imports of this page are:
import { Component } from '@angular/core';
import { LoadingController } from 'ionic-angular';
import { TutorProvider } from '../../providers/tutor/tutor';
import { EmailComposer } from '@ionic-native/email-composer';
import { CallNumber } from '@ionic-native/call-number';
import { EmailPage } from '../../pages/email/email';


@Component({
  selector: 'page-maths',
  templateUrl: 'maths.html',
})
export class MathsPage {

  mathTLists: any ; //This object mathTlists is our link to the t
//All the imposrt from above are injected in the constructer.
  constructor(public tutor: TutorProvider, private emailComposer:
  }
//load all the objects in the list.
ionViewDidLoad(){
//Loading Controller imported via loading controller.
  let loading = this.loadingController.create({
      content: 'Please wait...' //this string message will be dis
    });
      loading.present(); //This prsent method will display the lo
      setTimeout(() => { //setTimout represents when it closes.
      loading.dismiss(); //this will implement the dismiss.
      this.mathTLists = this.tutor.mathtutors; //setting mathtuto
  }, 1000); //this will load the controller for 2 seconds (it can

}
```



```
<!-- ngfor will display the varaibles from the array (for Mathstutors) in a ou
tutor provider. -->
  <ion-item *ngFor ="let mathTList of mathTLists">
  <ion-thumbnail item-start>
<!-- this image source references from the vairable using binding via '{{ }}'
   <img src = "{{mathTList.img}}" >
  </ion-thumbnail>
    <!--  binding reference for name  -->
    <h2>{{mathTList.name}}</h2>
     <!--  binding reference for university-->
    <p>{{mathTList.Uni}}</p>
    <!--  binding reference for qualification-->
    <p>{{mathTList.qual}}</p>
    <!-- email(..) method is called after button is clicked this then uses the
    information in the tutor provider-->
    <button ion-button clear item-end (click) ="sendEmail()">Email</button>
    <!-- call(..) method is also called after button is clicked this then uses
    information in the tutor provider-->
    <button ion-button clear item-end (click) ="call()">Call</button>
  </ion-item>
```

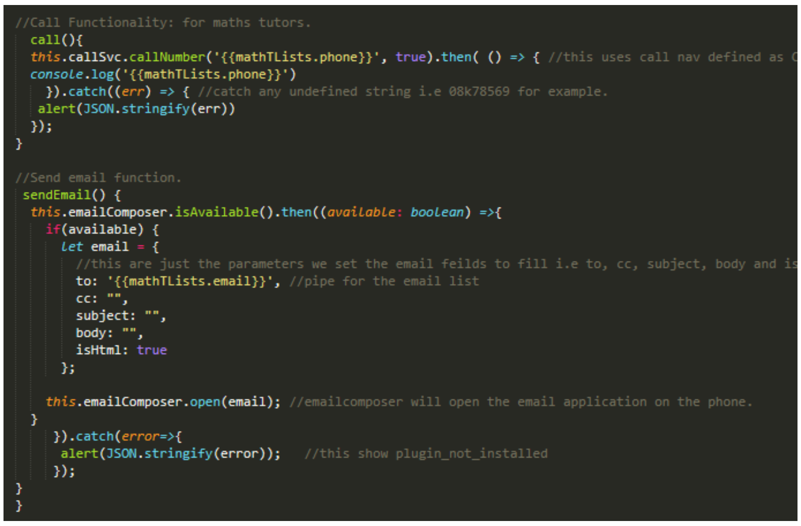
ionViewDidLoad function as 'this.mathTLists = this.tutor.mathtutors;' (Figure 5) . This initialisation is now recognised by our provider. Now that we have made a sufficient path from the provider to the typescript file we can now display the data onto the html file. (Figure 5) illustrates how we use biding ("{{mathTList.img}}") in order to display the data from the provider along with 'ngFor' to display all the variables accordingly. Hence, this process is repeated for the entire subject list.

Call function

The call feature was created by importing the ‘CallNumber’ (via Ionic Cordova plugin) and unfortunately required deployment for illustration. As you can see from (Figure 6) a simple function was created labeled ‘call’, inside call the call plugin was initialized through a promise. This promise was set to true and contained a string input value for the phone number. For the import number binding was used. To call the call function, a reference is inserted via a clicked ion tag in the html file.

Email function

The email feature was implemented by importing the ‘EmailCompser’ (via Ionic Cordova plugin) and also required deployment. Similar to the call function it uses a promise however in this instance an array called ‘email’ is created with parameters such as ‘to’ (using binding from tutor provider), ‘cc’, ‘subject’ and ‘body’ to replicate an email field (Figure 6). Once the email function is called in the html



```
//Call Functionality: for maths tutors.
  call(){
  this.callSvc.callNumber('{{mathTLists.phone}}', true).then( () => { //this uses call nav defined as C
  console.log('{{mathTLists.phone}}')
    }).catch((err) => { //catch any undefined string i.e 08k78569 for example.
   alert(JSON.stringify(err))
  });
}

//Send email function.
 sendEmail() {
  this.emailComposer.isAvailable().then((available: boolean) =>{
    if(available) {
      let email = {
        //this are just the parameters we set the email feilds to fill i.e to, cc, subject, body and is
        to: '{{mathTLists.email}}', //pipe for the email list
        cc: "",
        subject: "",
        body: "",
        isHtml: true
      };

    this.emailComposer.open(email); //emailcomposer will open the email application on the phone.
  }
     }).catch(error=>{
      alert(JSON.stringify(error));   //this show plugin_not_installed
     });
}
}
```

file, ‘this,emailCompser.open’ declaration is satisfied through the promise resulting in the launching of an email application, otherwise an alert error will be presented.

Evaluation:

It is important to highlight certain concerns encountered in the process of developing this app. Some functional requirements were left out during development. This is as a result of the limited development time frame. These components could be improved upon to deliver a more robust user experience. Worthy of note is the limited functionality of the tutor and students. Future work and upgrade can accommodate the following functionalities:

Student:

Rate & Review Tutor:

Users can rate the tutor as per the quality of services received. According to the teaching abilities, time adherence and behavior user can post the reviews about the concerned tutor.

Direct & Social Media Login:

Here, the users can log into an already created account. Or else, they can sign up via their email or any other social media account.

Book A Tutor: After reading all the reviews about the tutor, users can book him/her so as to start the subjects and courses as selected and charted out by the user.

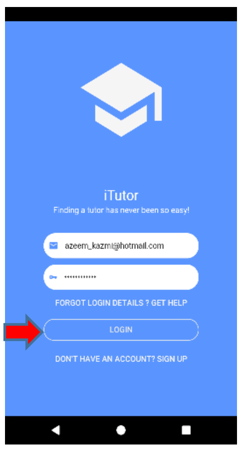
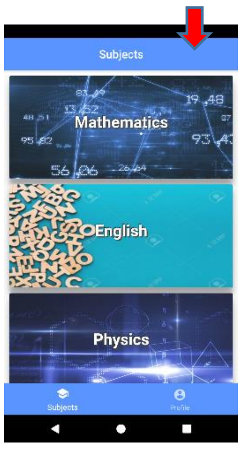
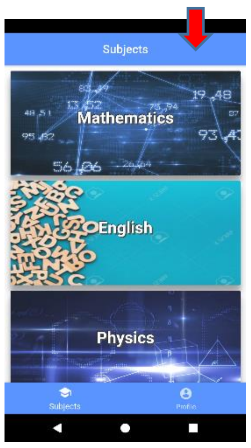
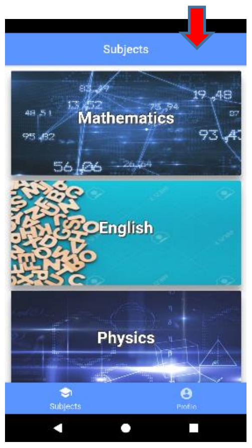
Change Availability timings: As per the number of the tuitions queued, the timings can be changed and altered by the tutors according to their convenience.

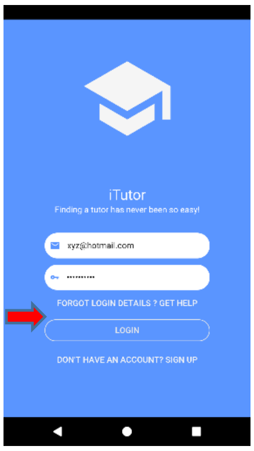
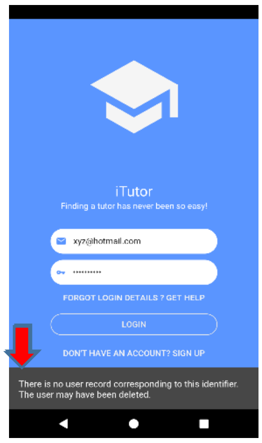
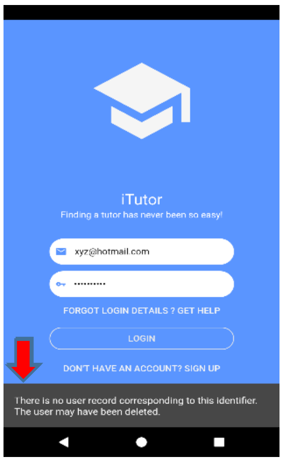
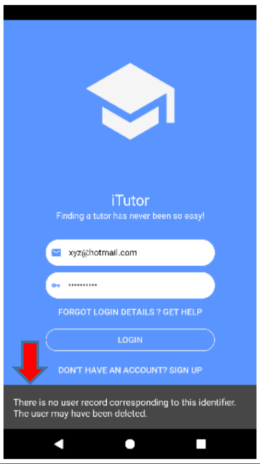
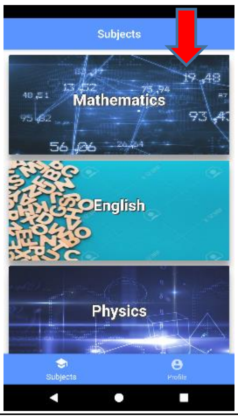
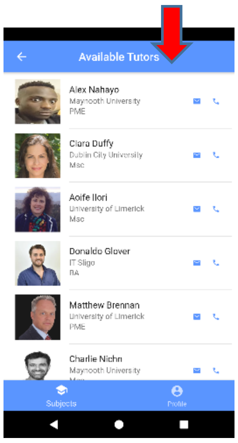
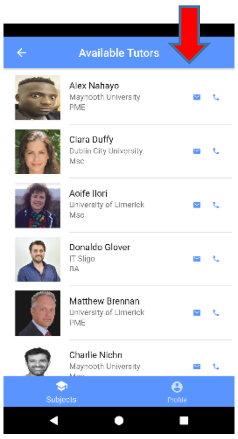
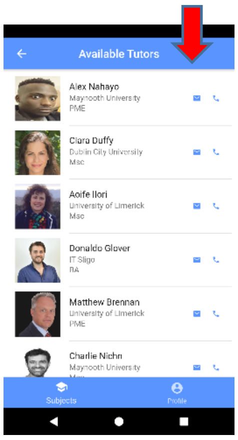
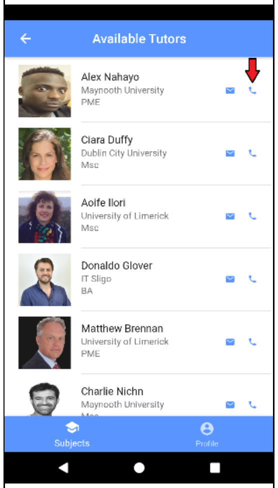
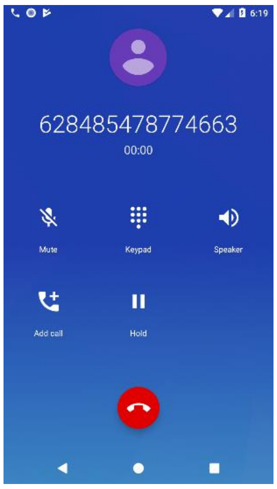
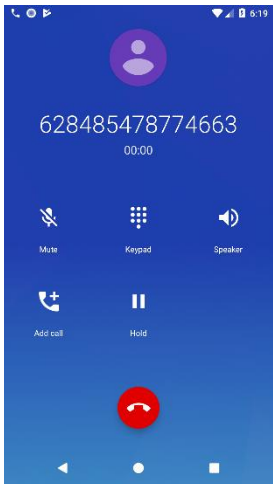
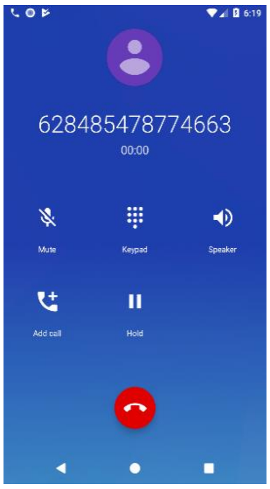
Tutor:

Chat with Student: The tutor can communicate with the students for answering their queries and can also update them with the change in timings schedules if arises and vice-versa.

iTutor Mobile Application Testing:

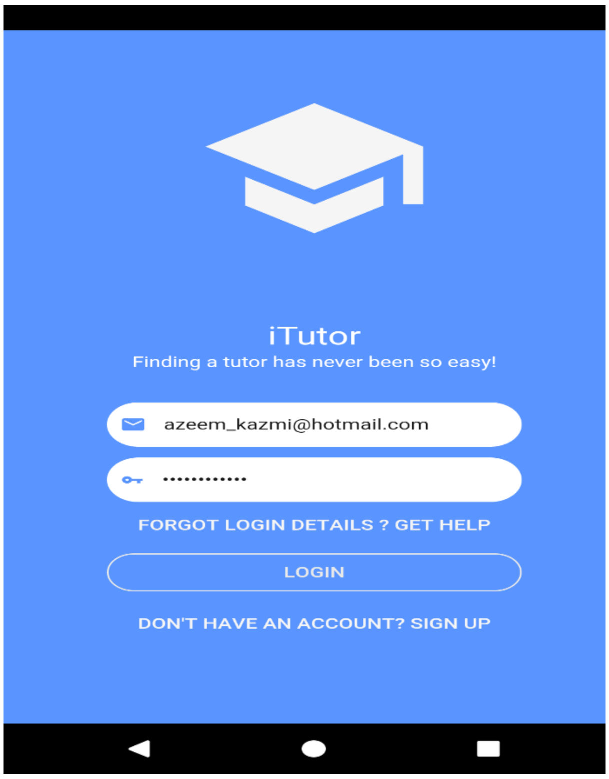
Purpose: The purpose of testing iTutor application is to check the errors in the application.	Prerequisites for this test**: None**
Test Run Information: **Tester:** Group G2 **Date of Test:** 19/12/2018	**Application**: iTutor 1.0 **Browser** [used]: Google Chrome Version 71.0.3578.98 **Database**: Firebase **Operating System:** Windows
	Required Configuration: [browser setup] No special setup needed
Delivery/RESULT: As this is an App in its Beta stage, the embedded functionalities are working well and the test cases run successfully. The iTutor application is running smoothly and error free. The app is user friendly. Though the App delivery is mostly based on the student/user perspective, there is room for improvement to include functional features to support tutors as well.	

Test Case	TEST STEP/INPUT	EXPECTED RESULTS	ACTUAL RESULTS	Overall Result
1.	User enter valid username and password. And click on login button.  iTutor Finding a tutor has never been so easy! azeem_kazmi@hotmail.com FORGOT LOGIN DETAILS ? GET HELP LOGIN DON'T HAVE AN ACCOUNT? SIGN UP	Once a valid user signed in homepage will be display on the screen with the list of subjects.  Subjects Mathematics English Physics Subjects Profile	Show a screen of home page with the list of the subjects.  Subjects Mathematics English Physics Subjects Profile	Test Successful – actual output is the same as the expected output.  Subjects Mathematics English Physics Subjects Profile

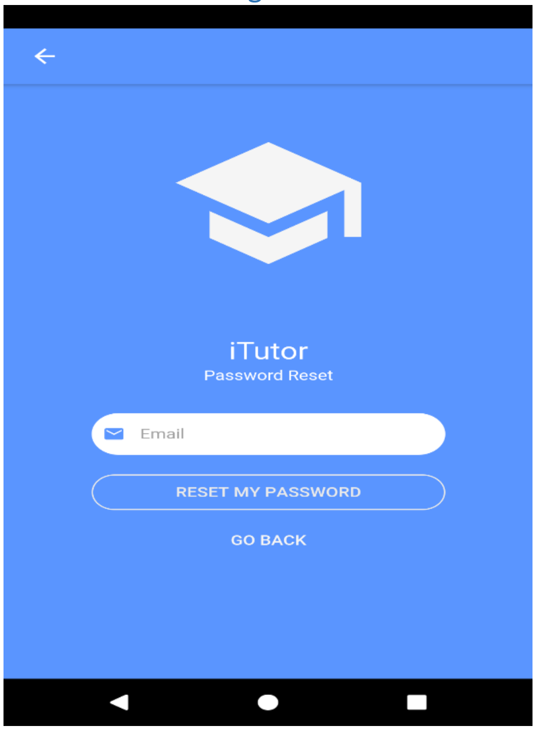
2.	An unauthorized user entre username and password.  iTutor Finding a tutor has never been so easy! xyz@hotmail.com FORGOT LOGIN DETAILS ? GET HELP LOGIN DON'T HAVE AN ACCOUNT? SIGN UP	A pop up message will show on the screen: "There is no user record corresponding identifier. The user may have been deleted.  iTutor Finding a tutor has never been so easy! xyz@hotmail.com FORGOT LOGIN DETAILS ? GET HELP LOGIN DON'T HAVE AN ACCOUNT? SIGN UP There is no user record corresponding to this identifier. The user may have been deleted.	Show a pop up message on the screen: "There is no user record corresponding identifier. The user may have been deleted.  iTutor Finding a tutor has never been so easy! xyz@hotmail.com FORGOT LOGIN DETAILS ? GET HELP LOGIN DON'T HAVE AN ACCOUNT? SIGN UP There is no user record corresponding to this identifier. The user may have been deleted.	Test Successful – actual output is the same as the expected output.  iTutor Finding a tutor has never been so easy! xyz@hotmail.com FORGOT LOGIN DETAILS ? GET HELP LOGIN DON'T HAVE AN ACCOUNT? SIGN UP There is no user record corresponding to this identifier. The user may have been deleted.
	User clicked on Mathematics button in Homepage.  Subjects Mathematics English Physics Subjects Profile	List of the mathematics tutors will display on the screen with their details.  Available Tutors Alex Nahayo Maynooth University PME Clara Duffy Dublin City University Msc Aoife Ilori University of Limerick Msc Donaldo Glover IT Sligo BA Matthew Brennan University of Limerick PME Charlie Nichn Maynooth University Subjects Profile	Show the list of all the mathematics tutors with their details.  Available Tutors Alex Nahayo Maynooth University PME Clara Duffy Dublin City University Msc Aoife Ilori University of Limerick Msc Donaldo Glover IT Sligo BA Matthew Brennan University of Limerick PME Charlie Nichn Maynooth University Subjects Profile	Test Successful – actual output is the same as the expected output.  Available Tutors Alex Nahayo Maynooth University PME Clara Duffy Dublin City University Msc Aoife Ilori University of Limerick Msc Donaldo Glover IT Sligo BA Matthew Brennan University of Limerick PME Charlie Nichn Maynooth University Subjects Profile
2.	User choose a tutor in available tutors list and press the call button to contact.  Available Tutors Alex Nahayo Maynooth University PME Clara Duffy Dublin City University Msc Aoife Ilori University of Limerick Msc Donaldo Glover IT Sligo BA Matthew Brennan University of Limerick PME Charlie Nichn Maynooth University Subjects Profile	A native mobile phone calling app will appear on the screen with the contact number of the selected tutor.  6:19 6284854787774663 00:00 Mute Keypad Speaker Add call Hold	Show the native mobile calling app on the screen with contact number of the selected tutor.  6:19 6284854787774663 00:00 Mute Keypad Speaker Add call Hold	Test Successful – actual output is the same as the expected output.  6:19 6284854787774663 00:00 Mute Keypad Speaker Add call Hold

Appendix A – iTutor App Overview:

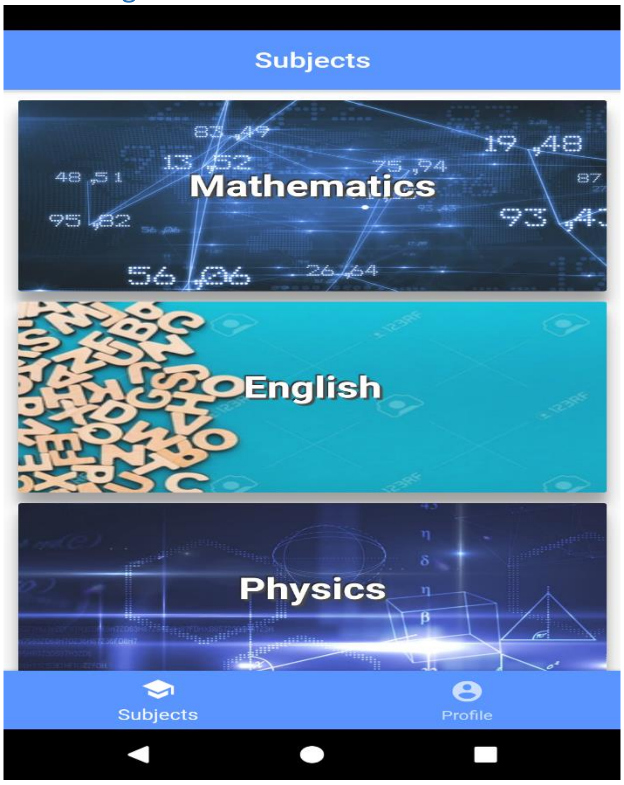
Login page:


iTutor
Finding a tutor has never been so easy!
azeem_kazmi@hotmail.com
••••••••••••
FORGOT LOGIN DETAILS ? GET HELP
LOGIN
DON'T HAVE AN ACCOUNT? SIGN UP

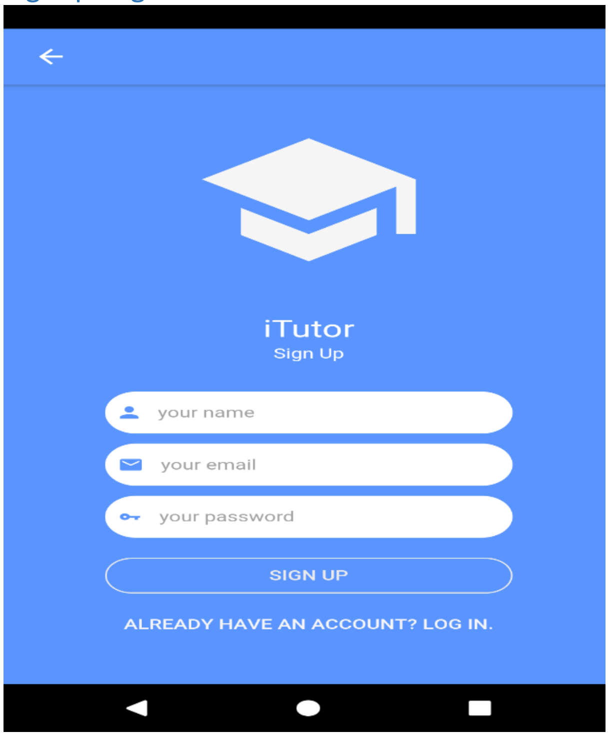
Password Reset Page:



Home Page:

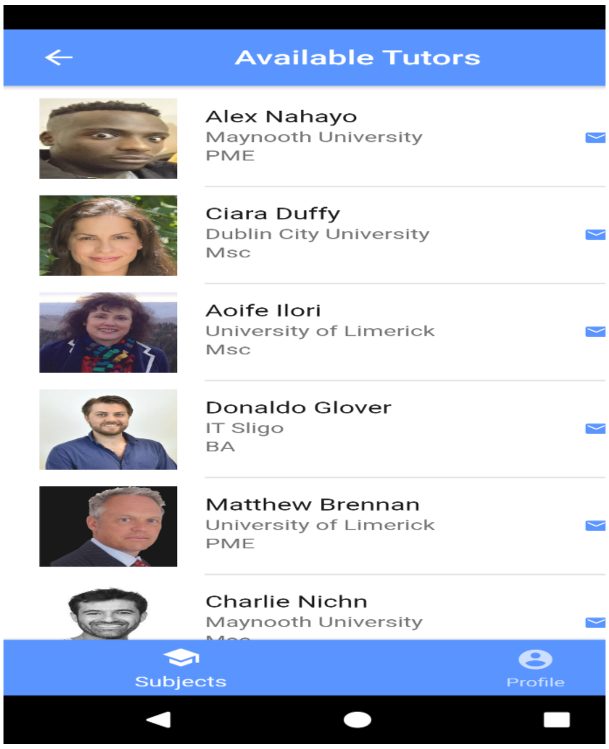


Signup Page



Tutor List Page:

Email button Page:



Available Tutors

Alex Nahayo
Maynooth University
PME

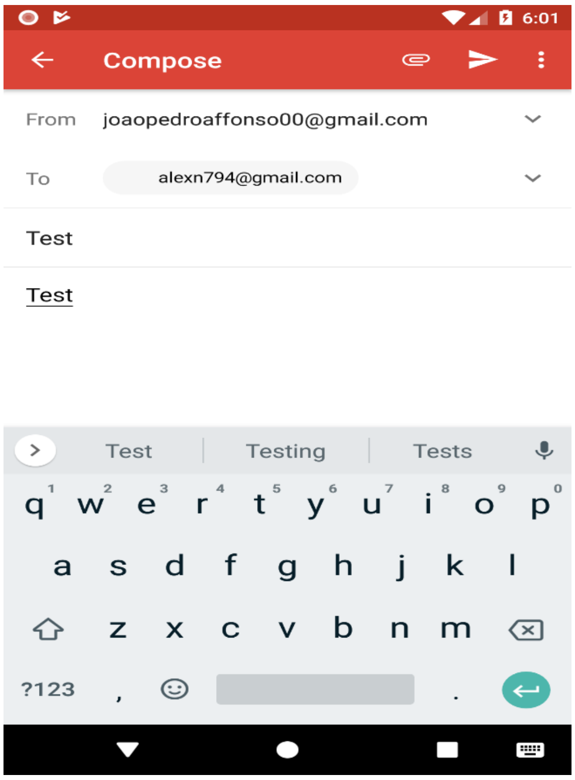
Ciara Duffy
Dublin City University
Msc

Aoife Ilori
University of Limerick
Msc

Donaldo Glover
IT Sligo
BA

Matthew Brennan
University of Limerick
PME

Charlie Nichn
Maynooth University



Compose

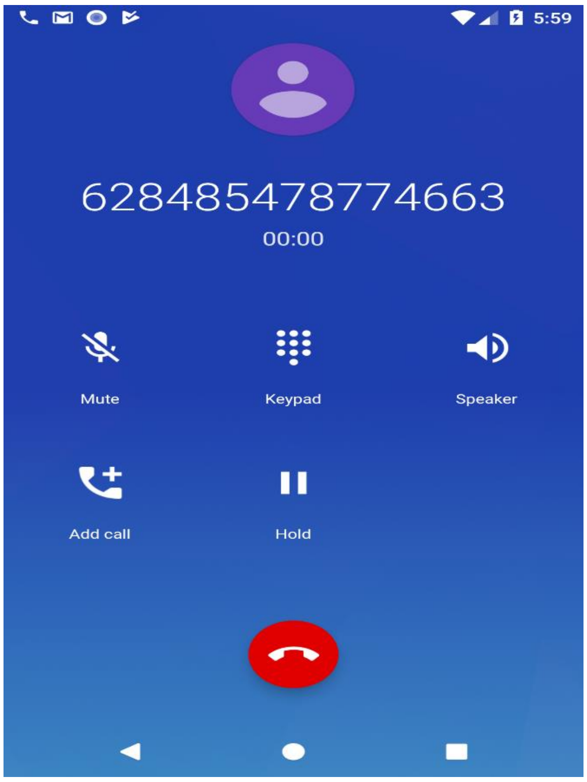
From joaopedroaffonso00@gmail.com

To alexn794@gmail.com

Test

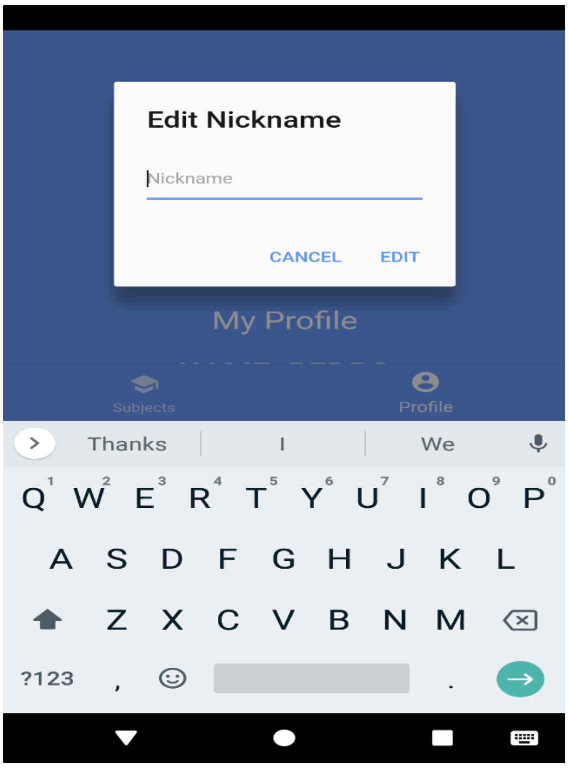
Test

Call button page:



6284854787774663
00:00

Edit profile page:



Edit Nickname

Nickname

CANCEL EDIT