Archivos importantes en js

Nuevo.js

/signup

Recibe la información enviada desde el cliente y las guarda en las variables username, email, password, nme, lastname, place, years, imei. Busca en base de datos el correo que recibió, y si no existe, guarda la información en un nuevo documento en base de datos.

Dentro de ese proceso, antes de guardar los datos, encripta la contraseña. Después, genera un token dentro del servidor, no es el de firebase.

Si el correo existe, entonces no guarda la información.

```
router.post('/signup', async(req, res) => {
  try {
      // Receiving Data
      let resultado;
      const { username, email, password, nme, lastname, place, years, imei } = req.body;
      const valida = await User.findOne({ email: req.body.email })
      if (!valida) {

          // Creating a new User
          const user = new User({
              nme,
              lastname,
              place,
              years,
              username,
              email,
              password,
              imei
          });
          user.password = await user.encryptPassword(password);
          await user.save();
          // Create a Token
          const token = jwt.sign({ id: user.id }, config.secret, {
              expiresIn: 60 * 60 * 24 // expires in 24 hours
          });
          /* resjson({ auth: true, token });*/
          res.status(404).send("The email doesn't exists")
      }
      else{
          console.log("USUARIO ENCONTRADO: ", valida);
          // AQUI PUEDES LLAMAR A TU SIGUIENTE MIDDLEWARE O DEVOLVER EL RESULTADO
          //next();
          res.status(200).json(valida);
      }
  } catch (e) {
      console.log("problema",e)
      res.status(500).send('There was a problem registering your user');
  }
```

/signin

Recibe información del cliente y busca en la base de datos el email, si existe, entonces valida la contraseña y el imei.

```
router.post('/signin', async(req, res) => {
    try {
        const user = await User.findOne({ email: req.body.email })
        if (!user) {
            return res.status(404).send("The email doesn't exists")
        }
        const validPassword = await user.validatePassword(req.body.password, user.password);
        const validImei = await User.findOne({ imei: req.body.imei});
        if (!validPassword && !validImei) {
            return res.status(401).send({ auth: false, token: null });
        }
        const token = jwt.sign({ id: user._id }, config.secret, {
            expiresIn: '24h'
        });
        console.log('imei registrado:', req.body.imei,' imei ingresado:',user.imei, 'validimei',validImei,'pass:', validPassword);
        res.status(200).json({ auth: true, token });
    } catch (e) {
        console.log(e)
        res.status(500).send('There was a problem signin');
    }
});
```

/result

Recibe información del cliente. Por medio de una expresión regular determina en qué colección de la base de datos se va a guardar el tema del usuario que terminó el test.

Una vez que termina de guardar todos los temas en su respectiva colección, manda a llamar a iniciarNotificacion.

```
router.post('/result', async(req, res) => {
  const regEx1 = new RegExp('Tema[0-9]*0$|Tema[0-9]*6$','g')
        const regEx2 = new RegExp('Tema[0-9]*1$|Tema[0-9]*7$','g')
        const regEx3 = new RegExp('Tema[0-9]*2$|Tema[0-9]*8$','g')
        const regEx4 = new RegExp('Tema[0-9]*3$|Tema[0-9]*9$','g')
        const regEx5 = new RegExp('Tema[0-9]*4$','g')
        const regEx6 = new RegExp('Tema[0-9]*5$','g')
            // Creating a new User
    try{
        var array = Notification.dividirCadena(temas,", ")
        for (i=0;i<array.length;i++){
          if(regEx1.test(array[i])){
            console.log('sí coincide 1')
            var tema = array[i]
            const user = new Uno({
              usr,
              tema,
              token,
              tipo
            });
            await user.save();
          }
          else if(regEx2.test(array[i])){
            var tema = array[i]
            console.log('sí coincide 2')
            const user = new Dos({
              usr,
              tema,
              token,
              tipo
            });
            await user.save();
          }
          else if(regEx3.test(array[i])){
            var tema = array[i]
            console.log('sí coincide 3')
            const user = new Tres({
```

```
            await user.save();
          }
          else if(regEx6.test(array[i])){
            var tema = array[i]
            console.log('sí coincide 6')
            const user = new Seis({
              usr,
              tema,
              token,
              tipo
            });
            await user.save();
          }
          else {
            console.log("Algo falla, ninguno coincide")
          }
        }
          await Notification.iniciarNotificacion()
          console.log('me iniciarion');
          res.status(200).json(usr);
      }
      catch (e) {
        console.log("problema",e)
        res.status(500).send('There was a problem registering your user');
    }
});
```

Cron.js

iniciarNotificacion()

Pone en true a control, detiene todos los crons que estén ejecutándose e inicia el cron “uno”

```
async function iniciarNotificacion(){
      control=true
      todo()
      uno.start();
}
exports.iniciarNotificacion =  iniciarNotificacion;
```

CronJobs

Busca en base de datos los temas de la colección uno y obtiene todos los documentos. Los guarda en la variable documentos. Para cada elemento de documentos obtiene el token y busca la información de la notificación del tema en turno en la colección notificaciones. Una vez que lo encuentra, manda a llamar a enviarNotificacion y le pasa la información encontrada y el token para enviarlo a través de FCM.

Control se establece en false, se detiene el cron actual. Esto para que una vez detenido, el cron decida qué es lo siguiente que tiene que realizar. Si control está en true, no hace nada más que imprimir, si es false, pasa al siguiente cron por medio del método crons.

Si el documento que busca en base de datos no existe, solo se detiene y pasa al siguiente cron.

```
var uno = new CronJob(' */1 * * * *',  async ()=>{
  const documentos = await temasuno.find({tipo: 'notificacion'}).sort({$natural:-1}).limit()
  if(documentos){
      for (i =0; i<documentos.length; i++){
          tok = documentos[i]['token']
          const notificacion = await noti.findOne({nombre: documentos[i]['tema']}).sort({$natural: -1}).limit(1)
          enviarNotificacion(notificacion,tok)
      }
      control=false;
      uno.stop()
  }
  else{
      control=false;
      uno.stop()
  }
},()=>{control?console.log('ya se detuvo'):crons('dos')});
```

Crons()

Comienza el cron que está determinado por el parámetro que recibe.

```
async function crons(parametro){
    //Si es false significa que no están activas las cron
        switch(parametro){
            case 'uno':
                //El método start inicia el cronjob de la variable uno
                uno.start();
                break;
            case 'dos':
                dos.start();
                break;
            case 'tres':
                tres.start();
                break;
            case 'cuatro':
                cuatro.start();
                break;
            case 'cinco':
                cinco.start();
                break;
            case 'seis':
                seis.start();
                break;
        }

}
```

Todo()

Detiene todos los cronJobs declarados

```
function todo(){
    uno.stop();
    dos.stop();
    tres.stop();
    cuatro.stop();
    cinco.stop();
}
```

enviarNotificacion

Recibe la información o contenido de la notificación extraída de base de datos y los acomoda en una variable data con el formato de notificación para poder enviarlo a través de FCM.

Envía después data a través de sendPushToOneUser

```
async function enviarNotificacion (titulo, token)  {
    control=false;
    const data = {
        tokenId: token,
        titulo: titulo['titulo'],
        mensaje: titulo['mensaje'],
        imagen: titulo['url']
    }
    //Método para hacer la conexión a Firebase Cloud Message y enviar la notificación al dispositivo
    Notification.sendPushToOneUser(data);

}
```

dividirCadena

Es un método para convertir un string en un array por medio de un separador, que es el carácter que va a determinar cada cuando se va a separar la cadena y convertirse en un elemento del array. Replace elimina los caracteres especificados y Split es el que separa.

```
//Método para convertir un String en un array
function dividirCadena(cadenaADividir,separador) {
    //Borra los caracteres [ y ]
    var regex1 = cadenaADividir.replace(/\[/g, '');
    var regex = regex1.replace(/\]/g, '');
    /*El método split separa los n caracteres del String para convertirlos en elementos de un Array.
      Los separa con base en separador, que es un caracter o cadena que se debe encontrar dentro del
      string, que en este caso es ', '*/
    var arrayDeCadenas = regex.split(separador);
    return arrayDeCadenas;
}
exports.dividirCadena = dividirCadena;
```

Notificación.js

initiFirebase()

es el que hace la conexión a firebase por medio del archivo json que da el proyecto de firebase cuando haces la configuración de FCM. Para obtener ese json tienes que registrar la aplicación y te va a dar uno, ese no es el que va ahí, pero sí debes tenerlo en tu backend. Cuando configuras el FCM te da otro json y es ese el que se pone aquí. El vídeo para configurarlo está hasta abajo.

```
function initFirebase() {
    const serviceAccount = require('./archivodeFirebase.json');
    admin.initializeApp({
        credential: admin.credential.cert(serviceAccount),
    });
}

initFirebase();
```

sendPushToOneUser

Envia la información que recibe a través de FCM con el método sendMessage. Este mensaje es para un solo usuario.

```
async function sendPushToOneUser(notification) {

    const message = {
        token: notification.tokenId,
        data: {
            titulo: notification.titulo,
            mensaje: notification.mensaje,
            imagen: notification.imagen
        }

    }
    //console.log('hola',message);

    await sendMessage(message);
    await sleep(1000)


}
```

SendPushToTopic

Envía la información que recibe a través de FCM con el método sendMessage. Este mensaje es para todos los usuarios suscritos a un tema, según la documentación de Firebase.

```
function sendPushToTopic(notification) {
    const message = {
        topic: notification.topic,
        data: {
            titulo: notification.titulo,
            mensaje: notification.mensaje
        }
    }
    sendMessage(message);
}
```

sendMessage

Es el encargado de conectarse a FCM y enviar el mensaje que se le pasa por parámetro.

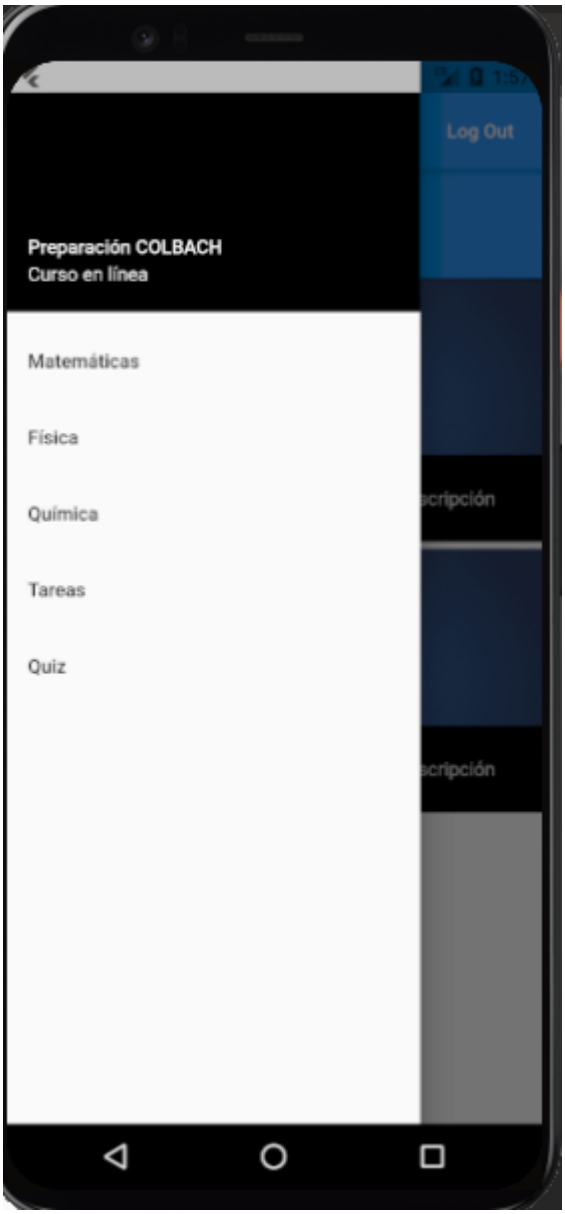
```
function sendMessage(message) {
    admin.messaging().send(message)
        .then((response) => {
            // Response is a message ID string.
            console.log('Successfully sent message:', response);
        })
        .catch((error) => {
            console.log('Error sending message:', error);
        })
}
```

En bd.js va la url de la base de datos a la que vas a hacer la conexión. Los archivos que tienen model en su nombre, son los modelos para poder crear nuevos documentos en base de datos.

APP Flutter

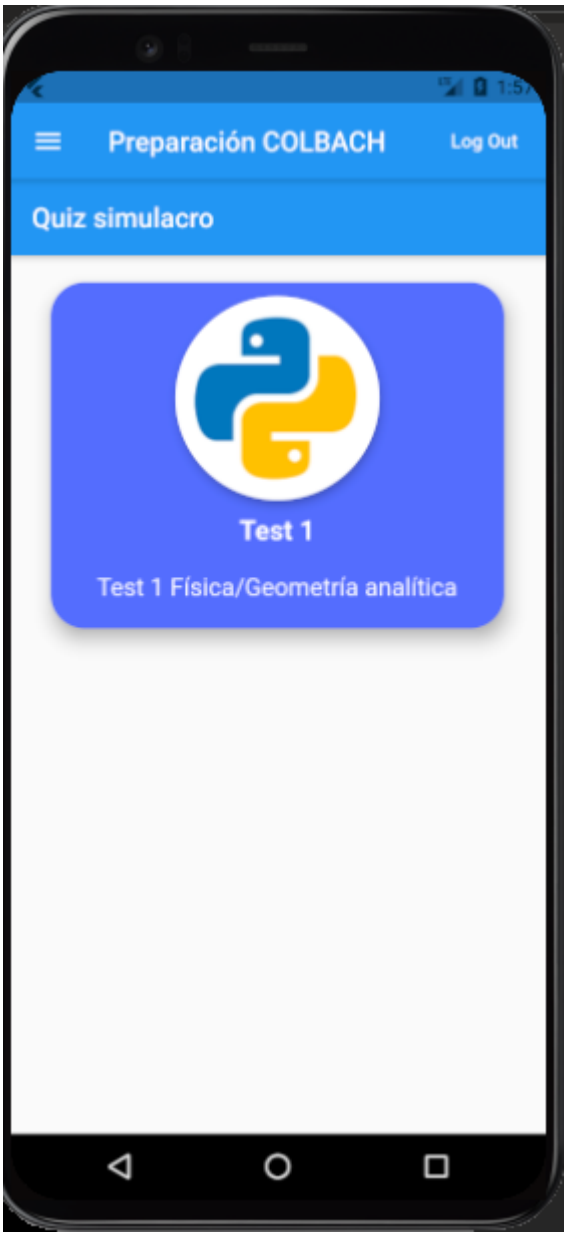
Pongo las pantallas porque casi todo el código de aquí es flutter y sus elementos (widgets). Los archivos están comentados y explican lo que hacen, aunque casi todos dicen únicamente la parte visual a la que corresponden pues explicarlos más a detalle sería explicar cada elemento de flutter y la teoría no la sé. Los fragmentos de código que aparecen no tienen pantalla. Los demás sí y como dije, están comentados para explicar qué hacen.

Pantallas correspondientes a main, Física y Química dentro de los archivos dart de la aplicación.



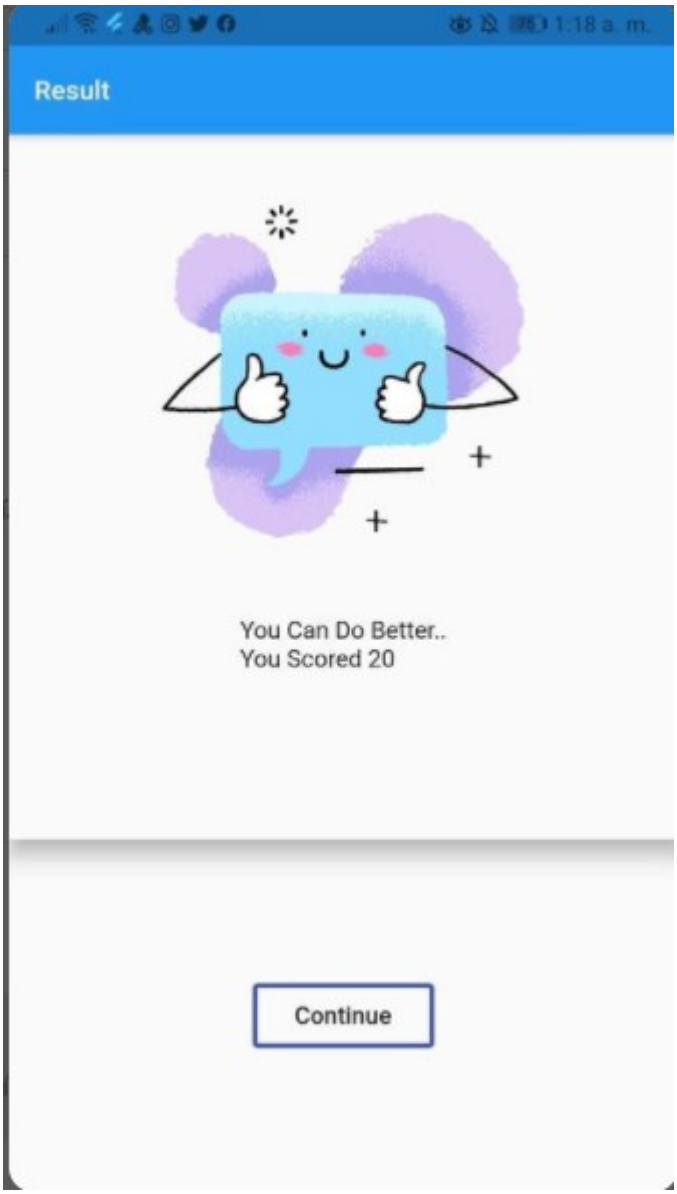
Menú lateral que aparece en las pantallas de los archivos Física, main, Química, quizhome y Tareas.

Pantalla que resulta del archivo Tareas en dart

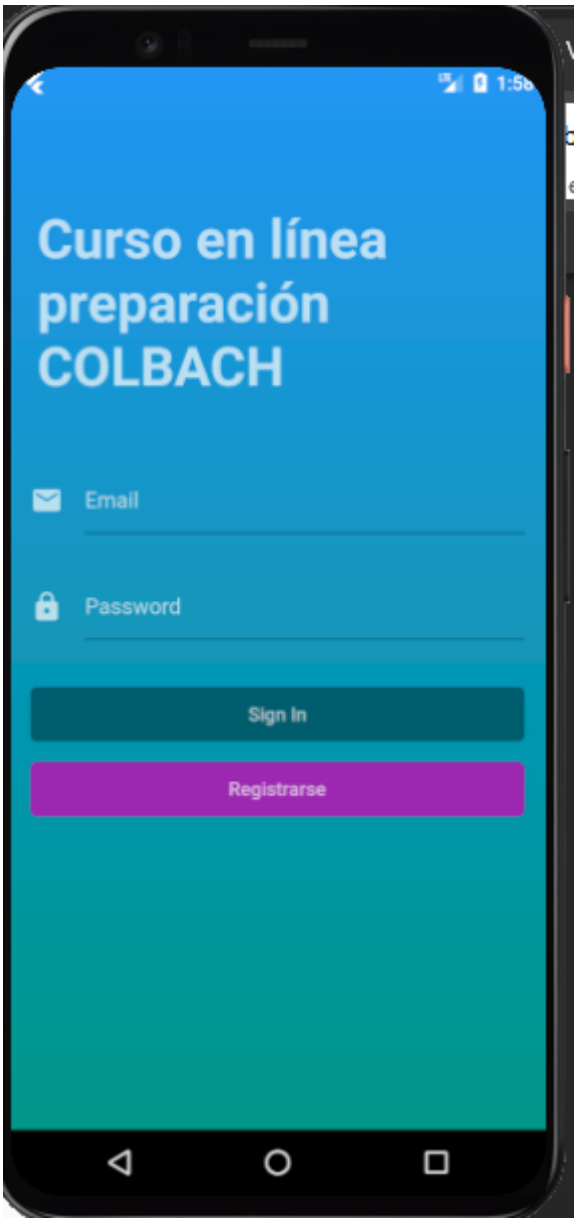


Pantalla que resulta del archivo quizhome en dart

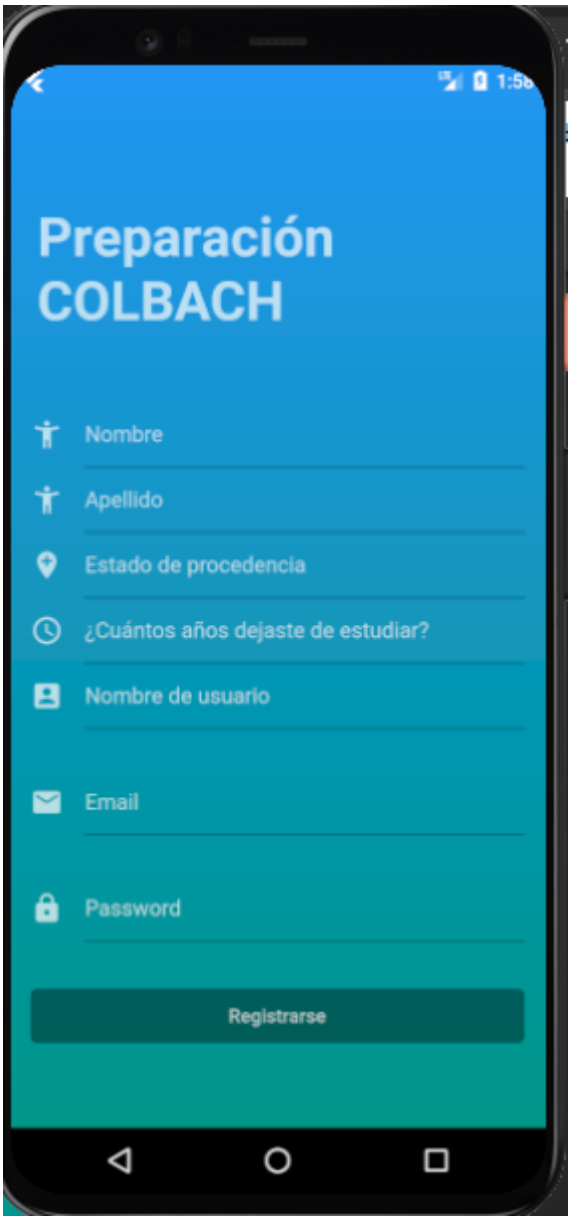
Pantalla que resulta de quizpage en dart



Pantalla que resulta del archivo resultpage en dart

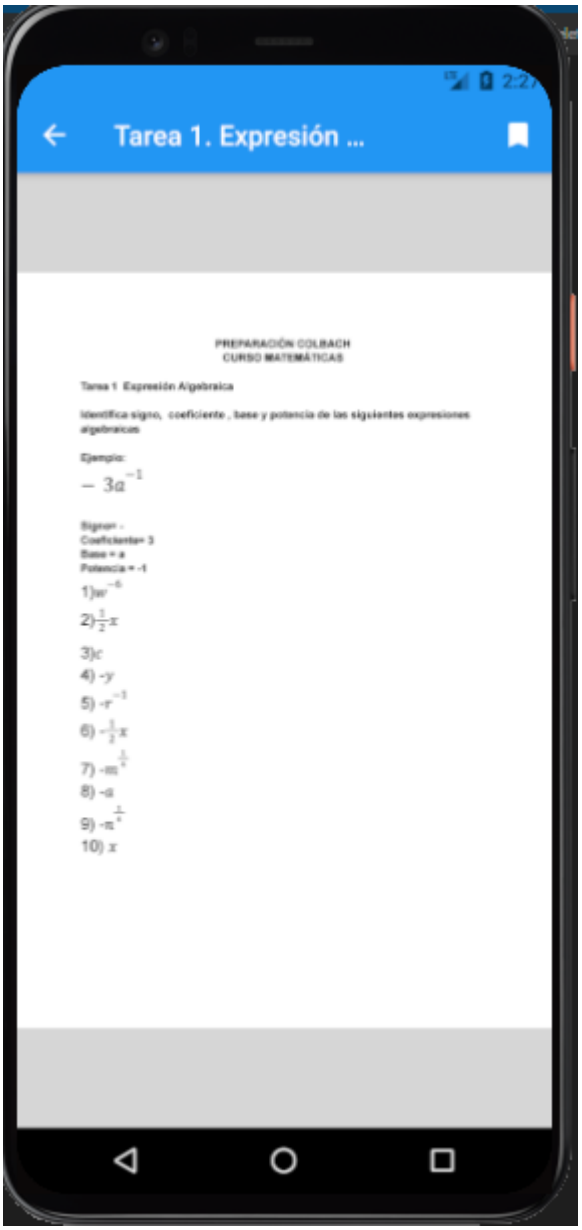


Pantalla que resulta del archivo loginPage en dart



Pantalla que resulta del archivo signupPage en dart

Pantalla que resulta del archivo VideoControlPage en los archivos de dart



Pantalla que resulta de los archivos pdfviewer, tarea2, tarea3, tarea4 … dentro de los archivos de dart

Respuesta.dart

Contiene la clase Respuesta con el método NotificaIncorrecto, que es el encargado de enviar la información de los temas al backend a través de /result. Obtiene el token FCM y crea un usrtoken.

```dart
class Respuesta {

  Notificaincorrecto(List tema) async{
    print('finale $tema');
    var token = await notificaciones.getToken();
    var usrToken = await notificaciones.getUsrToken();

  SharedPreferences sharedPreferences = await SharedPreferences.getInstance();
    Map data = {
      'usr': usrToken,
      'temas': tema.toString(),
      'token': token
    };
    print(data);
    var jsonResponse = null;

    //var response = await http.post(Uri.parse("http://192.168.56.1:3128/result"), body: data);
    //var response = await http.post(Uri.parse("https://aplicacion-2021.herokuapp.com/result"), body: data);
    var response = await http.post(Uri.parse("https://prueba-000.herokuapp.com/result"), body: data);
    print('despues del response');
    if(response.statusCode == 200) {
      print('200');
      jsonResponse = json.decode(response.body);
      print('Response status: ${response.statusCode}');
      print('Response body: ${response.body}');
      if(jsonResponse != null) {
        print('Correcto');
    }
    else {
      print('Error');
  }
}
  }
}
```

Quizpage.dart

Es en este archivo en donde se guardan las respuestas incorrectas. Lo demás deberás consultarlo en el vídeo de donde saqué el quiz.

En nextquestion es en donde se recorre el arreglo de preguntas extraídas del json. Aquí es en donde yo envío los temas incorrectos al backend llamando al método Notificaincorrecto.

```
void nextquestion() {
  canceltimer = false;
  timer = 30;
  setState(() {
    if (j < 10) {
      //Recorre todos los temas que se extrajeron del json
      i = random_array[j];
      j++;
    } else {
      //Si ya se recorrió todo, pasa a resultpage
      Navigator.of(context).pushReplacement(MaterialPageRoute(
        builder: (context) => resultpage(marks: marks),
      )); // MaterialPageRoute
      print("Hola $tema");
      //manda los temas incorrectos a Notificaincorrecto
      notificacion.Notificaincorrecto(tema);
    }
    btncolor["a"] = Colors.indigoAccent;
    btncolor["b"] = Colors.indigoAccent;
    btncolor["c"] = Colors.indigoAccent;
    btncolor["d"] = Colors.indigoAccent;
    disableAnswer = false;
  });
  starttimer();
}
```

Dentro de checkanswer es en donde guardo los temas incorrectos en un arreglo llamado tema. En la comprobación que hace el test para saber si debe pintarlo como correcto o incorrecto es en donde meto la opción de insertar el tema en el arreglo.

El arreglo mydata es el que contiene toda la información del json y los números del primer corchete corresponden al conjunto de texto definido en el json por { } (ver JSON-ESTRUCTURA en carpeta assets)

Link de donde saqué el quiz: https://www.youtube.com/watch?v=yHrpx4PoBzU

```
void checkanswer(String k) async{

  // in the previous version this was
  // mydata[2]["1"] == mydata[1]["1"][k]
  // which i forgot to change
  // so nake sure that this is now corrected
  if (mydata[2][i.toString()] == mydata[1][i.toString()][k]) {
    // just a print sattement to check the correct working
    // debugPrint(mydata[2][i.toString()] + " is equal to " + mydata[1][i.toString()][k]);
    marks = marks + 5;
    // changing the color variable to be green
    colortoshow = right;
  } else {
    // just a print sattement to check the correct working
    // debugPrint(mydata[2]["1"] + " is equal to " + mydata[1]["1"][k]);

    colortoshow = wrong;
    //
  }
  setState(() {
    // applying the changed color to the particular button that was selected
    btncolor[k] = colortoshow;

      canceltimer = true;
      disableAnswer = true;

  });
  // nextquestion();
  // changed timer duration to 1 second
  if(btncolor[k]==wrong){
    //Asigna el tema de la respuesta incorrecta al array tema
    tema.insert(n, mydata[3][i.toString()]);
    print(tema);
    n+=1;
  }
  Timer(Duration(seconds: 2), nextquestion);
}
```

Notas:

Para hacer funcionar el backend vas a tener que dar de alta en tu cuenta de firebase la aplicación y descargar los archivos json que te da y agregarlos a la app y al backend. El de firebase va en el cliente y el de FCM va en el backend.

También vas a tener que desplegar en heroku y hacer tu base de datos en atlas para correrlo desde tu pc o descargar mongodb en tu computadora y hacer la conexión local.

conexion

https://www.youtube.com/watch?v=w7OVmgncYPo

json FCM

https://www.youtube.com/watch?v=bbOmazof6KI

También vas a tener que crear tu colección para guardar los documentos que contengan la información que se va a enviar en forma de notificación.

Yo los guardé con este formato:


```
{
   "_id": {
      "$oid": "6128f9802318697d8310ca20"
   },
   "nombre": "Tema1",
   "titulo": "Tema 1",
   "mensaje": "Fórmula del desplazamiento",
   "url": "http://4.bp.blogspot.com/-AZix4BIyPyc/Vgn9khzIPmI/AAAAAAAAAb-c/pptJp7IaKSc/s1600/formula%2Bdel%2Bdesplazamiento.png"
}
```