Seguimiento Covid-19

APLICACIÓN HIBRIDA

Universidad Politécnica de Pachuca

Tabla de contenido

Historial de versiones

Fecha	Descripción	Autor	Comentarios
5/12/2020	Versión 1.0	Mario Sevilla	Documentación del proyecto Seguimiento Covid-19. Versión estable.

Introducción

Seguimiento Covid-19 tiene como objetivo ser una herramienta que facilite el registro y seguimiento de los síntomas que presenten los alumnos con Covid-19, así como el registro de los síntomas que presenten sus parientes cercanos y la vez sirva como registro para la toma de temperatura antes de permitir el ingreso a la universidad, esta información servirá para

Propósito

Proporcionar una herramienta que facilite el registro y seguimiento de síntomas tanto a la persona portadora de Covid-19 como al doctor encargado de atenderlo.

Alcance

Objetivo General

Desarrollar un sistema multiplataforma que sirva para llevar a cabo el seguimiento de alumnos y sus familiares han presentado síntomas de Covid-19. Enfocándose específicamente al desarrollo de la aplicación móvil para Smartphones con Android OS e iOS.

Requisitos Funcionales

Identificación del requerimiento:	RF01
Nombre del Requerimiento:	Autentificación de Usuario.
Características:	Los usuarios deberán identificarse para acceder a cualquier parte del sistema.
Descripción del requerimiento:	El sistema podrá ser consultado por cualquier usuario dependiendo del módulo en el cual se encuentre y su nivel de accesibilidad.
Requerimiento NO funcional:	•
Prioridad del requerimiento: Alta	

Identificación del requerimiento:	RF02
Nombre del Requerimiento:	Registro de síntomas del Usuario.
Características:	Los usuarios deberán poder registrar o listar sus síntomas, así mismo deberá incluir un apartado para la toma de su temperatura.
Descripción del requerimiento:	El sistema podrá ser capaz de registrar/listar los síntomas de los alumnos.
Requerimiento NO funcional:	•
Prioridad del requerimiento: Alta	

Identificación del requerimiento:	RF03
Nombre del Requerimiento:	Registro de síntomas de familiares del Usuario.
Características:	Los usuarios deberán poder registrar los síntomas que tienen sus familiares.
Descripción del requerimiento:	El sistema permitirá al usuario alumno poder registrar a sus familiares que tengan algún síntoma de Covid-19.
Requerimiento NO funcional:	•
Prioridad del requerimiento: Alta	

Identificación del requerimiento:	RF04
Nombre del Requerimiento:	El sistema debe estar disponible en Android y iOS.
Características:	El sistema tendrá la característica de ser soportado en los sistemas operativos Android y iOS.
Descripción del requerimiento:	El sistema podrá ser utilizado desde los sistemas operativos móviles Android y iOS mediante una aplicación de tipo hibrida.
Requerimiento NO funcional:	•
Prioridad del requerimiento: Alta	

Identificación del requerimiento:	RF05
Nombre del Requerimiento:	El sistema debe ser consumido desde una Api Rest.
Características:	El sistema deberá ser consumido mediante una API REST.
Descripción del requerimiento:	Las funcionalidades del sistema deberán ser consumidas mediante una API REST.
Requerimiento NO funcional:	•
Prioridad del requerimiento: Alta	

Requisitos no funcionales

Identificación del requerimiento:	RNF01
Nombre del Requerimiento:	Interfaz del sistema.
Características:	El sistema presentara una interfaz de usuario sencilla para que sea de fácil manejo a los usuarios del sistema.
Descripción del requerimiento:	El sistema debe tener una interfaz de uso intuitiva y sencilla.
Prioridad del requerimiento: Alta	

Requisitos de la interfaz externa

Interfaces de usuario

La interfaz con el usuario consistirá en un conjunto de ventanas con botones, listas y campos de textos. Ésta deberá ser construida específicamente para el sistema propuesto y, será visualizada desde el Smartphone.

Hardware Interfaces

Será necesario disponer de un Smartphone en perfecto estado con las siguientes características:

- Android 5.0 o superior.
- iOS 9.0 o superior.
- 150 Mb de espacio disponible para la app.

Software Interfaces

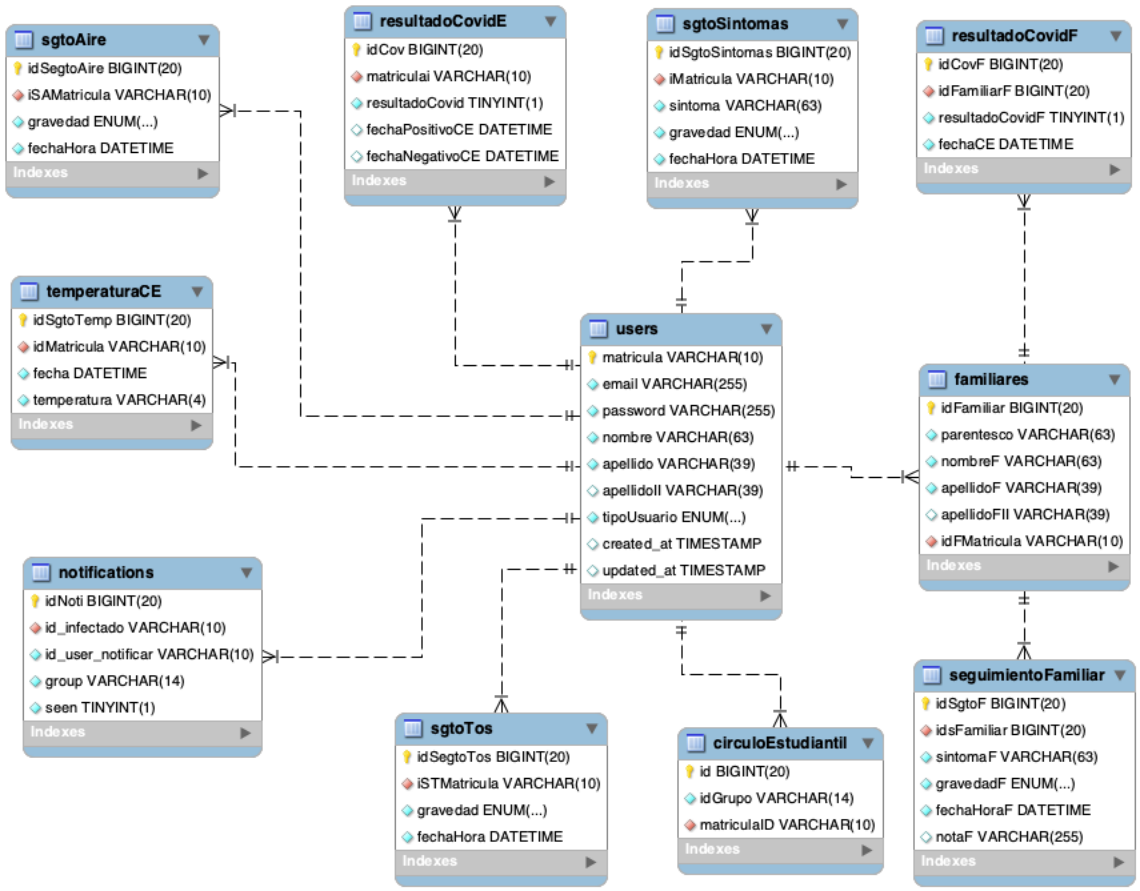
- Sistema Operativo: Android o iOS.

Historias de usuario

ID. De la historia de usuario	Rol	Característica/Funcionalidad	Razón/Resultado	Criterio de aceptación.
ID. 01	Como usuario	Quiero poder registrar/listar mis síntomas	Para llevar un control de mi progreso con la enfermedad.	
ID. 02	Como usuario	Quiero poder registrar mi temperatura	Para llevar un control de ella	

ID. 03	Como Usuario	Quiero tener disponible una versión para mi dispositivo móvil.	Para poder acceder desde mi teléfono de forma fácil.	
ID. 04	Como usuario.	Quiero que se notifique a mis compañeros de grupo si llego a dar positivo en la prueba de Covid-19.	Para que ellos tengan conocimiento y tomen sus debidas precauciones.	
ID. 05	Como Usuario.	Quiero tener la posibilidad de editar mis datos y síntomas.	Para poder realizar una corrección en caso de equivocarme al registrar un síntoma.	

Modelado de la base de datos



Diccionario de la base de datos

Nombre de la tabla: users
Descripción: Esta tabla servirá para almacenar los datos del alumno
Campos clave: matricula es llave primaria

Campo	Tamaño	Tipo de dato	Descripción
matricula	10	VARCHAR	Identificador único del alumno.
email	255	VARCHAR	Correo del usuario
password	15	VARCHAR	Contraseña de acceso del usuario.
nombre	63	VARCHAR	Nombre del usuario.
apellido	39	VARCHAR	Apellido del usuario.

apellidoII	39	VARCHAR	Apellido 2 del usuario en caso de tenerlo.
tipoUsuario	'alumno','maestro','administrativo'	ENUM	Tipo de usuario al que pertenece.
created_at		TIMESTAMP	
updated_at		TIMESTAMP	

Nombre de la tabla: familiares
Descripción: La tabla tiene como propósito registrar los familiares que tengan los alumnos
Campos clave: idFamiliar

Campo	Tamaño	Tipo de dato	Descripción
idFamiliar	10	INT	Identificador del familiar.
parentesco	63	VARCHAR	Identifica el tipo de parentesco que tiene con usuario, no es enum debido al gran numero de parentesco o relación sin parentesco con el usuario, por ejemplo un vecino.
nombreF	63	VARCHAR	Nombre del usuario.
apellidoF	39	VARCHAR	Apellido del usuario.
apellidoFII	39	VARCHAR	Apellido 2 del usuario en caso de tenerlo.
idMatricula	10	VARCHAR	Identificador del alumno para crear la relación.

Nombre de la tabla: temperaturaCE
Descripción:
Campos clave:

Campo	Tamaño	Tipo de dato	Descripción
idSgtoTemp	10	INT	Identificador del seguimiento.
idMatricula	10	VARCHAR	Identificador del usuario al que pertenecen los datos
fecha		DATETIME	Fecha del registro de los datos.
temperatura	2	VARCHAR	Temperatura tomada del usuario.

Nombre de la tabla: sgtoSintomas
Descripción:
Campos clave:

Campo	Tamaño	Tipo de dato	Descripción
idSgtoSintomas	10	INT	Identificador del seguimiento.
iMatricula	10	VARCHAR	Identificador del usuario al que pertenecen los datos
sintoma	63	VARCHAR	
gravedad	'ninguno','leve','moderada','grave'	ENUM	
fechaHora		DATE	Fecha del registro de los datos.

Los síntomas más habituales del Covid-19 son los siguientes:

- Fiebre
- Tos seca
- Cansancio

Otros síntomas menos comunes son los siguientes:

- Molestias y dolores
- Dolor de garganta
- Diarrea
- Conjuntivitis
- Dolor de cabeza
- Pérdida del sentido del olfato o del gusto
- Erupciones cutáneas o pérdida del color en los dedos de las manos o de los pies

Los síntomas graves son los siguientes:

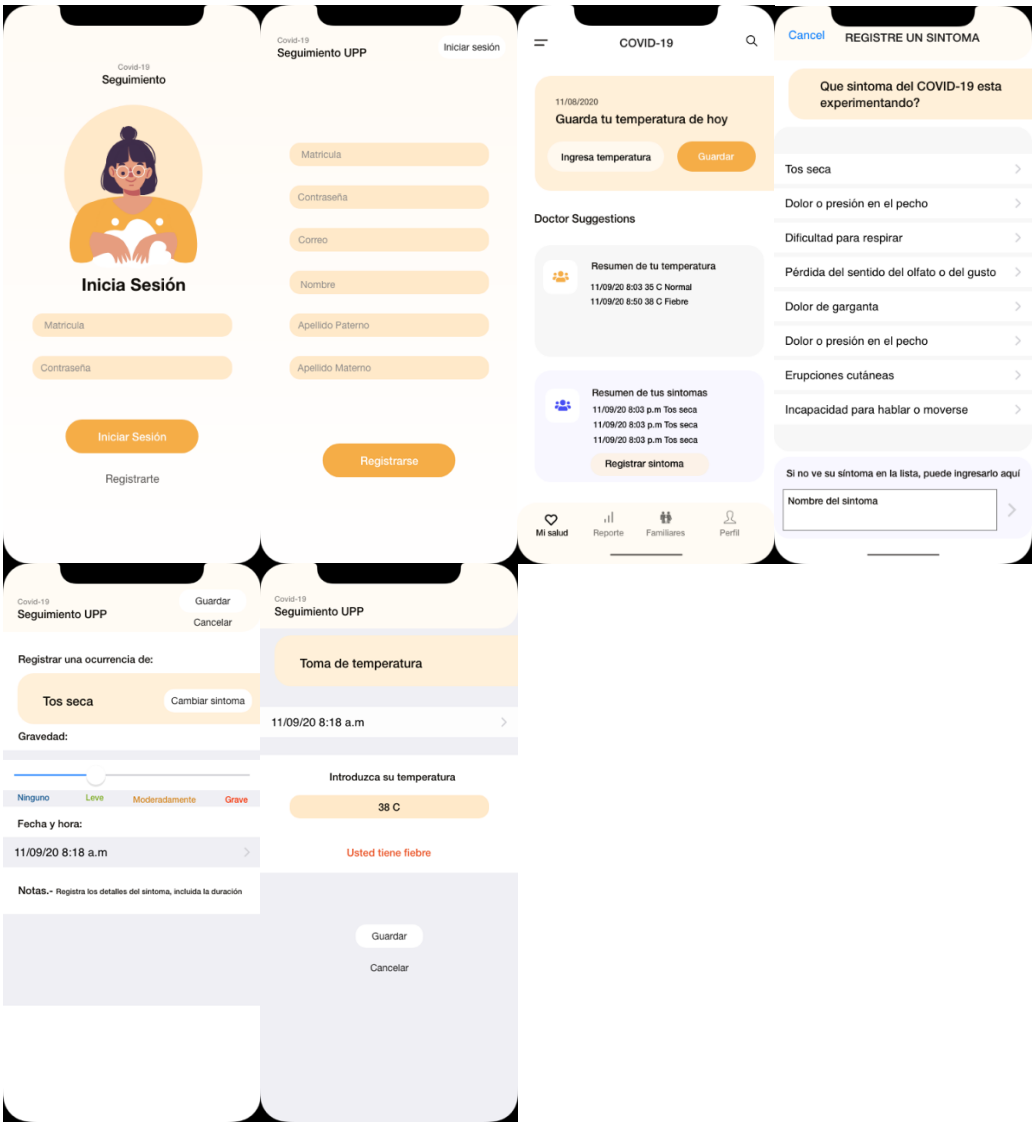
- Dificultad para respirar o sensación de falta de aire
- Dolor o presión en el pecho
- Incapacidad para hablar o moverse

Fuente: Organización Mundial de la salud.
https://www.who.int/es/emergencies/diseases/novel-coronavirus-2019/advice-for-public/q-a-coronaviruses#:~:text=sintomas

Diseño de interfaces

Wireframe

El wireframe se diseño inicialmente de un color claro, sin embargo el color fue cambiado a morado.



Interfaz de usuario final

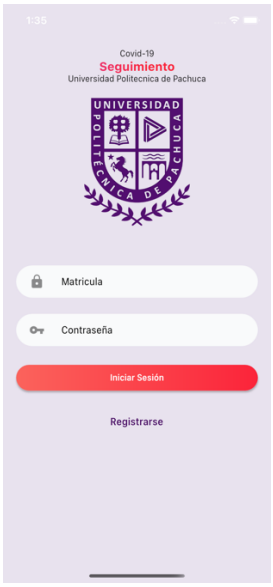


Ilustración 1 inicio de sesión

Ilustración 2 pantalla principal

Ilustración 3 Opción síntoma

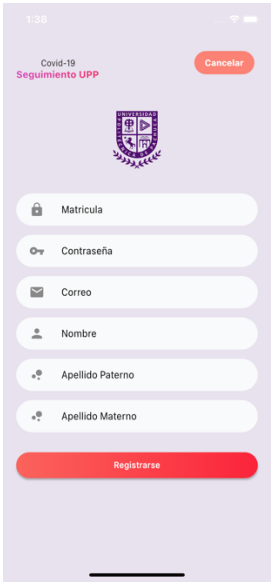


Ilustración 4 Registro de usuario

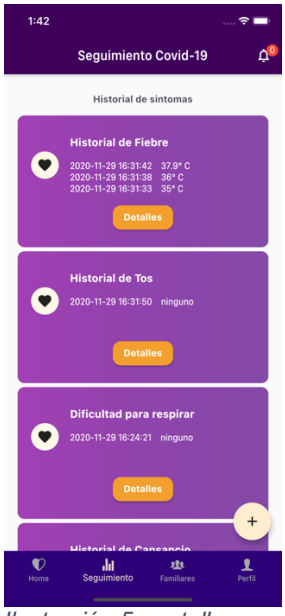


Ilustración 5 pantalla seguimiento

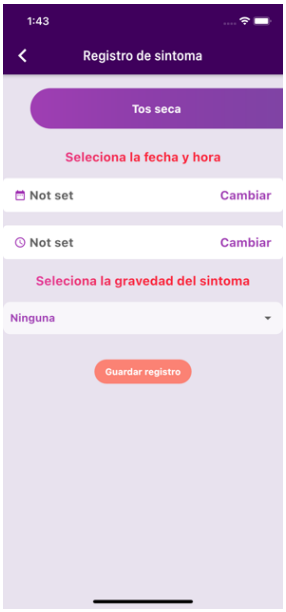


Ilustración 6 Registro síntoma

Ilustración 7 Detalles síntoma

Ilustración 8 Detalles lista

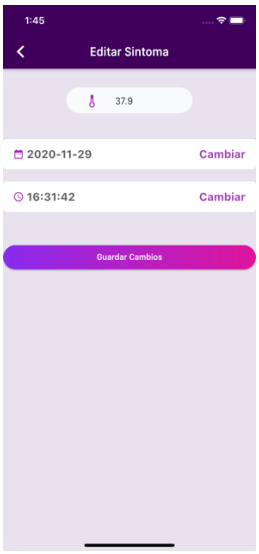


Ilustración 9 Editar síntoma

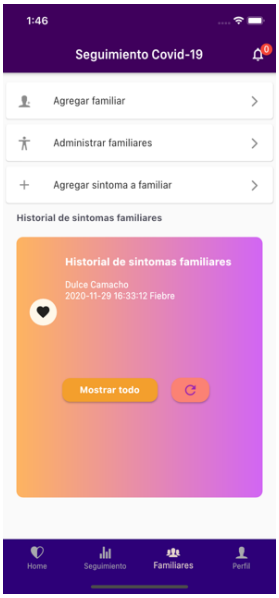


Ilustración 10 Seguimiento familiar

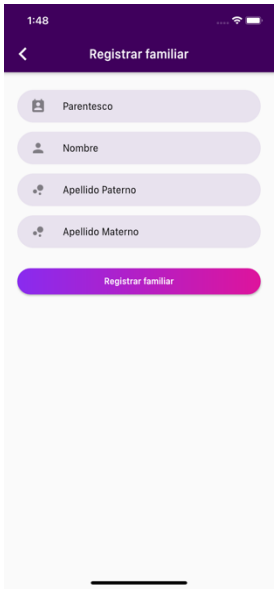


Ilustración 11 Registrar familiar

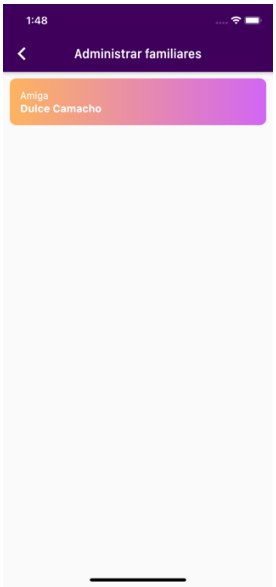


Ilustración 12 Admin familiares

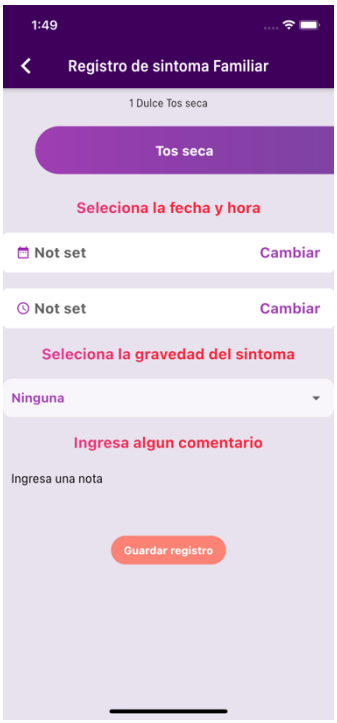


Ilustración 13 Registro síntoma familiares

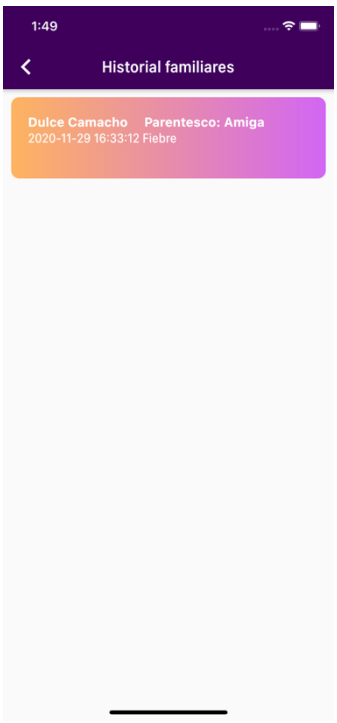


Ilustración 14 Historial de síntomas familiares

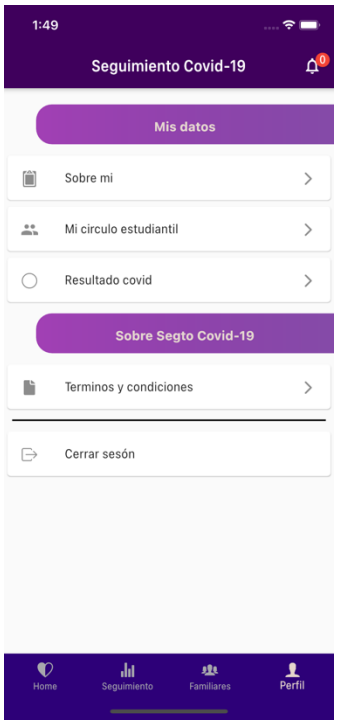


Ilustración 15 Perfil

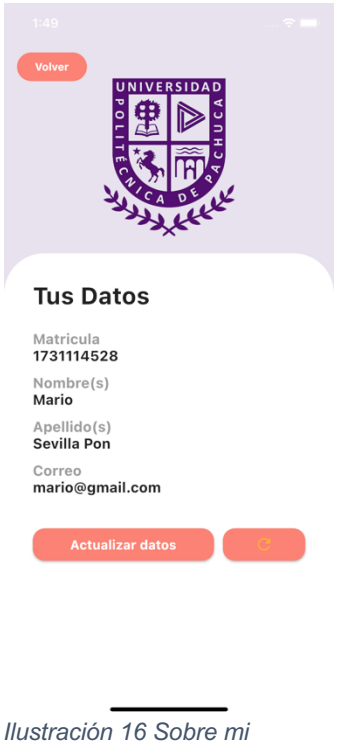


Ilustración 16 Sobre mi

Ilustración 17 Editar datos usuario

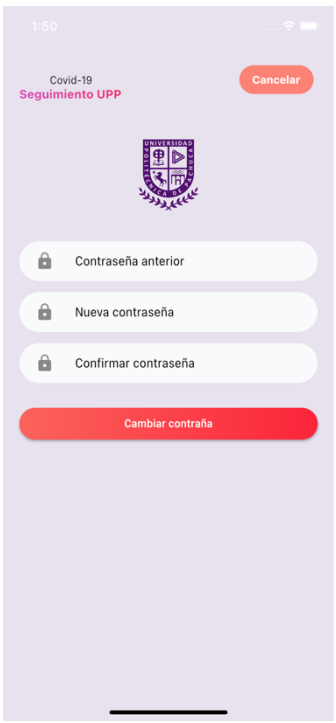


Ilustración 18 Cambiar contraseña

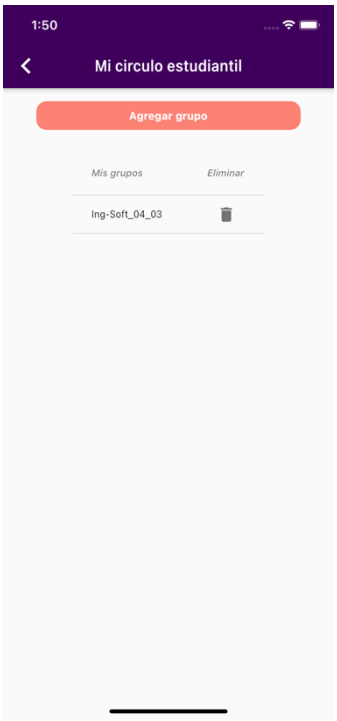


Ilustración 19 Mi circulo estudiantil

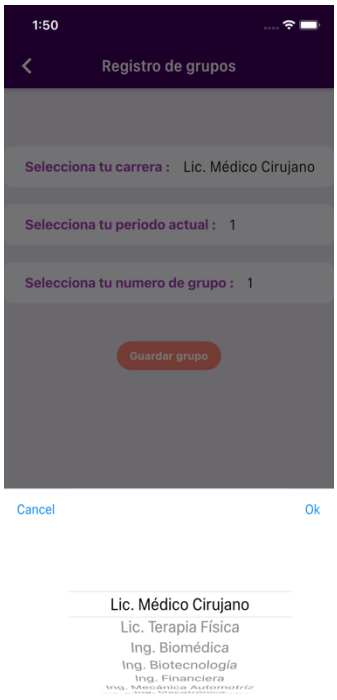


Ilustración 20 Agregar grupo

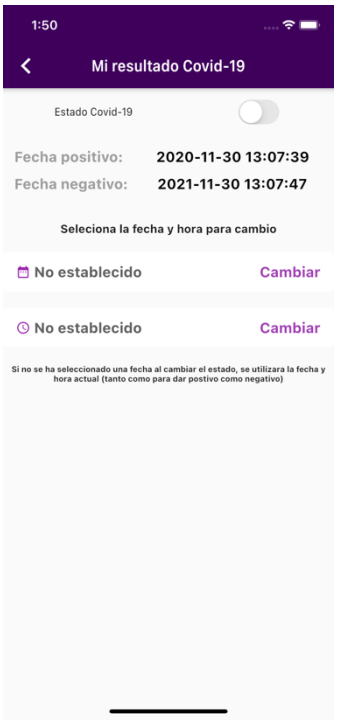


Ilustración 21 Resultado Covid-19

Api Rest en Laravel

Laravel es un framework de código abierto para desarrollar aplicaciones y servicios web con PHP 5 y PHP 7. Su filosofía es desarrollar código PHP de forma elegante y simple, evitando el "código espagueti". Fue creado en 2011 y tiene una gran influencia de frameworks como Ruby on Rails, Sinatra y ASP.NET MVC.

Proyecto Github

https://github.com/MaarioSevilla/apicloud/

Como correr el proyecto en Laravel

Conexión con la base de datos

Para realizar la conexión con la base de datos es necesario realizar los cambios en el archivo **.env** que se encuentra en la raíz de proyecto Laravel/.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=8889
DB_DATABASE=SeguimientoEstudiantil
DB_USERNAME=root
DB_PASSWORD=root
```

Nota: Todos los comandos deben ejecutarse dentro de la carpeta del proyecto.

Cargar las tablas en caso de no existir

php artisan migrate

Base de datos Nota

Las tablas fueron cargadas desde el la terminal con Laravel utilizando el comando antes mencionado, aunque no debería existir problema si se cargan desde el script sql.

Levantar el servidor

php artisan serv --host 192.168.10.9

Obviamente 192.168.10.9 hace referencia al servidor donde se aloja el proyecto.

Limpiar Caches

Si en un futuro se realiza algún mantenimiento y el proyecto empieza a dar problemas se recomienda ejecutar los siguiente comandos:

```
composer dump-autoload
php artisan cache:clear
php artisan optimize:clear
php artisan config:cache
```

Publicar el proyecto

Para publicar/preparar el proyecto para subirse a la red consulte la documentación oficial de laravel en la siguiente liga:

JWT AUTH

Json Web Token (JWT) es un estándar abierto (RFC 7519) y nos permite crear un método de autenticación en servicios API, para la conexión entre el cliente y el backend sea segura. Funciona de una forma sencilla, el cliente envía su usuario y contraseña, la API le retorna un token que enviará en todas las peticiones, para que ésta compruebe que tiene acceso a las acciones que se quieran realizar. JWT construidos sobre el algoritmo secreto HMAC o un par de claves pública / privada usando RSA o ECDSA

Controllers

Nota: Ruta donde se encuentran los controladores APP\Http\Controllers

Eloquent

El ORM Eloquent incluido con Laravel proporciona una implementación de ActiveRecord para trabajar con la base de datos. Cada tabla de la base de datos tiene un "Modelo" correspondiente que se utiliza para interactuar con esa tabla. Los modelos le permiten consultar datos en sus tablas, así como insertar nuevos registros en la tabla.

AuthController

auth:api

El auth:api middleware se usa dentro del constructor; No se puede acceder a las funciones dentro del controlador de autenticación sin tener el token válido.

```php
public function __construct() {
    $this->middleware('auth:api', ['except' => ['login', 'register']]);
}
```

Login

El método de inicio de sesión se utiliza para proporcionar acceso al usuario y se activa cuando /api/auth/login llama a la API. Autentica la matricula y la contraseña ingresados por el usuario. En respuesta, genera un token de autorización si encuentra un usuario dentro

de la base de datos. Viceversa, muestra un error si el usuario no se encuentra en la base de datos.

```
public function login(Request $request){

   $validator = Validator::make($request->all(), [
        'matricula' => 'required',
        'password' => 'required|string',
    ]);

    if ($validator->fails()) {
        return response()->json($validator->errors(), 422);
    }

    if (! $token = auth()->attempt($validator->validated())) {
        return response()->json(['error' => 'Unauthorized'], 401);
    }

    return $this->createNewToken($token);
}
```

Register

El método de registro se utiliza para crear un usuario cuando /api/auth/register sea llamado en la ruta. Primero, los valores de usuario como la matricula, el correo electrónico, la contraseña, nombre, apellidos y tipo de usuario se validan mediante el proceso de validación, y luego se registra al usuario si las credenciales de usuario son válidas. Luego, genera el JSON Web Token para proporcionar acceso válido al usuario.

```
public function register(Request $request) {
    $validator = Validator::make($request->all(), [
        'matricula' => 'required|max:10',
        'email' => 'required|max:255',
        'password' => 'required|max:255',
        'nombre' => 'required|max:63',
        'apellido' => 'required|max:39',
        'apellidoII' => 'max:39',
        'tipoUsuario' => 'required',
    ]);

    if($validator->fails()){
        return response()->json($validator->errors()->toJson(), 400);
    }

    $user = User::create(array_merge(
                $validator->validated(),
                ['password' => bcrypt($request->password)]
            ));

    return response()->json([
        'message' => 'User successfully registered',
        'user' => $user
    ], 201);
}
```

Logout

El método de cierre de sesión se llama cuando /api/auth/logout solicita la API y borra el token de acceso JWT pasado.

```
public function logout() {
    auth()->logout();

    return response()->json(['message' => 'User successfully signed
out']);
}
```

Refresh

El método de actualización crea un nuevo JSON Web Token en un período más corto, y se considera una mejor práctica generar un nuevo token para el sistema de autenticación de usuario. Invalida al usuario actualmente conectado si el token JWT no es nuevo.

```
public function refresh() {
    return $this->createNewToken(auth()->refresh());
}
```

userProfile

El método userProfile muestra los datos del usuario que inició sesión. Funciona cuando colocamos el token de autenticación en los encabezados para autenticar la solicitud de autenticación realizada a través de la /api/auth/user-profileAPI.

```
public function userProfile() {
    return response()->json(auth()->user());
}
```

createNewToken

La función createNewToken crea el nuevo token de autenticación JWT después de un período de tiempo específico, hemos definido la caducidad del token y hemos registrado los datos del usuario en esta función.

```
protected function createNewToken($token){
    return response()->json([
        'status' => 'ok',
        'token' => $token,
        'token_type' => 'bearer',
        'expires_in' => auth()->factory()->getTTL() * 60,
        'user' => auth()->user()
    ]);
}
```

Métodos Crud Controllers

Validaciones

Las validaciones tienen como función evitar hacer peticiones a la base de datos si no se cumple con la validación retornando cual de los valores no cumple dicha condición especificada. Las validaciones las encontraras en las funciones **Store**(que hace de create) y la función **update**.

```
    $validator = Validator::make($input, [
        'parentesco' => 'required|max:63',
        'nombreF' => 'required|max:63',
        'apellidoF' => 'required|max:39',
        'apellidoFII' => 'max:39',
        'idFMatricula' => 'required|max:10',
    ]);
```

Index

El método index que se encarga de realizar una consulta de todos los datos que contiene la tabla.

```
public function index()
{
    $familiares = Familiares::select("familiares.*")->get()->toArray();
    return response()->json($familiares);
}
```

Store (create)

El método store en realidad es el encargado de la creación de un nuevo campo en la tabla, **$request->all();** hace referencia al modelo, especificando que se requieren todos los campos, pasando por una validación, en caso pasar se intentara registrar los nuevos datos.

```
public function store(Request $request)
{
    $input = $request->all();
    $validator = Validator::make($input, [
        'parentesco' => 'required|max:63',
        'nombreF' => 'required|max:63',
        'apellidoF' => 'required|max:39',
        'apellidoFII' => 'max:39',
        'idFMatricula' => 'required|max:10',
    ]);
    if ($validator->fails()) {
        return response()->json([
            'ok' => false,
            'error' => $validator->messages(),
        ]);
```

```
    }
    try {
        Familiares::create($input);
        return response()->json([
            "ok" => true,
            "mensaje" => "Se registro con exito",
        ]);
    } catch (\Exception $ex) {
        return response()->json([
            "ok" => false,
            "error" => $ex->getMessage(),
        ]);
    }
}
```

Show

El método show o su equivalente son los métodos encargados de realizar los select a las tablas con las debidas restricciones es decir los datos que deseamos obtener de la consulta.

```
public function myfamily($idFMatricula)
{
    $familiares = Familiares::select("familiares.*")
        ->where("familiares.idFMatricula", $idFMatricula)
        ->orderBy('nombreF', 'ASC')
        ->get()->toArray();
    return response()->json([
        "ok" => true,
        "data" => $familiares,
    ]);
}
```

Update

El método update se encarga de actualizar los datos de un registro, como parámetros de entrada se piden todos los especificados en el modelo y el id, después de pasar por la validación, se pasará a buscar si realmente el id existe, en ese caso se le dirá que actualice a los nuevos datos introducidos.

```
public function update(Request $request, $idFamiliar)
{
    $input = $request->all();
    $validator = Validator::make($input, [
        'parentesco' => 'required|max:63',
        'nombreF' => 'required|max:63',
        'apellidoF' => 'required|max:39',
        'apellidoFII' => 'max:39',
        'idFMatricula' => 'required|max:10',
    ]);
    if ($validator->fails()) {
        return response()->json([
            'ok' => false,
            'error' => $validator->messages(),
        ]);
```

```
    }
    try {
        $familiares = Familiares::find($idFamiliar);
        if ($familiares == false) {
            return response()->json([
                "ok" => false,
                "error" => "No se encontro",
            ]);
        }
        $familiares->update($input);
        return response()->json([
            "ok" => true,
            "mensaje" => "Se modifico con exito",
        ]);
    } catch (\Exception $ex) {
        return response()->json([
            "ok" => false,
            "error" => $ex->getMessage(),
        ]);
    }
}
```

Destroy

El método destroy es el encargado de eliminar el registro especificado, este no es un delete lógico.

```
public function destroy($idFamiliar)
{
    try {
        $familiares = Familiares::find($idFamiliar);
        if ($familiares == false) {
            return response()->json([
                "ok" => false,
                "error" => "No se encontro",
            ]);
        }
        $familiares->delete([
        ]);
        return response()->json([
            "ok" => true,
            "mensaje" => "Se elimino con exito",
        ]);
    } catch (\Exception $ex) {
        return response()->json([
            "ok" => false,
            "error" => $ex->getMessage(),
        ]);
    }
}
```

Modelos

Dentro de los modelos especificamos las características de la tabla.

.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class TemperaturaCE extends Model
{
    public $table = "temperaturaCE";

    protected $primaryKey = 'idSgtoTemp';

    protected $fillable = [
        'idMatricula',
        'fecha',
        'temperatura'
    ];

    public $timestamps = false;

}
```

Rutas Api

Directorio: Routes\Api
Dentro de este directorio se especifica el método que se realizara (post, get, put, delete, update), el controlador al que hacen referencia y la ruta en la que se podrá consultar.

```php
Route::group([
    'middleware' => 'api',

], function ($router) {
    Route::post('/login', [AuthController::class, 'login']);
    Route::post('/register', [AuthController::class, 'register']);
    Route::post('/logout', [AuthController::class, 'logout']);
    Route::post('/refresh', [AuthController::class, 'refresh']);
    Route::get('/user-profile', [AuthController::class, 'userProfile']);
});

//00 user queries
Route::get('/myuser/{matriculaID}',
'App\Http\Controllers\UsController@myUser');
Route::resource('/user', 'App\Http\Controllers\UsController')->except([
    'create', 'edit'
]);
```

Tipo de respuesta Json

En la mayoría de los casos a excepción de dos (el conteo de notificaciones y la obtención de las mismas) el formato de respuesta será parecido al siguiente:

```
{
    "ok": true,
    "data": [
        {
            "idFamiliar": 1,
            "parentesco": "Amiga",
            "nombreF": "Dulce",
            "apellidoF": "Camacho",
            "apellidoFII": "Camacho",
            "idFMatricula": "17311"
        }
    ]
}
```


Es importante tener en cuenta al momento de la decodificación de la respuesta.

Aplicación Hibrida

Proyecto Github

https://github.com/MaarioSevilla/segtoCovid19

Flutter

Flutter es el kit de herramientas de UI de Google para realizar hermosas aplicaciones, compiladas nativamente, para móvil, web y escritorio desde una única base de código.

Dirección del servidor

Para modificar la dirección del servidor en la aplicación, se tiene que modificar la constante que se encuentra en Providers/globals.dart

```
const urlServer = "http://172.20.10.8:8000";
```

Controllers

La función Singin recibe como parámetros la matricula y contraseña, se crea un mapa con estos datos, hacemos la petición al servidor, con un máximo de intentos de 8 con una duración de 30 segundos cada uno, esto solo pasara si la excepción es por socket o time excepcion es decir el servidor no contesto o se tardo demasiado en procesar la petición, si las credenciales son correctas se almacenara el token y la matricula.

```
//funcion singin
signIn(String matricula, String pass) async {
  SharedPreferences sharedPreferences = await
SharedPreferences.getInstance();
  Map data = {
    'matricula': matricula,
    'password': pass
  };
  try{
    var jsonResponse = null;
    String myUrl = "$serverUrl/api/login";
    final r = RetryOptions(maxAttempts: 8);
    var response = await r.retry(
          () => http.post(myUrl,
          body: data).timeout(Duration(seconds: 30)),
      retryIf: (e) => e is SocketException || e is TimeoutException,
    );
    if (response.statusCode == 200) {
      jsonResponse = json.decode(response.body);
      print('Response status: ${response.statusCode}');
      print('Response body: ${response.body}');
      if (jsonResponse != null) {
        setState(() {
          _isLoading = false;
```

```
        });
        sharedPreferences.setString("token", jsonResponse['token']);
        sharedPreferences.setString("matricula", matricula);
        globals.matricula=matricula;
        Navigator.of(context).pushAndRemoveUntil(
            MaterialPageRoute(builder: (BuildContext context) => Menu()),
(
            Route<dynamic> route) => false);
      }
    }
    else {
      setState(() {
        _isLoading = false;
      });
      print(response.body);
      _toast.toastmsg("Credenciales incorrectas");
    }
  }on SocketException {
    print('No se pudo establecer conexion con el servidor 😑');
    _toast.toastmsg("No se pudo establecer conexión con el servidor");
  } on HttpException {
    print("Couldn't find the post 😱");
  } on FormatException {
    print("Bad response format 👎");
  }
}
```

AddData

Value almacenara el token obtenido al inicio de sesión e ira acompañado en la mayoría de las peticiones que realicemos.

```
void addDataTos(String _gravedad, String _fechaHora) async {
  final prefs = await SharedPreferences.getInstance();
  final key = 'token';
  final value = prefs.get(key ) ?? 0;
  String _matricula = await _statusProvider.checkLoginStatus();
  print('prueba resultado $_matricula');
  if(_matricula!=null){
    try{
      String myUrl = "$serverUrl/api/sgtotos";
      final response = await  http.post(myUrl,
          headers: {
            'Accept':'application/json'
          },
          body: {
            "iSTMatricula":        "$_matricula",
            "gravedad":      "$_gravedad",
            "fechaHora":       "$_fechaHora"
          } ) ;
      status = response.body.contains('error');
      var data = json.decode(response.body);
      if(status){
        print('data : ${data["error"]}');
```

```
        _toast.toastmsg("Algo salio mal");
      }else{
        print('data : ${data["token"]}');
        _save(data["token"]);
        _toast.toastmsg("Se ha registrado con exito");
      }
    }on SocketException {
      print('No se pudo establecer conexion con el servidor 😑');
      _toast.toastmsg("No se pudo establecer conexión con el servidor");
    } on HttpException {
      print("Couldn't find the post 😱");
    } on FormatException {
      print("Bad response format 👎");
    }
  }
}
```

EditarData

```
//function for update or put
void editarData(String idSgtoTemp, String _idMatricula , String _fecha,
String _temperaturaController) async {
  final prefs = await SharedPreferences.getInstance();
  final key = 'token';
  final value = prefs.get(key) ?? 0;

  String myUrl = "$serverUrl/api/temperatura/$idSgtoTemp";
  http.put(myUrl,
      headers: {
        'Accept':'application/json',
        'Authorization' : 'Bearer $value'
      },
      body: {
        "idMatricula": "$_idMatricula",
        "fecha": "$_fecha",
        "temperatura": "$_temperaturaController"
      }).then((response){
    print('Response status : ${response.statusCode}');
    print('Response body : ${response.body}');
    if (response.statusCode == 200) {
      _toast.toastmsg("Se ha actualizado con exito");
    }
  });
}
```

RemoveRegister

```
//function for delete
void removeRegister(String idSgtoTemp) async {
  final prefs = await SharedPreferences.getInstance();
  final key = 'token';
  final value = prefs.get(key ) ?? 0;
```

```
  String myUrl = "$serverUrl/api/temperatura/$idSgtoTemp";
  http.delete(myUrl,
      headers: {
        'Accept':'application/json',
        'Authorization' : 'Bearer $value'
      } ).then((response){
    print('Response status : ${response.statusCode}');
    print('Response body : ${response.body}');
    if (response.statusCode == 200) {
      _toast.toastmsg("Se ha eliminado con exito");
    }
  });
}
```

GetData

```
getAllData(String _sintoma) async {
  String _matricula = await _statusProvider.checkLoginStatus();
  print('prueba resultado $_matricula');
  String myUrl;
  if(_sintoma=="fiebre"){
    myUrl = "$serverUrl/api/temperatura/showallbymat/$_matricula";
  }else if(_sintoma=="Tos seca"){
    myUrl = "$serverUrl/api/sgtotos/showallbymat/$_matricula";
  }else if(_sintoma=="Cansancio"){
    myUrl = "$serverUrl/api/sgtosymptoms/showallbymat/$_matricula/1";
  }else if(_sintoma=="Molestias y dolores"){
    myUrl = "$serverUrl/api/sgtosymptoms/showallbymat/$_matricula/2";
  }else if(_sintoma=="Dolor de garganta"){
    myUrl = "$serverUrl/api/sgtosymptoms/showallbymat/$_matricula/3";
  }else if(_sintoma=="Diarrea"){
    myUrl = "$serverUrl/api/sgtosymptoms/showallbymat/$_matricula/4";
  }else if(_sintoma=="Conjuntivitis"){
    myUrl = "$serverUrl/api/sgtosymptoms/showallbymat/$_matricula/5";
  }else if(_sintoma=="Dolor de cabeza"){
    myUrl = "$serverUrl/api/sgtosymptoms/showallbymat/$_matricula/6";
  }else if(_sintoma=="Pérdida del sentido del olfato"){
    myUrl = "$serverUrl/api/sgtosymptoms/showallbymat/$_matricula/7";
  }else if(_sintoma=="Erupciones cutáneas"){
    myUrl = "$serverUrl/api/sgtosymptoms/showallbymat/$_matricula/8";
  }else if(_sintoma=="Dificultad para respirar"){
    myUrl = "$serverUrl/api/sgtoaire/showallbymat/$_matricula";
  }else if(_sintoma=="Dolor o presión en el pecho"){
    myUrl = "$serverUrl/api/sgtosymptoms/showallbymat/$_matricula/9";
  }else if(_sintoma=="Incapacidad para hablar o moverse"){
    myUrl = "$serverUrl/api/sgtosymptoms/showallbymat/$_matricula/10";
  }
  try{
    final r = RetryOptions(maxAttempts: 4);
    final response = await r.retry(
      // Make a GET request
          () => http.get(myUrl).timeout(Duration(seconds: 15)),
      // Retry on SocketException or TimeoutException
      retryIf: (e) => e is SocketException || e is TimeoutException,
    );
    //guardado y actualizacion de mapas
```

```
    return json.decode(response.body);
  }on SocketException {
    print('No se pudo establecer conexion con el servidor 😑');
    _toast.toastmsg("No se pudo establecer conexión con el servidor");
    return '{ok: true, data: []}';
  } on HttpException {
    print("Couldn't find the post 😱");
    return '{ok: true, data: []}';
  } on FormatException {
    print("Bad response format 👎");
    return '{ok: true, data: []}';
  }
}
```

SharedPreferences status provider

La matricula es almacena para despues ser utilizada en las diferentes consultas que se realizan en el sistema.

```
checkLoginStatus()  async {
  String _matricula;
  sharedPreferences = await SharedPreferences.getInstance();
  if(sharedPreferences.getString("matricula") == null) {
    print('no lo guardo');
    return null;
  }else{
    print('si lo guarde');
    _matricula=sharedPreferences.getString("matricula");
    return _matricula;
  }
}
```

Uso de los helper Controllers

Para utilizer los controller se tiene que importar primero la dirección donde se encuentra el controlador que se desea usar y después instanciarlo para hacer uso de sus funciones.

```
DataBaseHelperFamily databaseHelper = new DataBaseHelperFamily();

databaseHelper.removeRegister(widget.list[widget.index]['idFamiliar'].toS
tring());
```

Maping Data

Para traer datos desde el servidor creamos un map (esto depende de la forma en la que viene el json) despues se crea una lista dinámica especificando que parte del mapa se almacenara ahí.

```
Future<List> getData() async {
  sharedPreferences = await SharedPreferences.getInstance();
  if(sharedPreferences.getString("matricula") == null) {
```

```
    print('no lo guardo');
  }else{
    _matricula=sharedPreferences.getString("matricula");
  }
  String myUrl = "$serverUrl/api/family/myfamily/$_matricula";
  final response = await http.get(myUrl);
  Map<String, dynamic> map = json.decode(response.body);
  List<dynamic> data = map["data"];
  return data;
}
```

InitState

El init state cargamos todos los metodos o funciones que necesitamos se carguen al iniciar alguna pantalla.

```
@override
void initState() {
  super.initState();
  this.getData();
}
```

setState

El setState es utilizado para que los cambios se vean reflejar cambios.

```
setState(() {
  getData();
});
```

Evitar abusar de la función setState, dado que esto puede traer problemas en la memoria del dispositivo

FutureBuilder

```
new FutureBuilder<List>(
  future: getData(),
  builder: (context, snapshot) {
    if (snapshot.hasError) print(snapshot.error);
    return snapshot.hasData
        ? new ItemList(
      list: snapshot.data,
    )
        : new Center(
            child: new CircularProgressIndicator(),
    );
  },
)
```

```
class ItemList extends StatelessWidget {
  final List list;
  ItemList({this.list});

  @override
  Widget build(BuildContext context) {
    return new ListView.builder(
      itemCount: list == null ? 0 : list.length,
      itemBuilder: (context, i) {
        return new Container(
          padding: const EdgeInsets.all(10.0),
          child: new GestureDetector(
            onTap: () => Navigator.of(context).push(
              new MaterialPageRoute(
                  builder: (BuildContext context) => new FamilyMember(
                    list: list,
                    index: i,
                  )),
            ),
            child: new Container(
              margin: EdgeInsets.symmetric(vertical: 0),
              width: double.infinity,
              //largo del cuadro contenedor
              height: 72,
              decoration: BoxDecoration(
                borderRadius: BorderRadius.circular(10),
              ),
              child: DecoratedBox(
                decoration: BoxDecoration(
                  borderRadius: BorderRadius.circular(10),
                  gradient: LinearGradient(
                    colors: [
                      Color(0xFFFF961F).withOpacity(0.7),
                      Color(0xFFC027F1).withOpacity(0.7),
                    ],
                  ),
                ),
                child: Padding(
                  padding: const EdgeInsets.all(0.0),
                  //row es de lado
                  child: Column(
                    children: <Widget>[
                      new ListTile(
                        title: new Text(
                          list[i]['parentesco'].toString(),
                          style: TextStyle(fontSize: 16.0, color:
Colors.white),
                        ),
                        subtitle: new Text(
                          list[i]['nombreF'].toString()+'
'+list[i]['apellidoF'].toString(),
                          style: TextStyle(fontWeight:
FontWeight.bold,fontSize: 16.0, color: Colors.white),
                        ),
                      ),
                    ],
```

```
                  ),
                ),
              ),
            ),
          ),
        );
      },
    );
  }
}
```

```
checkLoginStatus() async {
  sharedPreferences = await SharedPreferences.getInstance();
  if(sharedPreferences.getString("token") == null) {
    Navigator.of(context).pushAndRemoveUntil(MaterialPageRoute(builder:
(BuildContext context) => LoginScreen()), (Route<dynamic> route) =>
false);
  }else{
    globals.matricula=sharedPreferences.getString("matricula");
    Navigator.of(context).pushAndRemoveUntil(MaterialPageRoute(builder:
(BuildContext context) => Menu()), (Route<dynamic> route) => false);
  }
}
```