MANUAL TECNICO DE APP
ASILO ESPERANZA

Enma Sofía López Rojas	**LR230079**
Kevin Enrique Martínez Martínez	**MM230084**
José Gerardo Marroquín Vásquez	**MV230090**
José Ernesto Sorto González	**SG202883**
Roger Eduardo Vásquez Portillo	**VP223250**

LINK DE VIDEO https://youtu.be/Qo7tHX1NlEY

Contenido

INTTRODUCCION 1

DESARROLLO 1

HERRAMIENTAS BASICAS 1

CONFIGURACIONES PREVIAS 1

GRANTTPRO 1

ESTRUCTURA DEL PROYECTO 1

CARPETAS 1

ARCHIVO: APP.JS 3

SCREENS: LOGICA DE LAS VISTAS CREADAS 5

GESTION DE CITAS 5

DATABASE 9

INTTRODUCCION

Se ha desarrollado una App de Gestión de citas para el Asilo Nueva esperanza, con esta aplicación se llevará un registro tanto de pacientes como de doctores, adicional a estos registro tiene como eje fundamental gestionar adecuadamente el registro y estados de las citas realizadas para los diferentes pacientes.

Previo al desarrollo de la app se hizo un estudio de las necesidades a solventar con la aplicación de esta app, por lo que en este manual se basará en las configuraciones previas al inicio del desarrollo, así como la estructura actual del proyecto el cual de momento esta trabajando a un 50% de todas las funcionalidades que tendrá como proyecto terminado

DESARROLLO

HERRAMIENTAS BASICAS

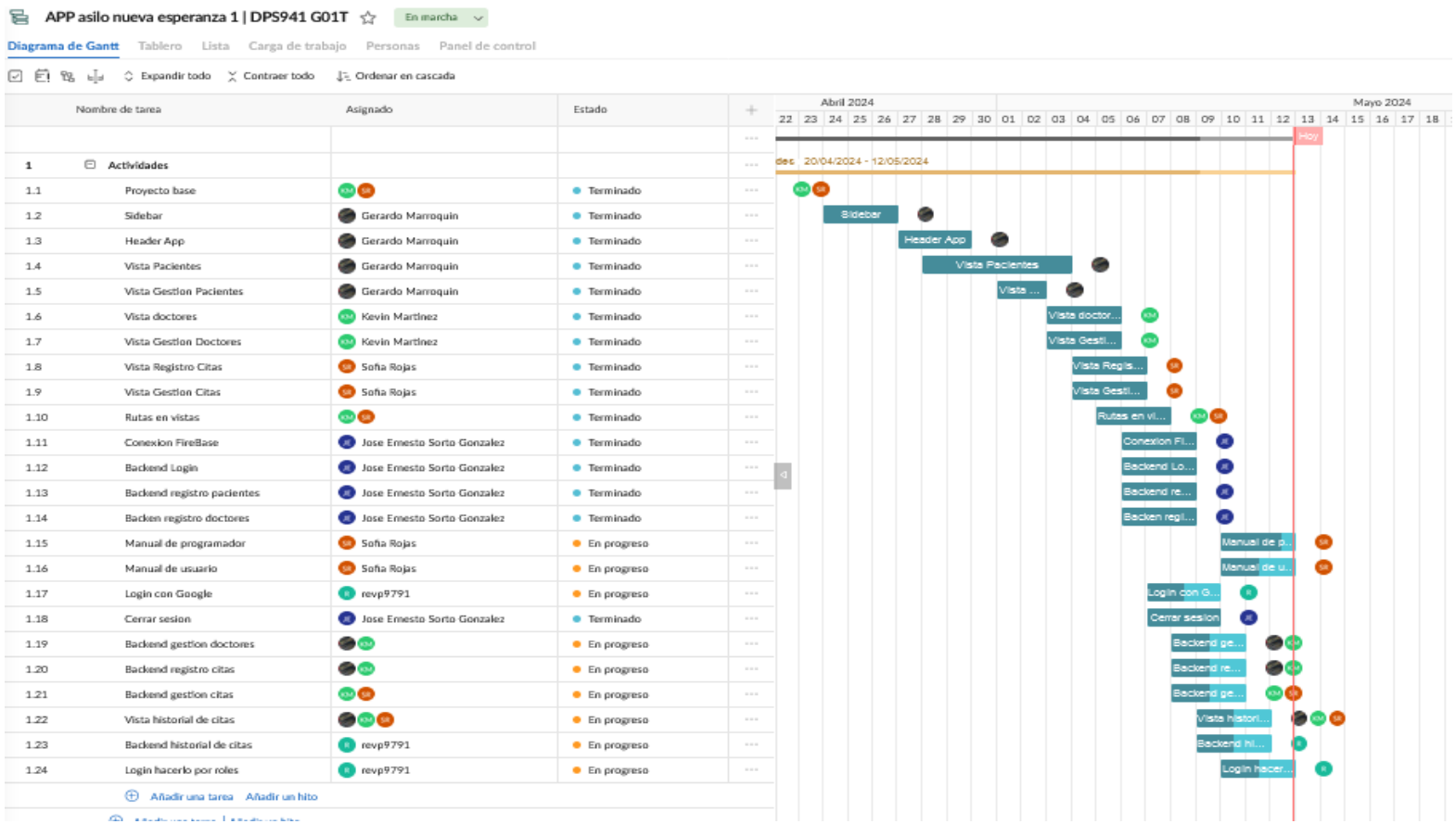
Para iniciar con el desarrollo de la app es necesario conocer las herramientas básicas para la realización del proyecto; por lo que detallaremos que herramientas son estas: como el lenguaje de programación se ha utilizado JavaScript y además se ha utilizado la biblioteca de React para brindar una interfaz más amigable con el usuario, y como IDE de desarrollo se ha utilizado Visual Studio Code versión 1.89.1 “x64”, una vez tengamos presente las herramientas básicas para la creación de una app procederemos a ir detallando algunas de las partes más importantes del proyecto desarrollado.

CONFIGURACIONES PREVIAS

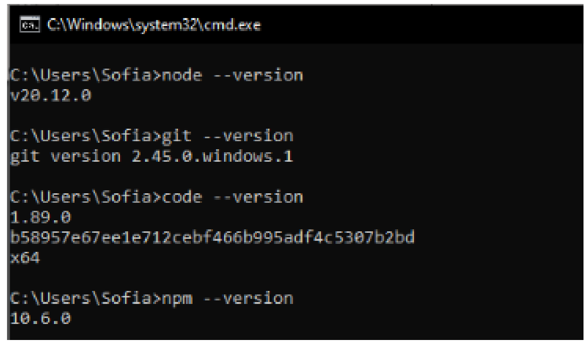
Una vez se tenga claro el IDE a utilizar y el lenguaje mediante el cual se desarrollará es necesario configurar el entorno de desarrollo, las vistas creadas inicialmente fueron creadas en expo go, posteriormente se han trasladado al IDE que se ha utilizado para unir las diferentes vistas, pero adicional a esto dentro del pc a utilizar y del IDE que se está utilizando se deben de hacer una serie configuraciones de herramientas para el desarrollo de la app.

GRANTTPRO

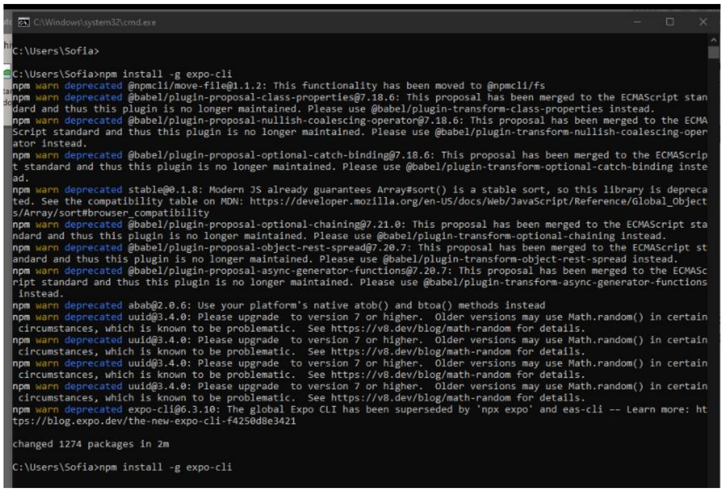
Para la realización de la app se organizó mediante un gestor de proyecto en el cual nos fuimos asignando las actividades a realizar como se muestra en la siguiente imagen (https://app.ganttpro.com/#/project/1715488817704/gantt)

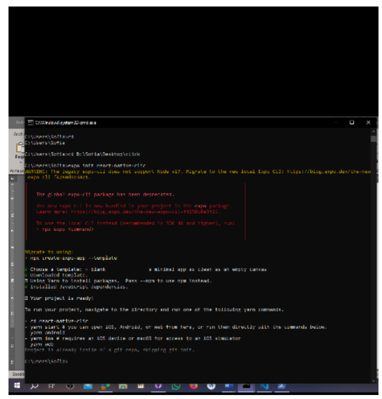


Entre las configuraciones están las siguientes Para iniciar con el desarrollo de una app utilizando JavaScript y React Native se debe de contar con un entorno preparado con las siguientes herramientas para el desarrollo



Adicional a estas herramientas también se puede hacer uso de expo-clic para ello colocamos en la terminal el comando **npm install -g expo-cli**



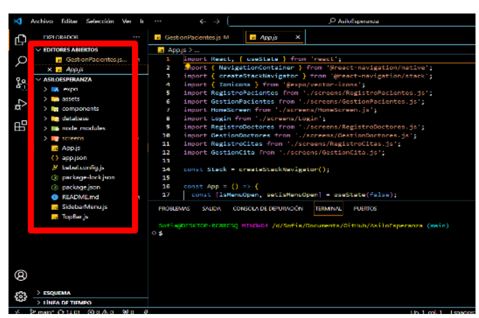


Una vez instalado estas herramientas procedemos a crea el proyecto y ver la estructura mediante la cual se ira trabajando.

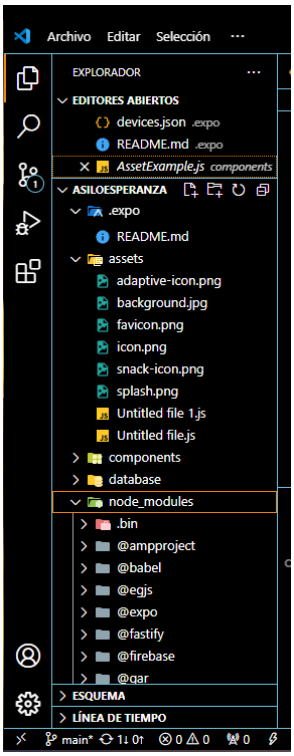
ESTRUCTURA DEL PROYECTO

CARPETAS

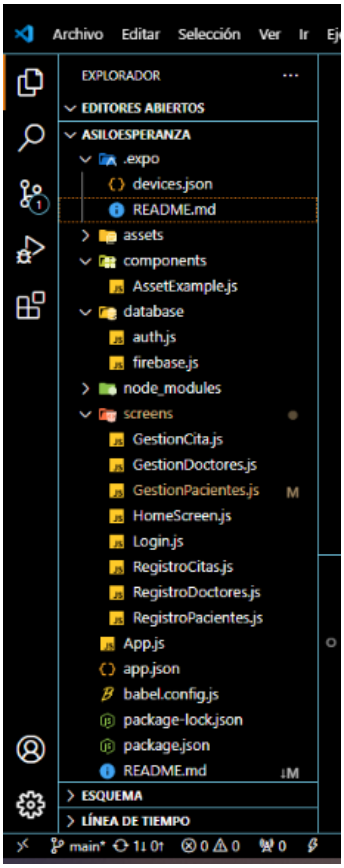
Una vez se hayan creado la carpeta del proyecto y exportemos las vistas creadas en expo.web la estructura del proyecto nos quedó de la siguiente forma:



En la creación del proyecto se generaron algunos archivos predeterminado para el funcionamiento de nuestra app; en este caso podemos mencionar algunos de estos archivo, los cuales son la carpeta .expo; esta carpeta servirá como almacenaje de información sobre los dispositivos emparejados con el proyecto, la carpeta assets; nos servirá para guardar los recursos multimedia utilizados en el proyecto, node_modules esta ultima no se genera al inicio del proyecto sino a medida que instalamos dependencias del proyecto por lo que la consideraremos como creación automática.



Las siguientes carpetas son parte ya del desarrollo realizado para la app del asilo tambien daremos un poco de detalle antes de entrar a explicar que contiene los archivos dentro de estas carpetas

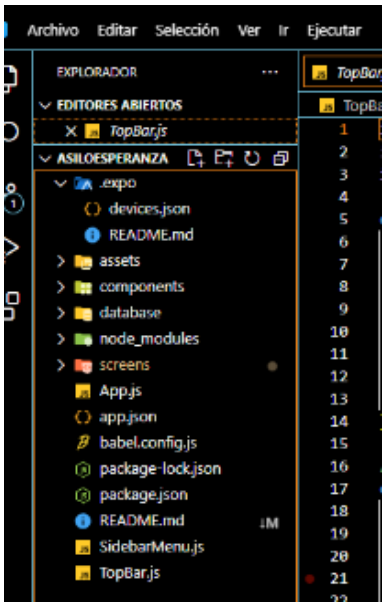


En la carpeta components tenemos un ejemplo de como mostrar una imagen y texto de forma local, mientras que en la carptade database, se ha establecido la configuración e inicialización de la base de datos; dentro de la carpeta que tiene como titulo Screens se han guardado tosdas las vista y configuraciones realizadas a cada modulo que se ha creado, pasando desde el login Los registros de los diferentes usuarios de la app, así como el manejo de los datos que se obtiene de los registro previamente realizados

En el archivo Apps.js se ha definido los componentes de navegación necesarios para react, lo que facilita la movilidad entre pantallas.

En el archivo de nombre app.json; se definen aspectos importantes respecto a la versiones y plataformas que se podrán comunicar.

Dentro de los últimos archivos tenemos el package-lock.json el cual primordial para gestionar y configurar las dependencias que se instalen a lo largo de todo el proyecto.



En el archivo Package.json se nos muestras las dependencias y las versiones que está utilizando el proyecto.

En SidebarMenu se estableció el sidebar que dentro proyecto sirve como menú de todos los módulos creados.

Y el archivo TopBar.js es la barra superior que se muestra en el proyecto donde se coloca el nombre de la vista que se observa.

Una vez con la estructura del proyecto un poco más clara detallaremos la lógica de programación que se ha utilizado en los archivos más relevantes del proyecto

ARCHIVO: APP.JS

Este fragmento de código define la estructura principal de una aplicación de React Native que utiliza React Navigation para la navegación entre diferentes pantallas. Aquí está una descripción de lo que hace:

```
import React, { useState } from 'react';
import { NavigationContainer } from '@react-
navigation/native';
import { createStackNavigator } from '@react-
navigation/stack';
import { Ionicons } from '@expo/vector-icons';
import RegistroPacientes from
'./screens/RegistroPacientes.js';
import GestionPacientes from
'./screens/GestionPacientes.js';
import HomeScreen from './screens/HomeScreen.js';
import Login from './screens/Login';
import RegistroDoctores from
'./screens/RegistroDoctores.js';
import GestionDoctores from
'./screens/GestionDoctores.js';
import RegistroCitas from
'./screens/RegistroCitas.js';
import GestionCita from
'./screens/GestionCita.js';
```

Importaciones: El código importa varios módulos necesarios para la configuración de la navegación y para definir las pantallas de la aplicación. Importa React y useState de React, así como varios componentes de navegación de

React Navigation como NavigationContainer y createStackNavigator. También importa el componente Ionicons del paquete @expo/vector-icons para usar iconos en la barra de navegación.

Componente App: El componente App es el componente principal de la aplicación. Define un estado isMenuOpen con useState para controlar si el menú está abierto o cerrado.

Función toggleMenu: Define una función toggleMenu que cambia el estado de isMenuOpen entre true y false cuando se llama. Esta función se utilizará para abrir y cerrar el menú.

NavigationContainer y Stack.Navigator: Envuelve el conjunto de rutas de la aplicación dentro de un NavigationContainer proporcionado por React Navigation. Dentro de este contenedor, define un Stack.Navigator que contendrá todas las pantallas de la aplicación.

```
const Stack = createStackNavigator();

const App = () => {
  const [isMenuOpen, setIsMenuOpen] =
useState(false);

  const toggleMenu = () => {
    setIsMenuOpen(!isMenuOpen);
  }

  return (
     <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Login"
          component={Login}
          options={{ headerShown: false }}
        />
        <Stack.Screen
          name="Home"
          component={HomeScreen}
          options={{ headerShown: false }}
        />
        <Stack.Screen
          name="Form"
          component={RegistroPacientes}
          options={{
            title: 'Registro Pacientes',
            headerStyle: {
              backgroundColor: 'black',
            },
            headerTintColor: 'white',
            headerBackTitleVisible: false,
            headerBackImage: () => (
              <Ionicons name="arrow-back"
size={24} color="orange" />
            ),
          }}
        />
        <Stack.Screen
          name="GestionPacientes"
          component={GestionPacientes}
          options={{
            title: 'Gestion de pacientes',
```

Stack.Screen: Define las diferentes pantallas de la aplicación dentro del Stack.Navigator. Cada pantalla está representada por un componente y tiene opciones de navegación, como el título, el estilo del encabezado y los botones de navegación personalizados.

Opciones de pantalla: Para cada pantalla, se definen opciones de navegación que incluyen el título de la pantalla, el estilo del encabezado (backgroundColor, headerTintColor), y el componente a mostrar cuando se navega hacia atrás (headerBackImage).

En resumen, este archivo define la estructura de navegación de una aplicación de React Native utilizando React Navigation, con varias pantallas y opciones de navegación configuradas para cada pantalla.

```
            headerStyle: {
              backgroundColor: 'black',
            },
            headerTintColor: 'white',
            headerBackTitleVisible: false,
            headerBackImage: () => (
              <Ionicons name="arrow-back"
size={24} color="orange" />
            ),
          }}
        />
        <Stack.Screen
          name="RegistroDoctores"
          component={RegistroDoctores}
          options={{
            title: 'Registro de doctores',
            headerStyle: {
              backgroundColor: 'black',
            },
            headerTintColor: 'white',
            headerBackTitleVisible: false,
            headerBackImage: () => (
              <Ionicons name="arrow-back"
size={24} color="orange" />
            ),
          }}
        />

    </NavigationContainer>

  );
}

export default App;
```

SCREENS: LOGICA DE LAS VISTAS CREADAS

GESTION DE CITAS

Este fragmento de código es un componente funcional de React Native que crea un formulario para registrar detalles de una cita médica. Aquí hay una descripción de lo que hace:

Importaciones: Importa React y useState de React, así como varios componentes de React Native necesarios para construir la interfaz de usuario, como View, Text, TextInput y TouchableOpacity.

Estado local: Utiliza el hook useState para definir y gestionar el estado local de diferentes campos del formulario, como fechaCita, nombrePaciente, nombreDoctor, edad, categoriaCita y observacion.

Función handleRegistro: Define una función llamada handleRegistro que se ejecuta cuando se presiona el botón "Guardar". Esta función imprime en la consola los valores de todos los campos del formulario.

```
import React, { useState } from 'react';
import { View, Text, TextInput,
TouchableOpacity } from 'react-native';

export default function App() {
  const [fechaCita, setFechaCita] =
useState('');
  const [nombrePaciente, setNombrePaciente]
= useState('');
  const [nombreDoctor, setNombreDoctor] =
useState('');
  const [edad, setEdad] = useState('');
  const [categoriaCita, setCategoriaCita] =
useState('');
  const [observacion, setObservacion] =
useState('');

// trae los datos de las constantes de
arriba.
  const handleRegistro = () => {
    console.log('Fecha de Cita:',
fechaCita);
    console.log('Nombre del paciente:',
nombrePaciente);
    console.log('Nombre del Doctor:',
nombreDoctor);
    console.log('Edad:', edad);
    console.log('Categoria de Cita:',
categoriaCita);
    console.log('Observacion:',
observacion);
  };

  const inputStyle = {
    padding: 5,
    borderWidth: 1,
    borderColor: 'orange',
    borderRadius: 10,
    marginBottom: 15,
    backgroundColor: 'white',
  };

  const buttonStyle = {
    backgroundColor: 'orange',
    padding: 12,
```

Estilos: Define estilos para los componentes del formulario, como el estilo del botón y el estilo de entrada de texto.

Interfaz de usuario: Renderiza la interfaz de usuario del formulario dentro de las vistas de React Native (View). Incluye campos de entrada de texto para la fecha de la cita, el nombre del paciente, el nombre del doctor, la edad, la categoría de la cita y las observaciones. También incluye un botón "Guardar" (TouchableOpacity) que ejecuta la función handleRegistro cuando se presiona.

En resumen, este componente de React Native crea un formulario simple para ingresar detalles de una cita médica y proporciona una función para manejar el registro de esta información.

```
    borderRadius: 10,
    alignItems: 'center',
  };

  const buttonText = {
    color: 'black',
    fontWeight: 'bold',
  };
//retorna a la vista
  return (
    <View style={{ flex: 1, backgroundColor:
'white', justifyContent: 'center', padding:
20 }}>
      <View style={{ backgroundColor:
'lightgray', borderRadius: 15, padding: 20
}}>
        {}
        <View style={{ marginBottom: 10 }}>
          <Text style={{ marginBottom: 10,
fontWeight: 'bold' }}>Fecha de Cita:</Text>
          <TextInput
            style={inputStyle}
            placeholder="DD/MM/AAAA"
            value={fechaCita}
            onChangeText={setFechaCita}
          />
        </View>

        {}
        <View style={{ marginBottom: 10 }}>
          <Text style={{ marginBottom: 10,
fontWeight: 'bold' }}>Nombre del
paciente:</Text>
          <TextInput
            style={inputStyle}
            placeholder=""
            value={nombrePaciente}
            onChangeText={setNombrePaciente}
          />
        </View>

        <View style={{ marginBottom: 10 }}>
          <Text style={{ marginBottom: 10,
fontWeight: 'bold' }}>Nombre del
Doctor:</Text>
```

```
          <TextInput
            style={inputStyle}
            placeholder=""
            value={nombreDoctor}
            onChangeText={setNombreDoctor}
          />
        </View>

        <View style={{ marginBottom: 10 }}>
          <Text style={{ marginBottom: 10,
fontWeight: 'bold' }}>Edad:</Text>
          <TextInput
            style={inputStyle}
            placeholder=""
            value={edad}
            onChangeText={setEdad}
          />
        </View>

        <View style={{ marginBottom: 10 }}>
          <Text style={{ marginBottom: 10,
fontWeight: 'bold' }}>Categoria de
Cita:</Text>
          <TextInput
            style={inputStyle}
            placeholder=""
            value={categoriaCita}
            onChangeText={setCategoriaCita}
          />
        </View>

        <View style={{ marginBottom: 10 }}>
          <Text style={{ marginBottom: 10,
fontWeight: 'bold' }}>Observacion:</Text>
          <TextInput
            style={{ ...inputStyle, height:
100 }}
            placeholder=""
            value={observacion}
            onChangeText={setObservacion}
            multiline={true}
            numberOfLines={6}
          />
        </View>
```

	<TouchableOpacity onPress={handleRegistro} style={buttonStyle}> <Text style={buttonText}>Guardar</Text> </TouchableOpacity> </View> </View>); }

DATABASE

Este código utiliza la biblioteca Firebase para interactuar con la base de datos Firestore y realizar operaciones de escritura (añadir documentos) en tres colecciones diferentes: "Doctor", "Paciente" y "Cita". Aquí está lo que hace cada parte del código:
 	
Importaciones de Firebase: Importa Firebase y el módulo Firestore.
 	
Configuración de la conexión a Firebase: Define la configuración de conexión a Firebase, que incluye la apiKey, authDomain, projectId, etc.
	const firebase = require('firebase/app');
	require('firebase/firestore');
	
	// Configura la conexion a Firebase
	const firebaseConfig = {
	apiKey: "AIzaSyBYQytt_OkWbm2AexhNcsdBXSUlHC9b6dg",
	authDomain: "dps-asilo-sv.firebaseapp.com",
	projectId: "dps-asilo-sv",
	storageBucket: "dps-asilo-sv.appspot.com",
	messagingSenderId: "186549304054",
	appId: "1:186549304054:web:595412bc63346015cacad7",
	measurementId: "G-QSVW3HW5BB"
	};
	
	// Inicializa Firebase con la configuracion
	firebase.initializeApp(firebaseConfig);
	
	// Obtiene una referencia a la base de datos Firestore
	const db = firebase.firestore();
	
	// Datos para el doctor
	const doctorData = {
	carnet: NaN,

Inicialización de Firebase: Inicializa Firebase con la configuración proporcionada.

Obtención de una referencia a Firestore: Obtiene una referencia a la base de datos Firestore para poder realizar operaciones en ella.

Datos para el doctor, paciente y cita: Define objetos JavaScript que representan datos para un doctor, un paciente y una cita.

Añadir documentos a las colecciones: Utiliza el método add() para añadir documentos a las colecciones "Doctor", "Paciente" y "Cita" en Firestore. Cada llamada a add() crea un nuevo documento en la colección correspondiente con los datos proporcionados.

```
  celular: 71368014,
  correo: "",
  especialidad: "Pediatra",
  jvpm: "12345",
  nombre: "Kevin Enrique Martinez Mar"
};

// Añade un documento a la coleccion
"Doctor"
db.collection('Doctor').add(doctorData)
  .then(docRef => {
    console.log('Documento de Doctor creado
con ID: ', docRef.id);
  })
  .catch(error => {
    console.error('Error al crear el
documento de Doctor: ', error);
  });

// Datos para el paciente
const pacienteData = {
  condicion: "En tratamiento",
  dirrecion: "1",
  dui: "02568794-3",
  edad: 73,
  estado: "en proceso",
  familiar: "1",
  fechaexp: "12/5/2024",
  nombre: "Juan",
  numero: 11111111,
  peso: 70
};

// Añade un documento a la coleccion
"Paciente"
db.collection('Paciente').add(pacienteData)
  .then(docRef => {
    console.log('Documento de Paciente
creado con ID: ', docRef.id);
  })
  .catch(error => {
    console.error('Error al crear el
documento de Paciente: ', error);
  });
```

Gestión de errores y éxito: Utiliza las promesas then() y catch() para manejar tanto los errores como el éxito de la operación de escritura en Firestore. En caso de éxito, se imprime el ID del documento creado en la consola. En caso de error, se imprime el error en la consola.

Ejecución: Se proporcionan instrucciones para instalar la biblioteca Firebase y ejecutar el script mediante node.

En resumen, este código sirve para conectar y configurar Firebase, añadir documentos a la base de datos Firestore en tres colecciones diferentes (Doctor, Paciente y Cita) y manejar posibles errores o éxito en estas operaciones de escritura

```
// Datos para la cita
const citaData = {
  nombrePaciente: 'Nombre del paciente',
  nombreDoctor: 'Nombre del doctor',
  edad: 'Edad del paciente',
  categoriaCita: 'Categoria de la cita',
  fecha: 'Fecha de la cita'
};

// Añade un documento a la coleccion "Cita"
db.collection('Cita').add(citaData)
  .then(docRef => {
    console.log('Documento de Cita creado con ID: ', docRef.id);
  })
  .catch(error => {
    console.error('Error al crear el documento de Cita: ', error);
  });


// Ejecucion
// npm install firebase
//node firebase_script.js
```