



Manual de técnico

NURSE CONTROL





Índice

- 01 Integrantes
- 02 Requerimientos de desarrollo
- 03 Estructura
- 04 Card.js
- 05 pacientes.js
- 06 pacientes.js
- 07 paciente/[id].js



Integrantes

- Alas Linares Alejandro Antonio AL192188
- De Paz Velásquez Vicente Daniel DV192307
- Duran Meléndez Gilberto Emmanuel DM192201
- Gutiérrez Borja Rafael Armando GB192205

Requerimientos de desarrollo



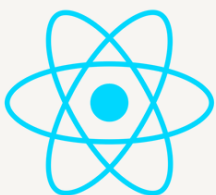
Android Studio

Herramienta de desarrollo que permite crear y utilizar emuladores del SO Android.



Firebase

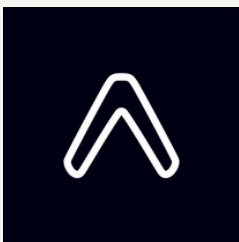
Servicio en la nube de google que permite el almacenamiento de datos y autenticación de usuarios.



React Native

React Native

Librería de javascript que permite desarrollar Cross-platform applications.

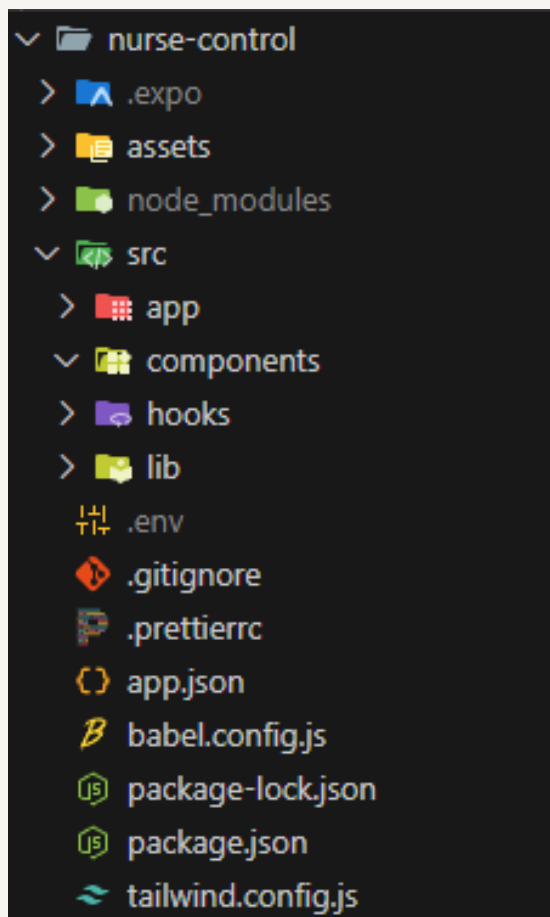


Expo

Ecosistema de herramientas que facilita el desarrollo de app con React Native.

Estructura

Para lograr una mejor organización manejamos todos los archivos dentro de la carpeta principal *src*, excepto por los *assets* o los archivos de configuración. A continuación presentamos una breve descripción de cada carpeta.



- **assets:** aloja archivos estáticos como imágenes o fuentes.
- **app:** debido a que utilizamos la librería *expo router* manejamos todas las rutas (pantallas) dentro de la carpeta *app*.
- **components:** aquí almacenamos los componentes reutilizables.
- **hooks:** en esta carpeta almacenamos *custom hooks*.
- **lib:** aquí almacenamos la configuración de diversas librerías como por ejemplo: *firebase*.
- **.env:** archivo local que almacena las variables de entorno
- **.gitignore:** archivo de configuración que previene subir archivos delicados al repo
- **.prettierrc:** archivo de configuración de la herramienta *prettier*.
- **app.json:** archivo de configuración de *expo*.
- **babel.config.js:** archivo de configuración del transpilador.
- **package.json:** archivo que maneja las librerías utilizadas por la app.
- **tailwind.config.js:** archivo de configuración de la librería de estilos *tailwind*.

Card.js

En muchas partes de la aplicación se utilizarán Cards para desplegar la información, por ello se ha agregado un componente reutilizable en: `[src/components/Card.js]`. El componente necesita la siguiente estructura de props

- **keys:** { header: { name: string, label: string }, body: { name: string, label: string }[] }
- **data:** Cloud Firestore data response
- **buttons:** { buttonText: string, icon: string, className: string, styles: object, action: function }[]

```
const Cards = ({ keys, data, buttons }) => {
  const theme = useTheme();

  return (
    <div>
      {data?.map((el, i) => (
        <Card key={el.id} className="mb-5">
          <Card.Title>
            title={` ${keys.header.label} ${el[keys.header.name]} `}
            subtitle=""
            style={{
              borderColor: 'transparent',
              backgroundColor: theme.colors.primaryContainer,
              marginBottom: 10,
            }}
          </Card.Title>
          {keys.body.map((subEl, j) => (
            <Card.Content key={i + j} style={{ margin: 1 }}>
              {subEl?.name === 'rol' ? (
                <Text variant="bodyMedium" style={{ fontWeight: 800 }}>{` ${
                  el[subEl.name]
                } `}</Text>
              ) : (
                <Text variant="bodyMedium">{` ${subEl.label} ${
                  el[subEl.name]
                } `}</Text>
              )}
            </Card.Content>
          ))}
          <Card.Actions>
            {buttons?.map((button, i) => (
              <Button
                key={i}
                icon={button.icon}
                mode="elevated"
                className={button.className}
                style={button.style}
                onPress={() => button.action(el)}
              >
                {button.buttonText}
              </Button>
            ))}
          </Card.Actions>
        </Card>
      ))}
    </div>
  );
}
```

pacientes.js

Este es el componente para la gestión de los pacientes, esta ruta muestra los Cards con la información de los pacientes, además de diversos botones para la gestión de los mismos.

```
const [data, setData] = useState(null);
const [isLoading, setLoading] = useState(false);
const theme = useTheme();
const [hasChanged, setHasChanged] = useState(false);

useFocusEffect(
  useCallback(() => {
    getData();
  }, [])
);

useEffect(() => {
  getData();
}, [hasChanged])

const getData = async () => {
  setLoading(true);
  const snapshots = await getDocs(collection(db, 'pacientes'));
  const res = snapshots.docs.map((doc) => {
    return {
      id: doc.id,
      ...doc.data(),
      fecha_nacimiento: moment(
        doc.data()['fecha_nacimiento'].seconds * 1000
      ).format('DD-MM-YYYY'),
      nombre_completo: `${doc.data().nombres} ${doc.data().apellidos}`,
    };
  });
  setData(res);
  setLoading(false);
};
```

Estados del componente, función para obtener los datos y hooks para actualizar el contenido cuando el componente obtenga el foco.

```
const editPaciente = (document) => {
  router.navigate(`/paciente/${document.id}`);
}

const deletePaciente = async (document) => {
  Alert.alert(
    '¿De verdad deseas eliminar al paciente?',
    'Haz click en "Sí" para eliminar, da click en "Cancelar para abortar la operación"',
    [
      { text: 'Sí', onPress: () => procedDelete(document) },
      { text: 'Cancelar', onPress: () => null },
    ]
  );
}

const procedDelete = async (document) => {
  try {
    const docRef = doc(db, "pacientes", `${document.id}`);
    await deleteDoc(docRef);
    Alert.alert(
      '¡Paciente eliminado con éxito!',
      'Hemos eliminado los registros del paciente',
      [
        { text: 'Ok', onPress: () => setHasChanged(true) },
      ]
    );
  } catch(e) {
    Alert.alert(
      '¡Algo salio mal!',
      'No pudimos eliminar al paciente, intentalo de nuevo más tarde',
      [
        { text: 'Ok', onPress: () => null },
      ]
    );
  }
}
```

Acciones de los botones eliminar y editar, con los alerts y efectos asociados a los mismos.

```
const keys = {
  header: { name: 'nombre_completo', label: '' },
  body: [
    { name: 'genero', label: 'Genero: ' },
    { name: 'fecha_nacimiento', label: 'Fecha de nacimiento: ' },
    { name: 'correo', label: 'Email: ' },
  ],
};

const buttons = [
  { buttonText: 'Editar', icon: 'account-edit', className: '', styles: {}, action: editPaciente },
  { buttonText: 'Eliminar', icon: 'delete', className: 'bg-red-200', styles: {}, action: deletePaciente },
]

const handleClick = () => {
  router.navigate(`/paciente/[id]`);
};
```

Keys, y buttons objects, usados como props para Cards component.

handleClick function, asociada al botón de agregar nuevo paciente.

pacientes.js

```
return (  
  <ScrollView  
    refreshControl={  
      <RefreshControl refreshing={isLoading} onRefresh={onRefresh} />  
    }  
    className="p-2 flex flex-1 mb-5"  
  >  
    {  
      !isLoading && (  
        <Text className="text-lg p-1.5">Listado de pacientes</Text>  
        <View className="my-2 flex-row ml-auto">  
          <Button  
            icon="head-plus"  
            mode="elevated"  
            className="self-start max-w-40"  
            contentStyle={{  
              backgroundColor: theme.colors.secondaryContainer,  
            }}  
            onPress={handleClick}  
          >  
            Agregar nuevo paciente  
          </Button>  
        </View>  
        <Cards keys={keys} data={data} buttons={buttons} />  
      </>  
    )  
  }  
</ScrollView>  
);
```

jsx usado para mostrar la información del paciente, junto con los botones de acción.

paciente/[id].js

Componente utilizado para crear/actualizar a los pacientes. La funcionalidad del componente varía según los datos enviados como slug a la ruta.

```
const theme = useTheme();
const { id } = useLocalSearchParams();
const [docToUpdate, setDocToUpdate] = useState(null);

const [
  control,
  handleSubmit,
  formState: { errors },
  reset,
] = useForm({
  resolver: yupResolver(schema),
});

useEffect(() => {
  const load = async () => {
    if(id !== '[id]') {
      const res = await getData(id);

      if(!res) return;
      reset({
        names: res.nombres,
        surnames: res.apellidos,
        email: res.correo,
        genre: res.genero,
        birthday: moment(res['fecha_nacimiento']).seconds * 1000,
      });
    }
  };
  load();
}, [id])

const getData = async (docId) => {
  const entry = await getDoc(doc(db, "pacientes", `${docId}`));
  const docRef = doc(db, "pacientes", `${docId}`);
  setDocToUpdate(docRef);
  return entry.exists() ? entry.data() : null;
}
```

Estados del componente, función para obtener los datos y hooks para actualizar el contenido cuando el componente obtenga el foco.

```
const onSubmit = async ({ names, surnames, email, birthday, genre }) => {
  const paciente = {
    nombres: names,
    apellidos: surnames,
    correo: email,
    fecha_nacimiento: birthday,
    genero: genre
  };

  if(id !== '[id]') paciente.id = id;
  let res = null;
  try {
    setloading(true);
    if(docToUpdate) {
      res = await updateDocument(paciente);
    } else {
      res = await addDoc(collection(db, 'pacientes'), paciente);
    }
    if (res.id) {
      Alert.alert(
        'Paciente ${res?.type === 'update' ? 'actualizado' : 'agregado'}',
        'El paciente ha sido ${res?.type === 'update' ? 'actualizado' : 'registrado'} con éxito',
        [{ text: 'OK', onPress: () => clean() }]
      );
    }
    setloading(false);
  } catch (e) {
    console.log(e);
  }
};

const updateDocument = async (doc) => {
  try {
    await updateDoc(docToUpdate, doc);
    return { id: '1234', type: 'update' };
  } catch(e) {
    return { type: 'update' };
  }
}
```

Acción para actualizar o agregar un nuevo paciente.

```
const clean = () => {
  reset({
    names: '',
    surnames: '',
    email: '',
    birthday: '',
  });
  if(id !== '[id]') return router.navigate('/pacientes');
  Alert.alert(
    '¿Quiere regresar a la lista de pacientes?',
    'Da click en ¡Sí! para navegar al listado de pacientes',
    [
      { text: '¡Sí!', onPress: () => router.navigate('/pacientes') },
      { text: '¡No!', onPress: () => null },
    ]
  );
};
```

Función para limpiar el formulario después de un ingreso exitoso.

Muestra un alert para preguntar si se desea agregar un paciente nuevo o si se desea navegar al listado.