UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

FACULTAD DE INGENIERÍA

CARRERA DE INGENIERIA EN SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Manual del Programador

Desarrollo e implementación de un demo de app móvil para brindar primeros auxilios psicológicos

Autor: Estrada Alejandro Rodrigo Javier

Director: Irene Robalino, Pedro Daniel

LOJA

2024

Índice

1. Introducción

Propósito del Manual

Este manual está diseñado para servir como una guía completa y detallada para los desarrolladores que trabajarán en la aplicación móvil de primeros auxilios psicológicos. Proporciona instrucciones claras y concisas sobre la configuración del entorno de desarrollo, la estructura del proyecto, el uso de herramientas y librerías específicas, así como la arquitectura utilizada para este proyecto. El objetivo es asegurar que todos los miembros del equipo de desarrollo puedan trabajar de manera eficiente y coherente, manteniendo la calidad y la continuidad del proyecto.

Descripción general del proyecto

La aplicación móvil de primeros auxilios psicológicos es una herramienta diseñada para ofrecer apoyo emocional y recursos de salud mental a los usuarios. Utiliza tecnologías modernas como React Native, Expo, y TypeScript, y está integrada con servicios avanzados como Hygraph para la gestión de contenido, Clerk para la autenticación de usuarios, y OpenAI para la interacción avanzada con el asistente virtual. La aplicación proporciona una plataforma accesible para la autoevaluación, educación en salud mental, y conexión con profesionales en caso de emergencia, con el objetivo de mejorar el bienestar emocional de sus usuarios.

2. Arquitectura de la aplicación

Descripción de la arquitectura

La aplicación PsychoApp sigue una arquitectura basada en componentes, lo que permite una separación clara de las responsabilidades y una mayor modularidad. Esta arquitectura facilita el mantenimiento, la escalabilidad y la reutilización del código. Cada componente cumple una función específica dentro de la aplicación, permitiendo que los desarrolladores trabajen en diferentes partes del sistema de manera independiente y paralela.

La arquitectura se divide en tres capas principales:

- **Vista**: Implementada con React Native y Expo, esta capa maneja la interfaz de usuario y las interacciones.
- **Controlador**: Contiene la lógica de la aplicación y los asistentes de inteligencia artificial.
- **Modelo**: Encargado de la gestión de datos y la autenticación, utilizando Hygraph y Clerk respectivamente.

Principios de diseño y estructura de proyecto

1. **Separación de responsabilidades**: Cada capa de la arquitectura tiene responsabilidades bien definidas, lo que facilita el desarrollo y el mantenimiento del sistema.
    - **Vista:** Maneja la interfaz de usuario y las interacciones del usuario. Los componentes de la UI se encuentran en el directorio ***presentation/components*** y las pantallas en el directorio ***presentation/screens.***
    - **Controlador:** Contiene la lógica de negocio en el directorio /src, incluyendo el manejo de flujos de trabajo, la integración con OpenAI para el asistente virtual y la lógica del asistente IA.

- **Modelo:** Gestiona la persistencia de datos y la autenticación de usuarios. La integración con Hygraph se encuentra en el directorio /config y la configuración de Clerk para la autenticación se maneja en el directorio /config.

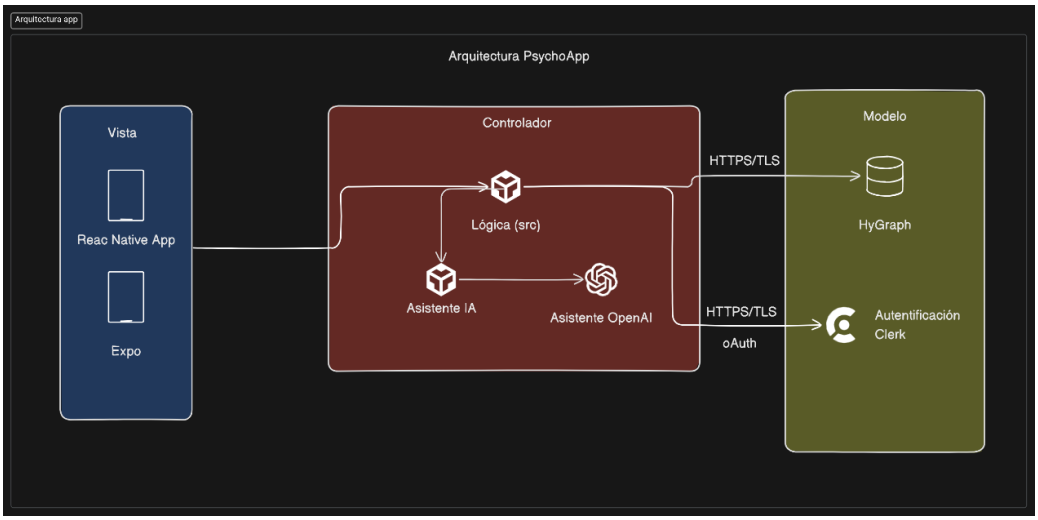
2. **Modularidad**: La aplicación se divide en módulos independientes que pueden desarrollarse y probarse de manera aislada. Esto mejora la capacidad de respuesta a cambios y nuevas funcionalidades.
3. **Reutilización de componentes:** Los componentes de UI y la lógica de negocio se diseñan para ser reutilizables en diferentes partes de la aplicación, lo que reduce la duplicación de código y mejora la consistencia.

Beneficios de la arquitectura basada en componentes

1. **Facilita el mantenimiento**: La clara separación de responsabilidades y la modularidad hacen que el código sea más fácil de entender y mantener. Los desarrolladores pueden identificar rápidamente dónde realizar cambios o arreglar errores.
2. **Mejora la escalabilidad**: La estructura modular permite añadir nuevas funcionalidades o mejorar las existentes sin afectar otras partes de la aplicación. Esto es crucial para soportar el crecimiento continuo del proyecto.
3. **Promueve la colaboración**: Diferentes equipos pueden trabajar en distintos componentes de la aplicación simultáneamente, sin interferencias, lo que mejora la eficiencia y la productividad.
4. **Facilita la reutilización del código**: Los componentes bien diseñados pueden ser reutilizados en múltiples lugares de la aplicación, lo que reduce el tiempo de desarrollo y asegura la coherencia en la interfaz de usuario y la lógica de negocio.

Descripción del diagrama de arquitectura

El diagrama de la arquitectura de PsychoApp ilustra cómo las diferentes capas y componentes interactúan entre sí:



1. **Vista**: Representada por React Native App y Expo, maneja toda la interfaz de usuario y las interacciones del usuario con la aplicación.
2. **Controlador**:
    - **Lógica (src)**: Gestiona la lógica de la aplicación y el flujo de datos entre la vista y el modelo.
    - **Asistente IA**: Encargado de manejar las funcionalidades del asistente de inteligencia artificial.
    - **Asistente OpenAI**: Utiliza la API de OpenAI para proporcionar respuestas y asistencia avanzada.
3. **Modelo**:
    - **Hygraph**: Se utiliza para la gestión de contenido y almacenamiento de datos, asegurando una comunicación segura a través de HTTPS/TLS.
    - **Autenticación Clerk**: Maneja la autenticación de usuarios utilizando OAuth para asegurar la privacidad y seguridad de los datos del usuario.

Cada una de estas capas y componentes está conectada a través de protocolos seguros como HTTPS/TLS para la transferencia de datos y OAuth para la autenticación,

asegurando que la aplicación no solo sea funcional y eficiente, sino también segura para los usuarios.

3. Configuración del entorno de desarrollo

Requisitos del sistema

Para asegurar una configuración de desarrollo fluida y efectiva, asegúrate de que tu sistema cumpla con los siguientes requisitos mínimos:

- **Sistema Operativo:** Windows 10/11, macOS 10.15 Catalina o superior, Linux (distribuciones modernas).
- **Memoria RAM:** Mínimo 8 GB (recomendado 16 GB o más).
- **Espacio en Disco**: Al menos 20 GB de espacio libre.
- **Procesador**: Procesador de 64 bits; Intel Core i5 o superior recomendado.
- **Software Adicional**:
    - Navegador web actualizado (Chrome, Firefox, etc.)
    - Emulador de Android y/o simulador de iOS (para pruebas en dispositivos virtuales).

Instalación de herramientas necesarias

Para configurar el entorno de desarrollo, sigue estos pasos para instalar las herramientas necesarias:

- **Node.js:**
    - Descarga e instala la última versión estable de Node.js desde https://nodejs.org/en
    - Para referencia, el proyecto actual utiliza la versión **v20.11.1**

- **Expo CLI**
    - Instala Expo CLI globalmente usando npm o yan: **npm install -g expo-cli**

Clonar repositorio desde Github

Una vez instaladas las herramientas procedemos a clonar el repositorio del proyecto siguiendo los siguientes pasos:

- **Clonar repositorio:** Dirígete al siguiente enlace y clona el repositorio de proyecto: https://github.com/javierestrada26/PyschoApp
- **Instalar dependencias:** Ingresa al proyecto desde tu IDE y ejecuta el siguiente comando para descargar las dependencias del proyecto: **npm i**

Configuración de variables de entorno

Las variables de entorno son cruciales para la configuración y seguridad del proyecto. Aquí se describen algunas variables importantes que necesitas configurar:

- **Archivo .env**: Crea un archivo .env en la raíz de tu proyecto y añade variables que se encuentran en archivo env.template. Estas variables de entorno corresponden a los servicios que se utilizan en la aplicación (Hygraph, OpenAI, Clerk)

Configuración de emuladores y dispositivos físicos:

- **Emulador de Android:** Si estás usando Android Studio, asegúrate de tener configurado un AVD (Android Virtual Device). Descárgalo desde el siguiente enlace : https://developer.android.com/studio/install?hl=es-419
- **Simulador de iOS:** Si estás en macOS, puedes usar Xcode para ejecutar simuladores de iOS.

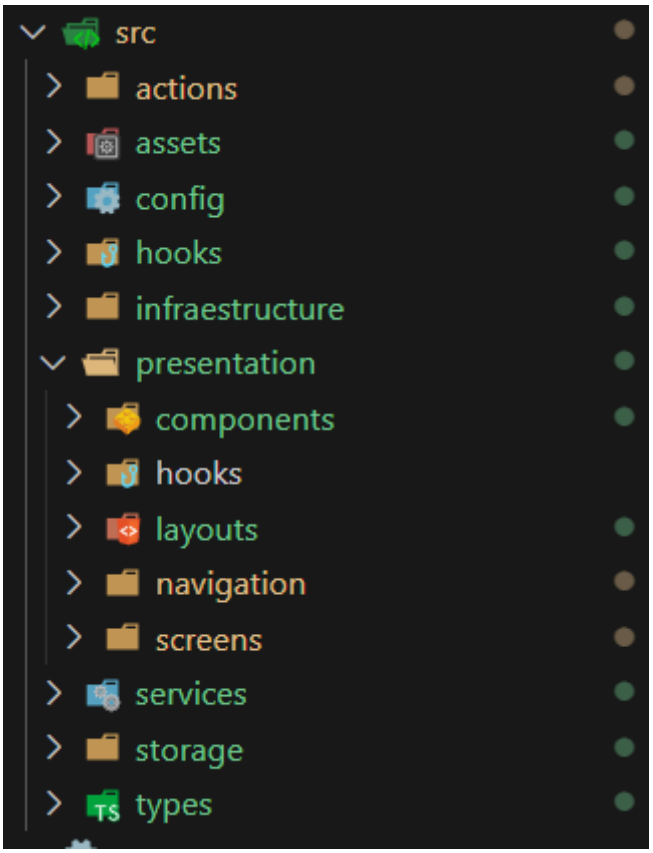
- **Dispositivo físico:** Instala la aplicación Expo Go en tu dispositivo móvil y escanea el código QR generado por Expo para probar la aplicación en un dispositivo físico.

Con esta configuración del entorno de desarrollo, deberías estar listo para empezar a trabajar en el proyecto PsychoApp, garantizando que todos los desarrolladores tienen una base sólida y uniforme para el desarrollo, pruebas y despliegue de la aplicación.

4. Estructura del proyecto

Descripción General de la estructura de directorios

La estructura del proyecto está organizada para facilitar la mantenibilidad, escalabilidad y claridad del código. La organización sigue una arquitectura de componentes, separando las responsabilidades en distintas carpetas. A continuación, se describe la estructura principal bajo el directorio src, que es el núcleo de la aplicación:



src/: Contiene todo el código fuente de la aplicación.

- **actions/**: Maneja las acciones que desencadenan cambios en el estado global o interactúan con servicios externos.
- **assets/**: Contiene los recursos estáticos de la aplicación, como imágenes, fuentes, íconos, etc.
- **config/**: Almacena configuraciones globales, constantes, y archivos de configuración del entorno.
- **hooks/**: Contiene hooks personalizados que encapsulan lógica reutilizable a lo largo de la aplicación.
- **infrastructure/**: Abarca la infraestructura de la aplicación, como la configuración de API, utilidades de red, o servicios básicos.
- **presentation/**: Encapsula todo lo relacionado con la presentación y la UI de la aplicación.
    - **components/**: Contiene los componentes reutilizables de la interfaz de usuario.
    - **hooks/**: Hooks específicos de la capa de presentación para manejar la lógica UI relacionada.
    - **layouts/**: Define las disposiciones o estructuras básicas de la UI que se reutilizan en varias pantallas.
    - **navigation/**: Maneja la lógica de navegación de la aplicación, incluyendo la configuración de rutas y stacks de navegación.
    - **screens/**: Contiene las pantallas de la aplicación, cada una representando una vista completa.
- **services/**: Incluye los servicios que interactúan con APIs externas, como llamadas a OpenAI, Clerk, o Hygraph.
- **storage/**: Maneja el almacenamiento local o persistente en la aplicación, como AsyncStorage o cualquier otra solución de persistencia.

- **types/**: Define el tipo de variables de entorno que se va a exportar para utilizar en nuestra aplicación.

Esta estructura bien organizada permite a los desarrolladores navegar el código de manera eficiente, comprender rápidamente dónde se encuentra la lógica relevante, y facilitar el desarrollo colaborativo dentro del equipo.

5. Configuración de servicios e inicio de proyecto

Servicios utilizados en la aplicación

La aplicación PsychoApp utiliza varios servicios externos para proporcionar funcionalidades avanzadas y manejar la gestión de datos, autenticación y procesamiento de lenguaje natural. A continuación, se describen los servicios principales:

Hygrahp

Hygraph es un sistema de gestión de contenido basado en GraphQL. En PsychoApp, se utiliza para manejar y servir datos estructurados que la aplicación consume, como artículos, recursos de ayuda y videos, etc. Hygraph permite una integración fluida con GraphQL, lo que facilita las consultas y mutaciones en el backend.

- **Configuración**
    - Diríjase a https://hygraph.com/ e inicie sesión con las credenciales proporcionadas
    - Selecciona el proyecto de psychoApp
    - Haga clic en Project settings en la parte inferior izquierda
    - Y copie la uri que se encuentra en la sección de enviroments
    - Diríjase al proyecto y en la variables de entorno ‘.env’, pegue la uri en la variable correspondiente

Clerk

Clerk es un servicio que proporciona autenticación, gestión de usuarios y manejo de sesiones de manera sencilla. En PsychoApp, Clerk se encarga de la autenticación de los usuarios, permitiendo el inicio de sesión, registro, y manejo de sesiones de manera segura utilizando OAuth.

- **Configuración**
    - Diríjase a https://clerk.com/ e inicie sesión con las credenciales asignadas
    - Haga clic en dashboard y seguidamente a API KEYS en la parte inferior izquierda
    - Copie la **Publishable key,** y en el proyecto pegue la llave en la variable de entorno correspondiente

OpenAI

OpenAI se utiliza en PsychoApp para proporcionar funcionalidades avanzadas de procesamiento de lenguaje natural, como asistencia conversacional y generación de contenido. La API de OpenAI permite a la aplicación ofrecer respuestas inteligentes y apoyo emocional a los usuarios a través de un asistente virtual.

Nota: Se debe iniciar sesión con las credenciales asignadas

- **Configuración**
    - Diríjase a https://platform.openai.com/apps e inicie sesión
    - Elija API y diríjase a dashboard, seguidamente a api key y creamos en caso de ser necesario una key nueva y copiamos el token generado
    - Vamos a nuestro proyecto y en las variables de entorno pegar en la variable correspondiente.
    - Nos dirigimos nuevamente al dashboard de OpenAi y a la sección de asistente, y copiamos el id de nuestro asistente

- Regresamos a nuestro proyecto y pegamos este id en la variable de entorno correspondiente

Inicio de proyecto

Una vez que se hayan configurado todos los servicios descritos anteriormente, el proyecto está listo para ser iniciado. Para comenzar a trabajar con la aplicación, sigue los siguientes pasos:

1. **Inicia el servidor de desarrollo con Expo**:
    - Ejecuta el siguiente comando en la terminal desde la raíz del proyecto: **npx expo start**
    - Esto abrirá una ventana con el panel de control de Expo. Desde aquí, puedes elegir ejecutar la aplicación en un emulador, simulador o en un dispositivo físico.
2. **Conexión con el dispositivo:**
    - **Dispositivo Físico**: Abre la aplicación Expo Go en tu dispositivo móvil y escanea el código QR que aparece en la pantalla del panel de Expo para iniciar la aplicación en tu dispositivo.
    - **Emulador/Simulador:** Asegúrate de tener un emulador de Android o un simulador de iOS en ejecución. Puedes iniciar la aplicación directamente desde el panel de control de Expo

Con esta configuración y pasos iniciales, estarás listo para comenzar a desarrollar y probar la aplicación PsychoApp en un entorno configurado correctamente.

6. Desarrollo

Modulo 1: Desarrollo de Inicio de sesión y registro

El módulo de Inicio de Sesión y Registro es fundamental para la gestión de usuarios en la aplicación de primeros auxilios psicológicos. Este módulo permite a los usuarios registrarse y acceder a la aplicación de manera segura y eficiente, utilizando el servicio de autenticación proporcionado por Clerk.

Integración Técnica de Clerk

- **Configuración**: El módulo está configurado en la aplicación mediante el uso del SDK de Clerk, el cual se integra directamente en el código de React Native.
- **Componentes Utilizados**:
    - **ClerkProvider:** Proporciona acceso a las funcionalidades de autenticación en toda la aplicación.
    - **SignedIn, SignedOut**: Componentes preconstruidos por Clerk que permiten la fácil implementación de las interfaces de registro e inicio de sesión.

```
import { ClerkProvider, SignedIn, SignedOut} from '@clerk/clerk-expo';
const clerkPubKey = process.env.REACT_APP_CLERK_FRONTEND_API;
function App() {
  return (
        <SignedIn>
          <NavigationContainer>
            <TabNavigator/>
          </NavigationContainer>

        </SignedIn>
        {/**SignOut */}
        <SignedOut>
          {/**<SlidesScreen/> */}
          <LoginScreen/>
        </SignedOut>
  );
}
```

Nota: Para más detalles diríjase al repositorio en Github: https://bit.ly/3T2ofPh

Pantalla de Inicio de sesión

La pantalla de Inicio de Sesión es el primer punto de interacción del usuario con la aplicación, permitiendo el acceso a través de cuentas Google utilizando OAuth, integrado con Clerk. Esta pantalla es clave para ofrecer una experiencia de inicio de sesión rápida, segura y sencilla, asegurando que los usuarios puedan acceder a la aplicación sin fricciones.

Funcionalidades Principales

1. **Autenticación con Google**:
    - La pantalla de inicio de sesión utiliza OAuth para permitir que los usuarios se autentiquen con sus cuentas de Google. Esto simplifica el proceso de inicio de sesión, eliminando la necesidad de crear y recordar nuevas contraseñas.
    - Clerk maneja el flujo de OAuth, asegurando que la sesión del usuario se gestione correctamente una vez autenticado.
2. **Manejo del Flujo OAuth**:
    - La función startOAuthFlow es utilizada para iniciar el proceso de autenticación con Google. Si el usuario completa exitosamente la autenticación, Clerk crea una sesión que es automáticamente activada para el usuario.
    - Se maneja la posibilidad de errores durante el flujo OAuth, asegurando que cualquier problema se capture y gestione adecuadamente.

```tsx
import { useCallback } from "react";
import { useWindowDimensions } from "react-native";
import * as WebBrowser from 'expo-web-browser';
import { useOAuth, useWarmUpBrowser } from '@clerk/clerk-expo';

WebBrowser.maybeCompleteAuthSession();

export const LoginScreen = () => {
  const { height } = useWindowDimensions();

  // Precalienta el navegador para mejorar la experiencia del usuario
  useWarmUpBrowser();

  // Manejo del flujo de inicio de sesión con Google
  const { startOAuthFlow } = useOAuth({ strategy: "oauth_google" });

  const onPress = useCallback(async () => {
    try {
      const { createdSessionId, setActive } = (await startOAuthFlow()) ?? {};

      if (createdSessionId && setActive) {
        setActive({ session: createdSessionId });
      } else {
        // Maneja la falta de una sesión activa si es necesario
      }
    } catch (err) {
      console.error("OAuth error", err);
    }
  }, []);

  return (
    // UI de la pantalla de inicio de sesión con un botón que llama a onPress
  );
};
```

Nota: Para más detalles diríjase al repositorio en Github: https://bit.ly/3X6apwv

Explicación del Código

1. **WebBrowser.maybeCompleteAuthSession**:
    - Se llama a esta función para completar cualquier sesión de autenticación web que podría haber quedado abierta, lo que es común en flujos OAuth.
2. **useWarmUpBrowser**:
    - Se utiliza esta función para precalentar el navegador del dispositivo. Esto mejora la experiencia del usuario al hacer que el inicio de sesión

sea más rápido cuando el flujo OAuth requiere la apertura de un navegador.

3. **startOAuthFlow**:
    - Esta función, proporcionada por Clerk, inicia el flujo de autenticación con Google. Si la autenticación es exitosa, se genera un createdSessionId y se utiliza setActive para establecer la sesión activa del usuario.
4. **useCallback**:
    - onPress está envuelto en useCallback para garantizar que la función de inicio de sesión solo se vuelva a crear si alguna de sus dependencias cambia, lo que mejora el rendimiento.

Módulo 2: Personal de Apoyo

El módulo de Personal de Apoyo se encarga de gestionar y mostrar la información del equipo de soporte emocional disponible en la aplicación. Este módulo integra Hygraph para consultar y obtener los datos del personal de apoyo, como nombre, profesión, ciudad, descripción, y formas de contacto. Los datos se traen desde una API GraphQL, permitiendo una gestión eficiente y flexible del contenido.

Funcionalidades Principales

1. **Integración con Hygraph**:
    - Este módulo se conecta a Hygraph, un CMS basado en GraphQL, para extraer la información relevante sobre el personal de apoyo. Hygraph permite la gestión centralizada del contenido, lo que facilita la actualización y el mantenimiento de la información sin necesidad de modificar el código de la aplicación.

2. **Consulta de Datos**:
    - Utilizando una consulta GraphQL, el módulo extrae datos como el nombre, profesión, ciudad, descripción, URLs de contacto (WhatsApp, correo electrónico), y la imagen del personal de apoyo. Esta información se presenta luego en la interfaz de usuario.
3. **Visualización del Personal de Apoyo**:
    - La información obtenida se utiliza para renderizar una lista de profesionales disponibles para proporcionar apoyo emocional. Los usuarios pueden ver detalles sobre cada profesional y contactarlos directamente a través de los enlaces proporcionados.

```
import { request, gql } from 'graphql-request'

const MASTER_URL = process.env.HYGRAPH_API_URL

// Función para obtener los datos del personal de apoyo
const getPersonal = async () => {
  await sleep()  // Simulación de un retardo para propósitos de UX o pruebas
  const query = gql`
    query Personal {
      staffs {
        id
        name
        profession
        city
        description
        whatsUrl
        mailUrl
        image {
          url
        }
      }
    }
  `
  const data = await request(MASTER_URL, query)
  return data
}
```

Nota: Para más detalles diríjase al repositorio en Github: https://bit.ly/3XliC1c

Visualización de los datos en el Frontend

En la pantalla de Detalles de Personal, se muestran los datos de un miembro específico del equipo de apoyo emocional. La visualización está organizada para ofrecer una experiencia clara y accesible al usuario, permitiéndole ver la información relevante del personal y facilitando la navegación en la aplicación.

```
import { useNavigation, useRoute } from '@react-navigation/native';
import { useEffect, useState } from 'react';
import { TouchableOpacity, View } from 'react-native';
import { Icon, Text } from 'react-native-paper';
import { PersonalContent } from './personalDetails/PersonalContent';

export const PersonalDetails = () => {
    const { params } = useRoute();
    const navigation = useNavigation();
    const [staff, setStaff] = useState(params.staff);

    return (
        <View>
            <View >
                <TouchableOpacity onPress={() => navigation.goBack()}>
                    <Icon source='arrow-left' color='white' size={30} />
                </TouchableOpacity>
                <Text variant='titleLarge' style={{ fontWeight: 'bold', color: 'white' }}>Detalles de
Personal</Text>
            </View>
            <PersonalContent staff={staff} />
        </View>
    );
};
```

Nota: Para más detalles diríjase al repositorio en Github: https://bit.ly/3TvKA8v

Descripción del Componente

El componente PersonalDetails es responsable de mostrar los detalles específicos de un miembro del personal seleccionado. A continuación, se presenta una descripción de cómo se gestionan y muestran los datos en el frontend:

Nota: Para más detalles diríjase al repositorio en Github: https://acortar.link/btQ0OI

Explicación del Código

1. **Navegación y Rutas**:

- useRoute para obtener los detalles del personal desde los parámetros de la ruta.
- useNavigation para manejar la navegación.

2. **Estado del Componente**:
    - useState para almacenar y utilizar directamente la información del personal pasada en params.

Módulo 3: Categorías

El módulo de Categorías permite a los usuarios explorar y navegar a través de diferentes categorías de contenido dentro de la aplicación. Estas categorías pueden representar distintos temas o tipos de apoyo emocional, facilitando que los usuarios encuentren rápidamente el contenido que más les interese o necesiten. Este módulo obtiene las categorías de forma dinámica a través de una API, asegurando que cualquier cambio en las categorías se refleje automáticamente en la aplicación sin necesidad de actualizar el código.

Funcionalidades Principales

1. **Carga Dinámica de Categorías**:
    - Las categorías se obtienen desde una API externa utilizando hyGraphApi, lo que permite gestionar y actualizar las categorías desde un CMS sin necesidad de modificar el código de la aplicación.

```
import { request, gql } from 'graphql-request'
const getCategories =  async () => {
  const query = gql`
  query GetCategory {
    categories {
      id
      name
      icon {
        url
      }
    }
  }
  `
  const data = await request(MASTER_URL, query)
  return data
}
```

Nota: Para más detalles diríjase al repositorio en Github: https://bit.ly/3XliC1c

2. **Renderización de la Lista de Categorías**:
    - Una vez obtenidas, las categorías se pasan al componente CategoriesList, que se encarga de renderizarlas en la interfaz de usuario. Esto permite a los usuarios navegar por las diferentes opciones disponibles.

Código de Implementación

```
export const Categories = () => {
    const [categories, setCategories] = useState([]);

    useEffect(() => {
        hyGraphApi.getCategories().then((resp) => setCategories(resp.categories));
    }, []);

    return (
        <View style={{ marginTop: 10 }}>
            <HomeHeading text="Categorías" />
            <CategoriesList categories={categories} />
        </View>
    );
};
```

Nota: Para más detalles diríjase al repositorio en Github: https://bit.ly/4dDfsf9

Explicación del código

1. **Estado y Efecto:**
    - **useState([]):** Almacena las categorías.
    - **useEffect:** Llama a la API para obtener las categorías cuando el componente se monta.
2. **Obtención de Categorías:**
    - La función getCategories se integró directamente en el useEffect para mayor simplicidad.
3. **Renderización:**
    - HomeHeading muestra el título.
    - CategoriesList muestra la lista de categorías.

Listado de Recursos según categoría

PublicationsByCategory permite a los usuarios visualizar un listado de publicaciones filtradas según la categoría seleccionada. Esto facilita que los usuarios accedan a contenido relevante y específico dentro de la categoría de su interés, mejorando la experiencia de navegación y personalización del contenido.

Funcionalidades Principales

1. **Filtrado de Publicaciones por Categoría**:
    - El módulo recibe como parámetro la categoría seleccionada y, basado en esta, realiza una llamada a la API para obtener las publicaciones más recientes correspondientes a dicha categoría.
2. **Carga Dinámica**:
    - Las publicaciones se cargan dinámicamente cada vez que se cambia de categoría, asegurando que el contenido sea siempre actualizado y relevante.

```
export const PublicationsByCategory = () => {

  const navigation = useNavigation()
  const param = useRoute().params as { category: string }

  const [lastestPublications, setlastestPublications] = useState([])
  useEffect(() => {
    console.log('Category',param.category);

    param&&getPublicByCategory()
  },[param])

  const getPublicByCategory=( )=>{
    hyGraphApi.getPublicByCategory(param.category)
    .then(resp=>{
      console.log(resp.newLastestPublics);
      setlastestPublications(resp?.newLastestPublics)
    })
  }
```

Nota: Para más detalles diríjase al repositorio en Github: https://bit.ly/3XmNC0W

Explicación del Código

1. **Obtención de la Categoría**:
    - El módulo utiliza useRoute para obtener la categoría seleccionada de los parámetros de la navegación, almacenándola en category.
2. **Estado del Componente**:
    - useState([]): Utilizado para almacenar las publicaciones más recientes filtradas por la categoría seleccionada.
3. **Efecto de Carga Dinámica**:
    - useEffect: Ejecuta la función getPublicByCategory cada vez que cambia la categoría, asegurando que se muestren las publicaciones correspondientes a la nueva selección.
4. **Obtención de Publicaciones**:

- getPublicByCategory: Llama a la API para obtener las publicaciones de la categoría seleccionada y actualiza el estado con los resultados.

Módulo 4: Actualización de la Sección de Recursos

NewLastestPublic se encarga de mostrar las publicaciones más recientes en la aplicación. Este módulo es esencial para mantener a los usuarios informados con el contenido más actualizado, asegurando que siempre tengan acceso a los recursos más relevantes y recientes disponibles.

Funcionalidades Principales

1. **Obtención de Publicaciones Recientes**:
   - El módulo realiza una llamada a la API para obtener las últimas publicaciones y las presenta al usuario en una lista, asegurando que el contenido mostrado esté siempre actualizado.

2. **Actualización Automática**:
   - La función de obtención de nuevas publicaciones se ejecuta automáticamente al cargar el componente, garantizando que la información presentada sea la más reciente.

```
export const NewLastestPublic = () => {
    const [lastestPublic, setLastestPublic] = useState([]);

    useEffect(() => {
        getNewLastestPublic();
    }, []);

    const getNewLastestPublic = async () => {
        const resp = await hyGraphApi.getNewLastestPublic();
        setLastestPublic(resp?.newLastestPublics);
    };

    return (
        <View style={{ marginTop: 10 }}>
            <HomeHeading text="Últimas Publicaciones" />
            <LastestPublicList lastestPublic={lastestPublic} />
        </View>
    );
};
```

Nota: Para más detalles diríjase al repositorio en Github: https://bit.ly/3XmNC0W

1. **Obtención de las Últimas Publicaciones**:
    - getNewLastestPublic: Función asíncrona que realiza una llamada a la API para obtener las publicaciones más recientes. Los resultados se almacenan en el estado lastestPublic.
2. **Estado del Componente**:
    - useState([]): Almacena las publicaciones más recientes obtenidas de la API.
3. **Efecto de Carga Automática**:
    - useEffect: Se asegura de que getNewLastestPublic se ejecute al montar el componente, de modo que el contenido mostrado esté siempre actualizado.
4. **Presentación del Contenido**:

- HomeHeading: Muestra el título "Últimas Publicaciones".
- LastestPublicList: Renderiza la lista de publicaciones recientes almacenadas en lastestPublic.

Módulo 5: Botón de Emergencia

El módulo de EmergencyScreen está diseñado para proporcionar asistencia inmediata a los usuarios mediante una llamada de emergencia vía Zoom. Este módulo obtiene un enlace de Zoom de una API y lo presenta al usuario cuando se activa la pantalla de emergencia. Es una función crítica que permite a los usuarios acceder a ayuda en situaciones urgentes.

Funcionalidades Principales

1. **Obtención Dinámica del Enlace de Zoom**:
    - Al acceder a la pantalla de emergencia, la aplicación realiza una llamada a la API para obtener un enlace de Zoom. Este enlace se utiliza para conectar al usuario con un asistente en vivo.
2. **Interfaz de Usuario Adaptativa**:
    - La pantalla se adapta al tamaño del dispositivo, asegurando que el enlace de emergencia se muestre correctamente independientemente de las dimensiones de la pantalla.

```
export const EmergencyScreen = () => {
    const { height } = useWindowDimensions();
    const [link, setLink] = useState<string>('');

    useEffect(() => {
        getLink();
    }, []);

    const getLink = async () => {
        const resp = await hyGraphApi.getLink();
        console.log("LINK", resp);
        setLink(resp);
    };
};
```

Nota: Para más detalles diríjase al repositorio en Github: https://bit.ly/3MoBXsk

1. **Obtención del Enlace de Zoom**:
    - getLink: Esta función realiza una llamada a la API para obtener el enlace de Zoom, que luego se almacena en el estado link para ser utilizado en la interfaz de usuario.
2. **Estado del Componente**:
    - useState<string>(''): Almacena el enlace de Zoom obtenido de la API.
3. **Carga Automática del Enlace**:
    - useEffect: Llama a getLink cuando el componente se monta para asegurar que el enlace esté disponible tan pronto como el usuario acceda a la pantalla.
4. **Adaptación de la Interfaz**:
    - useWindowDimensions(): Se utiliza para obtener las dimensiones de la pantalla y poder adaptar la interfaz según el tamaño del dispositivo.

Módulo 6: Asistente Virtual

El módulo de Chat con Asistencia Virtual ofrece a los usuarios la posibilidad de interactuar con un asistente virtual llamado Hannah. Este asistente está diseñado para proporcionar respuestas rápidas y útiles, basadas en la inteligencia artificial, directamente en la aplicación. Es una herramienta esencial para brindar soporte continuo y personalizado a los usuarios.

Funcionalidades Principales

1. **Interacción en Tiempo Real**:
    - Los usuarios pueden enviar y recibir mensajes en tiempo real, interactuando directamente con el asistente virtual para obtener la ayuda que necesitan.
2. **Asistente Virtual Inteligente**:
    - El asistente, basado en IA, está programado para responder a una variedad de consultas relacionadas con primeros auxilios psicológicos, proporcionando soporte inmediato y accesible.
3. **Indicadores de Carga**:
    - Mientras el asistente procesa una solicitud, un indicador de carga se muestra en pantalla, informando al usuario que su mensaje está siendo gestionado.

```jsx
export const ChatScreen = () => {
    const { messages, sendMessage, loading } = useChat();

    return (
        <View style={styles.container}>
            <View style={{ backgroundColor: '#005CAA', height: 120, borderRadius: 20 }}>
                <Tab
                    title='Hannah - Asistente Virtual'
                    icon={<MyIcon name='person' white />}
                    style={{ marginTop: 55, gap: 5 }}
                />
            </View>

            <GiftedChat
                messages={messages}
                onSend={newMessages => sendMessage(newMessages)}
                user={{ _id: 1 }}
                inverted={true}
            />
            {loading && <LoadingIndicator />}
        </View>
    );
};
```

Nota: Para más detalles diríjase al repositorio en Github: https://bit.ly/3WwTUJT

1. **Interfaz de Chat**:
    - GiftedChat: Componente principal que maneja la interfaz de chat, mostrando los mensajes y permitiendo a los usuarios enviar nuevos mensajes.
2. **Estado y Manejo de Mensajes**:
    - useChat: Hook personalizado que gestiona la lógica del chat, incluyendo el envío y la recepción de mensajes, así como el estado de carga.
3. **Encabezado del Chat**:
    - Tab: Muestra el nombre del asistente virtual "Hannah" junto con un ícono personalizado, proporcionando una interfaz amigable y reconocible.
4. **Indicador de Carga**:
    - LoadingIndicator: Muestra un indicador de carga cuando el asistente virtual está procesando una solicitud, asegurando que el usuario esté al tanto del progreso.

Modulo 7: Test de Personalidad

El módulo de Tests de Personalidad permite a los usuarios realizar evaluaciones de personalidad directamente desde la aplicación. Utilizando un formulario externo, el test está diseñado para proporcionar a los usuarios una mejor comprensión de su estado emocional y mental, facilitando así un apoyo psicológico más personalizado y efectivo.

Funcionalidades Principales

1. **Inicio del Test**:
    - Los usuarios pueden iniciar el test de personalidad mediante un botón que despliega una vista web dentro de la aplicación.
2. **Visualización del Test**:
    - El test se carga en un componente WebView, lo que permite a los usuarios completar el formulario sin salir de la aplicación.
3. **Cierre del Test**:
    - Los usuarios pueden cerrar el WebView y regresar a la pantalla anterior, manteniendo la experiencia de usuario fluida y sin interrupciones.

```
export const Tests = () => {
    const [showWebView, setShowWebView] = useState(false);
    const navigation = useNavigation();

    const handleStartTest = () => {
        setShowWebView(true);
    };

    const handleCloseWebView = () => {
        setShowWebView(false);
    };

    return (
        <View style={{ flex: 1, marginTop: 50, backgroundColor: '#FFFFFF' }}>
            {showWebView && (
                <View style={{ flex: 1 }}>
                    <WebView
                        source={{ uri: process.env.REACT_NATIVE_TEST_URI }}
                        style={styles.webview}
                    />
                    <View style={styles.closeButtonContainer}>
                        <Button title="Cerrar" onPress={handleCloseWebView} />
                    </View>
                </View>
            )}
        </View>
    );
};
```

Nota: Para más detalles diríjase al repositorio en Github: https://bit.ly/3WwTUJT

1. **Visualización del WebView**:
    - WebView: Componente que carga el formulario del test de personalidad utilizando una URL externa. La URL es configurada mediante una variable de entorno para asegurar flexibilidad y seguridad en la gestión de rutas externas.
2. **Inicio y Cierre del Test**:
    - handleStartTest: Función que muestra el WebView al usuario.
    - handleCloseWebView: Función que cierra el WebView, permitiendo al usuario volver a la pantalla principal.
3. **Gestión de Estado**:
    - useState(false): Maneja la visibilidad del WebView para iniciar y finalizar el test de personalidad.
4. **Variable de Entorno**:
    - process.env.REACT_NATIVE_TEST_URI: Esta variable de entorno almacena la URL del formulario del test, permitiendo cambios de la ruta sin necesidad de modificar el código fuente.

Módulo 8: Perfil de Usuario

El módulo de Perfil de Usuario permite a los usuarios visualizar y modificar su información personal dentro de la aplicación. Esta funcionalidad es esencial para mantener una experiencia personalizada y consistente, facilitando la gestión de la identidad del usuario y asegurando que la aplicación se ajuste a sus necesidades individuales.

Funcionalidades Principales

1. **Visualización del Perfil**:
    - Los usuarios pueden ver sus datos personales, como el nombre completo, y otra información relevante dentro de su perfil.
2. **Edición de la Imagen de Perfil**:
    - Permite a los usuarios seleccionar una nueva imagen de perfil desde su galería de fotos, que se puede ajustar antes de guardarse.
3. **Cierre de Sesión**:
    - Los usuarios tienen la opción de cerrar sesión de manera segura, gestionada a través de Clerk.

```
export const ProfileScreen = () => {
    const { signOut } = useClerk();
    const { user } = useUser();

    const pickImage = async () => {
        const result = await ImagePicker.launchImageLibraryAsync({
            mediaTypes: ImagePicker.MediaTypeOptions.Images,
            allowsEditing: true,
            quality: 0.75,
            base64: true,
        });

        if (!result.canceled) {
            const base64 = `${result.assets[0].base64}`;
            user?.setProfileImage({
                file: base64,
            });
        }
    };
```

Nota: Para más detalles diríjase al repositorio en Github: https://bit.ly/3AFm12j

1. **Selección y Actualización de la Imagen de Perfil**:
    - ImagePicker.launchImageLibraryAsync: Abre la galería de fotos del dispositivo, permitiendo al usuario seleccionar una imagen.
    - user?.setProfileImage: Actualiza la imagen de perfil del usuario con la imagen seleccionada, codificada en base64 para asegurar la compatibilidad.
2. **Gestión de Sesión**:
    - signOut: Función que permite al usuario cerrar sesión de manera segura.