Documentación de la Aplicación

Aquí explicamos de manera detallada como funciona el código y como se encuentra estructurada nuestra aplicación.

HomeView.js

Este es el componente principal, la primera pantalla que el usuario ve al abrir la app. En ella el usuario puede buscar lugares turísticos de diferentes ciudades y categorías.

Estado Local y Redux: Usamos el estado local para gestionar la ciudad seleccionada, la categoría y la búsqueda. Los lugares se obtienen desde Redux, que mantiene el estado global de la app.

Selector de Ciudad y Categoría: Aquí el usuario puede elegir qué ciudad y qué tipo de lugar está buscando. Cuando elige una opción, la app hace una nueva búsqueda de lugares y actualiza la lista.

Búsqueda: Hay un campo de texto donde el usuario puede escribir el nombre de un lugar para filtrar la lista. Si el nombre del lugar coincide con la búsqueda, se muestra en la lista.

Carga de Lugares: Cuando se selecciona una nueva ciudad o categoría, la app hace una llamada a la API para obtener los lugares. Mientras espera la respuesta, muestra un indicador de carga.

Lista de Lugares: La lista de lugares obtenidos de la API se muestra en una lista. Cada lugar se puede seleccionar para ver más detalles.

PlaceDetailView.js

Este componente se encarga de mostrar los detalles completos de un lugar cuando el usuario hace clic en un lugar de la lista.

Recibe el lugar: Usamos route.params para recibir el lugar seleccionado desde la pantalla anterior. Luego, este lugar es pasado al componente PlaceDetail para mostrar más información.

PlaceDetail: Aquí se muestran los detalles completos del lugar, como su nombre, dirección, teléfono y sitio web. Además, se incluyen los horarios de apertura si están disponibles.

FavoriteButton: En esta vista también aparece un botón que permite agregar o quitar el lugar de los favoritos del usuario.

PlaceItem.js

Este es el componente que representa un lugar individual en la lista.

- **Navegación**: Cuando el usuario toca un lugar, lo lleva a la vista de detalles de ese lugar, usando navigation.navigate para pasarle el objeto completo del lugar.
- **Estilos**: Cada lugar tiene un diseño con el nombre y la dirección del lugar.

FavoriteButton.js

Este botón permite a los usuarios agregar o eliminar lugares de sus favoritos.

- **Gestión de Favoritos**: Se revisa si el lugar ya está en los favoritos del usuario. Si lo está, se ofrece la opción de quitarlo; si no, el botón permite agregarlo.
- **Acción y Confirmación**: Después de cada acción, el usuario recibe una notificación en forma de alerta, informándole si el lugar fue agregado o eliminado correctamente.

WebsiteButton.js

Este componente muestra un botón que permite a los usuarios visitar el sitio web relacionado con un lugar.

Función del Enlace: Si el sitio web está disponible, se abre cuando el usuario toca el botón, llevándolo al navegador del dispositivo.

LoadingIndicator.js

Es un simple componente que muestra un indicador de carga mientras se obtienen los datos de la API.

Redux - Gestión del Estado Global

PlacesSlice.js

Este archivo es donde manejamos el estado de los lugares y los favoritos dentro de Redux.

- **Acciones**:
    - **setPlaces**: Se usa para actualizar la lista de lugares.
    - **addFavorite:** Agrega un lugar a los favoritos del usuario.
    - **removeFavorite:** Elimina un lugar de los favoritos.
- **Cargar Favoritos**: Con createAsyncThunk, cargamos los favoritos desde el almacenamiento local cuando la app se inicia, usando AsyncStorage para persistir los datos.

Store.js

Aquí configuramos el store de Redux, que guarda el reducer places.

- **Cargar Favoritos**: Al iniciar la app, despachamos la acción loadFavorites() para obtener la lista de favoritos guardada previamente.

Servicios

api.js

Este archivo contiene las funciones para interactuar con la API y obtener los lugares.

- **getPlaces**: Obtiene una lista de lugares filtrados por ciudad, categoría y palabra clave. Solo muestra los primeros 30 resultados.
- **getPlaceDetails**: Obtiene los detalles completos de un lugar usando su ID. Esta información incluye nombre, dirección, teléfono, sitio web y más.

Almacenamiento Local

LocalStorage.js

Aquí gestionamos todo lo relacionado con el almacenamiento de los favoritos usando AsyncStorage.

- **saveFavorites**: Guarda la lista de favoritos en AsyncStorage para que persistan entre sesiones.
- **loadFavorites**: Recupera los favoritos guardados de AsyncStorage cuando el usuario vuelva a la app.

Vistas

FavoritesView.js

En esta pantalla se muestran todos los lugares que el usuario ha marcado como favoritos.

- **Lista de Favoritos**: Si el usuario tiene favoritos guardados, se muestran en una lista. Si no hay favoritos, se muestra un mensaje diciendo que aún no hay favoritos.
- **Navegación a Detalles**: Al tocar un lugar, se lleva al usuario a la pantalla de detalles de ese lugar.

HomeView.js

- **Búsqueda**: Permite buscar lugares por nombre, ciudad y categoría.
- **Filtros**: El usuario puede elegir la ciudad y la categoría de los lugares que desea ver, y la lista se actualiza con los nuevos resultados.

- **Carga de Lugares**: La app hace una solicitud a la API cada vez que se cambia la ciudad o la categoría, actualizando la lista de lugares.
- **Interfaz**: Los lugares se muestran en una lista filtrada, permitiendo al usuario buscar, ver más detalles y explorar diferentes opciones.

Flujo General de la Aplicación

1. **HomeView**: En esta pantalla, el usuario puede buscar lugares turísticos y filtrarlos por ciudad o categoría. Al tocar un lugar, se accede a la vista de detalles.
2. **PlaceDetailView**: Aquí el usuario ve toda la información de un lugar, junto con un botón para agregarlo o quitarlo de sus favoritos.
3. **FavoritesView**: Muestra los lugares que el usuario ha guardado en favoritos. Al tocar un favorito, se lleva al usuario a la vista de detalles del lugar

IA UTILIZADA:

ChatGPT: A nivel de código y facilidad de creación de métodos y evitar la rebundancia, GPT facilitó mucho la tarea de tener que crear código similar y así ahorrarnos tiempo en la creación de diversos documentos.

GROK: Utilizamos grok para aumentar las ideas, con grok hicimos preguntas de como mejorar la estructura de la aplicación y poder crearla de la mejor manera posible, así siendo atractiva para usuarios y responsive por completo.