GRADO EN INGENIERÍA INFORMÁTICA

PROGRAMACIÓN DE DISPOSITIVOS MÓVILES

UNIVERSIDAD
DE GRANADA

Práctica 1 : Aplicación Android: ToDoList por ubicación

David Serrano Domínguez (*davidserrano07@correo.ugr.es*)

6 de abril de 2025

Índice

1 Descripción de las funcionalidades de la aplicación **3**

2 Tecnologias Usadas para la implementación **3**

3 Pantallas principales de la aplicación e implementación **4**

3.1 Implementación de modal de ubicación 4

3.2 Implementación de Hook de ubicación 6

3.3 Almacenamiento utilizado para las tareas 9

3.4 Implementación de las notificaciones 10

3.5 Listado de tareas completadas 11

4 Conclusiones **12**

Objetivos

La práctica consiste en desarrollar una aplicación móvil que funcione completamente de forma local, sin necesidad de conexión a un servidor externo. Para esta primera entrega, se acordó realizar una aplicación del tipo ToDo List, pero con un enfoque más avanzado de lo habitual. En este caso, la aplicación utiliza la ubicación del usuario para mostrar las tareas urgentes correspondientes a su zona actual. No obstante, también se ofrece la opción de visualizar todas las tareas urgentes, independientemente de la ubicación.

ENLACE AL REPOSITORIO DE GITHUB : Repositorio de las prácticas

1. Descripción de las funcionalidades de la aplicación

El primer paso a la hora de realizar esta práctica fue establecer las funcionalidades básicas que quería que se tuvieran, es decir, lo que se llama el **MVP(Producto Mínimo Viable)**, en este caso se decidió que para darle una funcionalidad extra a las típicas listas de tareas que todos ya conocemos, se decidió que esta mostrara siempre las tareas de una zona que se le podría asignar a las mismas a la hora de crearlas, de manera que el usuario a la hora de revisar las tareas que tuviera pendientes solo pudiera ver las más urgentes de su ubicación actual, de manera que de un vistazo al entrar de manera rápida y sencilla supiera qué debía hacer en ese sitio y momento actual.

Una vez se implementó esto, se añadió de manera extra un **switch** que diera la posibilidad al usuario de ver todas sus tareas pendientes, aunque estas no fueran de la ubicación actual en la que se encontrase.

En cuanto a la gestión de tareas, la aplicación permite realizar las acciones básicas: **crear, marcar como completadas y eliminar tareas**. Se optó por no incluir la edición de tareas, ya que, dados los objetivos de esta aplicación, no se consideró una funcionalidad prioritaria. En su lugar, se decidió invertir ese tiempo en el desarrollo y optimización del funcionamiento relacionado con la **ubicación de dichas tareas**, además de **añadir notificaciones** que avisarán al usuario de la **cantidad de tareas que debe realizar en la ubicación actual** además de que cuando quede media hora o menos para la siguiente tarea de su ubicación.

Características finales de la aplicación:

- **Crear tareas de prioridad urgente o normal**
- **Marcar tareas como completadas**
- **Eliminar tareas**
- **Crear tareas con o sin ubicación**
- **Lista de tareas completadas tanto urgentes como normales**
- **Mostrar tareas urgentes de la ubicación actual**
- **Mostrar todas las tareas urgente independientemente de la ubicación**
- **Mostrar siempre todas las tareas normales siempre**
- **Notificar de la cantidad de tareas urgentes de la ubicación actual en primer plano**
- **Notificar media hora antes de la siguiente tarea de la ubicación actual en primer plano**

2. Tecnologias Usadas para la implementación

A continucación se explicara las tecnologías escogidas para la realización de la aplicación móvil y se explicará el motivo de la eleccion de las mismas.

Para el desarrollo de la app se decidio usar la libreria de **Javascript** llamada **React**, concretamente **React Native** que es una versión de dicha libreria enfocada al **desarrollo móvil** y que permite poder compilar las aplicaciones para ejeuctarlas en **cualquier sistema operativo**.

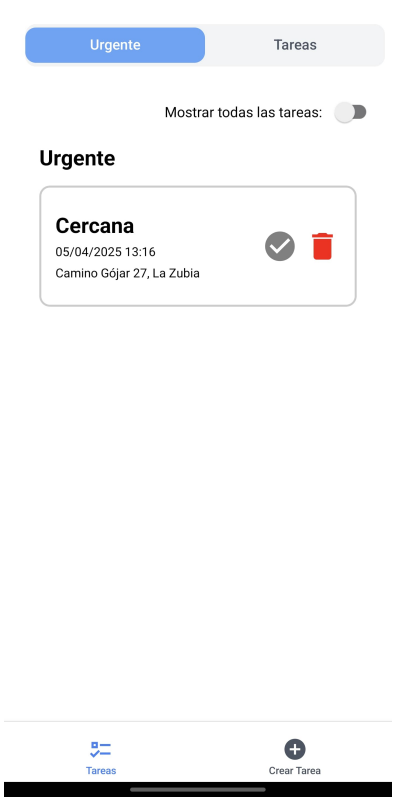
Además tras leer la **documentación oficial** ellos mismos recomiendan usar el framework de **Expo** para crear aplicaciones móviles de **React Native** de manera más sencilla y rápida, por tanto se decidio seguir dicha recomendación y usar dicho **framework** para el desarrollo.

A continuación se deja la documentación de ambas tecnologías:

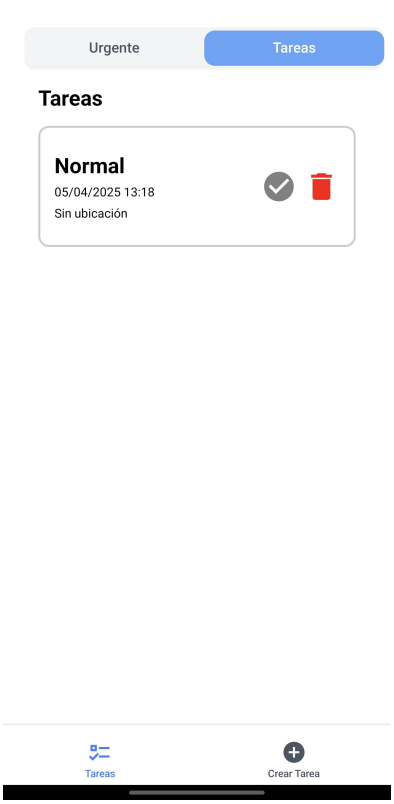
- **Documentación de React Native**
- **Documentación de Expo**

3. Pantallas principales de la aplicación e implementación

La aplicación consta principalmente de tres pantallas: **Tareas urgentes**, **Tareas normales** y **Crear nueva tarea**.



Pantalla de tareas urgentes



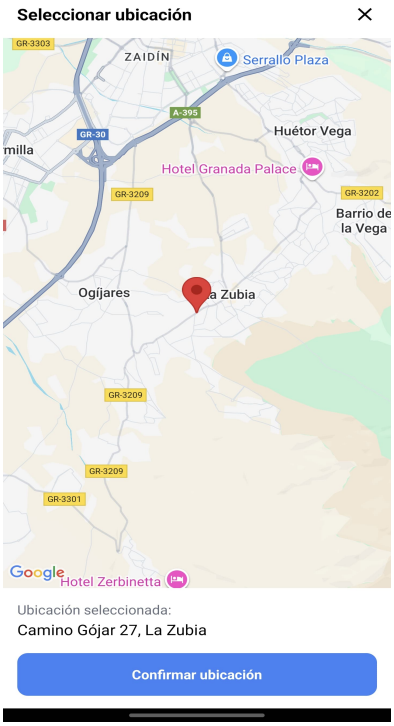
Pantalla de tareas normales

Pantall de crear nueva tarea

Estas serían las tres principales páginas de nuestra aplicación, en el caso de las páginas que listan las tareas se cambiara entre ellas mediante un deslizamiento horizontal que permitiria al usuario de manera rápida y sencilla poder ver sus tareas de **prioridad urgente o normal**.

3.1. Implementación de modal de ubicación

Además de las tres páginas anteriores nos encontramos con una **ventana modal** que se abriria al darle a **añadir ubicación** y que nos mostraria un **mapa** donde la posición inicial del mismo sería nuestra ubicación actual.



Modal de añadir ubicación

Mediante esta modal podriamos coger como ubicación de la tarea **nuestra posición actual** o en caso de deseear marcar la tarea en otro sitio podriamos buscar dicha posición en el **mapa y seleccionarla de manera fácil**, de manera que la tarea quedaría registrada para que nos saliera cuando nos encontraramos en dicha ubicación.
Una vez **seleccionada la ubicación que deseamos** que se le asigne a la tarea simplemente deberiamos darle a **confirmar ubicación** y quedaría registrada de manera **fácil y rápida**. Una vez explicado su funcionamiento vamos a pasar a explicar como se ha implementado esta lógica, y es que como hemos destacado anteriormente el **framework de Expo** esta preparado para crear aplicaciones móviles usando **React Native** por tanto este tiene un paquete preparado que al instalarlo podemos implementar toda esta lógica de manera rápida y sencilla.
Este paquete sería **react-native-maps** el cual incluye un componente llamado **MapView** que permite poder mostrar el mapa de manera sencilla como se hace en la aplicación y se muestra a continuación:

```
import MapView, { Marker, MapPressEvent } from "react-native-maps";
    <MapView
      style={styles.map}
      initialRegion={{
        latitude: location.coords.latitude,
        longitude: location.coords.longitude,
        latitudeDelta: 0.0922,
        longitudeDelta: 0.0421,
      }}
      onPress={handleSelectLocation}
    >
      {selectedLocation && (
        <Marker
          coordinate={{
            latitude: selectedLocation.latitude,
            longitude: selectedLocation.longitude,
          }}
        />
      )}
    </MapView>
```

Como podemos observar mediante el uso del componente anteriormente dicho, el de marker y el evento de

MapPresEvent podemos de manera fácil y rápida almacenar la que el usuario seleccione, además que mediante la **prop** que recibe el componente de **MapView** podemos mandar también de manera sencilla la posición inicial del mapa.
A continuación se muestra la función implementada para almacenar la **ubicación seleccionada** por el usuario:

```
  const handleSelectLocation = async (event: MapPressEvent) => {
    const coords = event.nativeEvent.coordinate;
    setSelectedLocation({
      latitude: coords.latitude,
      longitude: coords.longitude,
      altitude: 0,
      accuracy: 0,
      altitudeAccuracy: 0,
      heading: 0,
      speed: 0,
    });
```

Como podemos ver recibimos un **evento** del tipo que hemos importado mediante el paquete de **react-native-maps** el cual contiene la información que deseamos.
Extraemos dicha información y lo guardamos en un estado que tenemos creado para guardar la ubicación.
Para guardar la ubicación lo haremos usando objetos del tipo **location** perteneciente al paquete de **expo-location** de **Expo** y que se importaria de la siguiente manera:

```
import * as Location from "expo-location"

const [selectedLocation,setSelectedLocation] = useState<LocationObjectCoords | null>(null);
```

3.2. Implementación de Hook de ubicación

Mediante el uso del paquete explicado anteriormente para almacenar la ubicación seleccionada en el mapa se ha conseguido implementar un **hook** que permite de **manera global en toda la aplicación obtener la ubicación actual del usuario** en cualquier momento, a continuación se muestra su implementación:

```
import * as Location from "expo-location";
import { useState, useEffect, useCallback } from "react";
import AsyncStorage from "@react-native-async-storage/async-storage";

type LocationError = {
  message: string;
  code?: string;
};

export default function useLocation() {
  const [location, setLocation] = useState<Location.LocationObject | null>(
    null
  );
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState<LocationError | null>(null);

  const loadStoredLocation = useCallback(async () => {
    try {
      setIsLoading(true);
      const storedLocation = await AsyncStorage.getItem("location");
      if (storedLocation) {
        setLocation(JSON.parse(storedLocation));
        return true;
      }
      return false;
    } catch (error) {
```

```
      setError({ message: "Error loading stored location" });
      return false;
    } finally {
      setIsLoading(false);
    }
  }, []);

  const obtenerUbicacion = useCallback(async () => {
    try {
      setIsLoading(true);
      setError(null);

      const { status } = await Location.requestForegroundPermissionsAsync();
      if (status !== "granted") {
        setError({
          message: "Location permission denied",
          code: "PERMISSION_DENIED",
        });
        return;
      }

      const currentLocation = await Location.getCurrentPositionAsync({
        accuracy: Location.Accuracy.High,
      });

      setLocation(currentLocation);
      await AsyncStorage.setItem("location", JSON.stringify(currentLocation));
    } catch (error) {
      setError({
        message: "Error getting location",
        code: error instanceof Error ? error.message : "UNKNOWN_ERROR",
      });
    } finally {
      setIsLoading(false);
    }
  }, []);

  useEffect(() => {
    let isMounted = true;

    const initializeLocation = async () => {

      if (!isMounted) return;

      setIsLoading(true);

      try {

        const hasStoredLocation = await loadStoredLocation();



        if (!hasStoredLocation && isMounted) {
          await obtenerUbicacion();
        }
      } catch (error) {
        if (isMounted) {
          setError({
            message: "Error initializing location",
            code: "INIT_ERROR",
          });
        }
      } finally {
        if (isMounted) {
          setIsLoading(false);
        }
      }
    };

```

```
96      initializeLocation();
97
98
99      return () => {
100       isMounted = false;
101     };
102   }, [loadStoredLocation, obtenerUbicacion]);
103
104   return {
105     location,
106     obtenerUbicacion,
107     isLoading,
108     error,
109     setIsLoading,
110   };
111 }
```

Como podemos observar mediante el uso de este paquete no solo obtenemos la ubicación si no que también mediante el uso de la función **requestForegroundPermissionAsync** pedimos permisos al usuario para que este ceda su posición geográfica.
Además a este paquete se le añade el paquete de **React Native** que nos permite almacenar la ubicación del usuario para no tener que estar calculandola mientras nos encontremos en la misma ubicación, este paquete sería el **import AsyncStorage from "@react-native-async-storage/async-storage"** el cual nos permite tener almacenamiento local en el teléfono.

3.3. Almacenamiento utilizado para las tareas

De manera similar se ha hecho uso del paquete de **AsyncStorage** para almacenar las distintas tareas creadas en el teléfono y tenerlas siempre disponibles de manera local.

```
import AsyncStorage from "@react-native-async-storage/async-storage";
import DateTimePicker, {
  DateTimePickerEvent,
} from "@react-native-community/datetimepicker";
import * as Crypto from "expo-crypto";
import { useState } from "react";

  const params = useLocalSearchParams();
  const [title, setTitle] = useState("");
  const [description, setDescription] = useState("");
  const [date, setDate] = useState(new Date());
  const [showDatePicker, setShowDatePicker] = useState(false);
  const [showTimePicker, setShowTimePicker] = useState(false);
  const [location, setLocation] = useState<Location | null>(
    params.location ? JSON.parse(params.location as string) : null
  );
  const [priority, setPriority] = useState("Normal");
  const [duration, setDuration] = useState("");
  const handleSubmit = async () => {
    try {
      const newTask: Task = {
        id: Crypto.randomUUID(),
        title,
        description,
        duration: parseInt(duration),
        location: location || {
          coords: {
            latitude: 0,
            longitude: 0,
          },
          address: "",
        },
        priority,
        completed: false,
        date: date,
      };


      const existingTasks = await AsyncStorage.getItem("tasks");
      const tasks = existingTasks ? JSON.parse(existingTasks) : [];

      const updatedTasks = [...tasks, newTask];


      await AsyncStorage.setItem("tasks", JSON.stringify(updatedTasks));


      setTitle("");
      setDescription("");
      setLocation(null);
      setPriority("Normal");


      router.back();
    } catch (error) {
      console.error("Error saving task:", error);
    }
  };
```

Como podemos observar la lógica seguida para almacenar las tareas sería:

- **Crear un objeto tarea con los valores que almacenamos en los estados creados con anterioridad**
- **Obtener el array de objetos de tarea ya existente en el almacenamiento**
- **Añadir a dicho array la nueva tarea que hemos creado**
- **Volver a insertar el nuevo array de tareas en el AsynStorage**
- **Establecer valores de los estados por defecto de nueva para la creación de la siguiente tarea**

Además de manera preventiva todo esta envuelto en un **try, catch** para que en caso de fallo la aplicación siga funcionando y no se rompa

3.4. Implementación de las notificaciones

Lo siguiente que explicaremos será como se ha llevado a cabo la **implementación de las notificaciones**, en este caso solo en **primer plano** ya que por falta de tiempo no se ha podido implementar para que se ejecuten en **segundo plano** aunque hubiera sido lo óptimo y lo que hubiera tenido más sentido.

Para mandar las notificaciones se ha instalado el paquete de **expo-notifications** el cual permite de manera fácil y rápida mandar una **notificación push** al usuario.

```
import * as Notifications from "expo-notifications";

  if (nearbyTasks.length > 0) {
    try {
      await Notifications.scheduleNotificationAsync({
        content: {
          title: " Tareas  cercanas!",
          body: `Tienes ${nearbyTasks.length} ${
            nearbyTasks.length === 1
              ? "tarea cercana"
              : "tareas cercanas"
          } que requieren tu atenci n `,
        },
        trigger: null,
      });
      const doItNow = nearbyTasks.filter((task: Task) => {
        if (task.duration > 0) {

          const endTime = new Date(task.date).getTime();

          const duration = task.duration || 0;


          const startTime = endTime - duration * 60 * 1000;


          const timeToStart = (startTime - Date.now()) / (60 * 1000);

          console.log(`Tarea: ${task.title}`);
          console.log(
            `Tiempo hasta inicio necesario: ${timeToStart.toFixed(
              2
            )} minutos`
          );
          console.log(`Duraci n de la tarea: ${duration} minutos`);

```

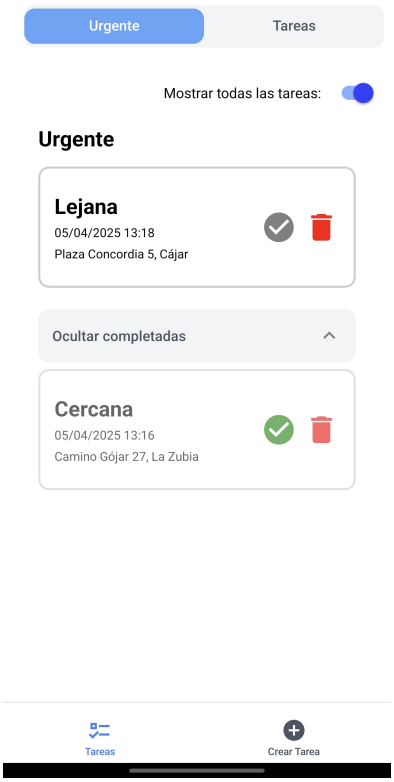
```
37
38            return timeToStart <= 30 && timeToStart >= 0;
39          }
40          return false;
41        });
42
43        doItNow.map(async (task: Task) => {
44          const endTime = new Date(task.date).getTime();
45          const duration = task.duration || 0;
46          const startTime = endTime - duration * 60 * 1000;
47          const timeToStart = (startTime - Date.now()) / (60 * 1000);
48
49          await Notifications.scheduleNotificationAsync({
50            content: {
51              title: ` Necesitas  empezar ${task.title}!`,
52              body: `Tienes ${timeToStart.toFixed(
53                0
54              )} minutos para empezar esta tarea que dura ${duration} minutos`,
55            },
56            trigger: null,
57          });
58        });
59      } catch (notificationError) {
60        console.error(
61          "Error enviando notificaciones:",
62          notificationError
63        );
64      }
65    }
```

En este caso la lógica mostrada anteriormente se ejecutaría siempre que estemos en el apartado de **tareas urgentes** y no tengamos activada la opción de mostrar todas las tareas, por tanto si se cumple ambas condiciones se ejecutaría la función que contiene la lógica anterior.

Primero **comprobariamos si tenemos algun tarea cercana** ya que en caso de que no entonces **no tendría sentido notificar** de nada al usuario, una vez realizada dicha comprobación obtenemos **la cantidad de tareas cercanas** y mandamos al usuario una notificación avisandole de cuantas tareas debe **realizar en su ubicación actual**.
Una vez mandada la notificación anterior pasariamos a comprobar si alguna de dichas tareas queda menos de **30 minutos** para su comienzo y en caso de que si por cada una de ellas mandamos una notificación al usuario diciendole que debería empezarla ya.

3.5. Listado de tareas completadas

Una vez vistas las principales páginas de la aplicación y el funcionamiento de la modal vamos a proceder a ver como se podrian ver mis **tareas ya completadas**.



Lista de tareas ya completadas

Como podemos ver se ha decidido **añadir una funcionalidad extra** que permite ver las **tareas ya completadas** independientemente de la **ubicación del usuario** asi de manera **fácil y sencill**a puede saber que tareas ya ha completado y cuales le quedan por completar **en un solo vistazo**.

4. Conclusiones

Para concluir considero que la aplicación obtenida es bastante práctica y vistoza para considerar que estuviera perfecta faltaría añadirle el trabajo en segundo plano de obtener la ubicación del usuario y en caso de pasar por una ubicación en la que existiera alguna tarea a realizar mandar un notificación al usuario de que debe realizar x tareas en dicha ubicación además de añadir también una notificación que avise al usuario cuando quede menos de 30 minutos para empezar una tarea que tuviera programado para x hora.