PROYECTO AGENDA DE CONTACTOS

Samuel Madrazo y Mia García

- **Introducción:**
  Vamos a crear una aplicación móvil que permite mantener una agenda de contactos. En esta agenda podremos añadir, eliminar, editar y mostrar contacto, a través de diferentes pantallas.

- **Objetivos:**
  - Crearemos un sistema de navegación para poder movernos libremente por las pantallas y gestionaremos los posibles errores que puedan surgir de ello.
  - Nuestra app mantendrá persistencia de datos, para evitar que se borren entre sesiones.
  - Buscaremos separar la lógica de datos de la lógica gráfica de como mostrar dichos datos con una arquitectura MVVM..
  - Exploraremos la posibilidad de cargar ciertos datos de internet, en concreto de una API.
  - Intentaremos gestionar la conectividad y sus permisos.
  - Diseñaremos una estructura de control de errores para que, si algo falla, haya un “plan B” y la app no colapse.
  - Compartiremos nuestra app con todos los posibles usuarios, de la manera más accesible posible.

- Desarrollo y decisiones clave de diseño/desarrollo:
  - CRUD:
    A lo largo del programa crearemos métodos de añadir,actualizar, eliminar y mostrar contactos. Serán creados en distintas capas y archivos, de forma que se estarán enlazando entre sí y cumpliendo la función necesaria en cada capa.

  - Arquitectura MVVM
    Usaremos la arquitectura MVVM para separar la vista gráfica y la interacción del usuario con cómo se procesan los datos internamente. Para ello usaremos la librería Hilt, una librería de inyecciones que facilita la aplicación de dicha arquitectura. También contaremos con un viewModel donde se almacenan las funciones que llaman las pantallas de la view y que se conectará con el repositorio, donde se gestionan los datos internamente.

  - Navegación:
    Usamos NavController para movernos entre las pantallas, pasando parámetros cuando sea necesario, y una sealed class Screens para definir las rutas.
  - Persistencia:

Usaremos la librería Room y su forma de trabajar para almacenar los contactos en la base de datos y que así se mantengan entre sesiones. Nuestra base de datos consistirá de una sola tabla Contact. Diseñaremos la interfaz DAO y las clases Database e Entity para poder definir el esquema de la tabla, sus métodos CRUD y los atributos de la tabla en sí.

- API:
  Usaremos la librería Retrofit, además de una interfaz de servicio de Api y una clase de respuesta de Api. Esto facilitará la conexión y carga de datos de la API, además de transformarlo en datos que Kotlin puede entender.

- Conectividad
  Implementamos clases que nos permiten comprobar el estado de red actual

- Gestión de errores:
  Debido a que nuestra aplicación depende de agentes externos (conexión a Internet y a la API) que no controlamos, hemos añadido código que recoge excepciones y valores en nulo a la hora de recoger datos de la API. De este modo, si la conexión a la API falla, asignamos una imagen por defecto y evitamos que la aplicación deje de funcionar.
  Adicionalmente, añadimos código relacionado con la navegación que evite volver a un estado anterior en los casos en los que genere problemas. Ej: Volver atrás al detalle de un contacto después de eliminarlo.

- Compartir:
  Generemos una apk de debug desde Android Studio para que la aplicación se pueda ejecutar fuera del entorno de desarrollo y así otros usuarios puedan ejecutarla.

- Conclusiones:

  A pesar de ser una aplicación relativamente simple, requiere una gran cantidad de trabajo que es difícil imaginarse cuando empiezas. Tienes que unir muchos sistemas distintos para que funcionen correctamente, comprobar una gran cantidad de posibles casos de uso y tener mil cosas en cuenta.
  Aunque fuera mucho esfuerzo, sentimos un inmenso orgullo de haberlo logrado y creemos que los conocimientos que tenemos ahora tras acabar son muy superiores a los de antes.