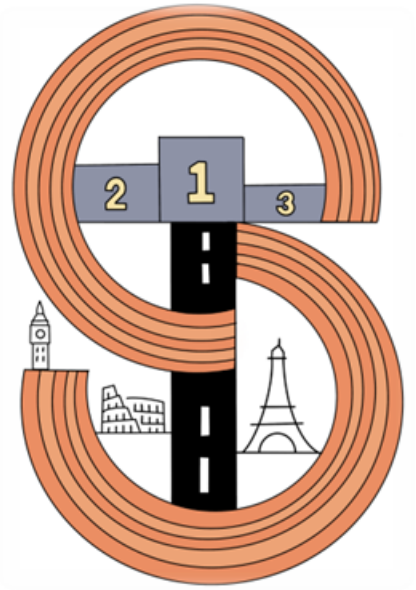


Proyecto

CODE

Participantes:

Brenda Castro Blanco
Cristina Comesaña Rodríguez

DESCRIPCIÓN INICIAL

Esta App Software contará con dos partes fundamentales:

- Frontend: Parte visible con la que interactúa el usuario final consumidor del contenido.
- Backend: Parte invisible con la que interactúa el usuario interno creador del contenido.

Será un agregador de contenido alrededor de un evento deportivo. Que va a solucionar necesidades, inquietudes..., del consumidor, como puede ser un deportista, turista, familiar....

Contará con:

- Marcadores con datos deportivos.
- Avisos importantes
- Alertas
- Personalización de las notificaciones
- Retroalimentación y feedback de los usuarios.

Podemos visualizar más gráficamente en un esquema que puede ser observado en el "Anexo 1".

CONTEXTO DEL PROYECTO

- Como objetivo inicial tenemos la creación de una app donde se registren eventos deportivos que serán visualizados por los usuarios que descarguen la aplicación pero que no tendrán permisos para la modificación de él.

- Para la realización del proyecto tomamos la decisión de introducirnos en el conocimiento de IDE de programación nuevo ya que será en app y un lenguaje nuevo que facilitará posteriormente la conexión con la base de datos y que hará más sencillo el modo de programación de la app.

- Para llegar al punto final realizaremos dos usuarios (que en la app no aparecen porque no nos dio tiempo de desarrollarlo, pero sería una de las optimizaciones de la aplicación).

  Un administrador y un usuario externo:

  - El administrador podrá incorporar a la aplicación los nuevos eventos con su fecha, nombre, lugar y su ID identificativo, además de poder modificarlos y eliminarlos.
  - Y el usuario externo simplemente tendrá una visualización de la aplicación con sus respectivos eventos y su registro pertinente que incorporaremos más adelante a la Base de Datos (seria creada con MONGODB, cosa que no nos dio tiempo de inicializarnos).

HERRAMIENTAS

IDE:

- Las herramientas que inicialmente decidimos utilizar son las siguientes:
    - Android Studio (IDE): es una herramienta que facilita el desarrollo de aplicaciones para la plataforma Android.

  Proporciona una interfaz gráfica para diseñar interfaces de usuario, escribir código, depurar y compilar aplicaciones.

  Incluye herramientas para emuladores, pruebas...

  Las características principales del IDE son:

    - Editor de código (autocompletado, sugerencias inteligentes...)
    - Diseñador de interfaces
    - Depuración de código
    - Emuladores para visualizar como se vería la aplicación en un dispositivo Android

  Permite utilizar dos lenguajes: Java y kotlin dando la opción de que si programas en java puedas convertir tu código a lenguaje kotlin.

  Además de tener Plugins y Extensiones pata mejorar la productividad.

LENGUAJE:

- El lenguaje utilizado es Kotlin:

Kotlin es un lenguaje de programación de código abierto creado por JetBrains. Se ha popularizado principalmente por su uso en el desarrollo de aplicaciones Android.

Sus características más notorias son:

- Lenguaje de tipo estático: Lo cual ayuda a la prevención de errores antes de la ejecución de la aplicación, ya que verifica el tipo de datos a la vez de la compilación.
- Multiplataforma: es decir permite escribir código una vez y ejecutarlo en diferentes plataformas.

Facilitando la creación de aplicaciones que se podrán poner en funcionamiento en varios dispositivos.

- Concisa y expresiva: Tiene incorporada una sintaxis concisa y legible, permitiendo escribir menos código y logrando las mismas funcionalidades que en Java.
- Interoperable con Java.

OTRAS HERRAMIENTAS:

- Lucid.app: para la creación del diagrama de usos
  https://lucid.app/lucidchart/invitations/accept/inv_2d0e7ce1-e9fc-4075-aec5-310525232b50.
- Creatly: Para la creación del diagrama de clases
  https://creately.com/es/planes/?ref=home
- PseInt: Para generar el pseudocódigo de la idea inicial
  https://pseint.sourceforge.net/
- Github: Para compartir el código con mi compañera de trabajo y con el evaluador del proyecto. Además de poder hacer cambios en el código desde otros dispositivos de trabajo sin que el código se deteriore y de error.
  https://github.com/
- Y para generar la documentación del proyecto utilizamos Documentos de Google.
  https://docs.google.com/document/d/18kByJHGUDns4XiO6IronDGtL3Lp1_r9zljsWS-5wb1E/edit?hl=es

PLANIFICACIÓN INICIAL

OBJETIVOS TÉCNICOS

CLASE MAIN:
- Menu

CLASE EVENTOS(clase padre):
ATRIBUTOS
- String nombreEvento;
- LocalDate fechaEvento;
- String lugarEvento;
- int id

CONSTRUCTORES
- Vacio
- Valores por defecto

TO STRING

METODOS ADICIONALES

- Método modificar
- Método añadir
- Método eliminar

CLASE INFORTURISTICA(clase hija):

ATRIBUTOS

- Atributos clase padre(nombreEvento, lugarEvento)
- String lugarInformacion;

CONSTRUCTOR

- (Heredado de la clase padre)

TO STRING

Nuestra app contará con una clase principal denominada "MAIN" que recibirá la información de los métodos de las otras clases creadas.
Esta clase contará con un menú en el cual el usuario dispondrá de diferentes elecciones para realizar lo de su interés. Esta clase será distinta dependiendo del usuario que entre, ya que si es un usuario externo(consumidor) solo podrá visualizar los eventos (to string de la clase requerida) con su determinada información pero no tendrá opción a modificar el evento.
Sin embargo si entra como un usuario de tipo administrador (que no quiere decir que es el programador) tiene derecho a modificar, añadir y eliminar eventos.
Posteriormente este usuario tendrá opción de visualizar el evento como usuario externo (to string).

Otra de las clases desarrollada en la aplicación es la clase "EVENTOS" que será la clase padre, contará con sus atributos propios (nombreEvento, lugarEvento, fechaEvento,id) , sus constructores(uno vacío y otro con los parámetros), no necesita getter ni setter ya que el lenguaje kotlin los genera automáticamente cuando una clase hija necesite la información de la clase, además tendrá un método "to string " que permitirá visualizar los datos del evento en pantalla (en este caso serie el emulador de android, pero se puede ver que funciona con el log cat sin necesidad de ejecutar el emulador todo el tiempo), esta clase contendrá los métodos propios (que utilizará el usuario administrador anteriormente nombrado), que son los siguientes:

- "anadirEvento": es un método que recibe los datos de el constructor salvo el id que se genera en una función a parte para utilizarlo como contador y que así cada evento nuevo tenga un id diferente (posteriormente comentaremos en las pruebas las mejoras que podríamos tener en esto y los errores que podrían causar), una vez generado ese id (como contador), se le añade los datos a su correspondiente id y en este caso se reconocería el evento para modificarlo o eliminarlo por el propio id que es único para ese evento.

- "modificarEvento": pediriamos mediante un mensaje al usuario administrador el id del evento que quiere modificar, por tanto si visualizamos el código recuperamos el id del evento determinado a modificar y le añadimos los datos nuevos mediante el método set que genera kotlin.

- "eliminar evento": pediríamos al usuario administrador por un mensaje visualizado en pantalla el id del evento a eliminar, y mediante una función

recuperaríamos el id del evento y mediante el método get eliminamos el evento deseado.

Otra clase creada es la clase "inforTuristica" que es una clase hija de "Eventos " que aparte de los atributos de la clase padre contendrá un atributo propio que es lugarInfor(String), contará de sus constructor (que visualizando el código podemos observar que lo recibe directamente de la herencia y añadiendo la nueva variable (atributo) no necesite un constructor propio) y el método tostring para poder visualizar la información que será recibida por parte del usuario administrador.

PSEUDOCÓDIGO

Como podemos observar en el "ANEXO 2" la validación de usuarios realizada a partir de una comprobación de DNI para saber si el usuario es correcto y posteriormente de una contraseña que determinara la entrada o no entrada del usuario a la aplicación.

DOCUMENTACIÓN

DIAGRAMA UML

Como podemos observar en el "ANEXO 3", tenemos un diagrama de clases en el que se puede ver las diferentes clases con los diferentes tipos de atributos, en la cual se especifican de qué tipo son (int, String, LocalDate) y si son públicos (+) o privados (-). También, se añaden los diferentes tipos de métodos que contiene cada clase y se especifica si son públicos (+) o privados (-). Además, podemos visualizar la interrelación entre ellos que vendrá especificada mediante flechas, en las cuales si son unidireccionales la información solo puede ir hacia la información de la flecha, en cambio si son bidireccional se comunican entre ellos.

Por otra parte, observamos que en nuestra aplicación, va a haber dos paquetes uno que contendrá el desarrollo completo de la aplicación y el otro que conectará la app con la base de datos mediante una clase nueva que hereda todos los atributos de eventos y se añadirá en la base de datos.

PRUEBAS

PLANIFICACIÓN INICIAL

Visualizando el código y mirando clase a clase ya tenemos idea de algunos errores que pueden llegar a surgir.

Para confirmarlo iremos prueba a prueba y analizando cada una de ellas viendo cada error que puede darse el caso.

Primero ejecutamos el código para visualizar si hay algún error inicial, esto es conocido como pruebas de caja blanca.
Para esto cabe explicar que en nuestro IDE no se puede visualizar por pantalla la ejecución para ver los errores (como si seria en el caso de otros IDES como Intellij), sin embargo está la opción del Logcat que es una herramienta de depuración que muestra en tiempo real todos los mensajes del sistema, estos pueden tener diferentes niveles de prioridad conocidos como filtros de mensaje (verbose, debug, info, warning, silent, error o fatal):

- V/verbose: es el nivel más bajo de prioridad, estos mensajes contienen información detallada y son útiles durante la fase de desarrollo y depuración.

- D/Debug: estos mensajes proporcionan información útil para la depuración de la app. Además, incluyen datos que pueden ayudar a los programadores a entender el flujo de ejecución y también a identificar problemas pequeños.

- I/Info: estos no indican errores, si no mensajes informativos, como puede ser indicar cuál es el estado de la aplicación en ese momento.

- W/Warning: estos mensajes advierten sobre posibles problemas que pueden darse pero que no van a impedir el funcionamiento de la aplicación.

- S/Silent: este nivel se utiliza para filtrar todos los mensajes y que solo salten los que son críticos y necesitan solución inmediata.

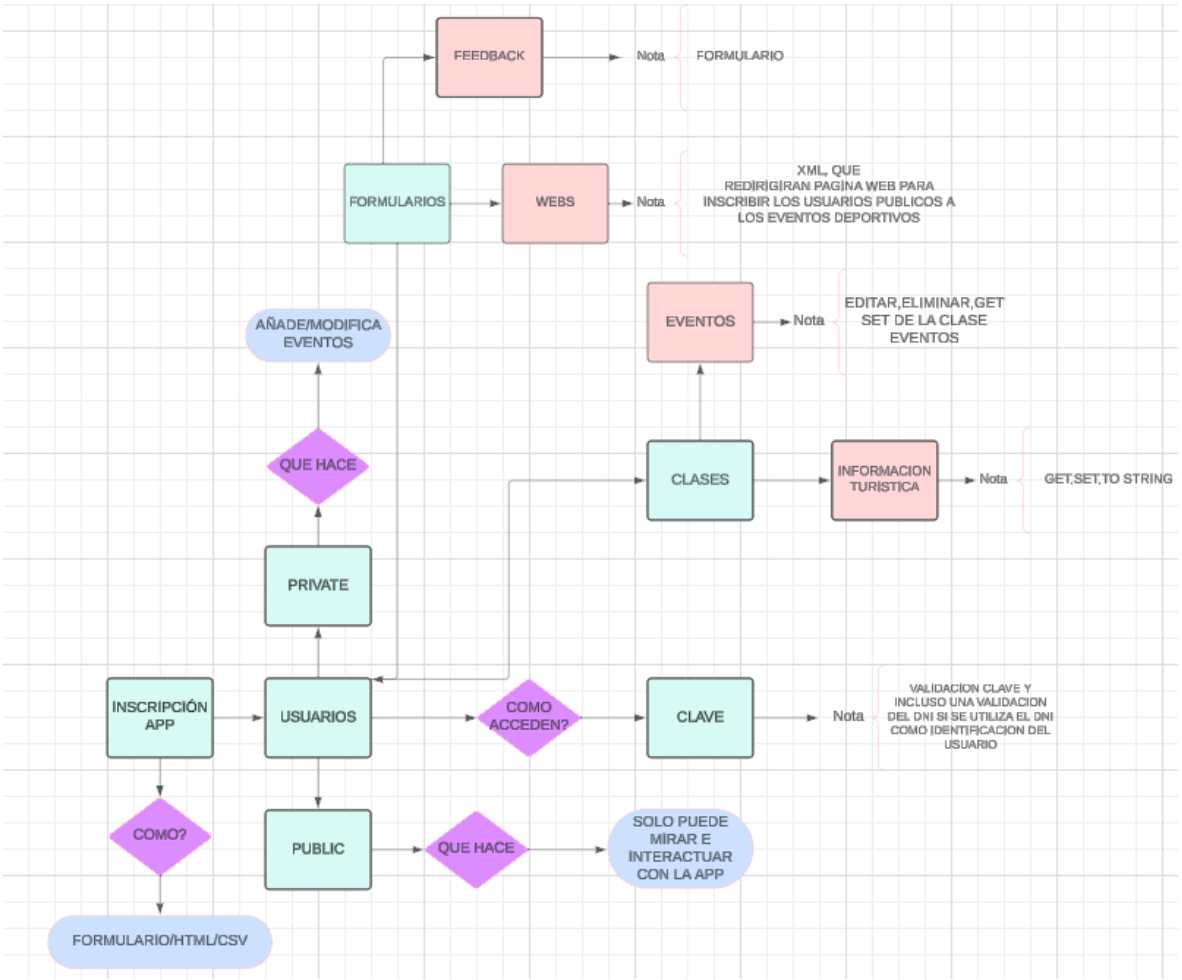
- E/Error: estos mensajes indican que ha ocurrido un error durante la ejecución de la aplicación, no la detienen pero puede darse el caso de que afecte a su buen funcionamiento, por lo que se necesita solucionar el error cuanto antes.

- F/Fatal: este tipo de mensajes indican que hay errores graves, los cuales hacen que la aplicación se detenga de forma inesperada. Suelen ser problemas que pueden llevar a la pérdida de datos.

ANEXOS

ANEXO 1

ESQUEMA IDEA PRINCIPAL:



(Se pueden ver más claros todos los detalles en el pdf que está situado en la carpeta de referencias compartida fuera de este documento)

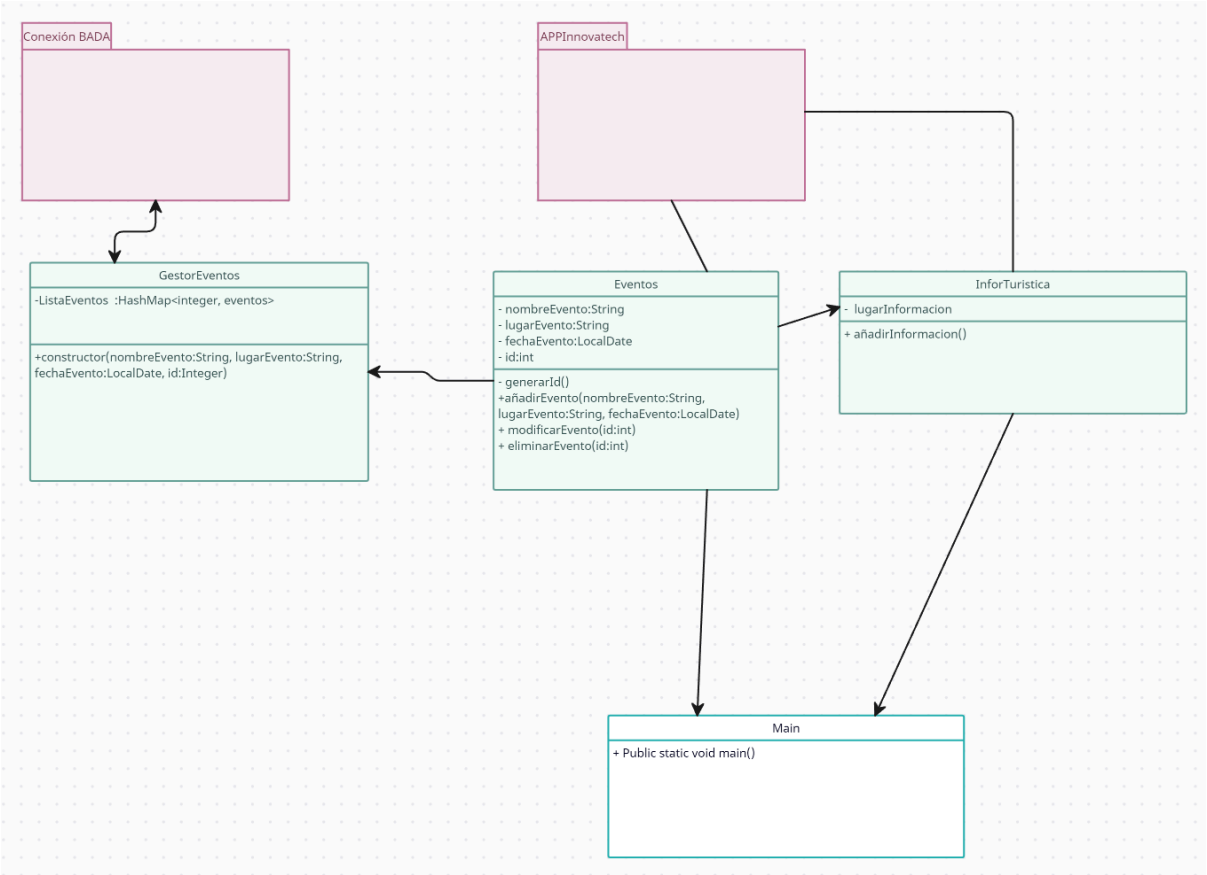
ANEXO 2

PSEUDOCÓDIGO:

```
1   Algoritmo usuarios
2       LEER DNI
3       Si DNI= FALSE Entonces
4   +       REPITE DNI
5       SiNo
6   +       PIDE CLAVE
7       Fin Si
8       Si CLAVE=FALSE Entonces
9   +       REPITE CLAVE
10      SiNo
11  +       ENTRA EN LA APP
12      Fin Si
13  FinAlgoritmo
14
```

ANEXO 3

DIAGRAMA DE CLASES:



ANEXO 4

DIAGRAMA DE CASOS DE USOS: